Journal of Grid Computing

QoS-based Task Group Deployment on Grid by Learning the Performance Data

--Manuscript Draft--

Manuscript Number:	GRID-D-12-00639R1
Full Title:	QoS-based Task Group Deployment on Grid by Learning the Performance Data
Article Type:	High-Performance Data Management & Mining on Computational Grids & Clouds
Keywords:	grid computing; task group deployment; performance data; advance QoS planning; task-resource mapping.
Corresponding Author:	Nithiapidary Muthuvelu, Masters Multimedia University Cyberjaya, Selangor MALAYSIA
Corresponding Author Secondary Information:	
Corresponding Author's Institution:	Multimedia University
Corresponding Author's Secondary Institution:	
First Author:	Nithiapidary Muthuvelu, Masters
First Author Secondary Information:	
Order of Authors:	Nithiapidary Muthuvelu, Masters
	Ian Chai, Ph.D
	Eswaran Chikkannan, Ph.D, Professor
	Rajkumar Buyya, Ph.D, Professor
Order of Authors Secondary Information:	
Abstract:	Overhead of executing fine-grain tasks on computational grids led to task group or batch deployment in which a batch is resized according to the characteristics of the tasks, designated resource, and the interconnecting network. An economic grid demands an application to be processed within the given budget and deadline, referred to as the quality of service (QoS) requirements. In this paper, we increase the task success rate in an economic grid by optimally mapping the tasks to the resources prior to the batch deployment. The task-resource mapping (Advance QoS Planning) is decided based on QoS requirement and by mining the historical performance data of the application tasks using a genetic algorithm. The mapping is then used to assist in creating the task groups. Practical experiments are conducted to validate the proposed method and suggestions are given to implement our method in a cloud environment as well as to process real-time tasks.
Response to Reviewers:	Reviewer #2: (1) Authors should better discuss complexity issues of their proposal, even at a theoretical level. Answer: This has been addressed briefly in section 7.2.4 Issues and Future Direction. As shown in Fig. 5, the proposed batch resizing strategy requires the meta- scheduler to keep monitoring, learning and updating the average deployment metrics for each task category-resource pair using a genetic algorithm. A frequent con- duct of advance QoS planning may delay the entire task group deployment as the genetic algorithm will increase the computation overhead at the meta-scheduler. With the current high-end machines, the overhead or latency can be reduced or hid- den by configuring the genetic algorithm to keep running in parallel as a separate thread. The meta-scheduler can obtain the latest, optimal task-resource mapping at any time from the genetic algorithm.

The advance QoS planning needs to be conducted frequently only when the grid status varies drastically at runtime. Practically, the meta-scheduler must have completed at least one task group deployment to each resource before the subsequent QoS planning. For simplicity, in our meta-scheduler, the planning is carried out once it has successfully processed GR_TOTAL × 2 task groups. We also conducted experiments in which the planning is performed after every GR_TOTAL × 1 task group deployment. We observed almost similar task success rate and budget utilisation as our grid status did not fluctuate much during the experiments.

(2) .... In fact, more experiments are necessary in order to prove that the proposed approach scales well on large-scale settings.

Answer:
This paper was prepared in 2011. At that time, we couldn't cater for more gri resources. Now (2014), we do not have the similar platform or configuration to run the experiments again. All the machines used for this experiments do not exist anymore.

Reviewer #3:

(1) The scope of application of the method proposed by the paper is quite limited as it is based on the assumption that the user has rough estimations on the processing time and the output file size of the application tasks. Many times the user can't make such estimations or, even worse, the estimations are completely wrong, hence making the method not applicable in the former case, or generating bad performances in the latter.

Answer:
Yes. This is the scope of the proposed environment, expecting the grid user to give the correct estimations of their BoT. Grid may delay the entire application processing time if the user do not have the basic info about their own BoT.

Sample scenario:
The user may have 1000 tasks, each may take up to 4 secs in his machine; in total: 67 mins. He may have the permission to use 3 grid machines, each may take 7 mins for executing 1 task. (Best case scenario) If the tasks are equally divided among the 3 machines, a machine will get a maximum of 334 tasks. Assume that each task requires 6 secs to be transmitted to and from the grid machine. In total he needs 39 mins for computation and 34 mins for communication => 73 mins. In this case, it is not worth running the BoT on grid. If the user has some estimations about his BoT size and execution time, then he can avoid such scenario, investing in the 3 grid machines. Hence, as a grid user, before using the grid, one should have some knowledge about his BoT.

(2) The authors seem to neglect the impact the "accessory" executions can have on the turnaround time, namely the tasks categorization, the sample task executions on the user's local machine, the executions performed on the resources for benchmarking, the computation of the deployment metrics, the computation of their average.

Answer:
Sample task executions on the user's local machine is done offline. The estimations from the sample executions are fed to the meta-scheduler when the user is ready to run the BoT on grid. This is indicated in section 4.

Tasks categorization is done using the task files and estimations (estimated CPU time, output file size) fed to the meta-scheduler. It is done using simple algorithm (data structure) which requires roughly 2-3 seconds.

Resource benchmarking is the core for determining the task category-resource average deployment metrics. This part of the strategy cannot be missed. The resource benchmarking and the computation of the deployment metrics are conducted in multi-threading manner (indicated in section 6) to hide the latency or delay so that it would not affect the turnaround time.

(3) the fact that it's the grid meta- scheduler the one which performs all the other computations and categorizations is not clear until late in the paper: this should be

stated clearly earlier in the paper.

Answer:
This is already indicated in section 1, page 2, last paragraph.

(4) The difference between the user's machine computational power and the resources computational power could be substantial, making it unreasonable to run sample executions locally, which could take days, while running them on the resources could take minutes.

Answer:
The user doesn't need to run the entire BoT for collecting the sample. As addressed in (1), in this paper we are covering the scenario where the user's machine performs better than the grid. This is indicated in section 1.

(5) the sentence "quality of service (QoS) requirements." is unfinished, the verb and the object are missing.

Answer:
Corrected.
"An economic grid demands an application to be processed within the given budget and deadline, referred to as the quality of service (QoS) requirements".

(6)
- what happens to the tasks when they exceeds the constraints?

Answer:
In this paper we are focusing on economic or commercial grid and lightweight tasks (indicated in section 1). If the user tasks exceed the constraints determined by the resource providers, definitely they cannot use the grid to run their tasks.

Our focus is lightweight tasks. We practically learn the performance data, keep grouping the tasks while ensuring the group obey the constraints. The whole batch resizing strategy ensures that optimal groups are created, obeying the constraints.

(7)
- tasks requirements acronyms could be shortened removing the redundant "task" in them (TFSize > FSize, ETCPUTime > ECPUTime). Alternatively the requirement OFSize should have the word "task" in it for uniformity.

Answer:
A couple of acronyms were used for the deployment metics (for resources, transmission time, processing overheads, processing cost, etc). Hence, we include "T" to indicate if the acronym is related to task.

OFSize = Output file size
Using "Task Output File Size" or "Output Task File Size" doesn't really sound correct.

(8)
-it's not clear how the benchmarks affect the ETCPUTime and what's the relation between them: what happens if the benchmarks show a different CPU time on the resources?

Answer:
The entire resource benchmarking is explained in papers [23,24] as indicated in section 4. Paper [24] has the complete experimental proof.

We can't expect for the user machine and grid machine to give us the same CPU time. The user machine might be better than the grid machine. We are doing benchmark to come up with a ratio [user machine CPU time : grid machine CPU time] for each task category.

(9)
Section 5:

	- - in "The process flow of of batch" there is an extra "of"; - - the sentence "The term task-resource allocation denotes that each resource is allocated tasks from each category." is not clear; Answer: Corrected. "The term task-resource allocation will be used to refer to the allocation or mapping of tasks from each category to each resource. " (10) Section 7: - a reference to the Grid Platform used in the experiments is missing; Answer: These grid machines are indicated in Table 1 (in section 7).

Noname manuscript No.
(will be inserted by the editor)

QoS-based Task Group Deployment on Grid by Learning the Performance Data

Nithiapidary Muthuvelu · Ian Chai · Eswaran Chikkannan · Rajkumar Buyya

Received: date / Accepted: date

Abstract Overhead of executing fine-grain tasks on computational grids led to task group or batch deployment in which a batch is resized according to the characteristics of the tasks, designated resource, and the interconnecting network. An economic grid demands an application to be processed within the given budget and deadline, referred to as the quality of service (QoS) requirements. In this paper, we increase the task success rate in an economic grid by optimally mapping the tasks to the resources prior to the batch deployment. The task-resource mapping (*Advance QoS Planning*) is decided based on QoS requirement and by mining the historical performance data of the application tasks using a genetic algorithm. The mapping is then used to assist in creating the task groups. Practical experiments are conducted to validate the proposed method and suggestions are given to implement our method in a cloud environment as well as to process real-time tasks.

Keywords Grid Computing · Task Group Deployment · Performance Data · Advance QoS Planning · Task-Resource Mapping

1 Introduction

In a typical grid, a grid meta-scheduler schedules and transfers the application task files to the resources, monitors the progress of the task executions, retrieves the output files from the resources, and finally compiles the output files. Task processing and communication overheads are inevitable in a grid as the resources are geographically distributed and shared by multiple users [4, 16, 35]. When utilising

Nithiapidary Muthuvelu, Ian Chai, Eswaran Chikkannan
Multimedia University, Persiaran Multimedia, 63100 Cyberjaya, Selangor, Malaysia
Tel.: +603-83125429
Fax: +603-83125264
E-mail: nithiapidary@mmu.edu.my

Rajkumar Buyya
Cloud Computing and Distributed Systems (CLOUDS) Lab, Dept. of Computer Science and Software Engineering, The University of Melbourne, Australia.

a grid for a large number of fine-grain tasks, the overheads affect the application turnaround time. Therefore, in our previous work [23,24], we proposed batch processing or *task group deployment* in which the fine-grain tasks of a bag-of-tasks (BoT) application are grouped into several task groups prior to the deployment. BoT tasks can be executed in parallel, thus, we can estimate the size of a task group for a resource based on the utilisation constraints of the resource and network, and the application budget and deadline as follows:

1. The estimated CPU time, wall-clock time, and storage/space consumption of a task group should be less than the maximum allowed CPU time, wall-clock time, and space utilisation of a resource.
2. The estimated transmission time of a task group and the output files should be less than the maximum allowed file transmission time for a resource.
3. The estimated task group turnaround time and processing cost should be less than the remaining application deadline and budget respectively.

Experiments in [23,24] prove that grouping the tasks according to these batch resizing policies highly reduces the application turnaround time while increasing the utilisation of the grid resources and the interconnecting network. However, not all the tasks can be processed within the application deadline and budget. For example, assume that a BoT contains 500 tasks $\{T_0, T_1, T_2, ..., T_{499}\}$ and the grid consists of 3 resources $\{R_0, R_1, R_2\}$.

At time t_1, tasks $T_0 - T_{20}$ are grouped in a batch according to the batch resizing policies, and deployed on R_0; at time t_2, $T_{21} - T_{40}$ are grouped for R_1; and at time t_3, $T_{41} - T_{61}$ are grouped for R_2. Once the meta-scheduler retrieves a processed batch from a resource, it proceeds with the next task group deployment to the resource. The tasks are grouped and deployed at runtime when a particular resource becomes available (first-come-first-serve manner). Hence, we can not assure that the given budget and deadline are sufficient to process the entire BoT.

An economic or a commercial grid emphasises two main parameters known as the user's quality of service (QoS) requirements: application *deadline* and *budget*, within which all the BoT tasks should be executed in a timely manner. The resource providers impose charges on the users for consuming the CPU cycles or storage space of their resources. Apart from gaining profit, this helps to control the usage of the resources [30,28,10,26]. In such an environment, we propose to optimally map the tasks to the resources based on the QoS which we refer to as *advance QoS planning*: the task-resource mapping is performed prior to the task group deployment in order to maximise the processed task count within the QoS.

In our method, the grid meta-scheduler classifies the user tasks and deploys some tasks on the grid resources to learn the processing capabilities of the resources and the processing needs of the tasks. Then, a genetic algorithm in the meta-scheduler estimates the task-resource mapping based on the QoS and the historical performance data of the tasks. Following that, the tasks will be grouped according to the mapping, the performance data, and the batch resizing policies prior to the deployment. The cycle of mining performance data and advance QoS planning is repeated at a regular basis so that the scheduling and deployment process adheres to the latest grid status. Experiments are conducted in a small-scale grid to analyse the task success rate within the specified QoS. We also suggest to extend our method to accommodate a cloud environment and real-time tasks.

The rest of the paper is organised as follows: Section 2 delivers the related work. Our scope and contribution are described in Section 3. Section 4 summarises the batch resizing policies. The design of our advance QoS planning is given in Section 5. Section 6 presents the process flow of the grid meta-scheduler. Section 7 delivers the experimental analysis and our suggestions for cloud environment. Finally, Section 8 concludes this paper.

2 Related Work

Batch processing in distributed computing minimises the task waiting time and resource idle time as well as maximises the resource and network utilisation. This is proved by James et al in [17] by grouping and deploying independent tasks on clusters. Li Hui et al [14] developed a framework in Gracie (a grid platform for bio-informatics computational tasks) for grouping thousands of lightweight tasks into a single group. They reduced the cost of submitting requests to the resources and the cost of starting up the remote data transmissions. The application throughput increased from 30.3 tasks/second to 50.7 tasks/second when 51242 tasks from the genome alternative splicing application [12] were submitted to a SMP workstation as a single batch. Snehal Antani [3] has conducted business value assessment for IBM for utilising WebSphere Compute Grid which delivers a cloud-enabled batch-processing platform for enterprises. In the Websphere platform, the tasks that require data from a particular repository will be grouped and processed near the repository in order to minimise the overhead and increase the I/O throughput. This strategy highly reduces the IT and operational cost of an enterprise.

The dynamic computing environment leads to batch resizing efforts, where the batch size is adapted based on certain priorities. In [32], Sodan et al determined the optimal number of tasks in a batch based on minimum and maximum batch size, average run-time of the tasks, machine size, number of running tasks in the machine, and minimum and maximum node utilisation. The authors in [22], grouped and deployed the data-intensive tasks on a campus grid. The batch size is decided in such a way that the batch tasks can be executed within an hour.

Batch resizing has been investigated well when handling task migration. Eric Mohr et al introduced lazy task creation [21] in which, when a processor becomes available at the time of the child task execution, the parent task will be un-grouped from the child task and deployed on the available processor. Dror G. Feitelson [9] introduced migration- and buddy-based task packing schemes for performing gang scheduling in a parallel environment. When a node has completed its tasks, it accepts the task group from a busy node. The size of the task group is determined based on the processing slot available at the requesting node. Maghraoui et al [20] created user jobs with special constructs to indicate the atomic tasks in the jobs. Upon resource unavailability, the reverted jobs are checked for their granularity and resized (split or merged) before being migrated to the designated nodes.

In an economic grid, the main priority is given to the application budget and deadline [2, 5]; one would process as many tasks as possible within the given budget and deadline. The authors in [28, 30] introduced capacity planning when executing application jobs in a commercial or an economic grid. The term *capacity* covers the deadline, budget, storage space, resource utilisation, and load balancing. Their goal is to reserve the resources for later job executions in order to maximise the QoS or capacity requirements.

A structured leaning on the computing environment is a necessity when deciding the batch size. The authors in [29] used Particle Swarm Optimisation to learn and mine the task processing time in order to decide the batch size. A couple of research works had been conducted in mining performance data to predict task turnaround time in order to support the task scheduling decision [11, 18, 27, 31, 34]. In this paper, we use data mining and learning to predict the task-resource mapping in order to maximise the task success rate within the QoS requirement.

3 Scope and Contribution

Simulations and practical experiments prove that task group deployment with batch resizing strategy highly reduces the application processing time [23, 24]. The details of the batch resizing strategy is given in Section 4. In this paper, the task group deployment is adapted to accommodate advance QoS planning; the statistical performance data of the processed tasks is learned and fed to a genetic algorithm for planning the optimal QoS-based task-resource mapping before creating the task groups.

Figure 1 shows the individual and task group deployment strategies as opposed to the task group deployment with advance task-resource mapping. Assume that a resource needs 4 minutes to process the allocated tasks via individual deployment; having a specific resource utilisation constraint (e.g. CPU utilisation of 1 minute or 2 minutes), a well-planned task group deployment can process most of the tasks within the time frame by minimising the resource idle time and task waiting time. The concern of this paper is the task success rate that can be achieved within a particular time frame. This is a common concern when a user reserves the resources in advance and expects the tasks to be completed within the reserved time period [6–8, 25, 33]. This expectation motivates us to associate the task group deployment with advance QoS planning to further increase the task success rate.

The proposed method is designed for handling computational and independent tasks of parametric and non-parametric sweep BoT. There is no special API needed to create the user tasks. However, we assume that the user has rough estimations on the processing time and the output file size of the application tasks.

4 Batch Resizing Policies and Techniques

We summarise the batch resizing policies [23, 24] before proceeding with our actual contribution in Section 5. Figure 2 depicts the entities (User Application, Meta-Scheduler, and Grid Resources) in a typical grid. (1) The user application is associated with its QoS and passed to the meta-scheduler. (2-5) The meta-scheduler distributes the application tasks to the available resources for execution. (7-8) The processed tasks are then passed to the user via the meta-scheduler.

A task (T) consists of file(s); e.g. the execution instruction, library, program, and input data. Each task has its requirements in terms of task file size (TF-

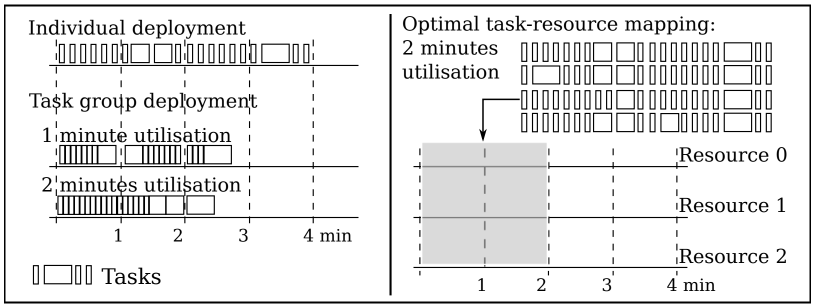


Fig. 1 (a) Individual Deployment; (b) Task Group Deployment; (c) Optimal Task-Resource Allocation.

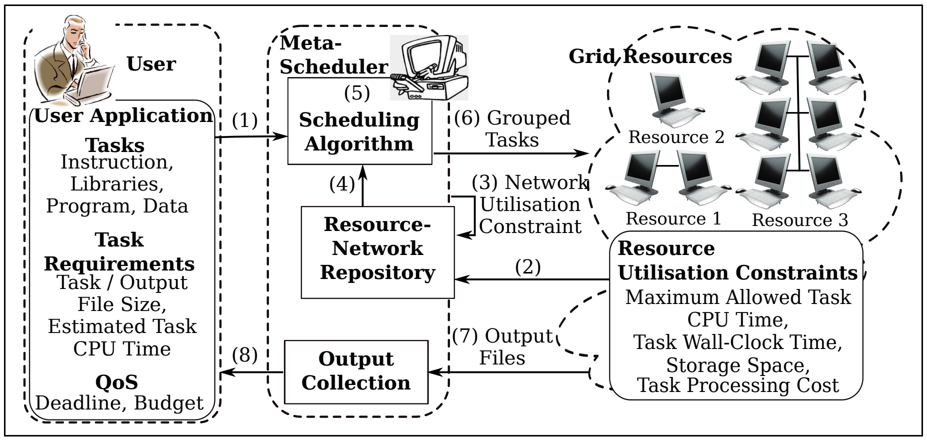


Fig. 2 Grid Entities and the Information Flow.

Size), output file size (*OFSize*), and estimated task CPU time (*ETCPUTime*). The *ETCPUTime* is the estimated task processing time provided by the user based on sample task executions on the user's local machine. The execution of a batch of tasks should adhere to the following utilisation constraints:

1. Resource utilisation constraints imposed by the resource provider to prevent the resource from being overloaded or misused [10,26]:
   - *Maximum Allowed CPU Time (MaxCPUTime)* for the execution of a batch at a resource.
   - *Maximum Allowed Wall-Clock Time (MaxWCTime)* that a batch can spend at the resource which includes the batch processing time and overhead (task waiting time, and task packing and unpacking overheads).
   - *Maximum Allowed Storage Space (MaxSpace)* that a batch and its output files can occupy at the resource at a time.
   - *Task Processing Cost (PCost)* per time unit charged by a resource.
2. Network utilisation constraint:
   - *Maximum Allowed File Transmission Time (MaxTransTime)* or the tolerance threshold that a meta-scheduler can wait for a batch and the relevant output files to be transmitted to and from a resource.

For a resource, R_j, when adding a task into a task group, TG, the resulting processing requirements of the TG should adhere to the following policies:

Policy 1: TG CPU time $\leq MaxCPUTime_{Rj}$
Policy 2: TG wall-clock time $\leq MaxWCTime_{Rj}$
Policy 3: TG and output transmission time $\leq MaxTransTime_{Rj}$
Policy 4: TG and output file size $\leq MaxSpace_{Rj}$
Policy 5: TG turnaround time $\leq$ Remaining *Deadline*
Policy 6: TG processing cost $\leq$ Remaining *Budget*
Policy 7: Number of tasks in $TG \leq$ Remaining BoT_{TOTAL}

BoT_{TOTAL} refers to the total tasks in the BoT. Policies 1-4 are on resource-network utilisation constraints, Policies 5-6 are on QoS, and Policy 7 is on task availability.

The varying processing capacities and workloads of the heterogeneous grid resources and the fluctuating interconnecting network require the meta-scheduler to learn and estimate the task CPU, wall-clock, and transmission times before applying the policies to create task groups. For this purpose, first, the application tasks are categorised based on their *TFSize*, *OFSize*, and *ETCPUTime*. Then, some tasks from the dominating categories are deployed on the resources to learn the performance of the resources and the network in regard to the task categories. The performance metrics are then used to decide the batch size for a resource according to the batch resizing policies. Periodic analysis is conducted to update the performance metrics, thus, to maintain the accuracy in deciding the batch size. The following sections further describe these batch resizing techniques.

4.1 Task Categorisation

The application tasks pass through three levels of categorisation and eventually divided into categories ($TCat$) based on their *TFSize*, *OFSize*, and *ETCPUTime* (provided by the user), and some pre-determined class intervals. An example of such categorisation is given in Fig. 3.

Level 1 divides the tasks into categories based on $TFSize$ of each task and the associated class interval, $TFSize_{CI}$. In Fig. 3, $TFSize_{CI}$ is set to 10 units and the resulting categories contain the tasks within the following $TFSize$ ranges:

$TCat_0$: 0 to $(1.5 \times TFSize_{CI})$ or $(0 < TFSize <15)$
$TCat_1$: $(1.5 \times TFSize_{CI})$ to $(2.5 \times TFSize_{CI})$ or $(15 \leq TFSize <25)$
$TCat_2$: $(2.5 \times TFSize_{CI})$ to $(3.5 \times TFSize_{CI})$ or $(25 \leq TFSize <35)$

In level 2, the categories from level 1 are further divided into sub-categories according to $ETCPUTime$ of each task and the class interval, $ETCPUTime_{CI}$. The resulting categories are divided into sub-categories in level 3 based on $OFSize$ and the class interval, $OFSize_{CI}$. In Fig. 3, $ETCPUTime_{CI} = 6$ and $OFSize_{CI} = 10$. A category is created only when there is at least one task belonging to that particular category. For each resulting $TCat$, the average task processing requirements are computed, namely average task file size ($AvgTFSize$), average estimated task CPU time (*AvgETCPUTime*), and average output file size ($AvgOFSize$) which will be used in the next technique (Section 4.2).

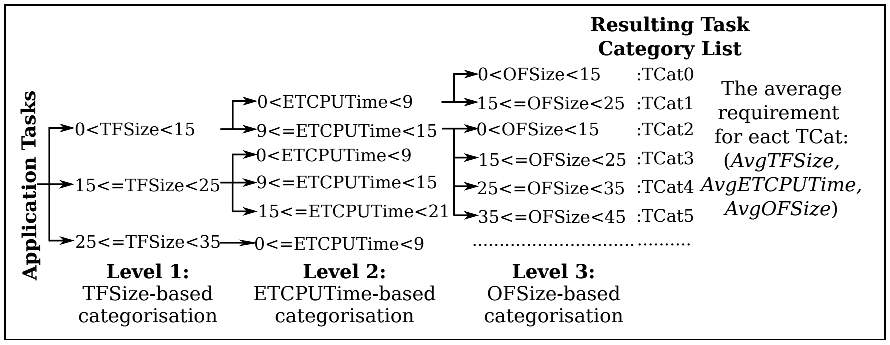


Fig. 3 Task Categorisation; $TFSize_{CI} = 10$, $ETCPUTime_{CI} = 6$, and $OFSize_{CI} = 10$.

4.2 Task Category-Resource Benchmarking

The *ETCPUTime* is estimated based on some sample executions on user's local machine; however, the CPU time of a task varies on different platforms. Therefore, some tasks from the dominating categories are benchmarked on the resources to learn how each resource and the interconnecting network react in regard to the categories. Upon retrieving a processed benchmark task, the following nine deployment metrics of the task are computed:

task file transmission time from meta-scheduler to resource ($MTRTime$); CPU time ($CPUTime$); wall-clock time ($WCTime$); output file transmission time from resource to meta-scheduler ($RTMTime$); turnaround time ($TRTime$); actual task processing time ($APTime$); resource overhead ($ROverhead$); processing overhead ($POverhead$); and processing cost ($PCost$); where,

$$ROverhead = WCTime - CPUTime \tag{1}$$

$$APTime = MTRTime + WCTime + RTMTime \tag{2}$$

$$POverhead = TRTime - APTime \tag{3}$$

$ROverhead$ covers task waiting time, and task packing and unpacking time at the resource; $POverhead$ is the overhead at the meta-scheduler; and $TRTime$ covers the time when the task is scheduled until its output files are retrieved by the meta-scheduler.

After completing all the benchmark tasks, the average of each deployment metric is computed for each task category-resource pair (*average analysis*). For a category k, the average deployment metrics on a resource j are expressed as average deployment metrics $TCat_k - R_j$, which consist of:

average task file transmission time ($AvgMTRTime_{k,j}$); average CPU time ($AvgCPUTime_{k,j}$); average wall-clock time ($AvgWCTime_{k,j}$); average output file transmission time ($AvgRTMTime_{k,j}$); average turnaround time ($AvgTRTime_{k,j}$); average actual task processing time ($AvgAPTime_{k,j}$); average resource overhead ($AvgROverhead_{k,j}$); average processing overhead ($AvgPOverhead_{k,j}$); average processing cost ($AvgPCost_{k,j}$). The average space ($AvgSpace_{k,j}$) consumed by the task and output files is obtained from $AvgTFSize$ and $AvgOFSize$ computed during task categorisation.

Not all the categories participate in this benchmark. Hence, their average deployment metrics are updated based on the average ratio of those categories participated in the benchmark and the average requirement details ($AvgTFSize$, $AvgETCPUTime$, and $AvgOFSize$) computed for each category during the categorisation process (Section 4.1). In short, this benchmark phase studies the response of the resources and the interconnecting network in respect to each category.

4.3 Periodic Average Analysis

Having the estimated average deployment metrics $TCat_k - R_j$ for all the categories, we can create the batches for a resource using the seven batch resizing policies. Due to task categorisation, Policy 7 is updated to control the total tasks that can be selected from a category.

Policy 7: Total tasks in TG from a $TCat_k \leq$ Remaining tasks in $TCat_k$ where, $k = 0,1,2,...,TCat_{TOTAL} - 1$ ($TCat$ ID); $TCat_{TOTAL}$ denotes total task categories.

When adding a task from a category, $TCat_k$, into a batch for a resource, R_j, the processing requirements of the task group will get accumulated from the average deployment metrics $TCat_k - R_j$. Tasks will be added until the resulting task group can satisfy all the seven batch resizing policies.

However, the deployment metrics $TCat_k - R_j$ may not reflect the grid after a time period [13] due to the fluctuating loads of the autonomous resources and the network which are shared by multiple users [11, 19]; the benchmark results can not be used for the entire BoT over a long period. Hence, the average analysis is conducted periodically to update the average deployment metrics $TCat_k - R_j$ according to the latest grid status. The overall process flow of our batch resizing strategy is given in Fig. 4.

5 Advance QoS Planning

In this paper, we propose to conduct advance QoS planning to increase the task completion rate within the given deadline and budget when deploying lightweight tasks on an economic grid. The average deployment metrics of the task categories will be fed into a genetic algorithm in order to optimally map the tasks to the resources based on the QoS. Following that, the tasks are grouped based on the batch resizing polices and dispatched to the designated resources. The process flow of batch resizing strategy with advance QoS planning is shown in Fig. 5.

After benchmarking, the meta-scheduler steps into advance QoS planning with the following strategy:

$TCat[TCat_{TOTAL}]$ refers to the file categories; $TCat[k]$ indicates the remaining tasks in category k or in $TCat_k$, where, $(0 \leq k < TCat_{TOTAL})$.

$GR[GR_{TOTAL}]$ refers to the grid resources; $GR[j]$ indicates resource j, where, $(0 \leq j < GR_{TOTAL})$.

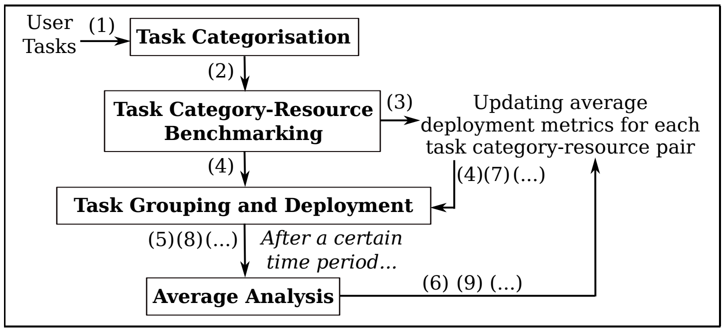

User Tasks
(1)
Task Categorisation
(2)
Task Category-Resource Benchmarking
(3)
Updating average deployment metrics for each task category-resource pair
(4)
(4)(7)(...)
Task Grouping and Deployment
(5)(8)(...)
After a certain time period...
Average Analysis
(6) (9) (...)

Fig. 4 Process Flow of the Batch Resizing Strategy.

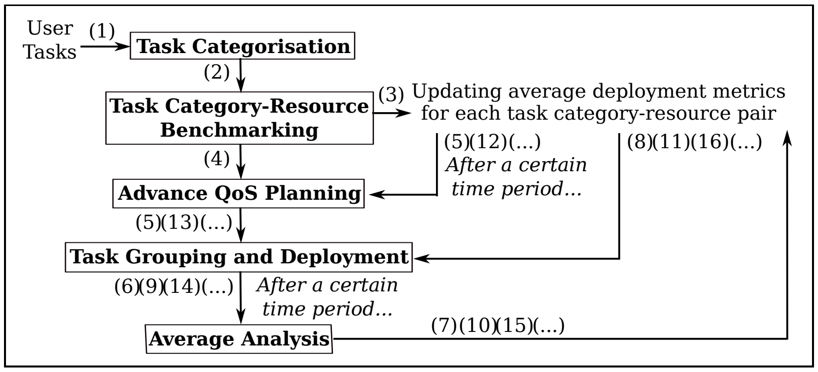

User Tasks
(1)
Task Categorisation
(2)
Task Category-Resource Benchmarking
(3)
Updating average deployment metrics for each task category-resource pair
(4)
(5)(12)(...)
After a certain time period...
(8)(11)(16)(...)
Advance QoS Planning
(5)(13)(...)
Task Grouping and Deployment
(6)(9)(14)(...)
After a certain time period...
Average Analysis
(7)(10)(15)(...)

Fig. 5 Process Flow of the Batch Resizing Strategy with Advance QoS Planning.

$AvgTRTime[GR_{TOTAL}][TCat_{TOTAL}]$ refers to the average turnaround time; $AvgTRTime[j][k]$ indicates the estimated average turnaround time of a task in category k on resource j.

$AvgPCost[GR_{TOTAL}][TCat_{TOTAL}]$ refers to the average processing cost; $AvgPCost[j][k]$ indicates the estimated average processing cost spent for executing a task in category k on resource j.

The term *task-resource allocation* will be used to refer to the allocation or mapping of tasks from each category to each resource. We represent task-resource allocation using $TRA[GR_{TOTAL}][TCat_{TOTAL}]$; $TRA[j][k]$ indicates the number of tasks allocated from a category k to resource j.

Assuming that $G = GR_{TOTAL}$ and $C = TCat_{TOTAL}$, then each $TRA[j][k]$ is determined in such a way that,

1. All the resources execute the tasks in parallel; they complete the task at different times. The estimated maximum turnaround time should be

less than the remaining deadline.

$$\begin{aligned}
maximum[&\sum_{k=0}^{C-1}(TRA[0][k] \times AvgTRTime[0][k]),\\
&\sum_{k=0}^{C-1}(TRA[1][k] \times AvgTRTime[1][k]),\\
&\sum_{k=0}^{C-1}(TRA[2][k] \times AvgTRTime[2][k]),\\
&\sum_{k=0}^{C-1}(TRA[...][k] \times AvgTRTime[...][k]),\\
&\sum_{k=0}^{C-1}(TRA[G-1][k] \times AvgTRTime[G-1][k])] \leq RemainingDeadline
\end{aligned}$$

2. All the resources impose utilisation charges. The estimated total processing cost should be less than the remaining budget.

$$\sum_{j=0}^{G-1}\sum_{k=0}^{C-1}(TRA[j][k] \times AvgPCost[j][k]) \leq RemainingBudget$$

3. The total allocated tasks should be less than the remaining BoT tasks.

$$\sum_{j=0}^{G-1}\sum_{k=0}^{C-1} TRA[j][k] \leq RemainingBoT_{TOTAL}$$

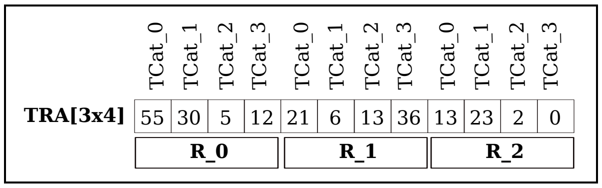
4. The total allocated tasks from each category should be less than the remaining tasks in the category.

for $k \leftarrow 0$ *to C-1*

$$\sum_{j=0}^{G-1} TRA[j][k] \leq TCat[k]$$

These four conditions serve as the fitness functions to a genetic algorithm. The algorithm evolves a population of chromosomes. Each chromosome represents the task-resource allocation in one-dimension $TRA[GR_{TOTAL} \times TCat_{TOTAL}]$. Assume that there are three resources and four task categories, a sample chromosome with its weight or values will be represented as in Fig. 6. The chromosome has 3×4 genes. Each gene indicates the number of tasks allocated from a particular $TCat$. The first four genes contains the allocated tasks from the four categories to resource R_0. This is followed by the task allocation for R_1 and R_2. In short, there are 216 tasks allocated for the resources; 89 tasks from $TCat_0$; 59 from $TCat_1$; 20 from $TCat_2$; and 48 from $TCat_3$.

The weight of each chromosome is evaluated using the fitness functions and those chromosomes with low fitness values will be removed from the population. New chromosomes will be created from the remaining chromosomes in order to maintain the population size for every evolution. Algorithm 1 depicts the refined fitness functions. The input to the algorithm:



	TCat_0	TCat_1	TCat_2	TCat_3	TCat_0	TCat_1	TCat_2	TCat_3	TCat_0	TCat_1	TCat_2	TCat_3
TRA[3x4]	55	30	5	12	21	6	13	36	13	23	2	0
	R_0				**R_1**				**R_2**			

Fig. 6 A Sample Chromosome Representation using $TRA[3 \times 4]$.

Algorithm 1: Fitness Function.

```
1  $FitnessValue = 0, TotalAllocatedTasks = 0$
2  $TotalProcessingCost = 0, gene = 0$
3  $AllocatedCategoryTasks[TCat_{TOTAL}]$ //Initialised to 0
4  $ResourceTurnaroundTime[GR_{TOTAL}]$ //Initialised to 0
5  //Getting the genes from the chromosome
6  for $j \leftarrow 0$ to $GR_{TOTAL}$ do
7      for $k \leftarrow 0$ to $TCat_{TOTAL}$ do
8          $TotalAllocatedTasks+ = TRA[gene]$
9          $AllocatedCategoryTasks[k]+ = TRA[gene]$
10         $TotalProcessingCost+ = TRA[gene] \times AvgPCost[j][k]$
11         $ResourceTurnaroundTime[j]+ = TRA[gene] \times AvgTRTime[j][k]$
12         $gene + +$
13 //The fitness functions
14 $TaskSelectionRatio = (TotalAllocatedTasks/RemainingBoT_{TOTAL})$
15 if $0.75 < TaskSelectionRatio \leq 1$ then
16     $FitnessValue + +$
17 if $TotalProcessingCost <= RemainingBudget$ then
18     $FitnessValue + +$
19 if $maximum(ResourceTurnaroundTime) <= RemainingDeadline$ then
20     $FitnessValue + +$
21 for $k \leftarrow 0$ to $TCat_{TOTAL}$ do
22     if $(0.9 < AllocatedCategoryTasks[k]/RemainingTasksInTCat_k) \leq 1.0$ then
23         $FitnessValue + +$
```

The chromosome ($TRA[GR_{TOTAL} \times TCat_{TOTAL}]$) randomly generated by the genetic algorithm, average processing cost ($AvgPCost[GR_{TOTAL}][TCat_{TOTAL}]$) and average turnaround time ($AvgTRTime[GR_{TOTAL}][TCat_{TOTAL}]$) of each $TCat_k - R_j$, and the remaining BoT tasks, category tasks, deadline, and budget.

In lines(6-12), the genes in the chromosome are retrieved to compute the total allocated tasks to all the resources, tasks allocated from each category, estimated total processing cost, and the estimated turnaround time to be consumed by each resource to process the allocated tasks. The resulting values are evaluated using the fitness functions:

1. **Lines(14-16)** The total allocated tasks is compared with the total remaining tasks. The chromosome fitness value will be incremented by one if the ratio is within 0.75 or 75% to 1.0 or 100%. The minimum ratio value can be increased ($> 75\%$) for better optimisation of the resulting task-resource allocation ($TRA[GR_{TOTAL} \times TCat_{TOTAL}]$).

2. **Lines(17-18)** The fitness value will be incremented if the estimated total processing cost can be accommodated by the remaining budget.
3. **Lines(19-20)** The fitness value will be incremented if the maximum of the resource turnaround times can be accommodated by the remaining deadline.
4. **Lines(21-23)** The tasks allocated from each category is compared with the remaining tasks in the category. The fitness value will be incremented if the ratio is within 0.9 or 90% to 1.0 or 100%. The minimum ratio value can be increased ($> 90\%$) in order to achieve a better optimisation.

A chromosome that satisfies all the fitness functions will produce a fitness value of $3 + TCat_{TOTAL}$. As mentioned in the fitness functions 1 and 4, if the minimum ratio value is increased, one can get a better chromosome. However, making the conditions to be more rigid may cause the evolution to end up with local minima. In short, the genetic algorithm produces an optimal task-resource allocation/mapping based on the performance data (average deployment metrics, $TCat_k - R_j$), remaining tasks, budget, and deadline. When creating a batch for a resource, this mapping will be referred for selecting a task from a particular category. Algorithm 2 explains the steps in deciding the batch size for a resource R_j. It determines the number of tasks from various categories that can be grouped for R_j. The algorithm requires the following input:

1. $TRA_{R_j}[TCat_{TOTAL}]$, the optimal task-resource allocation for R_j (the optimal chromosome produced for R_j).
2. Average deployment metrics of R_j which include
   $AvgCPUTime[TCat_{TOTAL}]$, $AvgWCTime[TCat_{TOTAL}]$,
   $AvgMTRTime[TCat_{TOTAL}]$, $AvgRTMTime[TCat_{TOTAL}]$,
   $AvgSpace[TCat_{TOTAL}]$, $AvgTRTime[TCat_{TOTAL}]$,
   and $AvgPCost[TCat_{TOTAL}]$.
3. Resource-network utilisation constraints of R_j which include $MaxCPUTime_{Rj}$, $MaxWCTime_{Rj}$, $MaxTransTime_{Rj}$, and $MaxSpace_{Rj}$.
4. Remaining budget, deadline, and category tasks.

In lines(5-6), the algorithm selects a task from a particular category based on the task availability (Policy 7) and the task-resource mapping. In line(7), the resulting task group ($TG[TCat_{TOTAL}]$) is checked for its accumulated processing requirements against the resource-network utilisation constraints using the batch resizing policies. If the policies are satisfied, the task is added into the group in line(9) and the processing requirements of the group are updated in lines(10-15). The action of task grouping is conducted by referring to the task selection in $TG[TCat_{TOTAL}]$. The task group is then dispatched to R_j.

6 Implementation of the Grid Meta-Scheduler

The process flow of the grid meta-scheduler with *Advance QoS Planning* module is given in Fig. 7. The meta-scheduler is developed using Java with multi-threading features.

1. ***Initial Preparation.*** (1) *Task Categorisation* retrieves task files from an user and categorises the tasks as explained in Section 4.1 according to the class

Algorithm 2: Determining the Size of a Task Group.

```
1  $TG[TCat_{TOTAL}]$ //Task group to be created; initialised to 0
2  $TG\_CPUTime = 0, TG\_WallclockTime = 0$
3  $TG\_TransmissionTime = 0, TG\_Space = 0$
4  $TG\_TurnaroundTime = 0, TG\_ProcessingCost = 0$
5  for $k \leftarrow 0$ to $TCat_{TOTAL}$ do
6      for $m \leftarrow 0$ to $RemainingTasksInTCat_k$ and $TRA_{R_j}[k]$ do
7          if
               $[(TG\_CPUTime + AvgCPUTime[k]) \leq MaxCPUTime_{Rj}]\&\&$
               $[(TG\_WallclockTime + AvgWCTime[k]) \leq MaxWCTime_{Rj}]\&\&$
               $[(TG\_TransmissionTime + AvgTransTime[k]) \leq MaxTransTime_{Rj}]\&\&$
               $[(TG\_Space + AvgSpace[k]) \leq MaxSpace_{Rj}]\&\&$
               $[(TG\_TurnaroundTime + AvgTRTime[k]) \leq RemainingDeadline]\&\&$
               $[(TG\_ProcessingCost + AvgPCost[k]) \leq RemainingBudget]$
           then
8              $TG[k]++$
9              $TG\_CPUTime+ = AvgCPUTime[k]$
10             $TG\_WallclockTime+ = AvgWCTime[k]$
11             $TG\_TransmissionTime+ = AvgTransTime[k]$
12             $TG\_Space+ = AvgSpace[k]$
13             $TG\_TurnaroundTime+ = AvgTRTime[k]$
14             $TG\_ProcessingCost+ = AvgPCost[k]$
15         else
16             Break //Break the loop and check the next TCat
17 //Note: $AvgTransTime[k] = AvgMTRTime[k]) + AvgRTMTime[k]$
```

intervals provided by the *Controller*. (2) It directs the task category list to the *Scheduling* module. Meanwhile, (3)(4) *Resource Planning*, running as a separate thread, keeps the information (hostname and the utilisation constraints ($MaxCPUTime$, $MaxWCTime$, $MaxSpace$)) of the participating grid resources to which the user has valid authorisations.

2. ***Benchmarking, Progress Monitoring, and Output Collection.*** (5) Having the task category and resource lists, *Benchmark Scheduling* selects benchmark tasks from the dominating categories and (6) dispatches the task files to the resources via *Task / Batch Deployment*. A task is dispatched to a resource once the resource has completed its current task. Hence, (7)(10) the *Progress Monitoring* will be informed about the dispatched task and the designated resource which then (8) instructs the *Output Collection* to trace the progress of the particular task. (9) Upon task completion, *Output Collection* retrieves the output files and (8) notifies the *Progress Monitoring*. For each resource, a set of *Scheduling*, *Task / Batch Deployment*, *Progress Monitoring*, *Output Collection*, *Deployment Analysis*, and *Average Analysis* modules are invoked as threads. The process of task deployment on a resource will not be affected or delayed by other resources; thus eliminating the synchronisation overhead. The steps (3-10) are repeated for each resource until all the benchmark tasks are successfully processed.

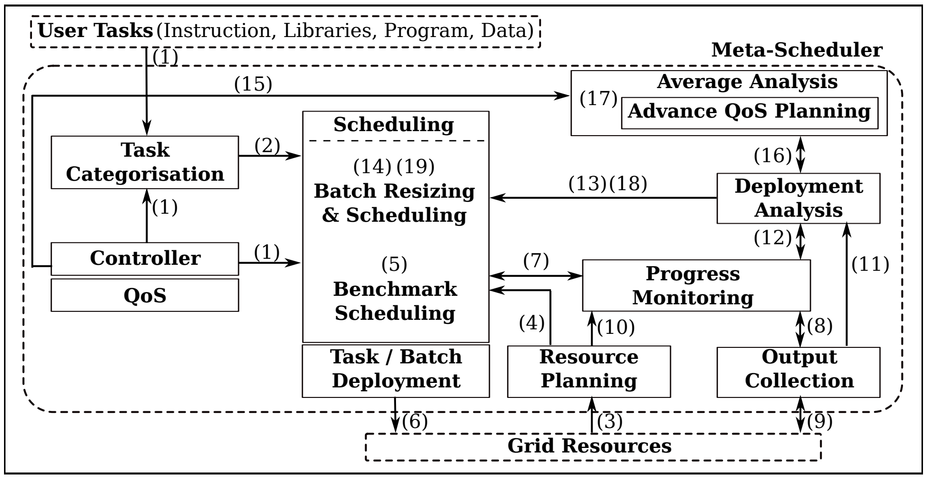


Fig. 7 Grid Meta-Scheduler with Advance QoS Planning Module.

3. ***Updating the Average Deployment Metrics.*** Upon retrieving the output files of a task, (11) *Deployment Analysis* will be updated by *Output Collection* in order to get the progress details of the processed task from *Progress Monitoring*. It then computes the nine deployment metrics of the task as explained in Section 4.2 and updates the remaining deadline and budget.

4. ***Advance QoS Planning.*** Once a particular resource completes its benchmark tasks, it is time to proceed with the first QoS planning. Assume that there are three resources (R_0, R_1, and R_2) and R_0 is the first one to complete all its benchmark tasks. (15) The *Controller* invokes an instance of *Average Analysis* to (16) gather the metrics of all the benchmark tasks processed by all the resources so far and (17) compute the average deployment metrics for each task category-resource pair ($TCat_k - R_j$). The average details are used by *Advance QoS Planning* to produce the optimal task-resource mapping based on the remaining budget and deadline as explained in Section 5.

5. ***Task Group Deployment and Average Analysis.*** The meta-scheduler will start to create task groups for R_0 according to the batch resizing policies and task-resource mapping. After conducting a number of task group deployment iterations for R_0, (15) the *Controller* will invoke an instance of *Average Analysis* specifically for R_0 to update its $TCat_k - R_0$ average deployment metrics based on the latest processed task groups.

6. ***Task Group Deployment and Periodic Average Analysis.*** (18) The *Deployment Analysis* passes the updated average details to *Batch Resizing & Scheduling* for (19)(6) the subsequent batch resizing and deployment activities for R_0. (16-19) The next *Average Analysis* for R_0 is invoked by the *Controller* after a certain number of task group deployments for regularly updating R_0's $TCat_k - R_0$ average deployment metrics. When R_1 or R_2 completes its benchmark tasks, the meta-scheduler performs the similar steps 6 and 7 to regularly update their performance metrics.

7. ***Periodic Average Analysis and Advance QoS Planning.*** The average analysis for each resource is conducted regularly in order to maintain the accuracy of the $TCat_k - R_j$ metrics. Similarly, the *Advance QoS Planning* will be invoked periodically as to update the task-resource mapping according to the latest $TCat_k - R_j$ metrics. The meta-scheduler repeats step 5 upon completing a certain number of task group deployment iterations. However, the frequency of conducting the advance QoS planning need to be monitored as genetic algorithms will increase the computation overheads at the meta-scheduler.

The genetic algorithm in the *Advance QoS Planning* module is implemented using Java Genetic Algorithms Package (JGAP) [1], an open-source toolkit. The meta-scheduler communicates with the resources using simple SSH, SCP, and RSH protocols for authentication, file transmission, and task execution purposes.

7 Performance Analysis

We conduct experiments in small scale environment using the proposed meta-scheduler to observe the impact of the advance Qos planning along with the batch resizing strategy in task group deployment. The next section presents some of the main configurations of the experiments.

7.1 Experimental Set-Up

Environment. As shown in Fig. 8, we will use five compute resources and the specifications of the machines are given in Table 1. Each resource is a single processing node with multiple cores. R_0 is located at the University of Melbourne (UNIMELB), Australia, whereas $R_1 - R_4$ are at the Multimedia University (MMU). The resource-network utilisation constraints of the participating resources are given in Table 2. The client is a dual-core machine (speed:2.40GHz, RAM:3GB), running on Linux.

BoT Application. Six computational programs are developed as trial applications: heat distribution, linear equation, finite differential equation, and three versions of Pi computations. 568 instances of these tasks are created with ($TFSize \leq 10KBytes$), ($ETCPUTime \leq 3.15minutes$), and ($OFSize \leq 5950KBytes$). These tasks will be deployed on the resources using various parameter sets. The majority of the tasks are considered fine-grain as 90.49% of the tasks have ($ETCPUTime \leq 2minutes$) and 79.93% of the tasks have ($OFSize \leq 1000KBytes$).

Task Group Deployment. In the experiments, the tasks are divided into categories according to the class intervals: $TFSize_{CI} = 1KByte$; $ETCPUTime_{CI} = 1minute$; and $OFSize_{CI} = 500KBytes$. Then, two tasks from the 20% of the dominating categories are selected as benchmark tasks for each resource. Once a resource completes its benchmark tasks, the average analysis for the resource will be conducted after every two iterations of task group deployments to the resource. Advance QoS planning will be carried out after the benchmarking, as well as after every 10 ($GR_{TOTAL} \times 2$) iterations of task group deployments completed by the meta-scheduler.

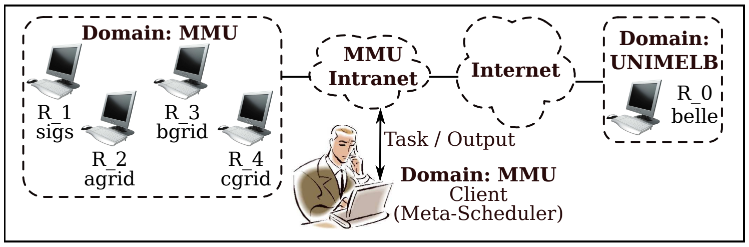


Fig. 8 Environmental Set-Up for the Experiments.

Table 1 Grid Resources

ID	Resource Name (Location)	Total Cores	Operating System, Speed, RAM
R_0	belle.csse.unimelb.edu.au (UNIMELB, Australia)	4	Ubuntu, 2.80GHz, 2GB
R_1	sigs.mmu.edu.my (MMU, Malaysia)	4	OpenSUSE, 2.40GHz, 2GB
R_2	agrid.mmu.edu.my (MMU, Malaysia)	2	Ubuntu, 2.40GHz, 1GB
R_3	bgrid.mmu.edu.my (MMU, Malaysia)	2	Ubuntu, 2.40GHz, 1GB
R_4	cgrid.mmu.edu.my (MMU, Malaysia)	2	Ubuntu, 2.40GHz, 1GB

Table 2 Resource-Network Utilisation Constraints

Utilisation Constraints	R_0	R_1	R_2	R_3	R_4
$MaxCPUTime(mins)$	5	4	6	5	4
$MaxWCTime(mins)$	10	8	10	15	10
$MaxSpace(MBytes)$	10	15	10	10	10
$MaxTransTime(mins)$	6	5	5	4	6
$TPCost$ (cost units per ms)	4	4	5	8	3

Genetic Algorithm. During the advance QoS planning, the genetic algorithm populates 80 chromosomes in each evolution. The best chromosome is selected based on one of the two termination conditions: when all the maximum fitness value is achieved or when the number of evolutions reaches 500.

Experiments. Two types of experiments are conducted in this analysis: (i) Task grouping based on the batch resizing strategy without advance QoS planning; and (ii) Task grouping based on the batch resizing strategy with advance QoS planning.

Note. We finalised the values given in our experiments and genetic algorithms (class intervals, total benchmark tasks, iterations of average analysis and QoS planning, total chromosomes, number of evolutions) after studying and observing a several experiments during the conduct of this research work.

7.2 Observations and Discussions

First, we deployed the 568 tasks on the five resources in a first-come-first-serve manner. On average, the individual task deployment consumed 141.40 minutes.

Task group deployment based on resource-network utilisation constraints (Policies 1-4) and task availability (Policy 7) exhibited a better performance with 109.29 minutes, a performance improvement of 22.71%. In the following observations, we analyse the task success rate within the specified deadline and budget when the task group deployment is conducted with and without the advance QoS planning.

7.2.1 Observation I - Process Flow

In this experiment, the meta-scheduler is provided with an application budget of 100000k cost units and a deadline of 75 minutes. The goal is to observe the process flow of the meta-scheduler and how the advance QoS planning reacts to the dynamic grid status.

17 categories were generated with the class intervals indicated in Section 7.1; thus, the highest fitness value that can be achieved is $3 + 17$. The resulting categories are:

0-330, 1-64, 2-20, 3-6, 4-22, 5-12, 6-12, 7-10, 8-10, 9-10, 10-8, 11-10, 12-16, 13-12, 14-8, 15-10, 16-8
(For example, 0-330 indicated 330 tasks in category 0 or $TCat_0$)

For the benchmarking, the meta-scheduler selected 20% of the dominating categories (three categories), namely, $TCat_0$, $TCat_1$, and $TCat_4$. Two tasks from each category were deployed on each resource; the total benchmark task is 30. We observed that R_0 was the first to complete its benchmark tasks. At this point, the remaining budget was 94086k cost units and the deadline was 69.5 minutes. The meta-scheduler then computed the average deployments metrics of all the $TCat_k - R_j$ pairs and proceeded with the advance QoS planning.

Table 3 presents the resulting optimal task-resource mapping with the fitness value of 11. The total tasks predicted by the genetic algorithm was 611; this task count is more than the remaining 538 tasks and expected to consume 127171k cost units in 170 minutes. Figure 9 shows the ratio of the actual vs predicted task count according to each category. The predicted task counts fell within the range 0.9 to 1.0 for 11 categories, yielding a fitness value of 11.

Subsequently, 13 tasks from $TCat_0$ were grouped for R_0. Meanwhile, R_1 managed to complete its benchmark tasks and 19 tasks from $TCat_0$ were grouped for R_1. The average analysis for each resource was repeated once the particular resource has successfully processed two task groups.

The meta-scheduler conducted the next advance QoS planning after completing $GR_{TOTAL} \times 2$ task group deployment iterations. Table 4 shows the total tasks and deployments completed before the second call for the advance QoS planning. The *Processed Tasks* includes six benchmark tasks and the tasks grouped in batches. The *Total Deployments* indicates the number of interactions between the resources and the meta-scheduler; i.e. the meta-scheduler deployed 32 tasks on R_3; six benchmark tasks via individual deployments; and 26 tasks via two task groups. In total, 191 tasks were processed via 41 deployments.

Finally, 480 out of 568 tasks were successfully processed within the deadline (75 minutes). The 480 tasks used up only 87330k cost units. As indicated in Table 4, 123 deployments were needed for processing the 480 tasks.

Table 3 Task-Resource Allocation, Fitness Value = 11

Resource	Task Categories																	Total Tasks
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	
R_0	65	11	5	0	3	5	0	3	2	3	5	3	2	3	2	7	3	122
R_1	65	6	5	2	0	3	3	1	1	3	2	1	6	3	2	6	0	109
R_2	65	15	2	1	3	8	3	3	2	10	2	3	4	1	3	3	5	133
R_3	65	11	2	2	3	7	3	0	3	3	5	3	0	2	5	3	6	123
R_4	57	11	5	1	3	3	3	3	2	3	6	0	4	2	3	10	8	124
Total	317	54	19	6	12	26	12	10	10	22	20	10	16	11	15	29	22	611
Actual	320	54	20	6	12	12	12	10	10	10	8	10	16	12	8	10	8	538

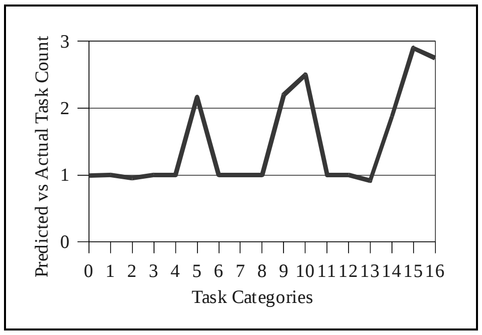


Fig. 9 Observation I: Ratio of the Actual vs Predicted Task Count.

Table 4 Benchmark and Task Group Deployment (Budget=100000k Cost Units, Deadline = 75 Minutes)

Resource	Before the Second QoS Planning		Overall Deployment	
	Processed Tasks	Total Deployments	Processed Tasks	Total Deployments
R_0	6+26	6+2	6+40	6+10
R_1	6+19	6+1	6+104	6+18
R_2	6+65	6+3	6+118	6+24
R_3	6+26	6+2	6+103	6+24
R_4	6+25	6+3	6+85	6+17
Total	191	41	480	123

7.2.2 Observation II - Comparison

In Observation I (Section 7.2.1), 480 tasks were successfully processed within 75 minutes using the batch resizing policies and the advance QoS planning. In this observation, we conducted experiment without QoS planning ($budget = 100000k$ cost units, $deadline = 75$ minutes); the size of a task group is determined merely based on the batch resizing policies.

Table 5 compares both the task deployments. Deployment without advance QoS planning processed 465 tasks in 75 minutes by consuming 90267k cost units; 90.27% of the budget was spent to complete 465 tasks. On the other hand, deployment with advance QoS planning used up only 87.33% of the budget and processed an additional 15 tasks.

Table 5 Budget=100000k cost units, Deadline=75 minutes

Task Group Deployment with Advance QoS Planning				
Resource	Processed Tasks	Total Deployments	Budget Spent (k)	Time Spent (mins)
R_0	46	16	15086	74.8
R_1	110	24	15084	74.8
R_2	124	30	18220	74.9
R_3	109	30	27528	74.9
R_4	91	23	11412	74.7
Total	480	123	87330	Max:74.9

Task Group Deployment without Advance QoS Planning				
Resource	Processed Tasks	Total Deployments	Budget Spent (k)	Time Spent (mins)
R_0	41	16	14824	73.8
R_1	103	22	15242	73.9
R_2	97	13	20459	74.9
R_3	133	31	28881	74.1
R_4	91	23	10861	72.6
Total	465	105	90267	Max:74.9

Table 6 $(100000k \leq Budget \leq 1200000k)$ Cost Units, $(75 \leq Deadline \leq 105)$ Minutes

	Experiments	**I**	**II**	**III**	**IV**	**V**
	Deadline (mins)	75	85	95	95	105
	Budget (k)	100000	100000	100000	1200000	1200000
With QoS Planning	Time Spent (mins)	74.9	82.7	81.5	94.3	99.9
	Budget Spent (k)	87330	99301	99717	112655	118566
	Processed Tasks	480	510	515	531	546
Without QoS Planning	Time Spent (mins)	74.9	81.2	80.3	95	96.3
	Budget Spent (k)	90267	99943	99655	117691	119756
	Processed Tasks	465	491	492	522	526

An important observation at R_2: The advance QoS planning spent 18220k to process 124 tasks, whereas in the deployment without QoS planning, 20459k was spent to process only 97 tasks. This proves the QoS benefit of task-resource mapping prior to task group deployment. A similar impact can be seen at R_1.

Following that, we conducted four sets of experiments (Experiment II-V) with varying budgets and deadlines as indicated in Table 6. In Experiment I, the deployment was constrained by the deadline, 75 minutes. Hence, we extended the deadline in Experiment II to 85 minutes and it can be noticed that the deployment in Experiment II was constrained by the budget, 100000k cost units. The budget constraint was confirmed in Experiment III by extending the deadline to 95 minutes. A similar configuration pattern was applied to the rest of the experiments.

All the experiments showed a similar time optimisation within the given time frame (Table 6); e.g. in Experiment I, the deployments with and without QoS planning used up 74.9 minutes; in Experiment II, the deployments consumed 82.7 minutes and 81.2 minutes respectively. The next concern is the task success rate and the budget spent within the time spent.

Figure 10 illustrates the observation in terms of percentages of the processed tasks and the budget spent. *Policies & QoS Planning* refers to the task grouping based on batch resizing policies and advance QoS planning, whereas *Policies* refers to the task grouping merely based on the batch resizing policies.

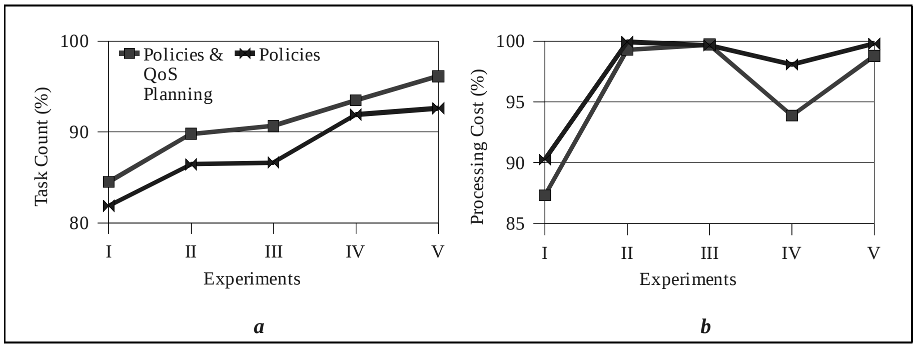


Fig. 10 Observation II, (a) Percentage of the Processed Task Count and (b) the relevant Cost or Budget Spent.

Figure 10(a) conveys that the deployment with advance QoS planning outperformed in terms of task success rate throughout the experiments. In Experiment III, both the deployments used up almost the same budget, but advance QoS planning managed to complete an additional 23 tasks (Fig. 10(b)). Experiment IV delivered a significant performance where advance QoS planning processed 93.40% tasks with only 93.88% of the budget. On the other hand, deployment without QoS planning spent 98.08% of the budget to complete only 91.90% tasks. In short, the optimal task-resource mapping increases the task success rate within the given time frame while minimising the overall task processing cost.

7.2.3 Observation III - Performance of the Genetic Algorithm

In our first observation (Section 7.2.1), 480 tasks were successfully processed in 75 minutes with 87330k cost units (out of 100000k). In this section, we conducted five experiments with varying budgets to analyse the performance of the genetic algorithm in optimising the available budget along with the deadline, 75 minutes.

Table 7 shows the experiment configurations and the resulting performance in terms of QoS utilisation and the total processed tasks. Experiment III reflects the one conducted in Observation I. The task success rate with the percentage of the budget spent are presented in Fig. 11.

The least number of processed tasks is recorded in Experiment I due to the budget constraint of 80000k cost units. Throughout the experiments, the task success rate increases as the budget increases.

R_3 imposes high processing charges as compared to other resources. Hence, in Experiment III (Observation I, Table 5), the advance QoS planning allocated fewer tasks to R_3 as compared to the task deployment without task-resource mapping. However, in this observation, more tasks were allocated to R_3 during Experiments V-VI as the available budget was more than enough to accommodate the deployment for 75 minutes. For example, 129 tasks were allocated to R_3 in Experiment V and 124 tasks in Experiment VI. On the other hand, only 109 tasks were allocated to R_3 in Experiment III (Table 5). This reflects the optimisation feature of the advance QoS planning. In spite of this optimisation, we did not see a convincing usage of the available budget as shown in Fig. 11 due to the deadline constraint.

Table 7 ($80000k \leq Budget \leq 400000k$) Cost Units, Deadline = 75 Minutes

Experiments	**I**	**II**	**III**	**IV**	**V**	**VI**
Budget (k)	80000	90000	100000	200000	300000	400000
Time Spent (mins)	64.3	73.4	74.9	74.9	74.9	74.9
Budget Spent (k)	79898	89719	87330	88891	91309	91518
Processed Tasks	404	481	480	483	490	490

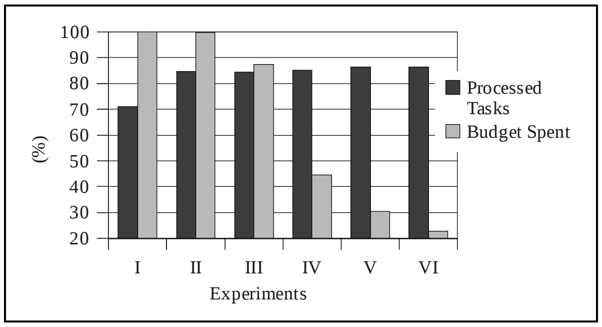


Fig. 11 Observation III, Processed Tasks and the Budget Spent.

7.2.4 Issues and Future Direction

Advance QoS planning increases the task success rate as compared to the conventional batch processing. However, there are a couple of issues need to be tackled in this method.

As shown in Fig. 5, the proposed batch resizing strategy requires the meta-scheduler to keep monitoring, learning and updating the average deployment metrics for each task category-resource pair using a genetic algorithm. A frequent conduct of advance QoS planning may delay the entire task group deployment as the genetic algorithm will increase the computation overhead at the meta-scheduler. With the current high-end machines, the overhead or latency can be reduced or hidden by configuring the genetic algorithm to keep running in parallel as a separate thread. The meta-scheduler can obtain the latest, optimal task-resource mapping at any time from the genetic algorithm.

The advance QoS planning needs to be conducted frequently only when the grid status varies drastically at runtime. Practically, the meta-scheduler must have completed at least one task group deployment to each resource before the subsequent QoS planning. For simplicity, in our meta-scheduler, the planning is carried out once it has successfully processed $GR_{TOTAL} \times 2$ task groups. We also conducted experiments in which the planning is performed after every $GR_{TOTAL} \times 1$ task group deployment. We observed almost similar task success rate and budget utilisation as our grid status did not fluctuate much during the experiments.

Another issue is on the scalability of advance QoS planning. Our experiments were conducted in a small-scale environment. How will the algorithm perform when there are tens of thousands of tasks with varying *TFSize*, *OFSize*, and *ETCPUTime* in an application? What if the environment consists of thousands of CPUs for executing the massive application tasks? In this scenario, we can adapt the clustering techniques to minimise the parameters in the QoS planning.

For example, upon benchmarking, the resources can be clustered according to their common characteristics. The neighbouring task categories or categories with similar processing requirements can be clustered as well. Then, the average performance metrics of each category cluster on each resource cluster will be computed to be fed to the genetic algorithm.

Besides these issues, the proposed task group deployment can be easily adapted to cloud computing. The cloud users may use multiple physical hardware, each running a number of instances or virtual machines (VMs). Similar to the grid resources, upon the invocation of the VMs, their performance can be benchmarked, and the application tasks can be grouped according to the capacities of the VMs. As cloud is based on pay-per-use model [15], the advance QoS planning can highly minimise the expenses and increase the utilisation of the VMs. In the case where the VMs are allocated from a clustered physical machines, then, the user can transfer the entire application tasks to one of the VMs and let the particular VM act as the meta-scheduler. This will reduce the latencies caused by multiple file transmissions between the user and the VMs, and progress monitoring of the tasks dispatched to the VMs.

The task group deployment can be adapted to handle real-time tasks; e.g. operations on weather information or road traffic data collected at every specific interval. Thousands of tasks with real-time data arrive at the meta-scheduler at regular intervals which need to be executed immediately in order to produce the timely response to the public or consumers. The tasks with minimal granularity can be grouped and passed to the resources or VMs, thus, minimising the relevant processing and communication overheads. The *OFSize* and *ETCPUTime* of the real-time tasks can be estimated based on the historical records of task execution. The task categorisation will be conducted online; when the tasks arrive at the meta-scheduler, they will be added into the existing categories according to their *OFSize* and *ETCPUTime*, and new categories will be created as needed. Subsequently, the average analysis and advance QoS planning will be conducted to create the task groups.

8 Conclusions

Batch resizing strategy highly decreases the turnaround time when deploying fine-grain tasks on a grid. The performance of the task group deployment can be improved to facilitate the economic grid in which the application deadline and budget play a critical role as the major constraints. In this paper, we proposed to conduct advance QoS planning to determine the optimal task-resource mapping prior to deploying the tasks on the grid resources. We learn the performance data of the application tasks at regular basis and the relevant statistical information is used to produce the task-resource mapping using a genetic algorithm. We developed a grid meta-scheduler for conducting experiments in a practical environment. Experiments revealed that the task group deployment with advance QoS planning outperforms in terms of task success rate as compared to the deployment without QoS planning. This strategy is targeted for those who execute BoT tasks within the reserved time frame in an economic or a commercial grid.

9 Acknowledgement

This paper is an extended version from ICA3PP 2010 [23]. We would like to acknowledge e-ScienceFund, Ministry of Science, Technology, and Innovation (MOSTI), Malaysia, and Endeavour Awards, Department of Innovation, Industry, Science and Research (DIISR), Australia, for supporting the research work and the development of the meta-scheduler described in this paper.

References

1. Jgap java genetic algorithm package. URL http://jgap.sourceforge.net/. Accessed on 30th March 2011
2. Abramson, D., Buyya, R., Giddy, J.: A computational economy for grid computing and its implementation in the nimrod-g resource broker. Future Generation Computer System **18**(8), 1061–1074 (2002)
3. Antani, S.: Batch processing with websphere compute grid: Delivering business value to the enterprise. Tech. rep., IBM (2010). URL http://www.redbooks.ibm.com/abstracts/redp4566.html
4. Baker, M., Buyya, R., Laforenza, D.: Grids and grid technologies for wide-area distributed computing. Softw. Pract. Exper. **32**(15), 1437–1466 (2002)
5. Barmouta, A., Buyya, R.: Gridbank: A grid accounting services architecture (gasa) for distributed systems sharing and integration. In: Proceedings of the 17th International Symposium on Parallel and Distributed Processing, p. 245.1. IEEE Computer Society, Washington, DC, USA (2003)
6. Castillo, C., Rouskas, G.N., Harfoush, K.: On the design of online scheduling algorithms for advance reservations and qos in grids. In: International Symposium on Parallel and Distributed Processing, pp. 1–10. California, USA (2007)
7. Castillo, C., Rouskas, G.N., Harfoush, K.: Online algorithms for advance resource reservations. Journal of Distributed and Parallel Computing **71**(7), 963–973 (2011)
8. Elmroth, E., Tordsson, J.: Grid resource brokering algorithms enabling advance reservations and resource selection based on performance predictions. Future Generation Computer System **24**(6), 585–593 (2008)
9. Feitelson, D.G.: Packing schemes for gang scheduling. In: Proceedings of the Workshop on Job Scheduling Strategies for Parallel Processing, pp. 89–110. Springer-Verlag, London, UK (1996)
10. Feng, J., Wasson, G., Humphrey, M.: Resource usage policy expression and enforcement in grid computing. In: Proceedings of the 8th IEEE/ACM International Conference on Grid Computing, pp. 66–73. IEEE Computer Society, Washington, DC, USA (2007)
11. Gao, Y., Rong, H., Huang, J.Z.: Adaptive grid job scheduling with genetic algorithms. Future Generation Computer Systems **21**(1), 151–161 (2005)
12. Guttmacher, A.E., Collins, F.S.: Genomic medicine–a primer. The New England Journal of Medicine **347**(19), 1512–1520 (2002)
13. Huang, P., Peng, H., Lin, P., Li, X.: Static strategy and dynamic adjustment: An effective method for grid task scheduling. Future Generation Computer Systems **25**(8), 884–892 (2009)
14. Hui, L., Yu, H., Xiaoming, L.: A lightweight execution framework for massive independent tasks. In: Workshop on Many-Task Computing on Grids and Supercomputers, pp. 1–9. IEEE (2008)
15. Huu, T.T., Koslovski, G.P., Anhalt, F., Montagnat, J., Primet, P.V.B.: Joint elastic cloud and virtual network framework for application performance-cost optimization. Journal of Grid Computing **9**(1), 27–47 (2011)
16. Jacob, B., Brown, M., Fukui, K., Trivedi, N.: Introduction to Grid Computing. IBM Publication (2005)
17. James, H., Hawick, K., Coddington, P.: Scheduling independent tasks on metacomputing systems. In: Proceedings of Parallel and Distributed Computing Systems, pp. 156–162. Fort Lauderdale, US (1999)
18. Li, H., Groep, D., Wolters, L.: Mining performance data for metascheduling decision support in the grid. Future Generation Computer Systems **23**, 92–99 (2007)

19. Liu, D., Cao, Y.: Computational intelligence and security. In: Y. Wang, Y.M. Cheung, H. Liu (eds.) CIS'06, chap. CGA: Chaotic Genetic Algorithm for Fuzzy Job Scheduling in Grid Environment, pp. 133–143. Springer-Verlag, Berlin, Heidelberg (2007)
20. Maghraoui, K.E., Desell, T.J., Szymanski, B.K., Varela, C.A.: The internet operating system: Middleware for adaptive distributed computing. International Journal of High Performance Computing Applications **20**(4), 467–480 (2006)
21. Mohr, E., Kranz, D.A., Halstead, R.H.J.: Lazy task creation: A technique for increasing the granularity of parallel programs. IEEE Transactions on Parallel and Distributed Systems **2**(3), 264–280 (1991)
22. Moretti, C., Bui, H., Hollingsworth, K., Rich, B., Flynn, P., Thain, D.: All-pairs: An abstraction for data-intensive computing on campus grids. IEEE Transactions on Parallel Distributed Systems **21**, 33–46 (2010)
23. Muthuvelu, N., Chai, I., Chikkannan, E., Buyya, R.: On-line task granularity adaptation for dynamic grid applications. In: Proceedings of the 10th International Conference on Algorithms and Architectures for Parallel Processing, vol. 6081, pp. 266–277 (2010)
24. Muthuvelu, N., Chai, I., Chikkannan, E., Buyya, R.: Batch resizing policies and techniques for fine-grain grid tasks: the nuts and bolts. Journal of Information Processing Systems **7**(2), 299–320 (2011)
25. Prodan, R., Wieczorek, M.: Negotiation-based scheduling of scientific grid workflows through advance reservations. Journal of Grid Computing **8**(4), 493–510 (2010)
26. Rahman, M., Ranjan, R., Buyya, R.: Cooperative and decentralized workflow scheduling in global grids. Future Generation Computer Systems **26**(5), 753–768 (2010)
27. Ramírez-Alcaraz, J.M., Tchernykh, A., Yahyapour, R., Schwiegelshohn, U., Quezada-Pina, A., González-García, J.L., Hirales-Carbajal, A.: Job allocation strategies with user run time estimates for online scheduling in hierarchical grids. Journal of Grid Computing **9**(1), 95–116 (2011)
28. Risch, M., Altmann, J.: Capacity planning in economic grid markets. In: Proceedings of the 4th International Conference on Advances in Grid and Pervasive Computing, GPC '09, pp. 1–12. Springer-Verlag, Berlin, Heidelberg (2009)
29. Sadasivam, G.S., Rajendran, V.V.: An efficient approach to task scheduling in computational grids. International Journal of Computer Science and Applications **6**(1), 53–69 (2009)
30. Siddiqui, M., Villazón, A., Fahringer, T.: Grid capacity planning with negotiation-based advance reservation for optimized qos. In: Proceedings of the 2006 ACM/IEEE conference on Supercomputing, pp. 103–118. ACM, New York, NY, USA (2006)
31. Smith, W., Foster, I., Taylor, V.: Predicting application run times with historical information. Journal of Parallel and Distributed Computing **64**, 1007–1016 (2004)
32. Sodan, A.C., Kanavallil, A., Esbaugh, B.: Group-based optimizaton for parallel job scheduling with scojo-pect-o. In: Proceedings of the 22nd International Symposium on High Performance Computing Systems and Applications, pp. 102–109. IEEE Computer Society, Washington, DC, USA (2008)
33. Takefusa, A., Nakada, H., Kudoh, T., Tanaka, Y.: An advance reservation-based co-allocation algorithm for distributed computers and network bandwidth on qos-guaranteed grids. In: Proceedings of the 15th international conference on Job scheduling strategies for parallel processing, pp. 16–34. Springer-Verlag, Berlin, Heidelberg (2010)
34. Talby, D., Feitelson, D.G.: Improving and stabilizing parallel computer performance using adaptive backfilling. In: Proceedings of the 19th IEEE International Parallel and Distributed Processing Symposium, p. 84.1. IEEE Computer Society, Washington, DC, USA (2005)
35. Venugopal, S., Buyya, R., Lyle, W.: A grid service broker for scheduling e-science applications on global data grids. Concurrency and Computation: Practice and Experience (CCPE) **18**, 685–699 (2006)

Dr. Nithiapidary received her B.IT degree from Universiti Tenaga Nasional, Malaysia, in August 2003, M.IT degree from the University of Melbourne, Australia, in December 2004, and Ph.D in IT from Multimedia University Malaysia in 2012. She is teaching at Multimedia University, Malaysia, since 2005. Her research interests include: Distributed and Parallel Processing, and Data Communication. She is a member of the IEEE Computer Society.

Dr. Ian Chai received his B.Sci. and M.Sci. in Computer Science from the University of Kansas and his Ph.D. in Computer Science from the University of Illinois at Urbana-Champaign. Since 1999, he has taught at Multimedia University in Cyberjaya, Malaysia.

Prof. Dr. C.Eswaran received his B.Tech, M.Tech, and Ph.D degrees from the Indian Institute of Technology Madras, India where he worked as a Professor in the Department of Electrical Engineering until January 2002. Currently he is working as a Professor in the Faculty of Information Technology, Multimedia University, Malaysia. Dr.C.Eswaran served as a visiting faculty and research fellow in many international universities. He has supervised successfully more than 25 Ph.D/M.S students and has published more than 150 research papers in reputed International Journals and Conferences. Prof.C. Eswaran is a senior member of IEEE.

Prof. Dr. Rajkumar Buyya is Professor of Computer Science and Software Engineering; and Director of the Cloud Computing and Distributed Systems (CLOUDS) Laboratory at the University of Melbourne, Australia. He is also serving as the founding CEO of Manjrasoft Pty Ltd., a spin-off company of the University, commercialising its innovations in Grid and Cloud Computing. He has authored and published over 300 research papers and four text books. The books on emerging topics that Dr. Buyya edited include, High Performance Cluster Computing (Prentice Hall, USA, 1999), Content Delivery Networks (Springer, Germany, 2008) and Market-Oriented Grid and Utility Computing (Wiley, USA, 2009). He is one of the highly cited authors in computer science and software engineering worldwide (h-index=48, g-index=104, 12500+ citations).

Software technologies for Grid and Cloud computing developed under Dr. Buyya's leadership have gained rapid acceptance and are in use at several academic institutions and commercial enterprises in 40 countries around the world. Dr. Buyya has led the establishment and development of key community activities, including serving as foundation Chair of the IEEE Technical Committee on Scalable Computing and four IEEE conferences (CCGrid, Cluster, Grid, and e-Science). The contributions and international research leadership of Dr. Buyya are recognised through the award of "2009 IEEE Medal for Excellence in Scalable Computing" from the IEEE Computer Society, USA. For further information on Dr. Buyya, please visit his cyberhome: www.buyya.com