Chapter 6

Scheduling Simulations

This chapter presents a performance evaluation of economic-based Grid resource management and scheduling. The GridSim toolkit is used to develop an economic Grid resource broker that supports the deadline and budget constrained (DBC) scheduling strategies and to quantify the broker's ability to dynamically select resources at runtime depending on their availability, capability, cost, and user quality of service requirements (QoS). The broker supports DBC algorithms with the four different optimisation strategies—cost, time, cost-time, and conservative time. The detailed performance evaluation of economic-driven scheduling algorithms is carried out through a series of simulations by varying the number of users, deadline, budget, and optimisation strategies and simulating geographically distributed Grid resources.

6.1 Economic Grid Resource Broker Simulation

We used the GridSim toolkit to simulate a Grid environment and a Nimrod-G like deadline and budget constrained scheduling system called economic Grid resource broker. The simulated Grid environment contains multiple resources and user entities with different requirements. The users create an experiment that contains an application specification (a set of Gridlets that represent application jobs with different processing) and quality of service requirements (deadline and budget constraints with optimization strategy). We created two entities that simulate users and the brokers by extending the GridSim class. When simulated, each user entity having its own application and quality of service requirements creates its own instance of the broker entity for scheduling Gridlets on resources.

6.1.1 Broker Architecture

The broker entity architecture along with its interaction flow diagram with other entities is shown in Figure 6.1. The key components of the broker are: experiment interface, resource discovery and trading, scheduling flow manager backed with scheduling heuristics and algorithms, Gridlets dispatcher, and Gridlets receptor. The following high-level steps describe functionality of the broker components and their interaction:

1. The user entity creates an experiment that contains an application description (a list of Gridlets to be processed) and user requirements to the broker via the experiment interface.
2. The broker resource discovery and trading module interacts with the GridSim GIS entity to identify contact information of resources and then interacts with resources to establish their configuration and access cost. It creates a Broker Resource list that acts as placeholder for maintaining resource properties, a list of Gridlets committed for execution on the resource, and the resource performance data as predicted through the measurement and extrapolation methodology.
3. The scheduling flow manager selects an appropriate scheduling algorithm for mapping Gridlets to resources depending on the user requirements (deadline and budget limits; and optimisation strategy—cost, cost-time, time, or time variant). Gridlets that are mapped to a specific resource are added to the Gridlets list in the Broker Resource.
4. For each of the resources, the dispatcher selects the number of Gridlets that can be staged for execution according to the usage policy to avoid overloading resources with single user jobs.

5. The dispatcher then submits Gridlets to resources using the GridSim's asynchronous service.
6. When the Gridlet processing completes, the resource returns it to the broker's Gridlet receptor module, which then measures and updates the runtime parameter, resource share available to the user. It aids in predicting the job consumption rate for making scheduling decisions.
7. The steps, 3–6, continue until all the Gridlets are processed or the broker exceeds deadline or budget limits. The broker then returns the updated experiment data along with processed Gridlets back to the user entity.

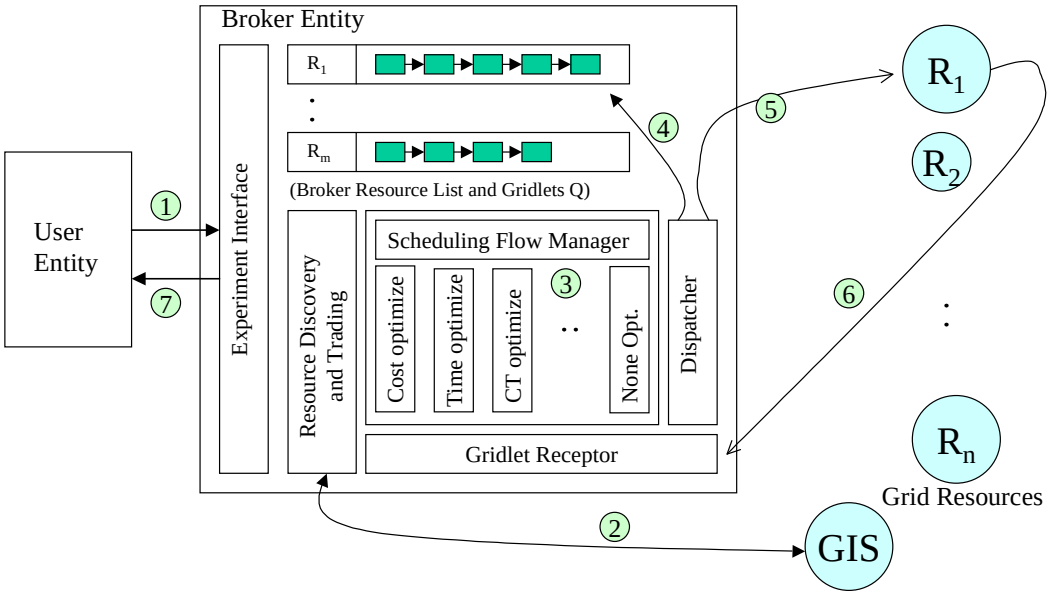


Figure 6.1: Economic Grid resource broker architecture and its interaction with other entities.

A class diagram hierarchy of the Grid broker package built using the GridSim toolkit is shown in Figure 6.2. The Grid broker package implements the following key classes:

`class Experiment` – It acts as a placeholder for representing simulation experiment configuration that includes synthesized application (a set of Gridlets stored in GridletList) and user requirements such as D and B-factors or deadline and budget constraints, and optimization strategy. It provides methods for updating and querying the experiment parameters and status. The user entity invokes the broker entity and passes its requirements via experiment object. On receiving an experiment from its user, the broker schedules Gridlets according to the optimization policy set for the experiment.

`class UserEntity` – A GridSim entity that simulates the user. It invokes the broker and passes the user requirements. When it receives the results of application processing, it records parameters of interest with the gridsim.Statistics entity. When it has no more processing requirements, it sends `END_OF_SIMULATION` event to the broker and gridsim.GridSimShutdown entities.

`class Broker` – A GridSim entity that simulates the Grid resource broker. On receiving an experiment from the user entity, it does resource discovery, and determines deadline and budget values based on D and B factors, and then proceeds with scheduling. It schedules Gridlets on resources depending on user constraints, optimization strategy, and cost of resources and their availability. When it receives the results of application processing, it records parameters of interest with the gridsim.Statistics entity. When it has no more processing requirements, it sends `END_OF_SIMULATION` event to the gridsim.GridSimShutdown entity.

`class BrokerResource` – It acts as placeholder for the broker to maintain a detailed record of the resources it uses for processing user application. It maintains resource characteristics, a list of Gridlets assigned to the resource, the actual amount of MIPS available to the user, and a report on the Gridlets processed. These measurements help in extrapolating and predicting the resource performance from the user point of view and aid in scheduling jobs dynamically at runtime.

`class ReportWriter` – A user-defined, optional GridSim entity which is meant for creating a report at the end of each simulation by interacting with the gridsim.Statistics entity. If the user does not want to create a report, then it can pass "`null`" as the name of the ReportWriter entity. Note that the users can choose any name for the ReportWriter entity and for the class name since all entities are identified by their name defined at the runtime.

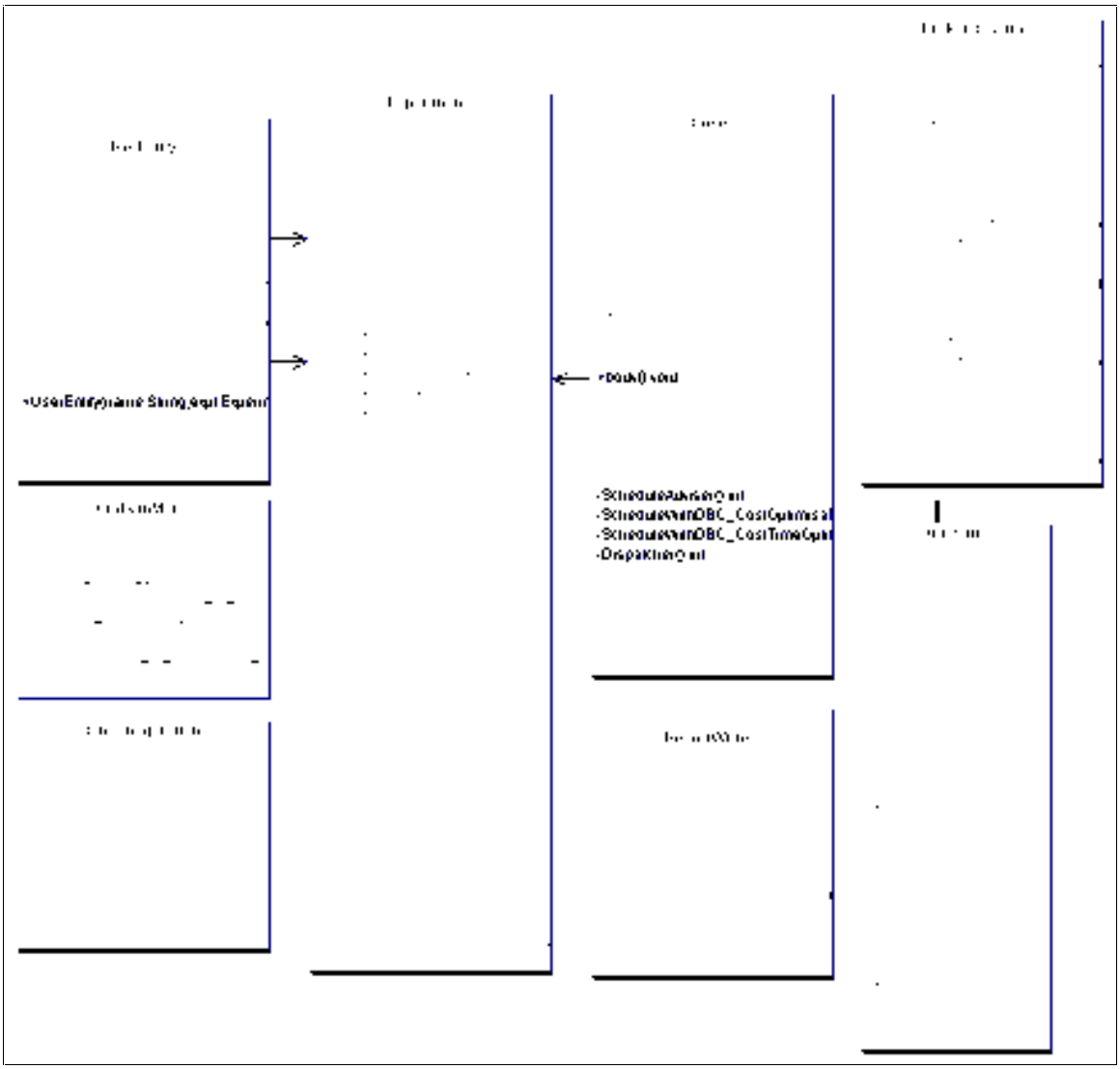


Figure 6.2: A class hierarchy diagram of Grid broker using the gridsim package.

An interactive class hierarchy diagram of the economic Grid resource broker (accessible from [104]) provides syntax and semantic information of data members and methods of each class discussed above.

6.1.2 Determining the Deadline and Budget

A D-factor close to 1 signifies the user's willingness to set a highly relaxed deadline, which is sufficient to process applications even when only the slowest resources are available. Similarly a B-factor close to 1 signifies that the user is willing to spend as much money as required even when only the most expensive resource is used. The jobs are scheduled on the Grid through user's broker. The broker uses these factors in determining the absolute deadline (see Equation 6.1) and budget (see Equation 6.2) values for a given execution scenario at runtime as follows:

Determining the Absolute Deadline Value:

$$Deadline = T_{MIN} + D_{FACTOR} * (T_{MAX} - T_{MIN})$$

Equation 6.1

where,

- T_{MIN} = the time required to process all the jobs, in parallel, giving the *fastest* resource the highest priority.
- T_{MAX} = the time required to process all the jobs, serially, using the slowest resource.
- An application with $D_{FACTOR} < 0$ would **never** be completed.
- An application with $D_{FACTOR} \geq 1$ would **always** be completed as long as some resources are available with minimal user-share throughout the deadline.

Determining the Absolute Budget Value:

$$Budget = C_{MIN} + B_{FACTOR} * (C_{MAX} - C_{MIN})$$

Equation 6.2

where,

- C_{MIN} = the cost of processing all the jobs, in parallel within deadline, giving the *cheapest* resource the highest priority.
- C_{MAX} = the cost of processing all the jobs, in parallel within deadline, giving the *costliest* resource the highest priority.
- An application with $B_{FACTOR} < 0$ would **never** be completed.
- An application with $B_{FACTOR} \geq 1$ would **always** be completed as long as some resources are available with minimal user-share throughout the deadline.

6.1.3 Scheduling Algorithms

We propose deadline and budget constrained (DBC) algorithms with four different optimisation strategies—cost optimisation, cost-time optimisation, time optimisation, and conservative-time optimisation—for scheduling task-farming applications on geographically distributed resources. The properties of DBC scheduling algorithms are shown in Table 6.1.

Table 6.1: Deadline and budget constrained adaptive scheduling algorithms.

Algorithm/ Strategy	Execution Time (Deadline, D)	Execution Cost (Budget, B)
Cost Opt	Limited by D	Minimize
Cost-Time Opt	Minimize when possible	Minimize
Time Opt	Minimize	Limited by B
Conservative-Time Opt	Minimize	Limited by B, but all unprocessed jobs have guaranteed minimum budget

The *cost-optimisation* scheduling algorithm uses the cheapest resources to ensure that the deadline can be met and the computational cost is minimized. The *time-optimisation* scheduling algorithm uses all the affordable resources to process jobs in parallel as early as possible. The *cost-time optimisation* scheduling is similar to cost optimisation, but if there are multiple resources with the same cost, it applies time optimisation strategy while scheduling jobs on them. The *conservative-time optimisation* scheduling

strategy is similar to the time-optimisation scheduling strategy, but it guarantees that each unprocessed job has a minimum budget-per-job.

We have incorporated DBC cost, time, and conservative time optimisation scheduling algorithms into the Nimrod-G broker and explored their ability in scheduling parameter sweep applications on the World-Wide Grid (WWG) testbed. A detailed evaluation of DBC cost, time, and cost-time scheduling algorithms by simulation for various scenarios is presented in the next sections.

6.2 Simulation Experiment Setup

To simulate application scheduling in GridSim environment using the economic Grid broker requires the modeling and creation of GridSim resources and applications that model jobs as Gridlets. In this section, we present resource and application modeling along with the results of experiments with quality of services driven application processing.

Table 6.2: WWG testbed resources simulated using GridSim.

Resource Name in Simulation	Simulated Resource Characteristics Vendor, Resource Type, Node OS, No of PEs	Equivalent Resource in Worldwide Grid (Hostname, Location)	A PE SPEC/ MIPS Rating	Resource Manager Type	Price (G$/PE time unit)	MIPS per G$
R0	Compaq, AlphaServer, CPU, OSF1, 4	grendel.vpac.org, VPAC, Australia	515	Time-shared	8	64.37
R1	Sun, Ultra, Solaris, 4	hpc420.hpcc.jp, AIST, Tokyo, Japan	377	Time-shared	4	94.25
R2	Sun, Ultra, Solaris, 4	hpc420-1.hpcc.jp, AIST, Tokyo, Japan	377	Time-shared	3	125.66
R3	Sun, Ultra, Solaris, 2	hpc420-2.hpcc.jp, AIST, Tokyo, Japan	377	Time-shared	3	125.66
R4	Intel, Pentium/VC820, Linux, 2	barbera.cnuce.cnr.it, CNR, Pisa, Italy	380	Time-shared	2	190.0
R5	SGI, Origin 3200, IRIX, 6	onyx1.zib.de, ZIB, Berlin, Germany	410	Time-shared	5	82.0
R6	SGI, Origin 3200, IRIX, 16	Onyx3.zib.de, ZIB, Berlin, Germany	410	Time-shared	5	82.0
R7	SGI, Origin 3200, IRIX, 16	mat.ruk.cuni.cz, Charles U., Prague, Czech Republic	410	Space-shared	4	102.5
R8	Intel, Pentium/VC820, Linux, 2	marge.csm.port.ac.uk, Portsmouth, UK	380	Time-shared	1	380.0
R9	SGI, Origin 3200, IRIX, 4 (accessible)	green.cfs.ac.uk, Manchester, UK	410	Time-shared	6	68.33
R10	Sun, Ultra, Solaris, 8,	pitcairn.mcs.anl.gov, ANL, Chicago, USA	377	Time-shared	3	125.66

6.2.1 Resource Modeling

We modeled and simulated a number of time- and space-shared resources with different characteristics, configuration, and capability as those in the WWG testbed. We have selected the latest CPUs models AlphaServer ES40, Sun Netra 20, Intel VC820 (800EB MHz, Pentium III), and SGI Origin 3200 1X 500MHz R14k released by their manufacturers Compaq, Sun, Intel, and SGI respectively. The processing capability of these PEs in simulation time-unit is modeled after the base value of SPEC CPU (INT) 2000 benchmark ratings published in [128]. To enable the users to model and express their application processing requirements in terms of MI (million instructions) or MIPS (million instructions per second), we *assume the MIPS rating of PEs is same as the SPEC rating*.

Table 6.2 shows characteristics of resources simulated and their PE cost per time unit in G$ (Grid dollar). These simulated resources resemble the WWG testbed resources used in processing a parameter sweep application using the Nimrod-G broker [102]. The PE cost in G$/unit time does not necessarily reflects the cost of processing when PEs have different capability. The brokers need to translate it into the G$ per MI (million instructions) for each resource. Such translation helps in identifying the relative cost of resources for processing Gridlets on them.

6.2.2 Application Modeling

We have modeled a task farming application that consists of 200 jobs. In GridSim, these jobs are packaged as Gridlets whose contents include the job length in MI, the size of job input and output data in bytes along with various other execution related parameters when they move between the broker and resources. The job length is expressed in terms of the time it takes to run on a standard resource PE with SPEC/MIPS rating of 100. Gridlets processing time is expressed in such a way that they are expected to take at least 100 time-units with a random variation of 0 to 10% on the positive side of the standard resource. That means, Gridlets' job length (processing requirements) can be at least 10,000 MI with a random variation of 0 to 10% on the positive side. This 0 to 10% random variation in Gridlets' job length is introduced to model heterogeneity in different tasks.

Algorithm: DBC_Scheduling_with_Cost_Optimisation()

1. RESOURCE DISCOVERY: Identify resources that can be used in this execution with their capability through the Grid Information Service.
2. RESOURCE TRADING: Identify cost of each of the resources in terms of CPU cost per second and capability to be delivered per cost-unit.
3. If the user supplies D and B factors, then determine the absolute deadline and budget based on the capability and cost of resources and user's requirements.
4. SORT resources by increasing order of cost.
5. SCHEDULING: Repeat while there exist unprocessed jobs in application job list with a delay of scheduling event period or occurrence of an event AND the time and process expenses are within deadline and budget limits:

[SCHEDULE ADVISOR with Policy]

   a. For each resource perform load profiling to establish the job consumption rate or the available resource share through measure and extrapolation.
   b. For each resource based on its job consumption rate or available resource share, predict and establish the number of jobs a resource can process by the deadline.
   c. For each resource in order:
      i. If the number of jobs currently assigned to a resource is less than the predicted number of jobs that a resource can consume, assign more jobs from unassigned job queue or from the most expensive machines based on job state and feasibility. Assign job to a resource only when there is enough budget available.
      ii. Alternatively, if a resource has more jobs than it can complete by the deadline, move those extra jobs to unassigned job queue.

6. [DISPATCHER with Policy]

Repeat the following steps for each resource if it has jobs to be dispatched:

- Identify the number of jobs that can be submitted without overloading the resource. Our default policy is to dispatch jobs as long as the number of user jobs deployed (active or in queue) is less than the number of PEs in the resource.

Figure 6.3: Deadline and budget constrained (DBC) scheduling with cost-optimization.

6.3 Deadline and Budget Constrained Cost Optimisation Scheduling

The steps for implementing DBC cost-optimisation scheduling algorithms within economic broker simulator are shown in Figure 6.3. This algorithm attempts to process jobs as economically as possible within the deadline and budget.

6.3.1 Scheduling Experiments with a Single User

In this experiment, we perform scheduling experiments with different values of deadline and budget constraints (DBC) for a single user. The deadline is varied in simulation time from 100 to 3600 in steps of 500. The budget is varied from G$ 5000 to 22000 in steps of 1000. For this scenario, we performed scheduling simulation for DBC *cost-optimization* algorithm. The number of Gridlets processed, deadline utilized, and budget spent for different scheduling scenario is shown in Figure 6.4–Figure 6.7. From Figure 6.4, it can be observed that for a tight deadline (e.g., 100 time unit), the number of Gridlets processed increased with the increase in budget value. Because, when a higher budget is available, the broker leases expensive resources to process more jobs within the deadline. Alternatively, when scheduling with a low budget value, the number of Gridlets processed increases as the deadline is relaxed (see Figure 6.5).

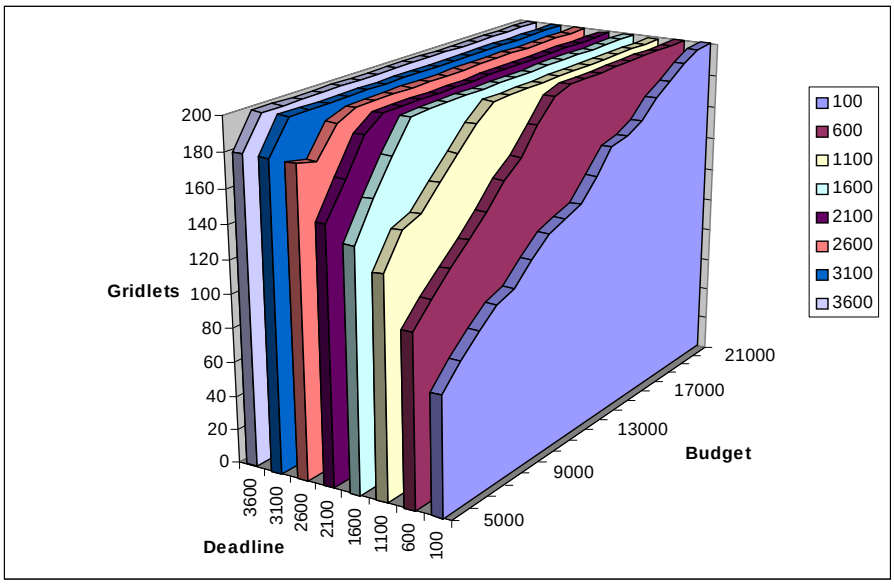


Figure 6.4: No. of Gridlets processed for different budget limits with a fixed deadline for each.

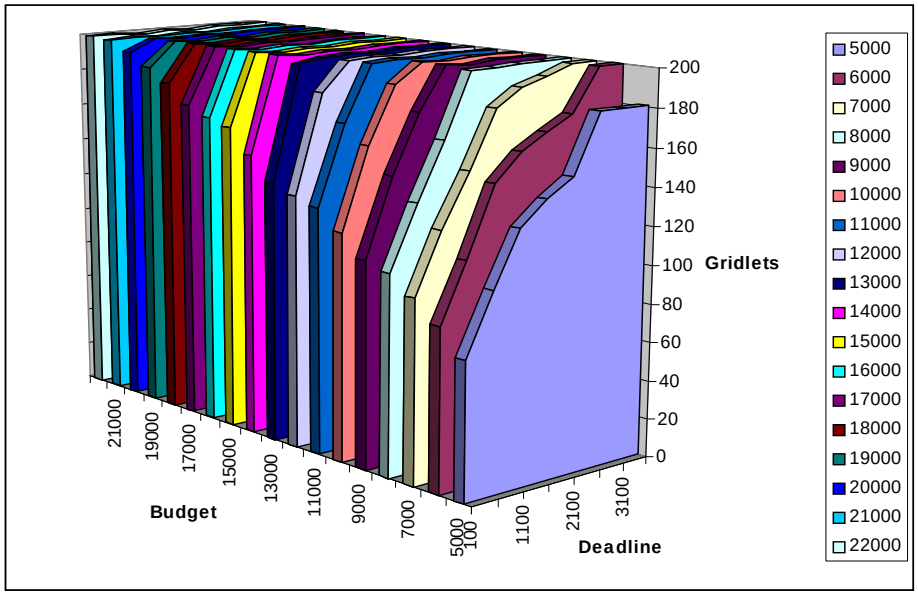


Figure 6.5: No. of Gridlets processed for different deadline limits with a fixed budget for each.

The impact of budget for different values of deadline is shown in Figure 6.6. In cost-optimization scheduling, for a larger deadline value (see time utilization for deadline of 3600), the increase in budget value does not have much impact on resource selection. This trend can also be observed from the budget spent for processing Gridlets with different deadline constraints (see Figure 6.7). When the deadline is too tight (e.g., 100), it is likely that the complete budget is spent for processing Gridlets within the deadline.

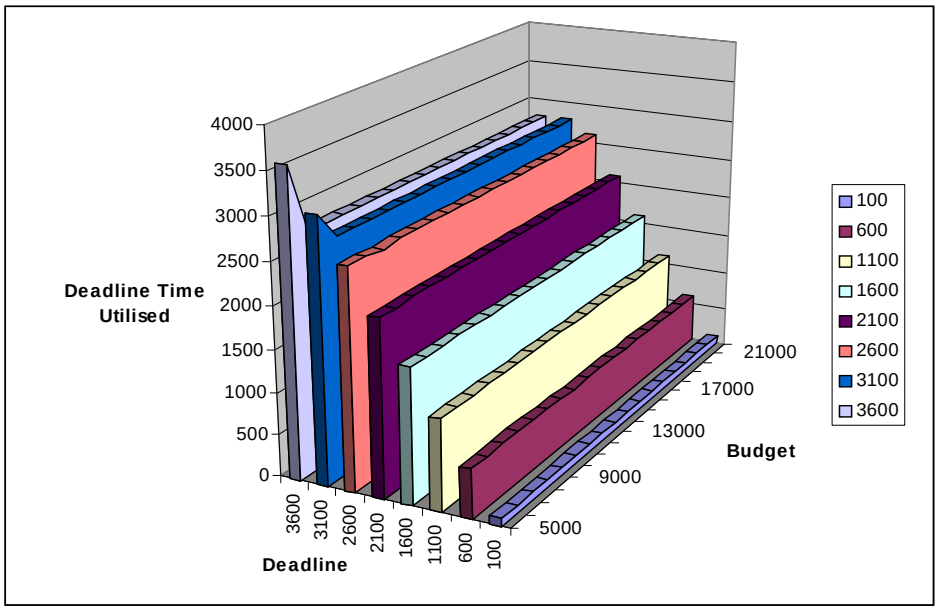


Figure 6.6: Deadline time utilized for processing Gridlets for different values of deadline and budget.

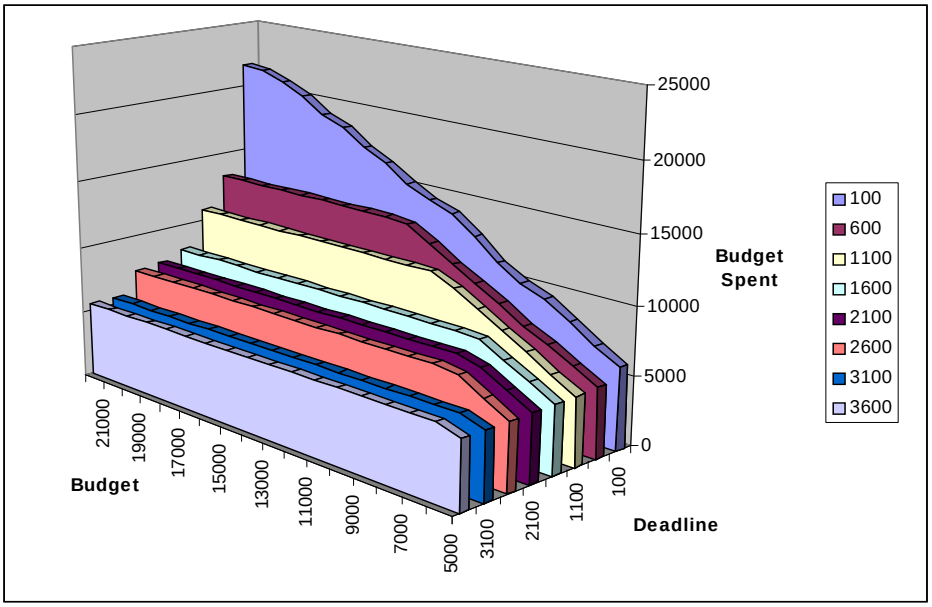


Figure 6.7: Budget spent for processing Gridlets for different values of deadline and budget.

Three diagrams (Figure 6.8–Figure 6.10) show the selection of resources for processing Gridlets for different budget values with a fixed deadline of 100, 1100, and 3100 (short, medium, and long deadline value) respectively. It can be observed that when the deadline is low, the economic broker also leases expensive resources to process Gridlets whenever the budget permits (see Figure 6.8). In this, all resources have been used depending on the budget availability. When the deadline is increased to a high value (a medium deadline of 1100), the broker processes as many Gridlets as possible on cheaper resources by the deadline (see Figure 6.9) and utilizes expensive resources if required. When the deadline is highly relaxed (a long deadline of 3100), the broker allocates Gridlets to the cheapest resource since it was able to process all Gridlets within this deadline (see Figure 6.10). In all three diagrams (Figure 6.8 –Figure 6.10), the left most solid curve marked with the label "All" in the resources axis represents the aggregation of all resources and shows the total number of Gridlets processed for the different budgets.

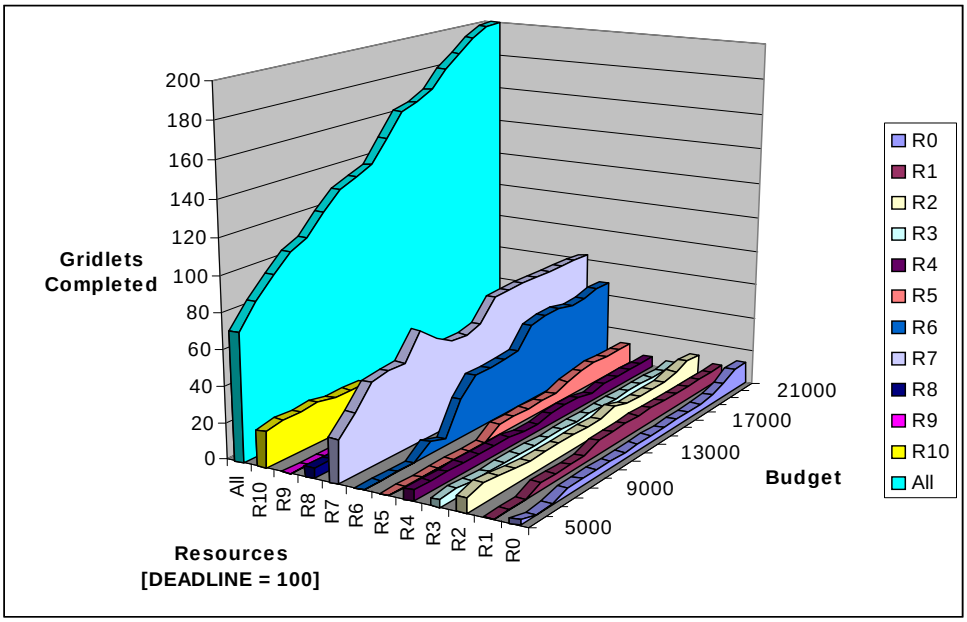


Figure 6.8: Gridlets processed on resources for different budget values with short deadline.

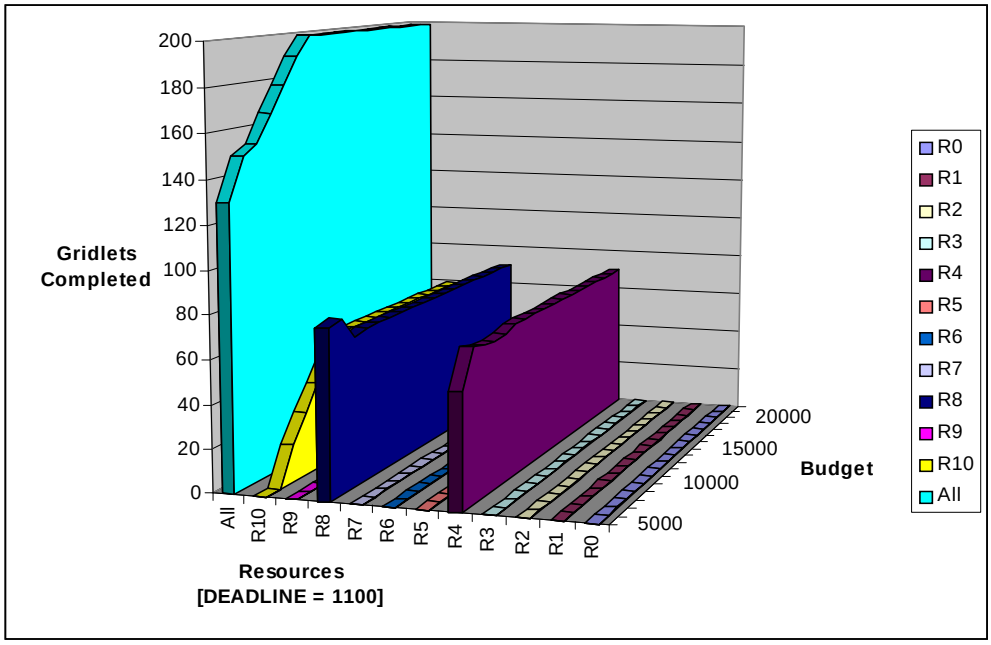


Figure 6.9: Gridlets processed on resources for different budget values with medium deadline.

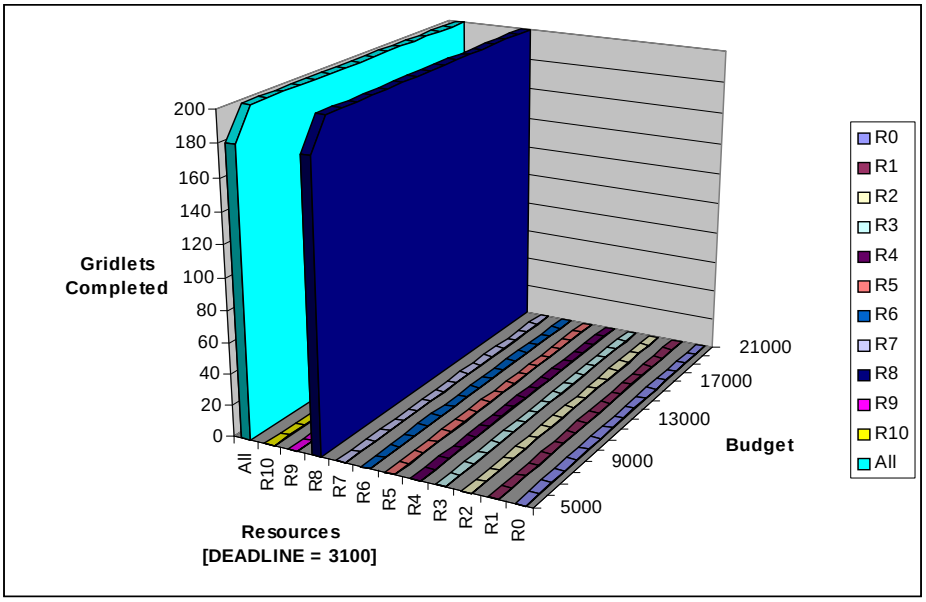


Figure 6.10: Gridlets processed on resources for different budget values with long deadline.

Let us now take a microscopic look at the allocation of resources at different times during the scheduling experimentation. The two graphs (Figure 6.11, Figure 6.13, and Figure 6.14) show a trace of leasing resources at different times during the scheduling experiment for processing Gridlets for different budget values with a fixed deadline of 100, 1100, and 3100 (short, medium, and long deadline values) respectively. It can be observed that when the deadline value is low, the economic broker also leases expensive resources to process Gridlets whenever the budget permits. The broker had to allocate powerful resources even if they are expensive since the deadline is too tight (see Figure 6.11 for Gridlets completed and Figure 6.12 for budget spent in processing). But this is not the case when the deadline is highly relaxed (see Figure 6.14)—the broker leased just one resource, which happened to process all Gridlets within the given deadline. From the diagrams (Figure 6.11 and Figure 6.12), it can be observed that the resource R7 has processed more

Gridlets than the resource R6, but had to spend more budget on the resource R6 since it is more expensive than the resource R7.

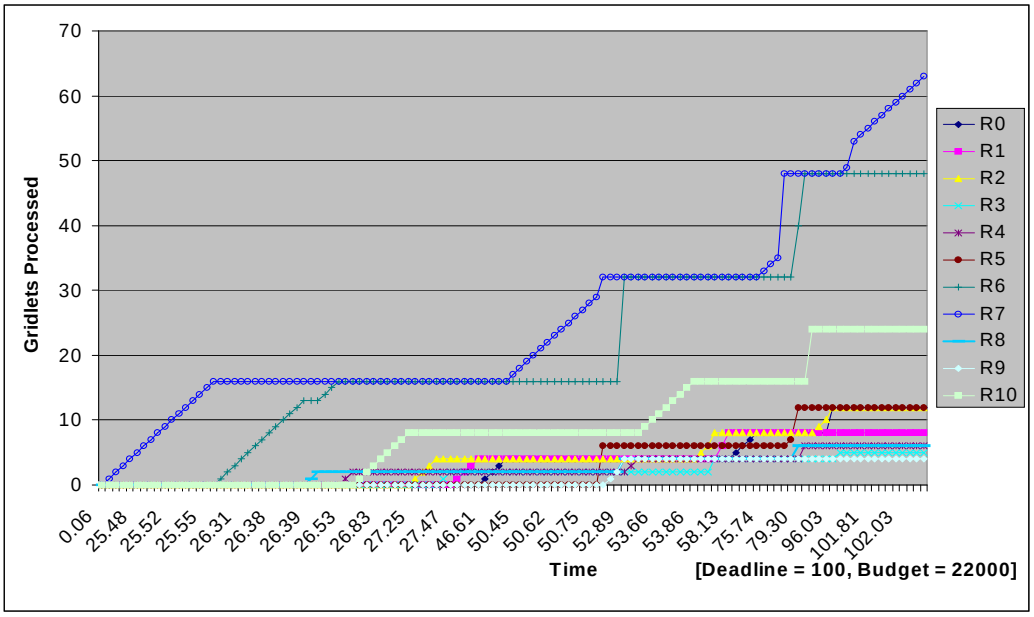


Figure 6.11: Trace of No. of Gridlets processed for a short deadline and high budget constraints.

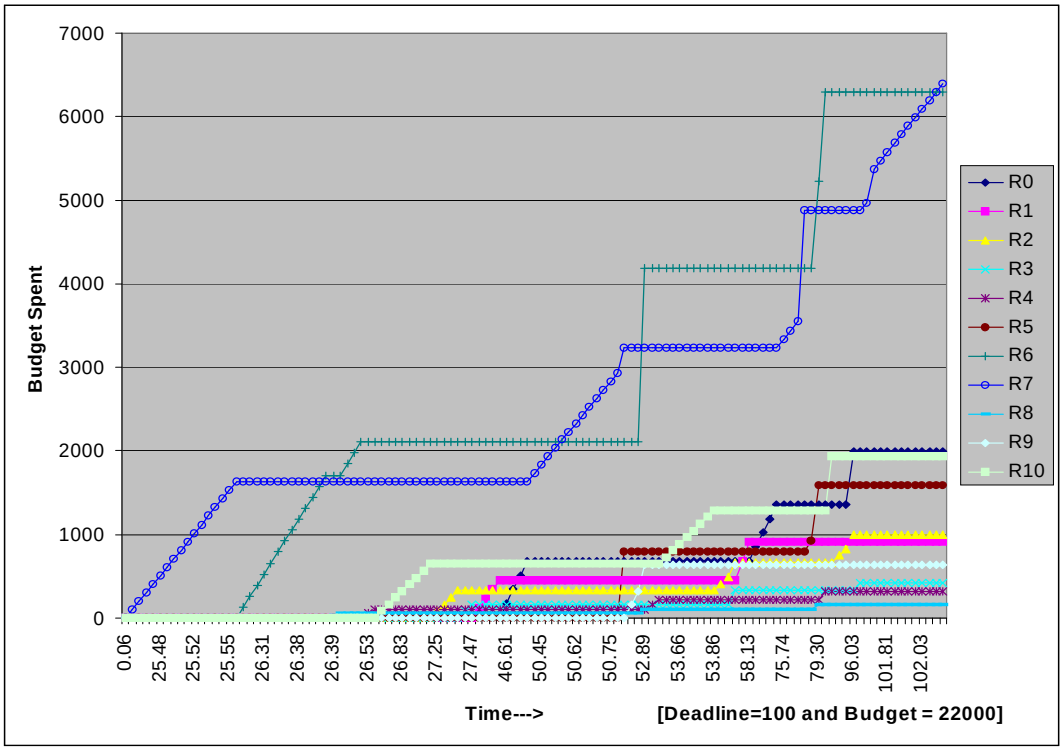


Figure 6.12: Trace of budget spent for short deadline and high budget constraints.

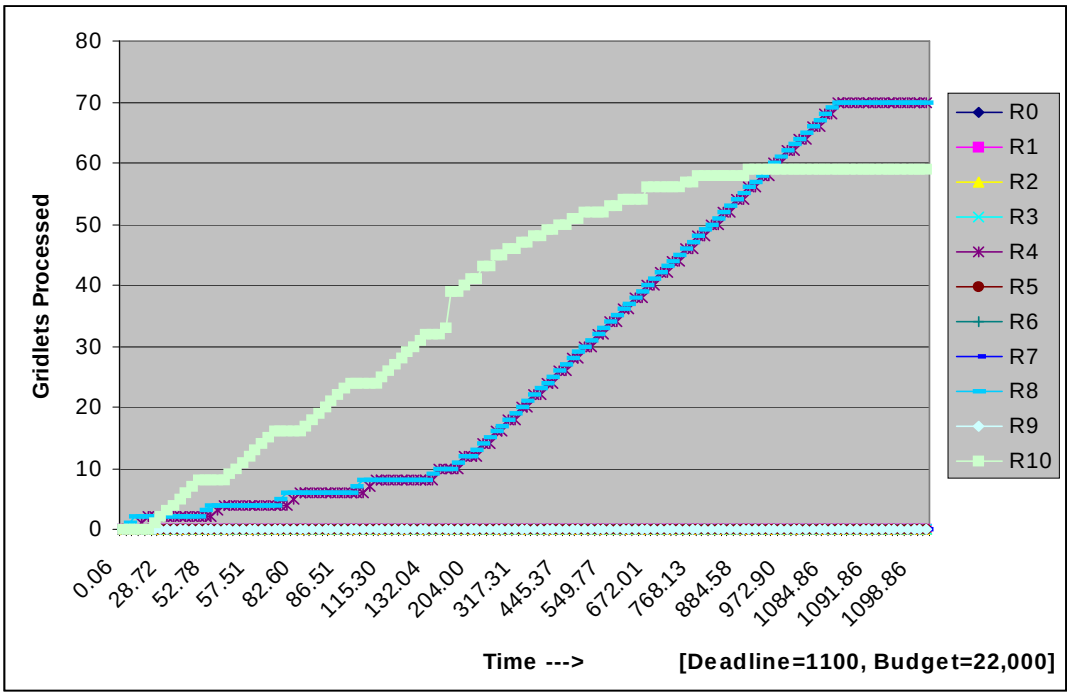


Figure 6.13: Trace of No. of Gridlets processed for a medium deadline and low budget constraints.

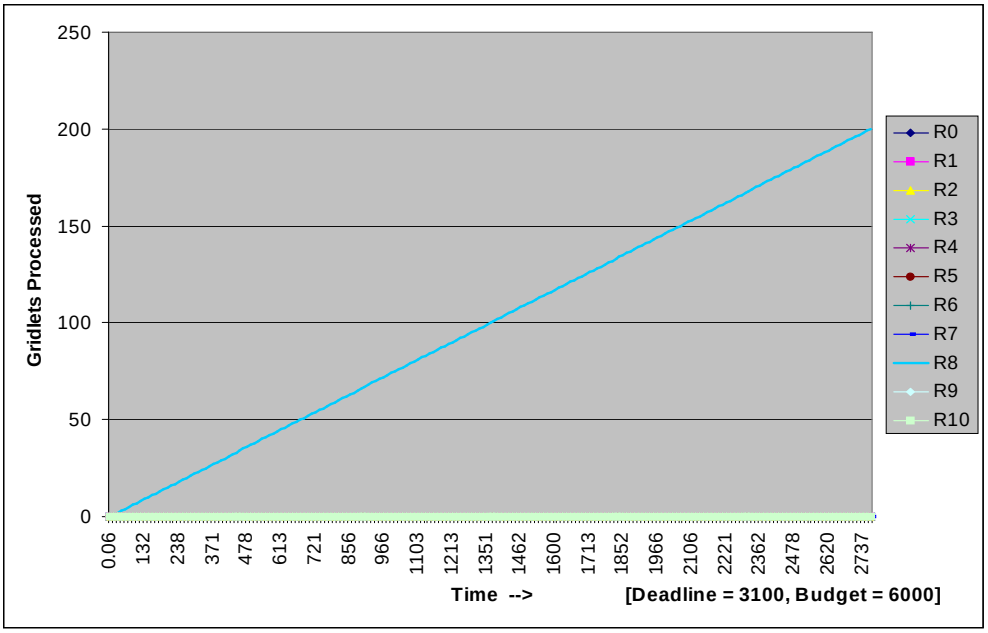


Figure 6.14: Trace of No. of Gridlets processed for a long deadline and low budget constraints.

A trace of the number of Gridlets committed to resources at different times depending on their performance, cost, and the user constraints (deadline and budget) and optimization requirements is shown in Figure 6.15 and Figure 6.16 for deadline values of 100 and 1100 time units respectively. In both graphs it can be observed the broker committed Gridlets to expensive resources only when it is required. It committed as many Gridlets as the cheaper resources can process by the deadline. The remaining Gridlets were assigned to expensive resources. The broker used expensive resources in the beginning and continued to use cheaper resources until the end of the experiment. This ability of economic Grid broker to select

resources dynamically at runtime demonstrates its adaptive capability driven by the user's quality of service requirements.

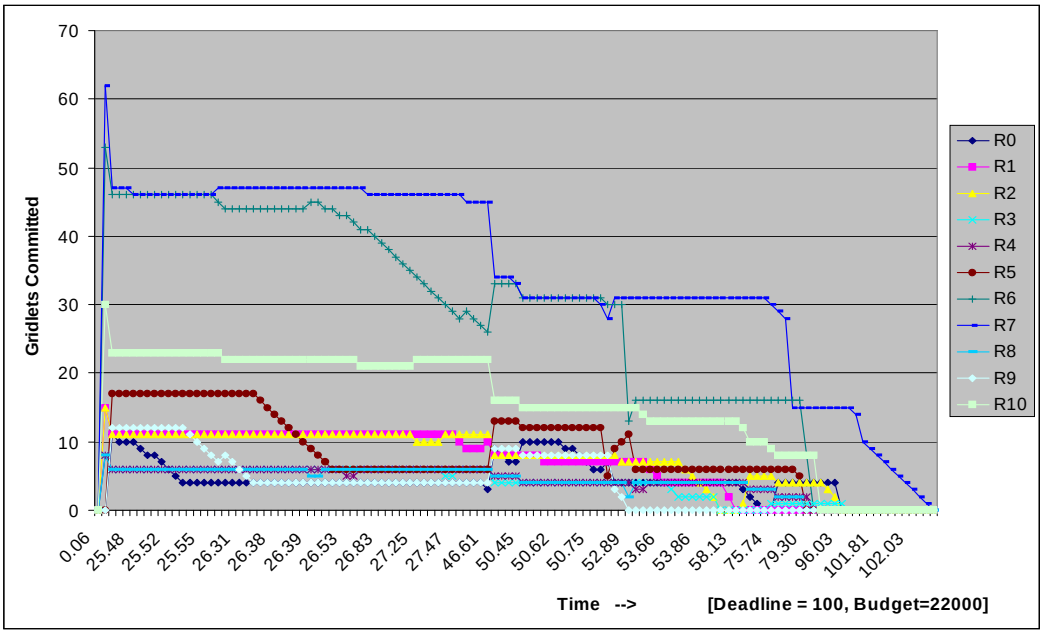


Figure 6.15: Trace of the no. of Gridlets committed for a short deadline and high budget constraints.

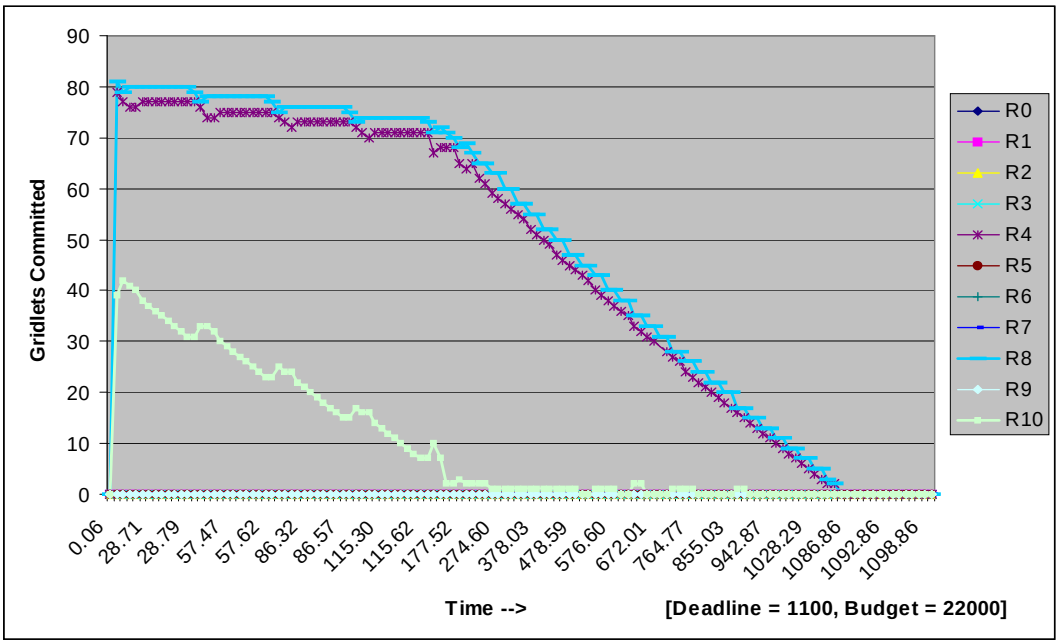


Figure 6.16: Trace of the no. of Gridlets committed for a medium deadline and high budget constraints.

6.3.2 Scheduling Experiments with Multiple Competing Users

In the second experiment, we explore distributed economic scheduling for a varying number of users competing for the same set of resources using the DBC constrained cost-optimisation scheduling algorithm. All users are modeled to have similar requirements to enable comparison among them and understand the overall scenario. Each user application contains 200 Gridlets with small variation as explained in

application modeling section. We modeled varying number of users in series from 1, 10, 20, and so on up to 100 and each with their own broker scheduling Gridlets on simulated WWG testbed resources (listed in Table 6.2). We explored scheduling of Gridlets for different budget values varied from 5000 to 22000 in step of 1000. For this scenario, we performed two scheduling experiments with two different values of deadline for DBC constrained *cost minimization* algorithm.

User Deadline = 3100 time units

The number of Gridlets processed, average time at which simulation is stopped, and budget spent for different scheduling scenario for each user with the deadline constraint of 3100 time units is shown in Figure 6.17, Figure 6.18, and Figure 6.19. From Figure 6.17, it can be observed that as the number of users competing for the same set of resources increase, the number of Gridlets processed for each user is decreasing because they have tight deadline. Whether there are few users (e.g., 1 or 10 users in this case), they are able to process all jobs in most cases when the budget is increased. Figure 6.18 shows the time at which broker terminated processing of Gridlets. When a large number of users are competing (e.g., 100) for resources, it can be observed that the broker exceeded the deadline. This is because the broker initially planned scheduling Gridlets for the period of deadline, but that schedule had to be terminated because competing users had already occupied high resource share well before the recalibration phase (the first establishment of the amount of resource share available to the user, which of course can change). Figure 6.19 shows the average budget spent by each user for processing Gridlets shown in Figure 6.17, which is also clear from the graphic similarity between both diagrams when a large number of users are competing for resources.

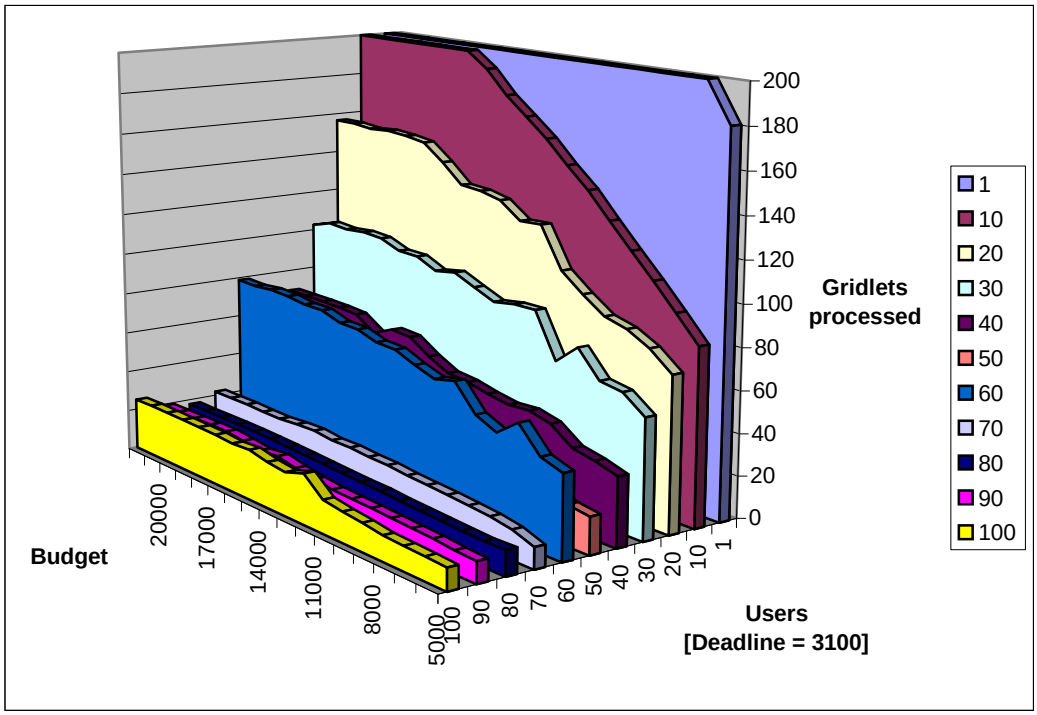


Figure 6.17: No. of Gridlets processed for each user when a varying number of users competing for resources.

When there are a large number of users arriving at different times, they are likely to impact on the schedule and the execution time of jobs already deployed on resources. The broker waiting for the return of jobs that are deployed on resources leads to the termination time exceeding the soft deadline, unless the execution of jobs is cancelled immediately.

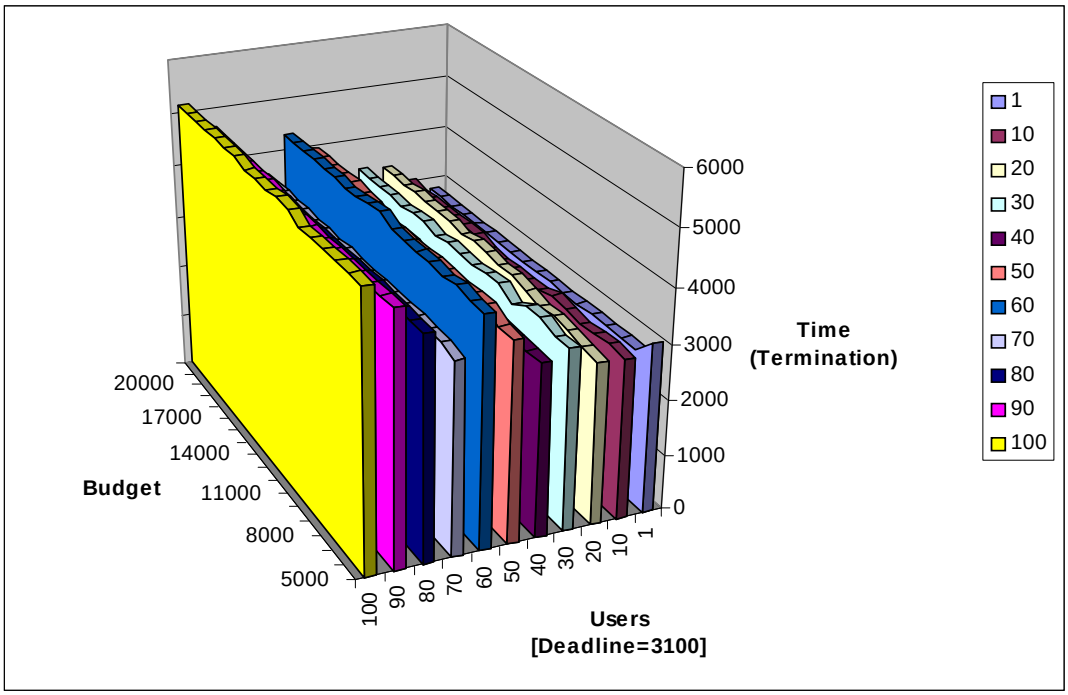


Figure 6.18: The average time at which the user experiment is terminated when a number of users are competing for resources.

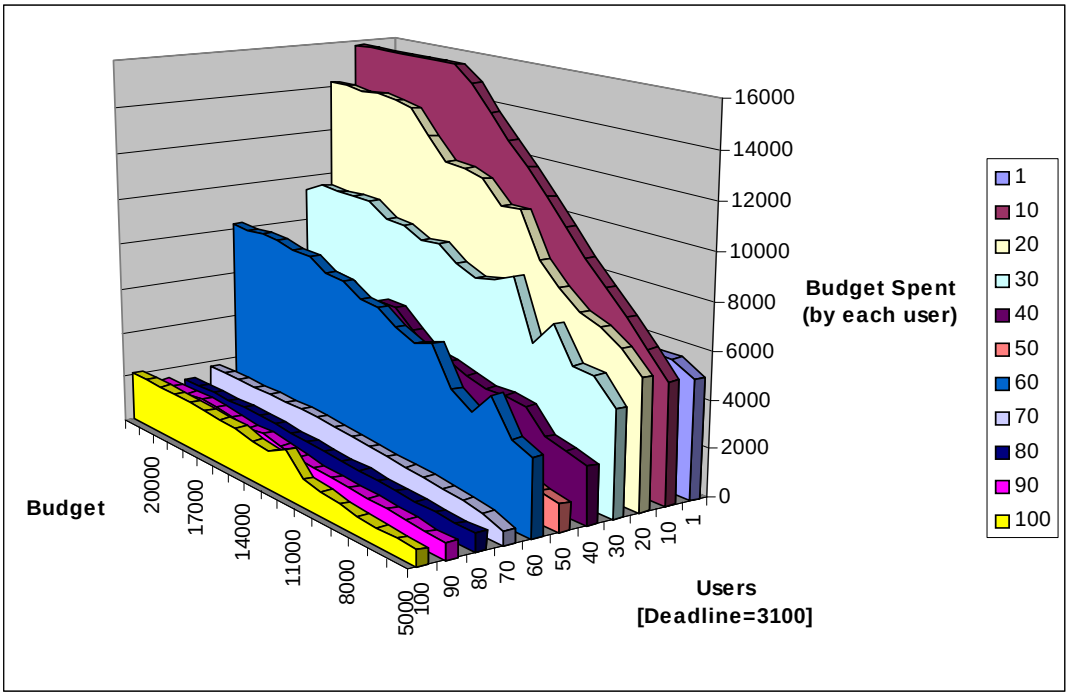


Figure 6.19: The average budget spent by each user for processing Gridlets.

User Deadline = 10000 time units

The number of Gridlets processed, average time at which simulation is stopped, and budget spent for different scheduling scenario for each user with the deadline constraint of 10000 time units is shown in

Figure 6.20, Figure 6.21, and Figure 6.22. In this experiment, the number of Gridlets processed for each user improved substantially due to the relaxed deadline constraint compared to the previous experiment (see Figure 6.17 and Figure 6.20). As the number of users competing for resources increased, the number of Gridlets processed for each user decreased. But when the budget is increased, the number of Gridlets processed increased as well. Unlike the previous experiment, the broker is able to learn and make better predictions on the availability of resource share and the number of Gridlets that can be finished by deadline. As the deadline was sufficient enough to revisit the past scheduling decisions, the broker is able to ensure that the experiment is terminated within the deadline for most of the time (see Figure 6.21). The average budget spent by each user for processing Gridlets is shown in Figure 6.22, which is also clear from the graphic similarity between Figure 6.20 and Figure 6.22 when a large number of users are competing for resources. However, up to the medium number of users (1-40 users), they were able to get many Gridlets processed, but decreased with the increasing number of users competing for resources, which means the increase in processing cost.

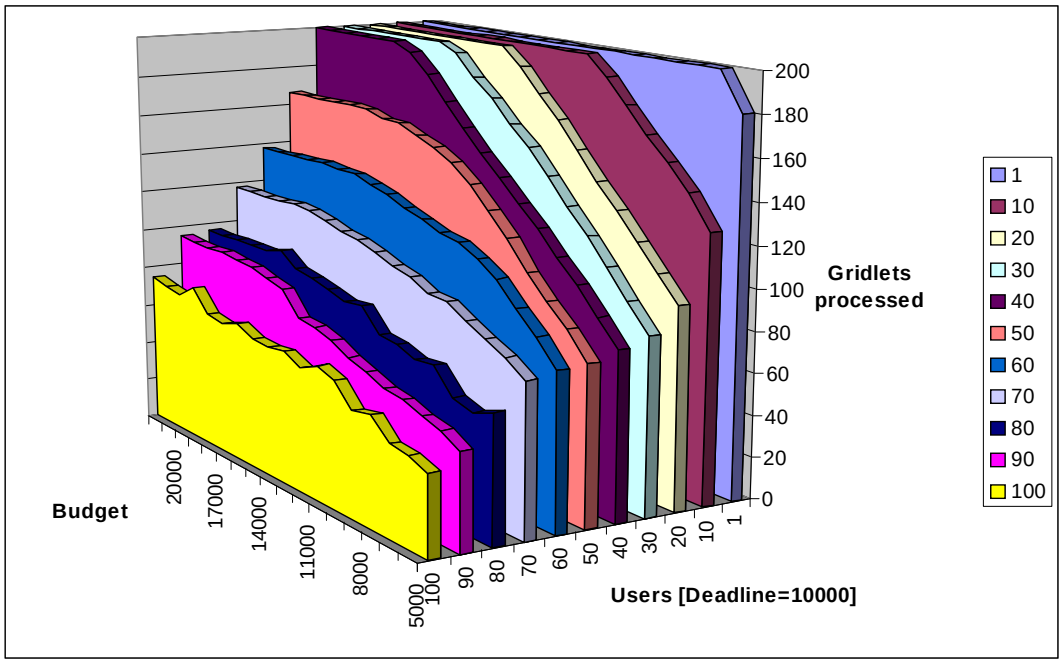


Figure 6.20: No. of Gridlets processed for each user with varying number of users competing for resources.

6.4 Deadline and Budget Constrained Time Optimisation Scheduling

In this experiment, we perform scheduling experiments with different values of deadline and budget constraints (DBC) for a single user using the DBC constrained time-optimisation scheduling algorithm shown in Figure 6.23. The deadline is varied in simulation time from 100 to 3600 in steps of 500. The budget is varied from G$ 5000 to 22000 in steps of 1000. The number of Gridlets processed, time spent, and budget spent for different scheduling scenario is shown in Figure 6.24, Figure 6.25, and Figure 6.26. The number of Gridlets processed increased with the increase in budget or deadline value (see Figure 6.25 for a tight deadline value say 100). This is because, when a higher budget is available, the broker is able to select expensive resources to process more jobs as quickly as possible. The increase in budget has impact not only on the number of Gridlets completed; it also has impact on the completion time. The application processing *completion time decreases* with the *increase in budget value* (see Figure 6.25). When a higher budget is available, the broker schedules jobs on even expensive resources depending on their capability and availability (e.g., resources R6 and R7) to complete the processing at the earliest possible time (see Figure 6.27). This also means that as the application processing completion time decreases, the processing cost increases as powerful and expensive resources are used in processing jobs (see Figure 6.28).

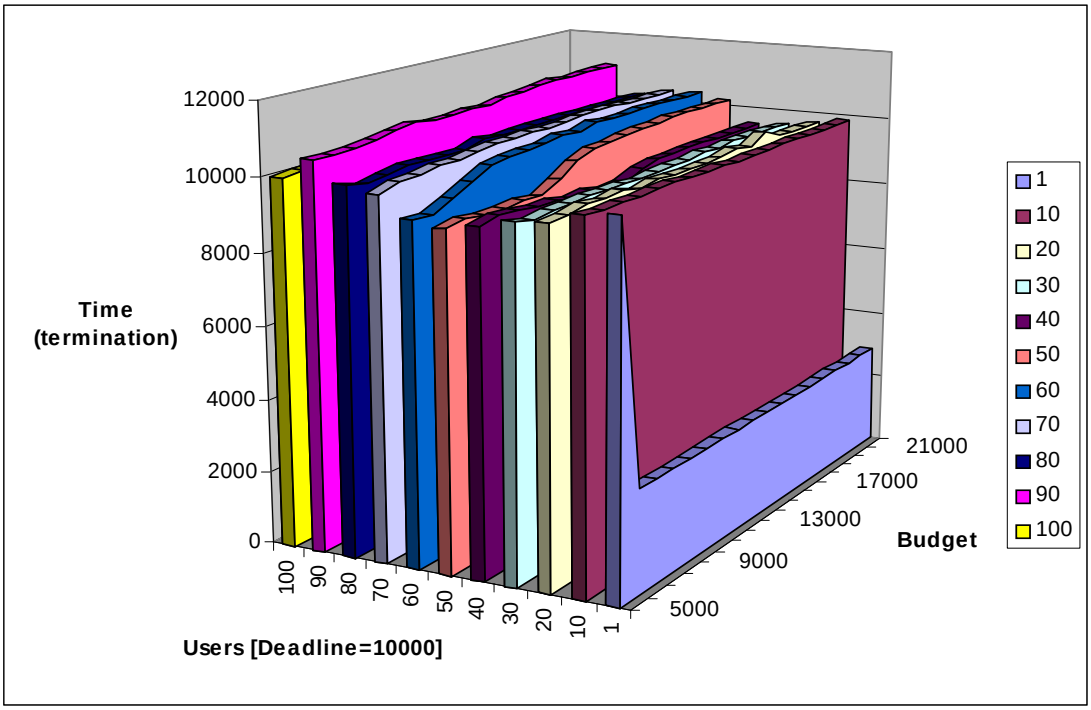


Figure 6.21: The average time at which the user experiment is terminated with varying number of users competing for resources.

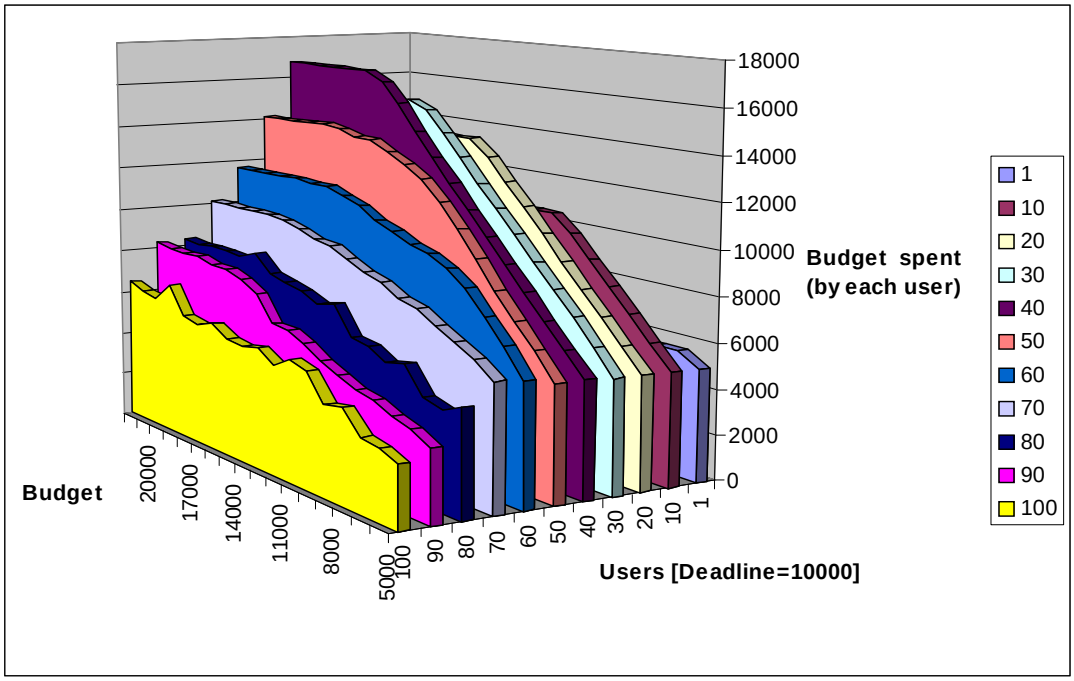


Figure 6.22: The average budget spent by each user for processing Gridlets.

Algorithm: DBC_Scheduling_with_Time_Optimisation()

1. RESOURCE DISCOVERY: Identify the resources and their capability using the Grid information services.
2. RESOURCE TRADING: Identify *the cost* of all resources and *the capability* to be delivered per cost-unit. The *resource cost* can be expressed in units such as processing cost-per-MI, cost-per-job, CPU cost per time unit, etc. and the scheduler needs to choose suitable unit for comparison.
3. If the user supplies D and B-factors, then determine the absolute deadline and budget based on the capability of resources and their cost, and the application processing requirements (e.g., total MI required).
4. SCHEDULING: Repeat while there exist *unprocessed jobs* and the current time and processing expenses are within the deadline and budget limits. [It is triggered for each scheduling event or whenever a job completes. The event period is a function of deadline, job processing time, rescheduling overhead, resource share variation, etc.]:

[SCHEDULE ADVISOR with Policy]

a.) For each resource, predict and establish the *job consumption rate* or *the available resource share* through the measure and extrapolation strategy taking into account the time taken to process previous jobs.

b.) If any of the resource has jobs assigned to it in the previous scheduling event, but not dispatched to the resource for execution and there is variation in resource availability, then move appropriate number of jobs to the Unassigned-Jobs-List. This helps in updating the whole schedule based on the latest resource availability information.

c.) Repeat the following steps for each job in the Unassigned-Jobs-List:

- Select a job from the Unassigned-Jobs-List.
- Create a *resource group* containing affordable resources (i.e., whose processing price is less than or equal to the remaining budget per job).
- For each resource in the resource group, calculate/predict the job completion time taking into account previously assigned jobs and the job completion rate and resource share availability.
- Sort resources in the resource group by the increasing order of job completion time.
- Assign the job to the first resource in the resource group and remove it from the Unassigned-Jobs-List if the predicted job completion time is less than the deadline.

5. [DISPATCHER with Policy]

Repeat the following steps for each resource if it has jobs to be dispatched:

- Identify the number of jobs that can be submitted without overloading the resource. Our default policy is to dispatch jobs as long as the number of user jobs deployed (active or in queue) is less than the number of PEs in the resource.

Figure 6.23: Deadline and budget constrained (DBC) time optimisation scheduling algorithm.

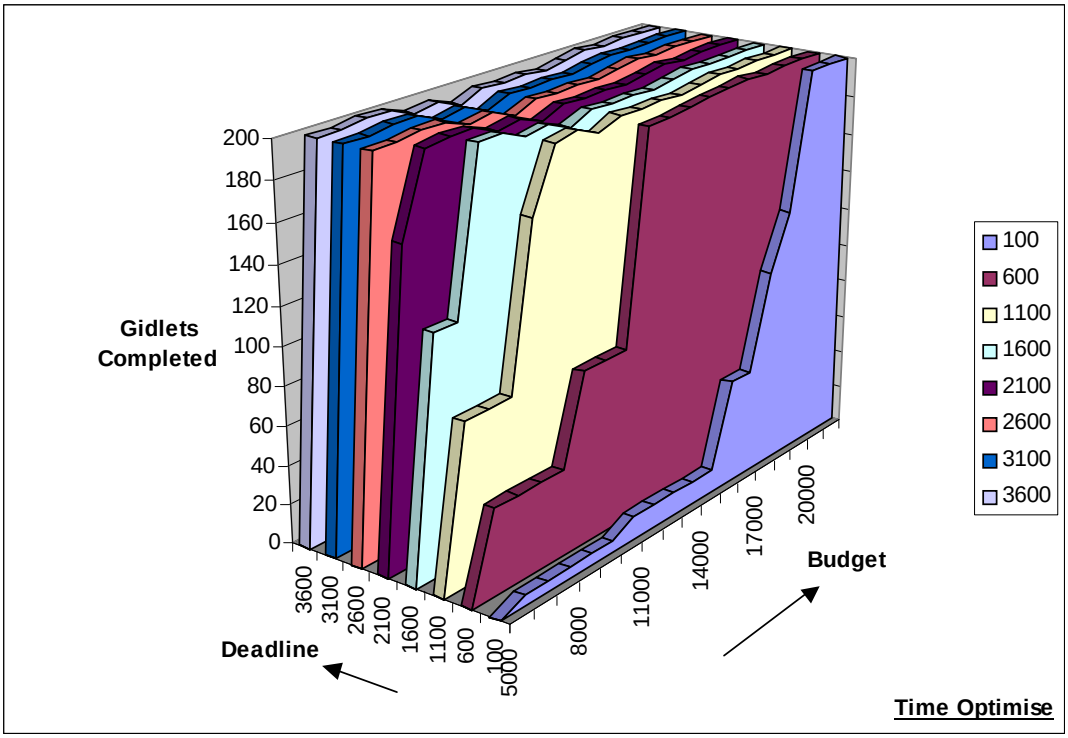


Figure 6.24: No. of Gridlets processed for different budget and deadline limits.

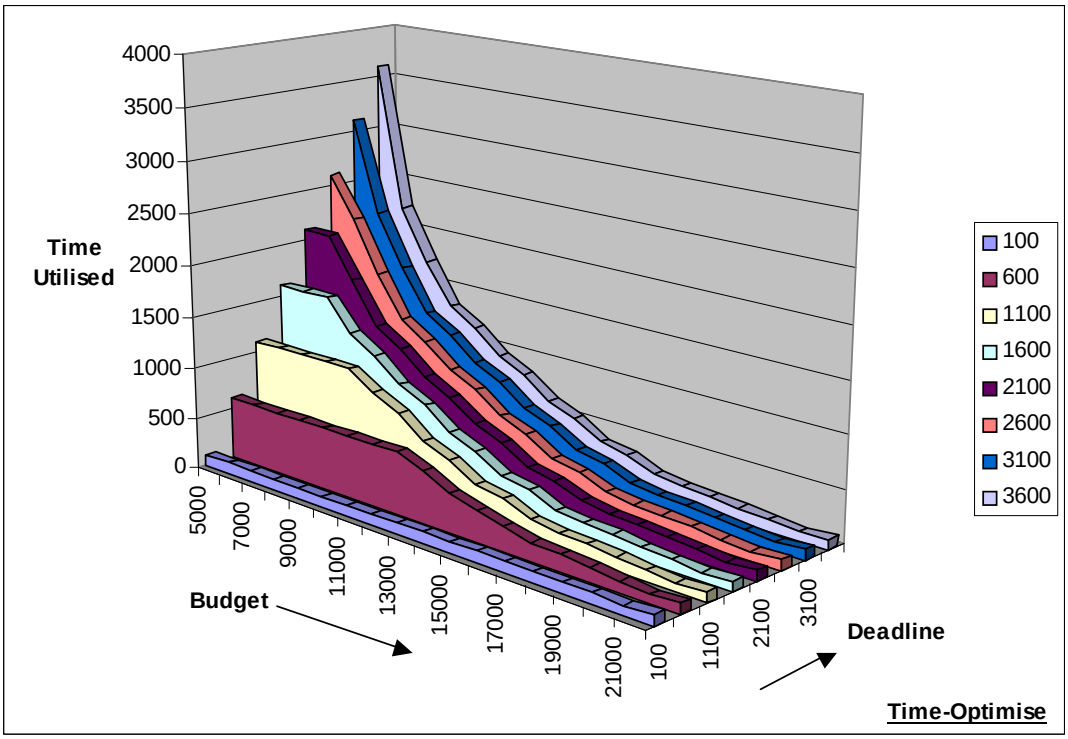


Figure 6.25: The time spent in processing Gridlets using the DBC time optimisation.

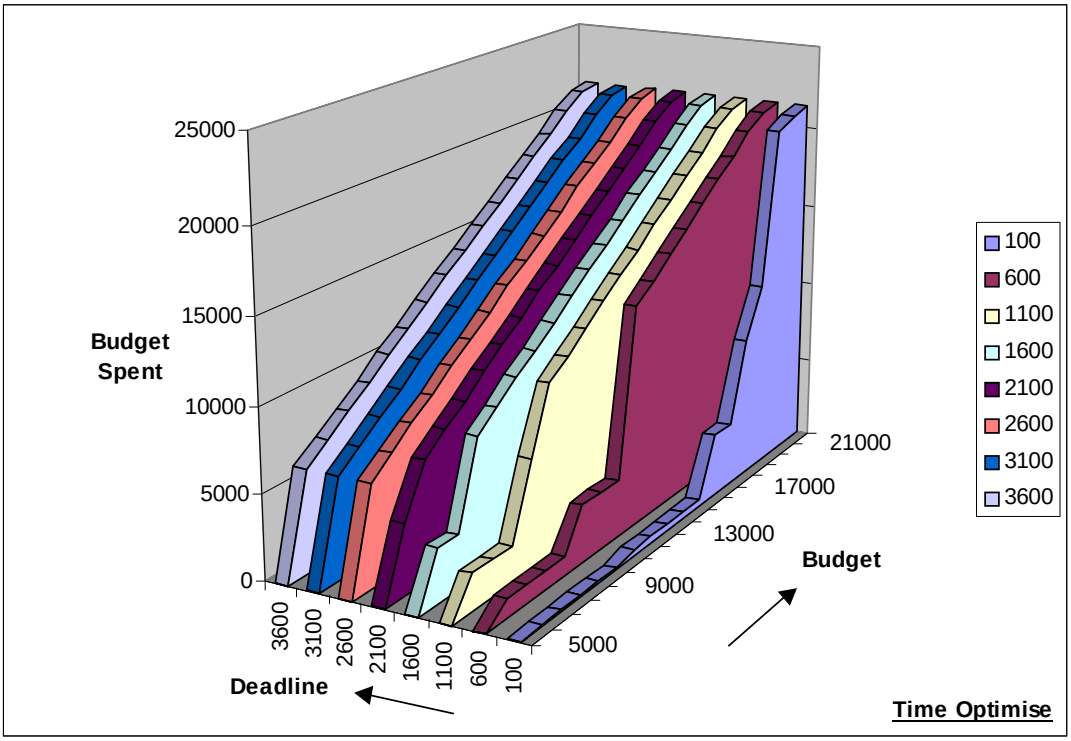


Figure 6.26: The budget spent in processing Gridlets using the DBC time optimisation.

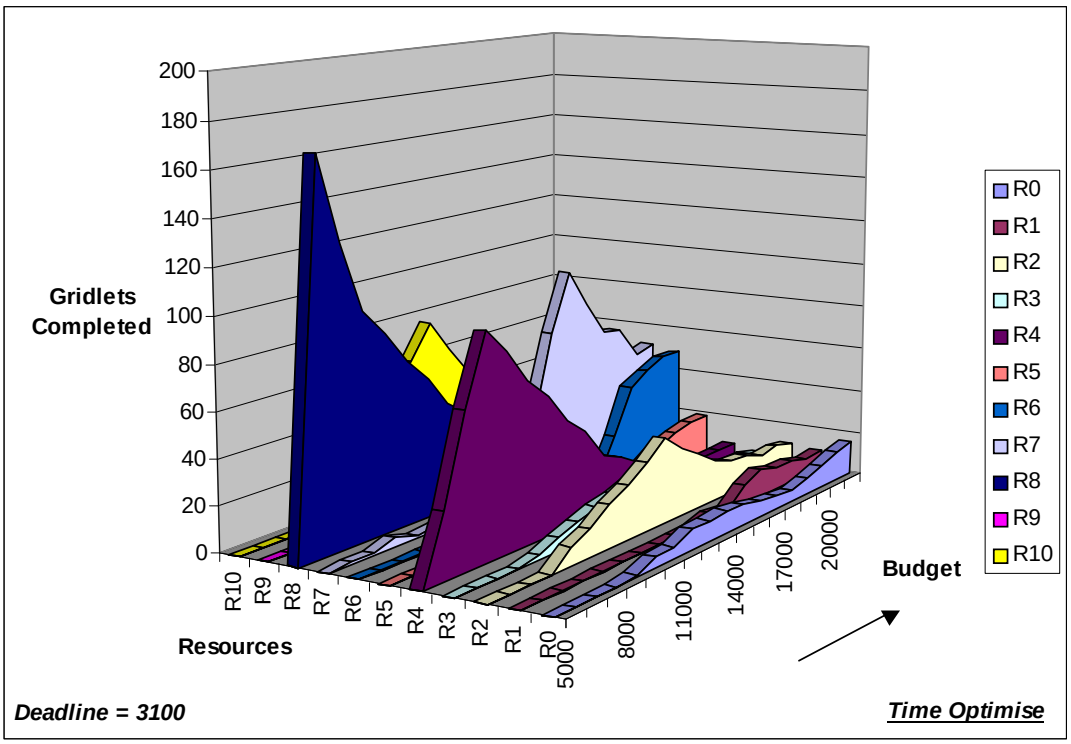


Figure 6.27: Selection of different resources for processing Gridlets for different budget limits.

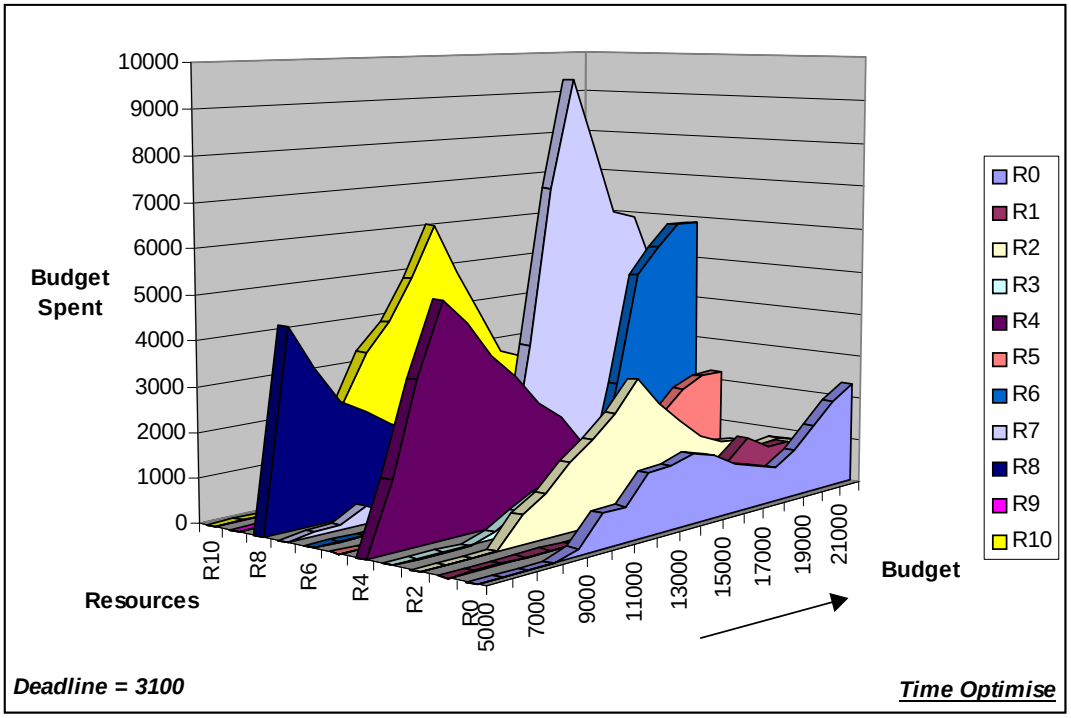


Figure 6.28: The budget spent in processing Gridlets on different resources for different budgets.

6.5 Comparing the Cost and Time Optimisation Scheduling

The completion time and the budget spent for processing application jobs scheduled using the cost and the time optimisation strategies is shown in Figure 6.29 and Figure 6.30. In both scheduling optimisation scenarios, the deadline value is set to 3100 time units and budget value is varied from 5000 to 22000 in steps of 1000. In general, as the budget value increases, the completion time decreases and the processing cost increases when the time-optimisation scheduling strategy is used; whereas, the completion time remains closer to the deadline and processing cost decreases when the cost-optimisation scheduling is used.

Note that, when the available budget per job is less than the cost of processing a job on any resource, no jobs are scheduled for processing in the case of time-optimisation scheduling. This can be observed from Figure 6.30 when the budget is 5000—the budget per job is less than the cost of processing even on the cheapest resource and no jobs are processed, hence the budget spent in processing is shown as 0. Such a condition can also be strictly enforced within the cost-optimisation strategy.

The time-optimisation scheduling algorithm uses as many resources as it can in parallel as long as the budget is available since minimizing the completion time is a major goal. Whereas, the cost-optimisation scheduling algorithm uses resources, giving the first preference to cheaper resources, as long as the deadline can be met, since minimizing the processing cost is a major goal. That means, the users can choose a scheduling strategy that meets their quality of service requirements. When the work is urgent and the results are needed as quickly as possible, they can choose the time optimisation strategy and place a limit on the processing expenses. If they do not have immediate requirement for results, they can choose the cost-optimisation scheduling strategy and minimize the processing cost.

In the DBC time optimization scheduling, the increase in budget value has much impact on resource selection and the completion time. When a higher budget is available, the increase in deadline to a larger value does not have much impact on a completion time or budget spent (see Figure 6.25 and Figure 6.26). This situation is very much different for the cost optimisation scheduling where deadline parameter drives the selection of resources.

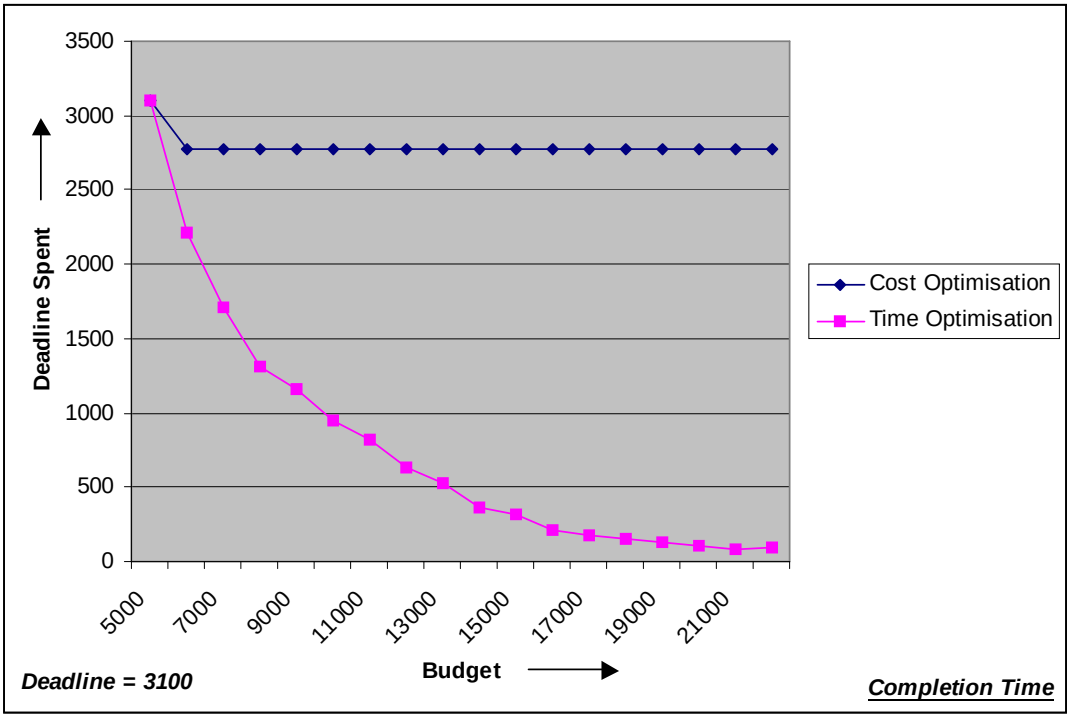


Figure 6.29: The time spent in processing application jobs using time cost and time optimisation scheduling algorithms given different budget limits.

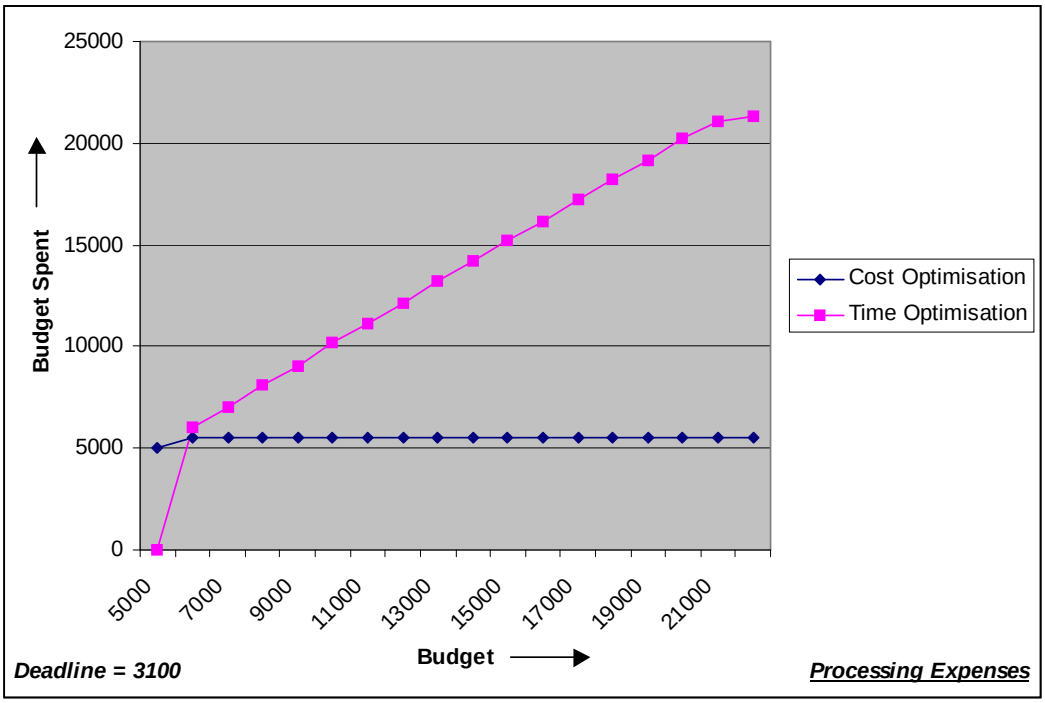


Figure 6.30: The budget spent in processing application jobs using time cost and time optimisation scheduling algorithms given different budget limits.

6.6 DBC Cost-Time Optimisation Scheduling

The DBC cost-time optimisation scheduling algorithm (shown in Figure 6.31) extends the cost-optimisation algorithm to optimise the time without incurring additional processing expenses. This is accomplished by applying the time-optimisation algorithm to schedule jobs on resources having the same processing cost.

Algorithm: DBC_Scheduling_with_Cost_Time_Optimisation()

1. RESOURCE DISCOVERY: Identify the resources and their capability using the Grid information services.
2. RESOURCE TRADING: Identify the cost of all resources and the capability to be delivered per cost-unit. The resource cost can be expressed in units such as processing cost-per-MI, cost-per-job, CPU cost per time unit, etc. and the scheduler needs to choose suitable unit for comparison.
3. If the user supplies D and B-factors, then determine the absolute deadline and budget based on the capability of resources and their cost, and the application processing requirements (e.g., total MI required).
4. SCHEDULING: Repeat while there exist unprocessed jobs and the current time and processing expenses are within the deadline and budget limits. [It is triggered for each scheduling event or whenever a job completes. The event period is a function of deadline, job processing time, rescheduling overhead, resource share variation, etc.]:

[SCHEDULE ADVISOR with Policy]

a. For each resource, predict and establish the job consumption rate or the available resource share through the measure and extrapolation strategy taking into account the time taken to process previous jobs.

b. SORT the resources by increasing order of cost. If two or more resources have the same cost, order them such that powerful ones (e.g., higher job consumption rate or resource share availability, but the first time based on the total theoretical capability, say the total MIPS) are preferred first.

c. Create resource groups containing resources with the same cost.

d. SORT the resource groups with the increasing order of cost.

e. If any of the resource has jobs assigned to it in the previous scheduling event, but not dispatched to the resource for execution and there is variation in resource availability, then move appropriate number of jobs to the Unassigned-Jobs-List. This helps in updating the whole schedule based on the latest resource availability information.

f. Repeat the following steps for each resource group as long as there exist unassigned jobs:

Repeat the following steps for each job in the Unassigned-Jobs-List depending on the processing cost and the budget availability: [It uses the time optimisation strategy.]

- Select a job from the Unassigned-Jobs-List.
- For each resource, calculate/predict the job completion time taking into account previously assigned jobs and the job completion rate and resource share availability.
- Sort resources by the increasing order of completion time.
- Assign the job to the first resource and remove it from the Unassigned-Jobs-List if the predicted job completion time is less than the deadline.

5. [DISPATCHER with Policy]

Repeat the following steps for each resource if it has jobs to be dispatched:

- Identify the number of jobs that can be submitted without overloading the resource. Our default policy is to dispatch jobs as long as the number of user jobs deployed (active or in queue) is less than the number of PEs in the resource.

Figure 6.31: Deadline and budget constrained (DBC) scheduling with cost-time optimisation.

6.6.1 Experiment Setup

The resources used in evaluating the performance of cost-time optimisation scheduling are show in Table

6.3. The characteristics of resources is same as those used in previous experiment except that the price of resource R4 is set to the same value as the resource R8 to demonstrate the superior ability of cost-time optimisation scheduling algorithm over the cost optimisation scheduling algorithm. It can be noted some of the resources in Table 6.3 have the same MIPS per G$. For example, both R4 and R8 have the same cost and so resources R2, R3, and R10.

A task farming application containing 200 jobs used in this scheduling experiment is the same as the one used in previous experiments.

Table 6.3: Resources used in Cost-Time scheduling simulation.

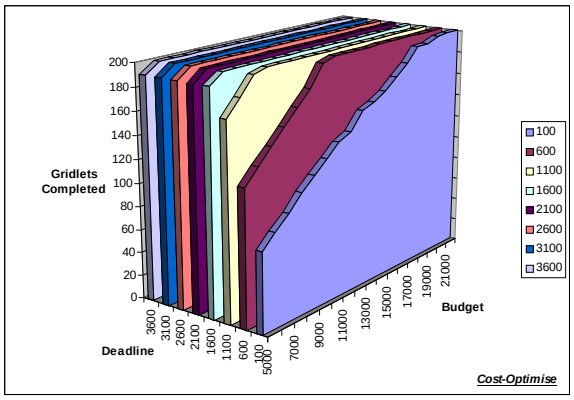
Resource Name in Simulation	Simulated Resource Characteristics Vendor, Resource Type, Node OS, No of PEs	Equivalent Resource in Worldwide Grid (Hostname, Location)	A PE SPEC/ MIPS Rating	Resource Manager Type	Price (G$/PE time unit)	MIPS per G$
R0	Compaq, AlphaServer, CPU, OSF1, 4	grendel.vpac.org, VPAC, Melb, Australia	515	Time-shared	8	64.37
R1	Sun, Ultra, Solaris, 4	hpc420.hpcc.jp, AIST, Tokyo, Japan	377	Time-shared	4	94.25
R2	Sun, Ultra, Solaris, 4	hpc420-1.hpcc.jp, AIST, Tokyo, Japan	377	Time-shared	3	125.66
R3	Sun, Ultra, Solaris, 2	hpc420-2.hpcc.jp, AIST, Tokyo, Japan	377	Time-shared	3	125.66
R4	Intel, Pentium/VC820, Linux, 2	barbera.cnuce.cnr.it, CNR, Pisa, Italy	380	Time-shared	1	380.0
R5	SGI, Origin 3200, IRIX, 6	onyx1.zib.de, ZIB, Berlin, Germany	410	Time-shared	5	82.0
R6	SGI, Origin 3200, IRIX, 16	Onyx3.zib.de, ZIB, Berlin, Germany	410	Time-shared	5	82.0
R7	SGI, Origin 3200, IRIX, 16	mat.ruk.cuni.cz, Charles U., Prague, Czech Republic	410	Space-shared	4	102.5
R8	Intel, Pentium/VC820, Linux, 2	marge.csm.port.ac.uk, Portsmouth, UK	380	Time-shared	1	380.0
R9	SGI, Origin 3200, IRIX, 4 (accessible)	green.cfs.ac.uk, Manchester, UK	410	Time-shared	6	68.33
R10	Sun, Ultra, Solaris, 8,	pitcairn.mcs.anl.gov, ANL, Chicago, USA	377	Time-shared	3	125.66

6.6.2 Scheduling Experiments with Cost and Cost-Time Optimisation Strategies

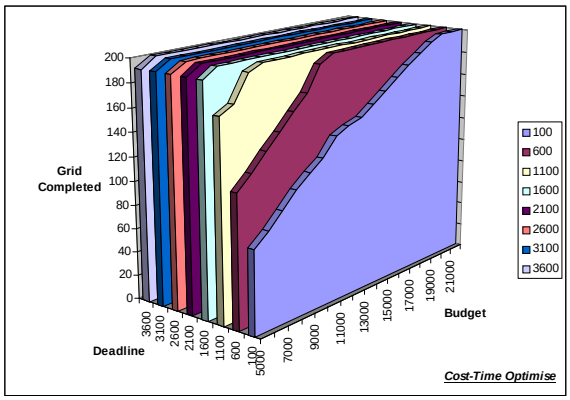
We perform both cost and cost-time optimisation scheduling experiments with different values of deadline and budget constraints (DBC) for a single user. The deadline is varied in simulation time from 100 to 3600 in steps of 500. The budget is varied from G$ 5000 to 22000 in steps of 1000. The number of Gridlets processed, deadline utilized, and budget spent for the DBC cost-optimisation scheduling strategy is shown in Figure 6.32a, Figure 6.32c, and Figure 6.32e, and for the cost-time optimisation scheduling strategy is shown in Figure 6.32b, Figure 6.32d, and Figure 6.32f. In both cases, when the deadline is low (e.g., 100 time unit), the number of Gridlets processed increases as the budget value increases. When a higher budget is available, the broker leases expensive resources to process more jobs within the deadline. Alternatively, when scheduling with a low budget value, the number of Gridlets processed increases as the deadline is relaxed.

The impact of budget for different values of deadline is shown in Figure 6.32e and Figure 6.32f for cost and cost-time strategies. For a larger deadline value (see the time utilization for deadline of 3600), the increase in budget value does not have much impact on resource selection. When the deadline is too tight

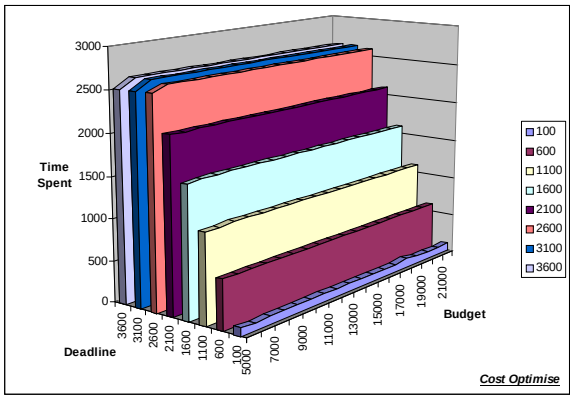
(e.g., 100), it is likely that the complete budget is spent for processing Gridlets within the deadline.



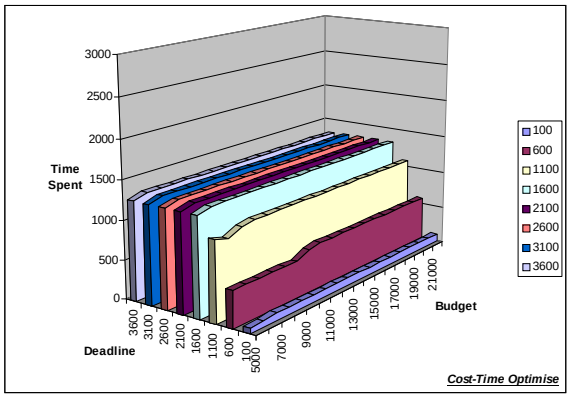
(a) No. of Gridlets processed.



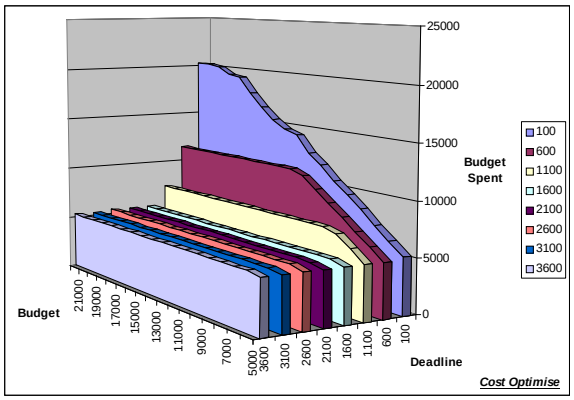
(b) No. of Gridlets processed



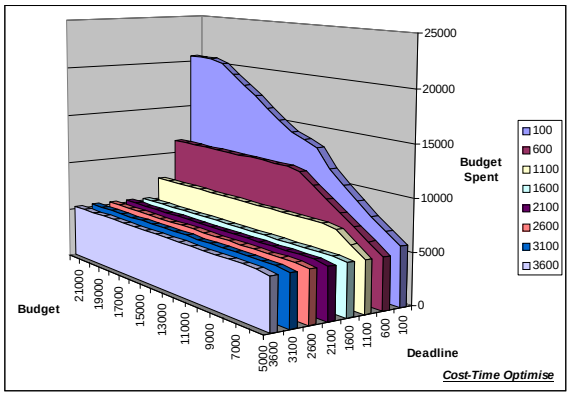
(c) Time spent for processing Gridlets.



(d) Time spent for processing Gridlets.



(e) Budget spent for processing Gridlets.



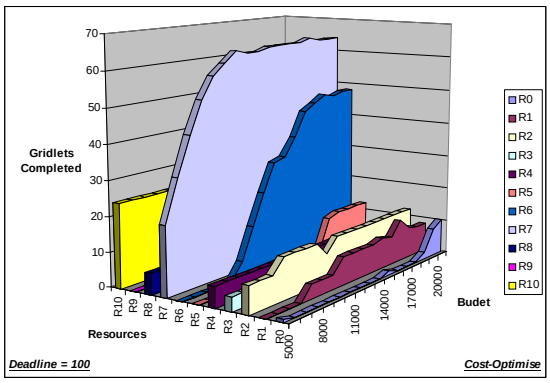
(f) Budget spent for processing Gridlets.

Figure 6.32: The number of Gridlets processed, time, and budget spent for different deadline and time limits when scheduled using the cost and cost-time optimisation algorithms.

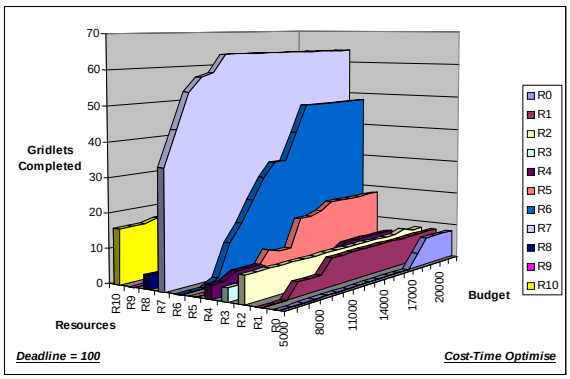
It can be observed that the number of Gridlets processed and the budget-spending pattern is similar for both scheduling strategies. However, the time spent for the completion of all the jobs is significantly different (see Figure 6.32c and Figure 6.32d), as the deadline becomes relaxed. For deadline values from 100 to 1100, the completion time for both cases is similar, but as the deadline increases (e.g., 1600 to 3600), the experiment completion time for cost-time scheduling optimisation strategy is much less than the

cost optimisation scheduling strategy. This is because when there are many resources with the same MIPS per G$, the cost-time optimisation scheduling strategy allocates jobs to them using the time-optimisation strategy for the entire deadline duration since there is no need to spent extra budget for doing so. This does not happen in case of cost-optimisation strategy—it allocates as many jobs that the first cheapest resource can complete by the deadline and then allocates the remaining jobs to the next cheapest resources.

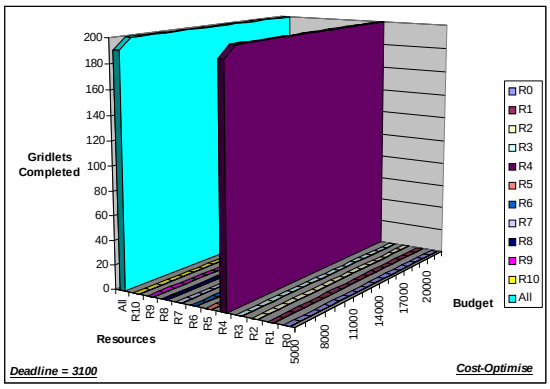
A trace of resource selection and allocation using cost and cost-time optimisation scheduling strategies shown in Figure 6.33 indicates their impact on the application processing completion time. When the deadline is tight (e.g., 100), there is high demand for all the resources in short time, the impact of cost and cost-time scheduling strategies on the completion time is similar as all the resources are used up as long as budget is available to process all jobs within the deadline (see Figure 6.33a and Figure 6.33b). However, when the deadline is relaxed (e.g., 3100), it is likely that all jobs can be completed using the first few cheapest resources. In this experiment there were resources with the same cost and capability (e.g., R4 and R8), the cost optimisation strategy selected resource R4 to process all the jobs (see Figure 6.33c); whereas the cost-time optimisation strategy selected both R4 and R8 (see Figure 6.33d) since both resources cost the same price and completed the experiment earlier than the cost-optimisation scheduling (see Figure 6.32c and Figure 6.32d). This situation can be observed clearly in scheduling experiments with a large budget for different deadline values (see Figure 6.34). Note that the left most solid curve marked with the label "All" in the resources axis in Figure 6.34 represents the aggregation of all resources.



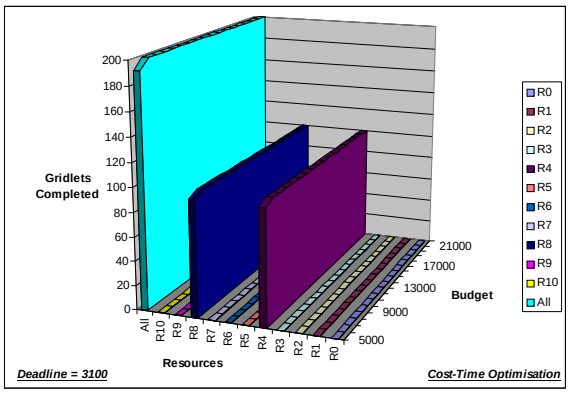
(a) Cost optimisation with a short deadline.



(b) Cost-time optimisation with a short deadline.



(c) Cost optimisation with a long deadline.

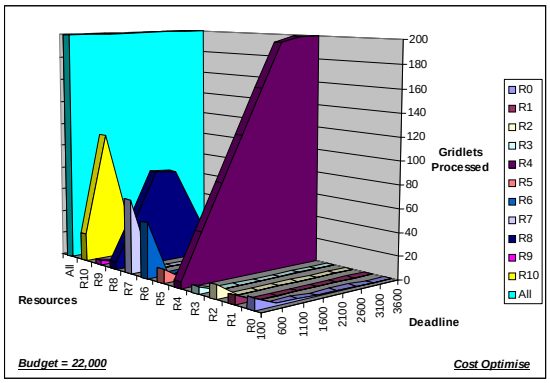


(d) Cost-time optimisation with a long deadline.

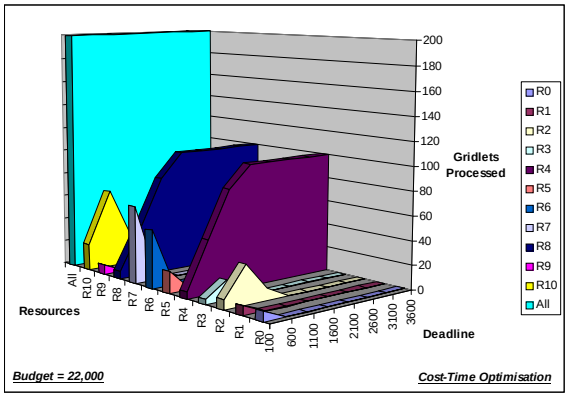
Figure 6.33: The number of Gridlets processed and resources selected for different budget values with a long deadline value when scheduled using the cost and cost-time optimisation algorithms.

As the deadline increases, the cost optimisation algorithm predominantly scheduled jobs on the resource R4 (see Figure 6.34a) whereas, the cost-time optimisation algorithm scheduled jobs on resources R4 and R8 (see Figure 6.34b), the first two cheapest resources with the same cost. Therefore, the application

scheduling using the cost-time optimisation algorithm is able to finish earlier compared to the one scheduled using the cost optimisation algorithm (see Figure 6.35) and both strategies have spent the same amount of budget for processing its jobs (see Figure 6.36). The completion time for cost optimisation scheduling continued to increase with increase of the deadline as the broker allocated more jobs to the resource R4 and less to the resource R8. However, the completion time for deadline values 3100 and 3660 is the same as the previous one since the broker allocated jobs to only resource R4. This is not the case with the cost-time optimisation scheduling since jobs are allocated proportionally to both resources R4 and R8 and thus minimizing the completion time without spending any extra budget.



(a) Resource selection when the budget is high.



(b) Resource selection when the budget is high.

Figure 6.34: The number of Gridlets processed and resources selected for different deadline values with a large budget when scheduled using the cost and cost-time optimisation algorithms.

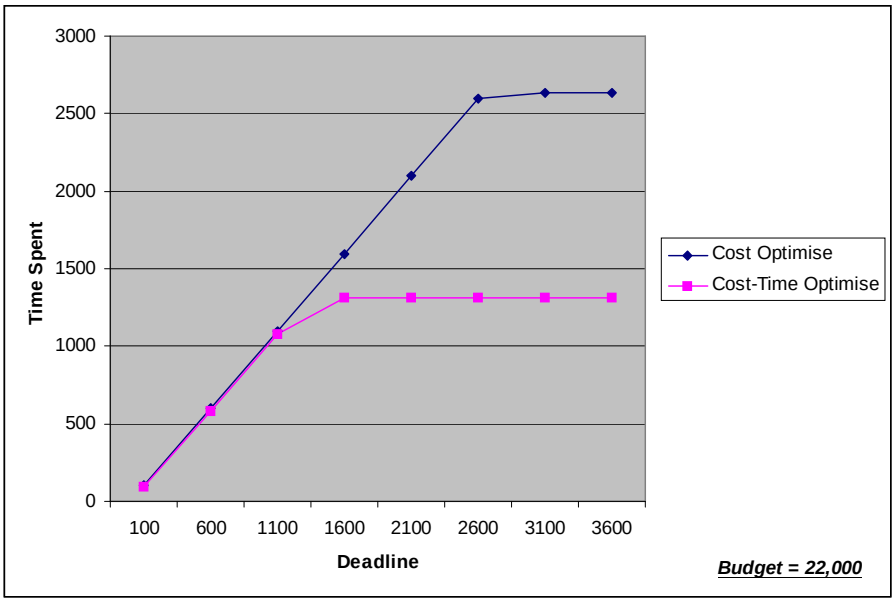


Figure 6.35: The time spent for processing application jobs for different deadline constraints with a large budget when scheduled using the cost and cost-time optimisation algorithms.

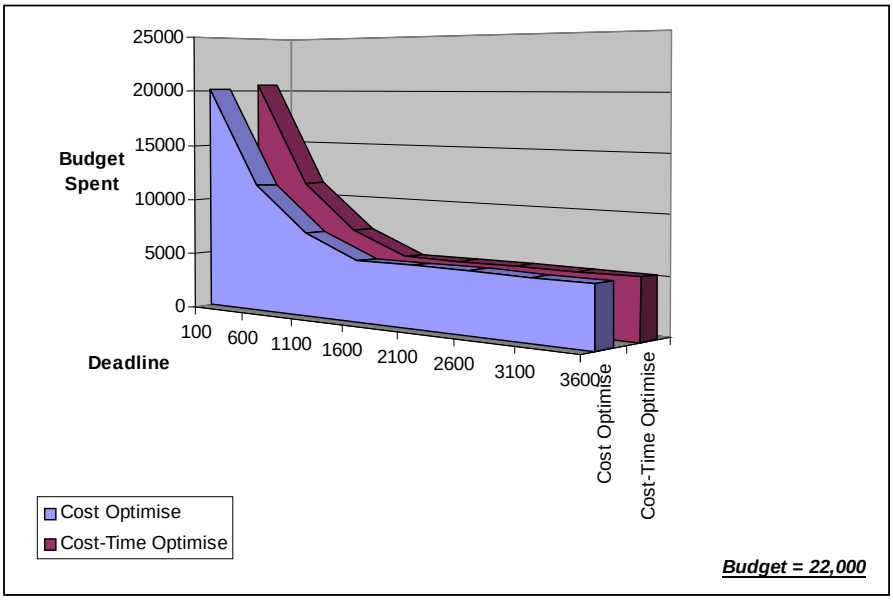


Figure 6.36: The budget spent for processing application jobs for different deadline constraints with a large budget when scheduled using the cost and cost-time optimisation algorithms.

Let us now take a microscopic look at the allocation of resources when a moderate deadline and large budget is assigned. A trace of resource allocation and the number of Gridlets processed at different times when scheduled using the cost and cost-time optimisation algorithms is shown in Figure 6.37 and Figure 6.38. It can be observed that for both the strategies, the broker used the first two cheapest resources, R4 and R8 fully. Since the deadline cannot be completed using only these resources, it used the next cheapest resources R2, R3, and R10 to make sure that deadline can be meet. The cost optimisation strategy allocated Gridlets to resource R10 only, whereas the cost-time optimisation allocated Gridlets to resources R2, R3, and R10 as they cost the same price. Based on the availability of resources, the broker predicts the number of Gridlets that each resource can complete by the deadline and allocates to them accordingly (see Figure 6.39 and Figure 6.40). At each scheduling event, the broker evaluates the progress and resource availability and if there is any change, it reschedules some Gridlets to other resources to ensure that the deadline can be meet. This can be observed in Figure 6.39 and Figure 6.40—the broker allocated a few extra Gridlets to resource R10 (cost-optimisation strategy) and resources R2, R3, and R10 (cost-time optimisation strategy) during the first few scheduling events.

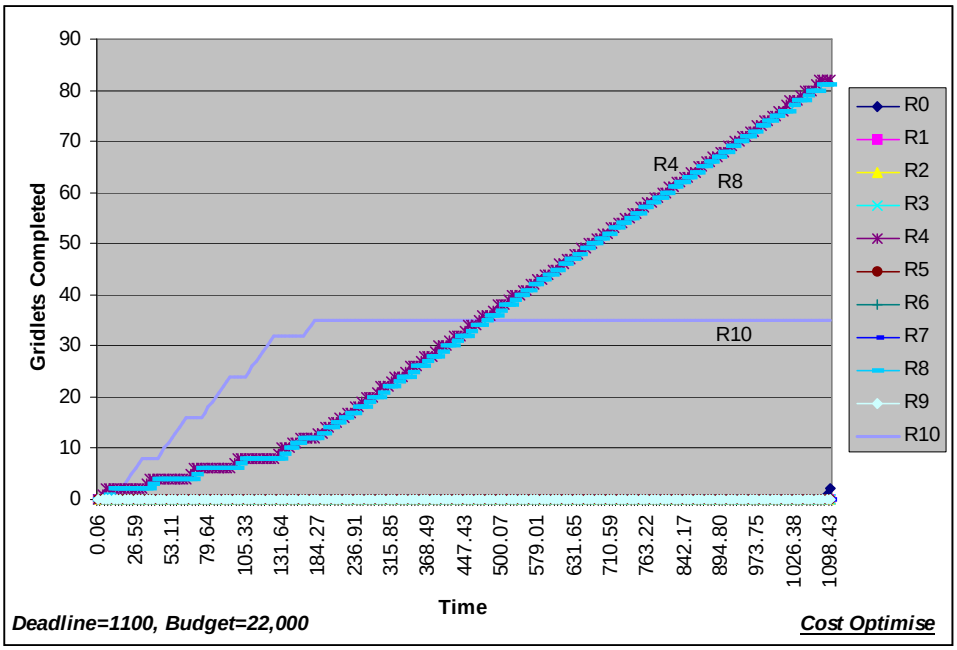


Figure 6.37: Trace of No. of Gridlets processed on resources for a medium deadline and high budget constraints when scheduled using the cost optimisation strategy.

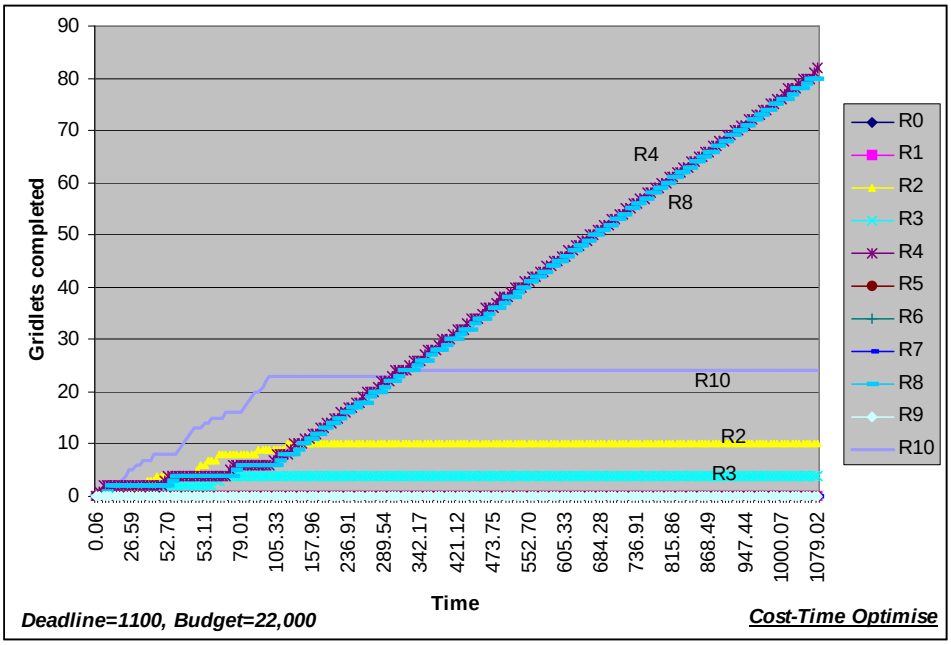


Figure 6.38: Trace of No. of Gridlets processed on resources for a medium deadline and high budget constraints when scheduling using the cost-time optimisation strategy.

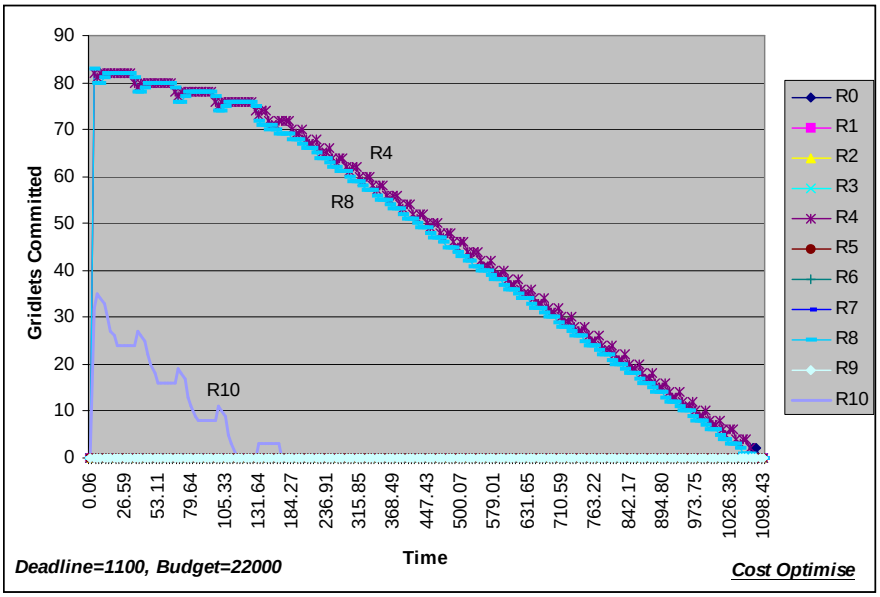


Figure 6.39: Trace of the number of Gridlets committed to resources for a medium deadline and high budget constraints when scheduled using the cost optimisation strategy.

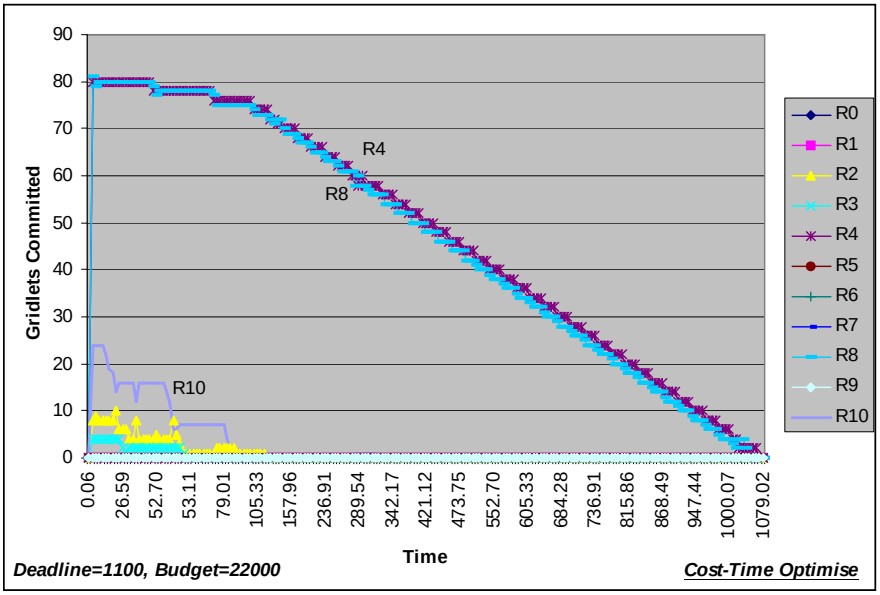


Figure 6.40: Trace of the number of Gridlets committed to resources for a medium deadline and high budget constraints when scheduled using the cost-time optimisation strategy.

In summary, when there are multiple resources with the same cost and capability, the cost-time optimisation algorithm schedules jobs on them using the time-optimisation strategy for the deadline period. The results of scheduling experiments for many scenarios with a different combination of the deadline and budget constraints, we observe that applications scheduled using the *cost-time* optimisation are able to complete earlier than those scheduled using the cost optimisation algorithm without incurring any extra expenses. This proves the superiority of the new deadline and budget constrained cost-time optimisation algorithm in scheduling jobs on global Grids.

6.7 Summary and Conclusion

We discussed the use of computational economy as a metaphor for devising scheduling strategies for large-scale applications on distributed resources. We used the GridSim toolkit in simulating an economic-based Grid resource broker that supports deadline and budget-based scheduling. We simulated and evaluated performance of scheduling algorithms with cost, time, and cost-time optimisation strategies for a variety of scenarios. The broker is able allocate resources depending on the users' quality of service requirements such as the deadline, budget, and optimisation strategy. The performance evaluation results at microscopic level reveal their impact on the application processing cost and time; and demonstrate the usefulness of allowing uses to trade-off between the timeframe and processing cost depending on their QoS requirements. Also, these extensive simulation studies demonstrate the suitability of GridSim for developing simulators for scheduling in parallel and distributed systems.

Software Availability

The economic Grid resource broker simulator with source code can be downloaded from the GridSim project website:

`http://www.buyya.com/gridsim/`