Batch Resizing Policies and Techniques for Fine-Grain Grid Tasks: The Nuts and Bolts

Nithiapidary Muthuvelu*, Ian Chai*, Eswaran Chikkannan*
and Rajkumar Buyya**

Abstract— The overhead of processing fine-grain tasks on a grid induces the need for batch processing or task group deployment in order to minimise overall application turnaround time. When deciding the granularity of a batch, the processing requirements of each task should be considered as well as the utilisation constraints of the interconnecting network and the designated resources. However, the dynamic nature of a grid requires the batch size to be adaptable to the latest grid status. In this paper, we describe the policies and the specific techniques involved in the batch resizing process. We explain the nuts and bolts of these techniques in order to maximise the resulting benefits of batch processing. We conduct experiments to determine the nature of the policies and techniques in response to a real grid environment. The techniques are further investigated to highlight the important parameters for obtaining the appropriate task granularity for a grid resource.

Keywords— Batch Resizing, Task Granularity, Global Grid, Application Turnaround Time

1. INTRODUCTION

Utilising a grid [1] for executing fine-grain tasks increases the overall application processing time due to the overhead involved in handling each small-scale task [2]. This overhead is mainly caused by the communication latency when transferring a particular task file from a scheduler to the designated resource and retrieving the processed task file from the resource [3-5].

This motivates the need for batch processing in a grid; we will refer to this as task group deployment. Multiple tasks are grouped and processed together mainly for reducing the processing overhead, especially in terms of task waiting time [6]. As shown in equation (1) and Fig. 1, the total overhead in deploying four tasks (T) individually can be reduced if the tasks are grouped and deployed together in a batch or a task group (TG). Section 4 of this paper produces the experimental results to prove the impacts of individual, fine-grain task processing in a grid. Moreover, the experiments reveal that deploying lightweight tasks on a grid leads to inefficient resource-network utilisation and unfavourable application throughput.

※ This paper is an extended version of ICA3PP 2010 [16]. Here, we would like to acknowledge e-ScienceFund, Ministry of Science, Technology, and Innovation, Malaysia, and Endeavour Awards, Department of Innovation, Industry, Science and Research, Australia, for supporting the research work and the development of the meta-scheduler described in this paper.

Manuscript received October 1, 2010; accepted January 25, 2011.

Corresponding Author: Nithiapidary Muthuvelu

* Multimedia University, Persiaran Multimedia, 63100 Cyberjaya, Selangor, Malaysia ({nithiapidary, ianchai, eswaran}@mmu.edu.my)

** Cloud Computing and Distributed Systems (CLOUDS) Laboratory, Dept. of Computer Science and Software Engineering, The University of Melbourne, 3053 Carlton, Victoria, Australia (raj@csse.unimelb.edu.au)

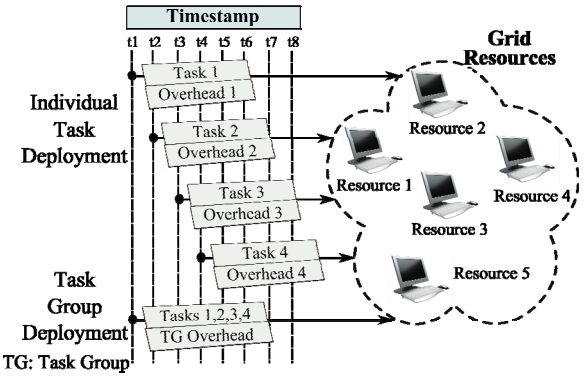

Fig. 1. Individual vs task group deployment

$$overhead_{T1} + overhead_{T2} + overhead_{T3} + overhead_{T4} > overhead_{TG} \quad (1)$$

This motivates batch processing in a grid whereby the fine-grain tasks are grouped into batches before being deployed on the resources. Here, the concern is the size of the batch or the task granularity; *"How many tasks can be grouped in a batch for a particular resource?*".

This concern leads us to understand the factors that affect the batch size before proceeding with the task grouping process. In this paper, we learn about the impacts of batch size and the overall task group deployment process on the participating grid entities. In conjunction with this motivation, we discover the policies and techniques to determine the size of a task group for a grid resource. We further investigate the batch resizing techniques to realise the important parameters involved in obtaining the appropriate task granularity.

The batch resizing policies and techniques are mainly for computation-intensive, bag-of-tasks (BoT) applications. All the tasks in the BoT are independent and have a similar compilation platform. Our goal is to reduce the overall application turnaround time while maximising the usage of resource and network capacities.

The rest of the paper is organised as follows: Section 2 presents the related work. Section 3 conveys the set-up of all the experiments conducted in this paper. In Section 4, Experiment Phase I is conducted to describe the motivation, policies, and issues involved in the batch resizing process. Section 5 explains the task categorisation and benchmarking techniques to decide the granularity for BoT applications. These two techniques are then implemented in a meta-scheduler explained in Section 6. Section 7 presents Experiment Phase II, which analyses the performance of the two techniques. The importance of periodic average analysis is realised in Section 8. Section 9 produces the results from Experiment Phase III (experiments on average analysis). Finally, Section 10 concludes the paper by suggesting future work.

2. RELATED WORK

Task resizing has become an interesting focus for research in recent years [7-9]. Maghraoui et al [10] used special constructs in the user job files to indicate the atomic computational units of the jobs. In a distributed platform, upon resource unavailability, these constructs are referred

accordingly to split or merge the computational units before migrating the jobs to other resources.

There are a number of simulations conducted that involve determining the granularity of a batch in parallel and distributed environments. Sodan et al [11] proposed to compute the batch size based on the average runtime of the jobs, machine size, number of running jobs in the machine, and minimum/maximum node utilisation. Their simulations did not consider the varying network usage or bottleneck, and it limits the flexibility of the job groups by fixing the upper and lower bounds of the number of jobs in the group.

The authors in [12,13] grouped the tasks based on resource's Million Instructions Per Second (MIPS) and task's Million Instructions (MI); e.g. for utilising a resource with 500 MIPS for 3 seconds, tasks were grouped into a single task file until the maximum MI of the file was 1500.

Realising the inaccuracy of considering MI and MIPS [14], the authors in [15] enhanced the task grouping process by considering the parameters from users (budget and deadline), applications (estimated task CPU time and task file size), utilisation constraints of the resources (maximum allowed CPU and wall-clock time, and task processing cost per time unit), and transmission time tolerance (maximum allowed task file transmission time). They used genetic algorithms in the simulations to divide the user jobs to all the available resources before proceeding with the actual task grouping and deployment activities. However, it was assumed that the task file size reflected the processing length of the task.

Following from that, we [16] enhanced the algorithm in [15] by considering the task file size apart from its processing length, the space availability at the resources, and the output file transmission time. We incorporated the task grouping policies and techniques in the algorithm that also supports an unlimited number of user tasks arriving at the scheduler at runtime.

This paper is an extended version of our previous work [16]. We developed a meta-scheduler called GridBatch (using Java and multi-threading features) with our proposed batch resizing policies and techniques. The GridBatch was tested in a grid environment with simple computational, independent tasks. All the parameters involved in the techniques are discussed in details with experiments in order to realise their impacts on the entire batch resizing process.

3. EXPERIMENTAL PLAN

Three phases of experiments will be conducted throughout the paper. This section provides the details of the grid resources and the BoT application used in the experiments.

3.1 Grid Resources

Table 1 lists the five grid resources that will be participating in the experiments. 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), Malaysia. The experimental set-up is given in Fig. 2. The client machine (operating system: Ubuntu, speed: 2.40GHz, RAM: 3GB) is located within the MMU domain.

3.2 BoT Application

The BoT grid application used throughout this paper is kind of non-parametric sweep applica-

Table 1. Grid resources

ID	Resource Name (Location)	CPUs	Operating System, Speed, RAM
R_0	belle.csse.unimelb.edu.my (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

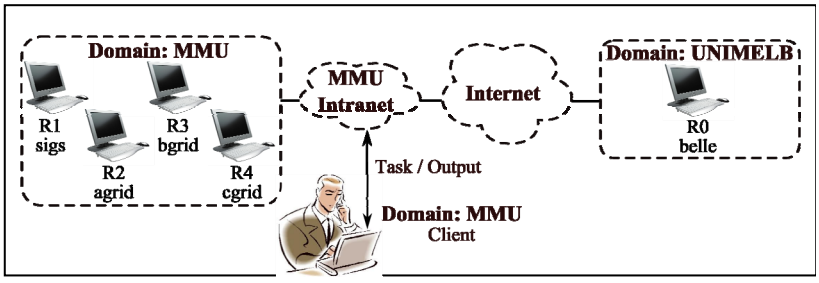

Fig. 2. Environmental set-up for the experiments

tion. The BoT comprises instances of six computational programs, namely, heat distribution, linear equations, finite differential equations, and three versions of Pi computations. The instances of each program are to be executed using various parameter sets.

There are 568 tasks in this BoT; the task file size ranges from 7-10 KBytes; estimated task CPU time ranges from 0.07-3.15 minutes; and the estimated output file size is 0.05-5950 KBytes. The majority of the tasks are considered fine-grain: 90.49% of the tasks have (CPU time $\leq$ 2minutes) and 79.93% of the tasks have (output file size $\leq$ 1000KBytes).

Application turnaround time refers to the total time taken to successfully process all the tasks in the BoT. It includes file transmission time, task waiting and execution time, as well as the overhead at the scheduler, interconnecting network, and the resources.

4. Batch Resizing: Motivation, Policies, and Issues

This section compares the overhead in individual task deployment with group-based task deployment, thus presenting the motivation towards batch processing in a grid environment. For this purpose, we conduct Experiment Phase I with the experimental set-up given in Fig. 2.

4.1 Performance Analysis – Experiment Phase I

We select 50 tasks with (CPU time $\leq$ 1minute) and (output file size $\leq$ 16KBytes) from the BoT for further deployment on the five resources. There are 10 experiments in this phase and the task granularity or batch size for each experiment is indicated in Table 2.

Experiment I reflects the individual task deployment where the tasks are transmitted to the resources one-by-one; the Task Granularity is set to 1. This induces 50 task executions and 100 file transmissions (50 for task file and 50 for output file transmissions). A task will be scheduled and dispatched to a resource when the resource has successfully completed the current task.

In Experiment II, two tasks are grouped (compressed) together for deployment on a resource.

Table 2. Task granularities for Experiment Phase I

Experiment	I	II	III	IV	V	VI	VII	VIII	IX	X
Task Granularity	1	2	4	6	8	10	12	14	16	18
Total Task groups	50	25	13	9	7	5	5	4	4	3
Total File Transmissions	100	50	26	18	14	10	10	8	8	6

This incurs 25 task groups and 50 file transmissions. In Experiment X, 18 tasks are grouped into a batch. Thus, only three task groups are created and sent to the first three grid resources.

Fig. 3 shows the performance charts of Experiment Phase I. The total time for executing the 50 tasks sequentially on a local machine is 9.21 minutes. Chart (a) reveals the overall application turnaround time for executing the tasks on the five grid resources.

The individual task deployment (Experiment I) consumes 5.51 minutes. The task group or batch deployment with granularity 2 (Experiment II) consumes 4.23 minutes, revealing an improvement of 23.23% as compared to the individual task deployment. The minimum turnaround time is 3.84 minutes with a performance improvement of 30.31%. This is achieved in Experiment VI with granularity 10. Here, only five groups are created; each contains 10 tasks. The five resources simultaneously process the five groups with less communication overhead.

However, the application turnaround time increases after Experiment VI. This is due to the imbalanced task allocation to the resources. For example, in Experiment VIII, the 50 tasks are divided into four groups and deployed on four resources. Each resource handles higher workloads sequentially (thus reducing the degree of parallelism) as opposed to Experiment VI in which all five resources are engaged with balanced workloads.

Chart (b) shows the total transmission time involved in all the experiments. R_1-R_4 are located within the same domain as the client machine with a better transmission speed as compared to R_0. Thus, frequent file transmissions with R_0 will increase the overall application transmission time.

Experiment I involves 50x2 transmissions, costing a total of 3.67 minutes; 11 task files are transmitted to R_0 (as indicated in Table 3). Experiment VI with 5x2 transmissions conveys a better performance of 90.21% by consuming only 0.37 minutes in total. The smallest transmission time (0.3 minute) is obtained when the granularity is 18. Here, only three groups are created and transmitted to R_0, R_1, and R_2.

Chart (c) further explores the average transmission time in terms of task and output files. The average task and output file transmission times for granularity 1 are 1.91 seconds and 2.49 seconds respectively. Having 50x2 transmissions, involves approximately 1.91x50 seconds and 2.49x50 seconds merely for the communication purpose. Granularity 18 with 3x2 transmissions involves only 2.86x3 seconds and 3.16x3 seconds for the task and output files.

Chart (d) reveals the average file size and the average utilisation of the network connection in each experiment. In Experiment I, the average task file size is 3.75 KBytes and the average task transmission rate is 6.07 KBytes/sec; the average output file size is 2.58 KBytes, which is transferred with 1.95 KBytes/sec. In Experiment VI, the average size of the five groups is 33.24 KBytes and the resulting transmission rate is 53.82 KBytes/sec. This shows that, task grouping leads to a better utilisation of the achievable network bandwidth. The highest network utlisation is achieved with granularity 18; 79.20 KBytes/sec and 34.81 KBytes/sec for task and output files respectively. Here, one should note that the size of the group is less than 16 KBytes. When

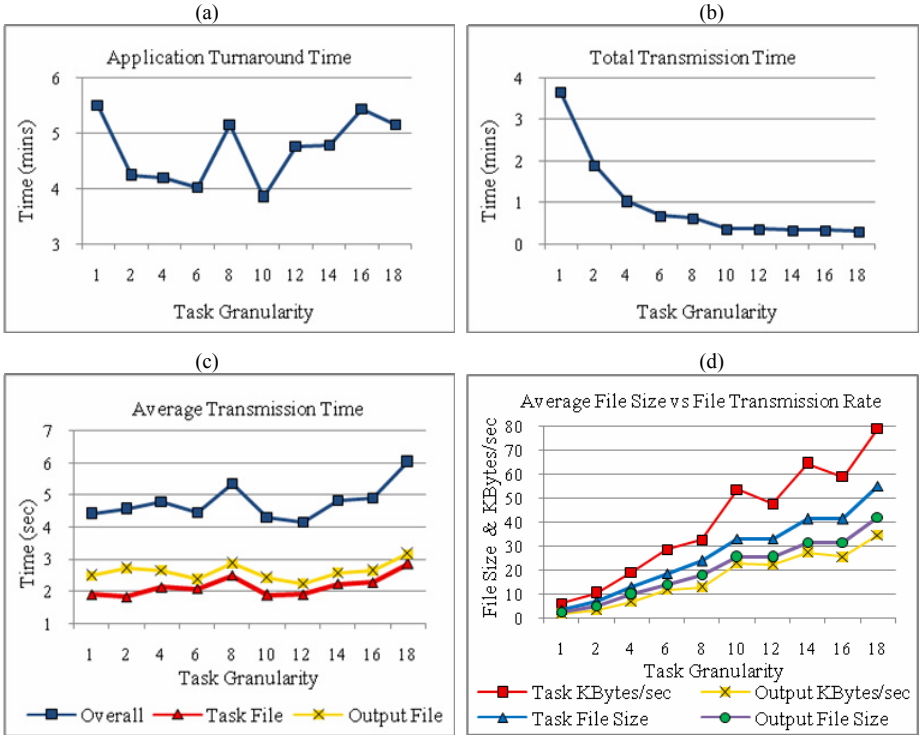

Table 3. Total task groups processed by each resource

Experiment	I	II	III	IV	V	VI	VII	VIII	IX	X
R_0	11	5	3	2	2	1	1	1	1	1
R_1	11	5	3	2	2	1	1	1	1	1
R_2	13	5	3	2	1	1	1	1	1	1
R_3	8	5	2	2	1	1	1	1	1	0
R_4	7	5	2	1	1	1	1	0	0	0
Total	50	25	13	9	7	5	5	4	4	3

Fig. 3. Performance charts – Experiment Phase I

grouping larger files, one may not get a similar impact since large files will overload the network, resulting in an unfavourable transmission time [17].

During Experiment V, we noticed a slight increase in the turnaround time due to the additional workloads assigned to R_0 by other users. In addition, more time is spent on file transmissions due to the fluctuating network conditions that increase the communication overhead. Having shared resources and communication overhead are inevitable in a grid environment.

4.2 Factors Influencing the Task Granularity

Experiment Phase I reveals the importance of task grouping before deploying the fine-grain tasks on a grid. When adding a task into a batch, the processing need of the batch will increase in terms of CPU time, wall-clock time, and the required storage space. This demands us to control the resulting granularity or the number of tasks in a batch. We discover the following four

main factors that affect the task granularity for a particular resource:

- The processing requirements of the tasks in a grid application.
- The processing speed and overhead of the grid resources.
- The utilisation constraints imposed by the providers to control the resource usage [18].
- The bandwidths of the interconnecting networks [19].

Fig. 4 depicts the information flow pertaining to the above-mentioned factors in a grid environment. The grid model contains three entities: User; Meta-Scheduler; and Grid Resources. The meta-scheduler gets the tasks from the user, groups the tasks into batches, and deploys the batches on the resources. The task granularity is decided based on the following factors:

- The processing requirements of each task in an application that include the task file size (TFSize), estimated task CPU time (ETCPUTime), and estimated output file size (OFSize).
- The utilisation constraints imposed by the resource providers to control the resource usage [18] that include the maximum CPU time (MaxCPUTime) allowed for executing a task, the maximum wall-clock time (MaxWCTime) a task can spend at the resource, and the maximum storage space (MaxSpace) that a task or a set of tasks (including the relevant output files) can occupy at a time. The MaxWCTime encompasses the CPU time and the processing overhead (waiting time and task packing/unpacking overhead) at the resource.
- The network utilisation constraint or the maximum time that a scheduler can wait for the task and output files to be transmitted to and from the resources (MaxTransTime).

Having these three input sets, the meta-scheduler can perform task grouping for a resource, R_j, based on the following five policies:

- **Policy 1: TG CPU time ≤ MaxCPUTime_{Ri}**
  The total CPU time of the task group should be less than the maximum allowed CPU time of a resource R_i.
- **Policy 2: TG wall-clock time ≤ MaxWCTime_{Ri}**

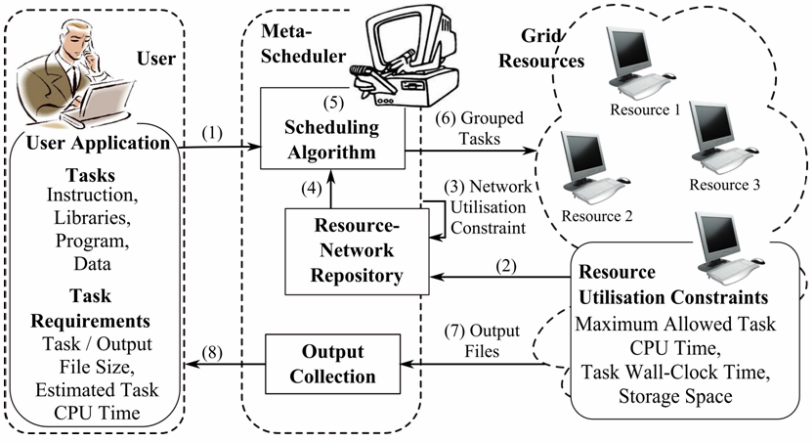

Fig. 4. The meta-scheduler and the information flow

The total wall-clock time of the task group should be less than the maximum allowed wall-clock time of a resource R_i.

- **Policy 3: TG and output transmission time ≤ $\mathbf{MaxTransTime_{Ri}}$**
  The transmission time of the task group and the relevant output files should be less than the maximum allowed file transmission time for a resource R_i.
- **Policy 4: TG and output file size ≤ $\mathbf{MaxSpace_{Ri}}$**
  The total size of the task group and the relevant output files should be less than the maximum allowed storage space of a resource R_i.
- **Policy 5: Number of tasks in TG ≤ Remaining $\mathbf{BoT_{TOTAL}}$**
  The number of tasks in a task group should be less than the remaining tasks in the BoT; BoT_{TOTAL} refers to the total tasks in the BoT.

The Policies 1-4 are related to resource-network utilisation constraints and Policy 5 is on task availability. There are some issues in using these policies due to the dynamic nature of a grid.

Issue I: Resource overhead and task wall-clock time. A grid resource can be a cluster of multiple nodes or a node with single or multiple cores. The wall-clock time of a task is influenced by the current processing load of the resource and the speed of the resource's local job scheduler. Policy 2 requires us to know the resource queuing system overhead in advance.

Issue II: Resource speed and task CPU time. Task CPU time differs according to a resource's processing capability; e.g a group of five tasks might be handled by Resource A smoothly, whereas it may exceed the maximum allowed CPU time or wall-clock time of Resource B, in spite of having a similar architecture as Resource A. In addition, a task with 20,000 instructions can be a fine-grain task for a machine that processes 10,000 instructions per second. However, a machine that processes 100 instructions per second will consider the same task as an average- or coarse-grain task. Policy 1 requires us to learn the resource speed and the processing need of the tasks prior to the task grouping process.

Issue III: Network condition and task transmission time. Task grouping increases the file size to be transmitted to and from the resources, and thus may overload the network. Moreover, the achievable bandwidth and latency of the interconnected network [19,20] vary at times; e.g. the network bandwidth at time t_x may support the transmission of a batch of seven tasks, however, at time t_y this may result in a heavily-loaded network (where $x < y$). Policy 3 requires us to determine the most appropriate file size depending on the current network status.

5. BATCH RESIZING TECHNIQUES: TASK CATEGORISATION AND BENCHMARKING

The three issues pertaining to conducting batch resizing policies (Section 4.2) can be viewed from two perspectives:

- **From a Task Perspective:** The issues are related to estimating task wall-clock time, CPU time, and task file transmission time.

- **From a Resource Perspective:** The issues are related to estimating the resource overhead, processing speed, and network condition.

In this paper, we focus on deploying computational tasks from BoT applications. A BoT consists of independent tasks that can be executed simultaneously. The order of the task executions can be random; e.g. T_1 can be executed before initiating T_0.

The resources operate on the tasks at their own pace. Each heterogeneous resource is associated with its intrinsic processing speed, overhead, and storage space. Moreover, the resources have their own workload or processing load. The individualistic and autonomous nature of the BoT tasks and the resources let us deal with the above-mentioned issues using two techniques: *Task Categorisation*; and *Task Category-Resource Benchmarking*.

In task categorisation, the BoT tasks are organised into categories according to their processing requirements. Following that, in the task category-resource benchmarking, sample tasks from the categories are scheduled and deployed on the grid resources in order to learn the behaviour of the resources and the interconnecting network on the task categories.

5.1 Task Categorisation

The tasks in a BoT may vary in terms of TFSize, ETCPUTime, and OFSize. When adding a task into a group, the TFSize, ETCPUTime, and OFSize of the task group are accumulated. Hence, the first concern is to ensure that the scheduler selects the most appropriate tasks from the BoT so that the resulting task group satisfies all of the five batch resizing policies.

A BoT may contain thousands of tasks. The second concern is that the task selection should be done in a timely manner, ensuring that the relevant overhead will not affect the application processing time. Thus, there is a need for proper task file management and searching strategies.

Here, we address the two concerns by arranging the tasks in a tree structure based on certain class interval thresholds applied to TFSize, ETCPUTime, and OFSize. This technique involves three levels of categorisation. In the first level, the tasks are divided into categories according to the task file size class interval ($TFSize_{CI}$). In the second level, the resulting categories are further divided according to the estimated task CPU time class interval ($ETCPUTime_{CI}$). Finally, in the third level, the tasks are divided based on the output file size class interval ($OFSize_{CI}$).

Algorithm 1 in Fig. 5 depicts the level 1 categorisation in which the tasks are divided into categories (TCat) based on TFSize of each task and the $TFSize_{CI}$. The range of a category is set according to $TFSize_{CI}$. For example, the range of:

$TCat_0$: 0 to (1.5 x $TFSize_{CI}$)
$TCat_1$: (1.5 x $TFSize_{CI}$) to (2.5 x $TFSize_{CI}$)
$TCat_2$: (2.5 x $TFSize_{CI}$) to (3.5 x $TFSize_{CI}$)

The category ID (TCatID) of a task is 0 if its TFSize is less than the $TFSize_{CI}$ (line 2,3). Otherwise, the mod and base values (line 5,6) of the TFSize are computed to determine the suitable category range. For example, when $TFSize_{CI}$ = 10 size unit,

tasks with (0 < TFSize <15) belong to $TCat_0$
tasks with (15 ≤ TFSize <25) belong to $TCat_1$
tasks with (25 ≤ TFSize <35) belong to $TCat_2$

Algorithm 1: Level 1 Task Categorisation.
Data: Requires TFSize of each T and TFSizeCI

```
1    for i := 0 to BOT_TOTAL-1 do
2        if T_i-TFSize < TFSize_CI then
3            TCatID := 0;
4        else
5            ModValue := T_i-TFSize mod TFSize_CI;
6            BaseValue := T_i-TFSize - ModValue;
7            if ModValue < TFSize_CI / 2 then
8                TCatID := (BaseValue / TFSize_CI) - 1;
9            else
10               TCatID := ((BaseValue + TFSize_CI) / TFSize_CI) - 1;
11           endif
12       endif
13   T_i belongs to TCat of ID TCatID
```

Note: $T_{i\text{-}TFSize}$ refers to the file size of the i^{th} task.

Fig. 5. Listing of level 1 task categorisation algorithm

This is followed by the level 2 categorisation in which the categories from level 1 are further divided into sub-categories according to the ETCPUTime of each task and $ETCPUTime_{CI}$. A similar categorisation algorithm is applied for this purpose. Subsequently, the level 3 categorisation divides the categories from level 2 into sub-categories based on OFSize and $OFSize_{CI}$. Fig. 6 presents an instance of task categorisation when $TFSize_{CI}$ = 10, $ETCPUTime_{CI}$ = 6, and $OFSize_{CI}$ = 10.

The categories at each level are created only when there is at least one task belonging to that particular category. For each resulting TCat, the average processing requirements are computed, namely, the average task file size (AvgTFSize), the average estimated task CPU time (AvgETCPUTime), and the average output file size (AvgOFSize). These average details will be used in a later technique. With this file organisation, one can easily locate the category that obeys the five policies and then select a task from the particular category to be added into a batch.

The order of the categorisation process can be altered; e.g. in level 1 categorisation, the tasks can be divided according to $ETCPUTime_{CI}$ instead of $TFSize_{CI}$. The resulting categories are not affected by the categorisation order, but merely depend on the class interval used at each level. Small class intervals can be used to increase the number of categories in order to achieve better accuracy when selecting a task for a batch. However, this will increase the overhead in searching

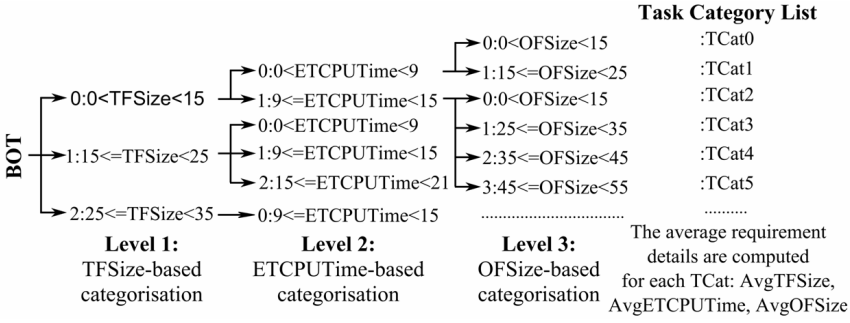

Fig. 6. Task categorisation – an example

for the most appropriate task category.

Our next step is to determine how the task categories react along with the resources and the interconnecting network. We achieve this using the second technique, task category-resource benchmarking.

5.2 Task Category-Resource Benchmarking

As mentioned in Section 4.2, the performance and overhead of the resources or the network cannot be estimated merely based on some written specifications. Hence, we suggest a benchmark phase in which a few tasks are selected from the BoT and deployed on the resources as to study the behaviour of the grid in response to the user tasks before scheduling the entire BoT.

First, we determine the dominating categories based on the total number of tasks in the categories. Then, we select p tasks from the first m dominating categories and send them to each resource. The total number of benchmark tasks, $BTasks_{TOTAL}$:

$$BTasks_{TOTAL} = m \times p \times total\_resources \tag{2}$$

Upon retrieving the processed output files of a benchmark task, the following eight 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); and processing overhead (POverhead).

The resource overhead (ROverhead) of a task refers to the waiting time and other overheads (task packing and unpacking time) during the task execution at the resource:

$$ROverhead = WCTime - CPUTime \tag{3}$$

The actual processing time of a task (APTime):

$$APTime = MTRTime + WCTime + RTMTime \tag{4}$$

However, there are overheads at the meta-scheduler that will be a part of the task turnaround time. Hence, the eighth deployment metric, task processing overhead (POverhead):

$$POverhead = TRTime - APTime \tag{5}$$

Finally, after completing all of the benchmark tasks, the average of each deployment metric is computed for each task category-resource pair. For a category k, the average deployment metrics on a resource j are expressed as average deployment metrics of $TCat_k$-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 ($AvgROverhead_{k,j}$); and average processing overhead ($AvgPOverhead_{k,j}$).

It can be noted that not all the categories are participating in this benchmark. Therefore, the average deployment metrics of those categories that missed the benchmark will be updated based on the average ratio of the categories that participated in the benchmark, in the following order:

$$AvgMTRTime_{i,j} = (AvgTFSize_i \times \sum_{k=0}^{m-1}(AvgMTRTime_{k,j} / AvgTFSize_k)) / m \quad (6)$$

$$AvgCPUTime_{i,j} = (AvgETCPUTime_i \times \sum_{k=0}^{m-1}(AvgCPUTime_{k,j} / AvgETCPUTime_k)) / m \quad (7)$$

$$AvgRTMTime_{i,j} = (AvgOFSize_i \times \sum_{k=0}^{m-1}(AvgRTMTime_{k,j} / AvgOFSize_k)) / m \quad (8)$$

$$AvgROverhead_{i,j} = (\sum_{k=0}^{m-1}(AvgROverhead_{k,j})) / m \quad (9)$$

$$AvgPOverhead_{i,j} = (\sum_{k=0}^{m-1}(AvgPOverhead_{k,j})) / m \quad (10)$$

$$AvgWCTime_{i,j} = AvgCPUTime_{i,j} + AvgROverhead_{i,j} \quad (11)$$

$$AvgAPTime_{i,j} = AvgMTRTime_{i,j} + AvgWCTime_{i,j} + AvgRTMTime_{i,j} \quad (12)$$

$$AvgTRTime_{i,j} = AvgMTRTime_{i,j} + AvgWCTime_{i,j} + AvgRTMTime_{i,j} + AvgPOverhead_{i,j} \quad (13)$$

where,

k denotes the TCatID in the benchmark and k takes specific values in the range $\{0,1,2,...,TCat_{TOTAL}\text{-}1\}$.

i denotes the TCatID missed the benchmark and i takes specific values in the range $\{0,1,2,...,TCat_{TOTAL}\text{-}1\}$.

j = 0,1,2,...,total grid resources-1.

m = Total categories in the benchmark.

The average space consumed by each $TCat_k$-R_j pair can be obtained from the AvgTFSize and AvgOFSize computed during the task categorization process.

In short, this benchmark phase studies the response of the grid resources and the interconnecting network on each task category. We select p tasks from the first m dominating categories for each resource. Increasing p and m will lead to a better accuracy in learning the behaviour of the grid environment. However, this will increase the $BTasks_{TOTAL}$ (the degree of individual task deployment) and reduce the total remaining tasks available for batch deployment.

6. THE GRIDBATCH META-SCHEDULER

This section presents the GridBatch meta-scheduler that performs the task grouping based on the proposed five batch resizing policies (Section 4) and two techniques (Section 5).

Fig. 7 shows the process flow of the entire GridBatch meta-scheduler system. There are eight modules involved in the meta-scheduler: *Controller, Task Categorisation, Scheduling, Task / Batch Deployment, Output Collection, Progress Monitoring, Resource Planning,* and *Deployment Analysis.* The *Scheduling* module encompasses three sub-modules, namely, *Benchmark*

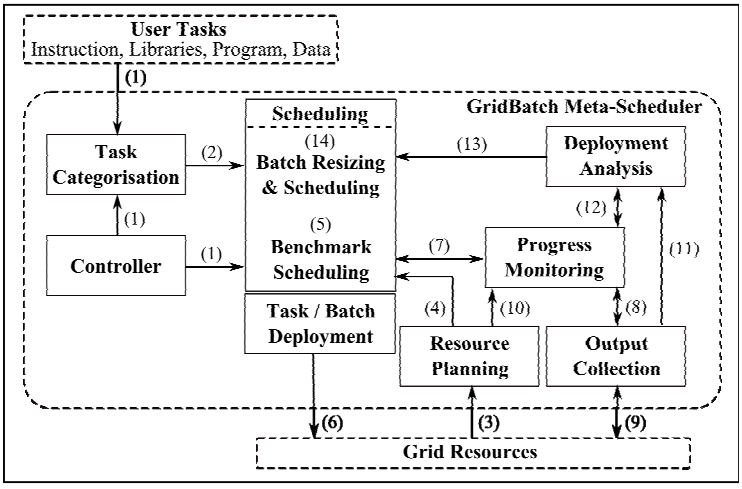

Fig. 7. Process flow of the GridBatch meta-scheduler

Scheduling, Batch Resizing & Scheduling, and *Task / Batch Deployment*.

(1) When the task files and the processing requirements are passed to the GridBatch, the *Controller* directs the value(s) of

- class intervals ($TFSize_{CI}$, $ETCPUTime_{CI}$, $OFSize_{CI}$) to the *Task Categorisation*, which will be used during the task categorisation process;
- *p* and *m* to the *Scheduling* module, which will be used during the benchmark phase; and
- the network utilisation constraint, MaxTransTime, to the *Scheduling* module, which will be used as the maximum time for transmitting the task and output files to and from a resource.

The process flow of the meta-scheduler includes two phases: Task Categorisation until the Task Category-Resource Benchmarking; and Task Deployment Analysis until the Batch Task Deployment.

6.1 Phase I: Task Categorisation till Task Category-Resource Benchmarking

(1) The *Task Categorisation* retrieves the task files from the user and categorises the tasks as explained in Section 5.1, according to the class intervals provided by the *Controller*. It then, prepares the task category list and (2) directs the list to the *Scheduling* module. Meanwhile, (3) the *Resource Planning* keeps the information of the participating grid resources to which the user has valid authorisations. (4) It presents a resource list that contains the IP addresses or host-names of the participating resources to the *Scheduling* module. It also passes the utilisation constraints (MaxCPUTime, MaxWCTime, MaxSpace) of the resources to the *Batch Resizing & Scheduling* module.

(5) Having the task and resource lists, the *Benchmark Scheduling* identifies the *m* dominating categories and selects *p* tasks from each category to be deployed on each resource. It schedules one task to one resource and (6) dispatches the task to the assigned resource via the *Task / Batch Deployment* module. In total, it has to schedule and deploy $BTasks_{TOTAL}$ individual tasks.

(7) Meanwhile, the *Progress Monitoring* will be informed about the dispatched task. (8) It then instructs the *Output Collection* to trace the progress of the dispatched task at the particular resource. (9) Upon detecting the completion of the task, the *Output Collection* retrieves the processed task files and (8) notifies the *Progress Monitoring*. (7) The *Progress Monitoring* then informs the *Scheduling* module that the particular resource is available for the next task; a task will be assigned to a resource once the resource completes the current task and becomes available for the next task. This mechanism also helps the meta-scheduler to identify the missing tasks after the task dispatching activity.

In addition, (10) the *Progress Monitoring* periodically gets a list of available grid resources from *Resource Planning* in order to keep track of any resource failure event that could cause the loss of the deployed task. (7) The module then notifies the *Scheduling* to re-invoke the deployment of the failed tasks. The steps (3-10) continue until all the $BTasks_{TOTAL}$ benchmark tasks are successfully deployed and processed.

In step (5), after the scheduling process, the module compresses the multiple files of a task into one file. For example, a task can be composed of an instruction file, a program or executable file, and a data file. The program and data files will be compressed into one file. Hence, the module will dispatch the instruction and the compressed files to the resource. Then, it will invoke the instruction file in the resource using a remote shell (rsh) that will do the necessary operations on the compressed files.

6.2 Phase II: Task Deployment Analysis till Batch Task Deployment

When collecting the output files of a task, (11) the *Output Collection* passes the task's details to *Deployment Analysis*. (12) The *Deployment Analysis* will get the progress details of the particular processed tasks from the *Progress Monitoring*. It then computes the eight deployment metrics of the task (MTRTime, CPUTime, WCTime, RTMTime, TRTime, APTime, ROverhead, POverhead). Upon computing the deployment metrics of all the benchmark tasks, the module will calculate the average value of each deployment metric for each task category-resource pair ($TCat_k$-R_j), as mentioned in Section 5.2.

Here, the *Resource Planning* does an additional job where (3) it retrieves the resource utilisation constraints from the resources in a periodic manner and (4) passes the constraint details to the *Batch Resizing & Scheduling*. (14) Having the resource-network utilisation constraints and the $TCat_k$-R_j average deployment metrics, the *Batch Resizing & Scheduling* builds a batch for a particular resource based on the five batch resizing policies formulated in Section 4.2.

The task categorisation process derives the need for enhancing Policy 5 to control the total number of tasks that can be selected from a category. Policy 5 can be expressed as follows:

Policy 5: Total tasks in TG from a $TCat_k \leq$ size_o f ($TCat_k$)
where, k = 0,1,2,...,$TCat_{TOTAl}$-1 (denoting the TCatID).

(14) First, the *Batch Resizing & Scheduling* loops through the task categories and determines the TCat that satisfies all the five policies. It selects a task from the particular TCat and adds it to the batch. The processing needs of the batch are updated based on the average deployment metrics of the TCat. The process continues. Whenever adding a task into a batch, the processing needs of the batch accumulate. Once the accumulated processing needs of the batch fit the re-

source-network utilisation constraints, the batch will be dispatched to the designated resource via the *Task / Batch Deployment*. The steps (3, 4, 14, 6-10) are repeated until all the remaining BoT_{TOTAL} tasks are successfully deployed and processed.

7. PERFORMANCE ANALYSIS – EXPERIMENT PHASE II

In this section, experiments are conducted in the environmental set-up shown in Fig. 2 in order to observe the impacts of the task categorization and task category-resource benchmarking techniques. We will use all the 568 tasks from the BoT for this analysis.

Table 4 shows the 10 experiments in Phase II and the observations on the resulting total task categories, benchmark tasks, and task groups. Experiment I reflects the individual task deployment. Experiments II-X involve task categorisation based on the class intervals ($TFSize_{CI}$, $ETCPUTime_{CI}$, and $OFSize_{CI}$) given in Table 4; two tasks from the first 20% of the dominating categories are selected to be deployed on each resource for the purpose of benchmarking.

First, we analyse an experiment (Experiment IV) in order to understand the process flow of the meta-scheduler. In Experiment IV, 17 categories are generated with $TFSize_{CI}$ = 1 KBytes, $ETCPUTime_{CI}$ = 1 minute, and $OFSize_{CI}$ = 500 KBytes. 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 (e.g. 0-330 indicated 330 tasks in category 0)

Then, the first three (20% of 17) dominating categories, namely, $TCat_0$, $TCat_1$, and $TCat_4$, are selected to participate in the benchmark phase; two tasks from each category are deployed for each resource ($BTasks_{TOTAL}$ = 30). After the benchmark phase, the average deployment metrics are computed for each $TCat_k$-R_j, as mentioned in Section 5.2.

Subsequently, task grouping is conducted for each resource based on the resource-network utilisation constraints stated in Table 5. Throughout the experiment, 199 task groups are created, thus it involves a total of (30+199)x2 transmissions (for both the task and output files). Table 6 lists the granularity and the number of groups created for each resource; e.g. for R_2, there are 6 benchmark tasks; 29 groups, each with one task; 3 groups, each with 3 tasks; 1 group with 15 tasks; and 2 groups, each with 20 tasks. In total, R_2 has received 41 groups (including the individual benchmark tasks) and processed 99 tasks. Overall, the experiment involves 229x2 transmissions to and from the five resources in order to process the 568 tasks. It can be noted that, after benchmarking, 153 tasks do not get included in any group. The processing requirements of

Table 4. Phase II experiments, configurations, and the observations

Experiment	I	II	III	IV	V	VI	VII	VIII	IX	X
TFSizeCI (KBytes)	-	1	5	1	1	1	1	1	1	1
ETCPUTimeCI (mins)	-	1	1	1	1	1	1	2	3	0.5
OFSizeCI (KBytes)	-	100	100	500	1000	1500	2000	100	100	100
Total Categories	-	58	58	17	11	9	8	56	54	63
Benchmark Tasks	-	76	76	30	20	10	10	68	62	84
Task Groups	-	170	168	199	220	260	261	152	143	189

Table 5. Resource-network utilisation constraints

Utilisation Constraints	R_0	R_1	R_2	R_3	R_4
MaxCPUTime (mins)	5	4	4	5	4
MaxWCTime (mins)	10	8	10	15	10
MaxSpace (MBytes)	10	15	10	10	10
MaxTransTime (mins)	6	5	5	4	6

Table 6. Benchmark and task group deployment, Experiment IV

Resource	Granularity											(Benchmark+ Groups):Tasks
	1 (Benchmark)	1	2	3	4	8	12	15	20	22	23	
R_0	6	24	2	2	-	-	-	-	-	-	2	36:86
R_1	6	35	2	-	1	-	-	-	-	3	-	47:115
R_2	6	29	-	3	-	-	-	1	2	-	-	41:99
R_3	6	31	-	3	2	-	7	-	-	-	-	49:138
R_4	6	34	3	4	-	9	-	-	-	-	-	56:130
Total	30	153	7	12	3	9	7	1	2	3	2	229:568

each of these are sufficient enough to meet the resource-network utilisation constraints.

Fig. 8 shows the performance charts of Experiment Phase II. The observations in Chart (a) and Chart (b) reveal that the task grouping highly reduces the overall application turnaround time and the total transmission time. The minimum task turnaround time (115.38 minutes) is achieved in Experiment VIII where 56 categories are generated with $TFSize_{CI}$ = 1 KBytes, $ETCPUTime_{CI}$ = 2 minutes, and $OFSize_{CI}$ = 100 KBytes. This experiment involves 68 benchmark tasks, and the remaining 500 tasks are grouped accordingly into 152 groups, leading to a total of 220x2 transmissions. The minimum total transmission time (12.54 minutes) is achieved with 205x2 transmissions in Experiment IX where 54 categories are generated with $TFSize_{CI}$ = 1 KBytes, $ETCPUTime_{CI}$ = 3 minutes, and $OFSize_{CI}$=100 KBytes.

We realize that a better performance is achieved when we increase the number of categories. However, Experiments II and III with 58 categories deliver an average performance as compared to Experiments VIII and IX with 56 and 54 categories, respectively. This is due to the number of benchmark tasks, $BTasks_{TOTAL}$, prior to the task group deployment and the overhead at the meta-

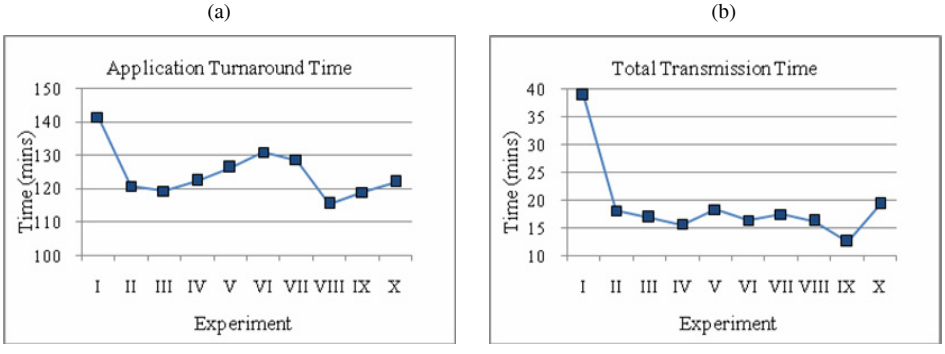

Fig. 8. Performance Charts – Experiment Phase II

scheduler.

For example, increasing the categories will increase the $BTasks_{TOTAL}$, which are to be deployed on the resources individually. In addition, the overhead at the meta-scheduler to loop through the list of categories for the task selection process will increase as well, leading to a higher application turnaround time. A similar effect can be seen in Experiment X with 63 categories and 84 benchmark tasks.

On the other hand, increasing the categories and the $BTasks_{TOTAL}$ will increase the accuracy of the meta-scheduler when computing the average deployment metrics of each $TCat_k$-R_j. This helps towards more accurate decisions on the task granularity of a batch. Thus, when deciding the class intervals ($TFSize_{CI}$, $ETCPUTime_{CI}$, and $OFSize_{CI}$), one should consider the priorities of the two factors: application turnaround time; and accuracy of task granularity.

8. Batch Resizing Technique: Periodic Average Analysis

The grid entities are autonomous in nature; they have their own workloads that are unknown to each other. The task categorisation and benchmarking help the meta-scheduler to learn the behaviour of a grid. However, as the grid operates autonomously in a dynamic environment, the deployment metrics of a task category may not reflect the latest grid after a time period [21]; the meta-scheduler will fail to keep up the accuracy level in deciding the appropriate batch size if it uses the benchmark results for deploying the entire BoT over time.

In this section, we propose to conduct a periodic average analysis to update the deployment metrics of each $TCat_k$-R_j pair according to the current grid status during the meta-scheduler runtime. This average analysis serves as a subsequent technique following the task categorisation and benchmarking techniques. There are three main concerns in performing this technique.

Concern I: The period size of average analysis. The meta-scheduler should analyse the processed task groups at regular intervals and update the deployment metrics of each $TCat_k$-R_j. Here, the concern is pertaining to the term 'regular intervals' or the 'period size'. If we increase the period size, the deployment metrics will not get updated along with the latest grid status frequently. With a small period size, the average analysis is performed frequently within short intervals. However, the meta-scheduler may not have completed any task group deployment before the subsequent average analysis. Moreover, this will increase the computation overhead at the meta-scheduler. Hence, in our GridBatch, the average analysis is conducted after each of the *AAIterations* iterations in the task group deployment. This ensures that there is at least one latest processed task group when conducting the subsequent average analysis.

Concern II: The various tasks in the task groups. Upon benchmarking, when creating a batch, multiple tasks from various categories are added into the group for deployment. The group is accepted by a resource as a single task. Upon execution, the deployment metrics can only be computed for a batch rather than for the individual tasks in the batch. Therefore, the average ratio computation method used in the benchmark phase (Section 5.2) cannot be employed to update the deployment metrics of $TCat_k$-R_j. The listing in Fig. 9 depicts the process flow used by the GridBatch to compute and update the average deployment metrics of each $TCat_k$-R_j from the latest processed task groups.

Concern III: The selection of task groups for average analysis. The average analysis is conducted at regular time intervals at runtime. Meanwhile, the number of processed task groups

During the average analysis, for a particular resource R_j,

1. Get a maximum of $AATasks_{TOTAL}$ latest processed task groups. Here, we select groups that are successfully processed by R_j within the last *AALatestTime* minutes.
2. Get the 'actual' deployment metrics of the selected task groups. These metrics can be retrieved from the progress report of the task groups.
3. The tasks in task groups come from various categories. Hence, identify the tasks and their categories in each group.
4. Based on the previous $TCat_k$-R_j average metrics, compute the 'estimated' task deployment metrics that each group is supposed to receive.
5. For each group, compute the ratios, 'estimated':'actual', of the eight deployment metrics.
6. Use these ratios to estimate and update the latest $TCat_k$-R_j average deployment metrics.
7. Not all the categories will participate in the $AATasks_{TOTAL}$ latest processed task groups. Update the $TCat_k$-R_j average details of those categories based on the ratios of the participated categories as explained in Section 5.2.

Fig. 9. Process flow of the average analysis

fetched by the meta-scheduler increases at runtime as well. Here, the concern is related to the number of groups that should be selected for an average analysis that can reflect the latest grid status. In our approach, whenever the average analysis is invoked, the meta-scheduler will select a maximum of $AATasks_{TOTAL}$ task groups processed by each resource within the last *AALastestTime* minutes. The batches that get processed earlier than the last *AALastestTime* minutes will not be considered for an average analysis.

9. Performance Analysis – Experiment Phase III

In this phase, we analyse the performance of the proposed periodic average analysis technique. For this purpose, we use the same configurations of Experiment IV (of Experiment Phase II); the tasks are divided into 17 categories based on $TFSize_{CI}$ = 1 KBytes, $ETCPUTime_{CI}$ = 1 minute, and $OFSize_{CI}$ = 500 KBytes; and $BTasks_{TOTAL}$ = 30.

There are four experiments conducted in this phase and the observations are given in Table 7. For example, in Experiment I, for each resource, the average analysis is conducted after every two iterations of task group deployment (*AAItearions* = 2).

During the analysis, all the tasks groups ($AATask_{TOTAL}$ = All) from the last 5 minutes (*AALatestTime* = 5 minutes) are selected for the average computations to update the average deployment metrics of $TCat_k$-R_j. As a result, 102 task groups are created throughout Experiment I to deploy the remaining 538 tasks. Table 8 shows the granularity and the number of groups created for all the resources. Note that, after the benchmarking, there are only 22 tasks that did not get included in any group. Experiment I involves only (30+102)x2 transmissions between the meta-scheduler and the resources as compared to (30+199)x2 transmissions in Experiment IV of Phase II (without periodic average analysis).

Fig. 10 shows the application turnaround time and the total transmission time involved in Experiment Phase III in comparison with Experiment Phase II. The observations prove that Experiments with periodic average analysis deliver better performance; e.g. Experiment I shows a

Table 7. Phase III experiments, configurations, and the observations

Experiment	I	II	III	IV
AALatestTime	5	10	10	20
AAIterations	2	2	4	6
$AATasks_{TOTAL}$	All	All	All	All
$BTasks_{TOTAL}$	30	30	30	30
Task Groups	102	108	103	109

Table 8. Benchmark and task group deployment, Experiment I

Resource	Granularity																(B+ Groups) :Tasks
	1 (B)	1	2	3	4	5	6	7	8	9	11	14	13	20	22	23	
R_0	6	15	-	-	1	-	-	-	-	-	-	-	-	-	-	2	24:71
R_1	6	3	1	8	3	-	1	-	-	-	-	-	-	-	3	-	25:119
R_2	6	-	4	5	1	-	1	1	-	1	-	-	-	3	-	-	22:115
R_3	6	3	4	4	4	3	-	-	1	-	-	-	3	-	-	-	28:107
R_4	6	1	7	3	3	1	3	2	3	-	1	3	-	-	-	-	33:156
Total	30	22	16	20	12	4	5	3	4	1	1	3	3	3	3	2	132:568
B: Benchmark																	

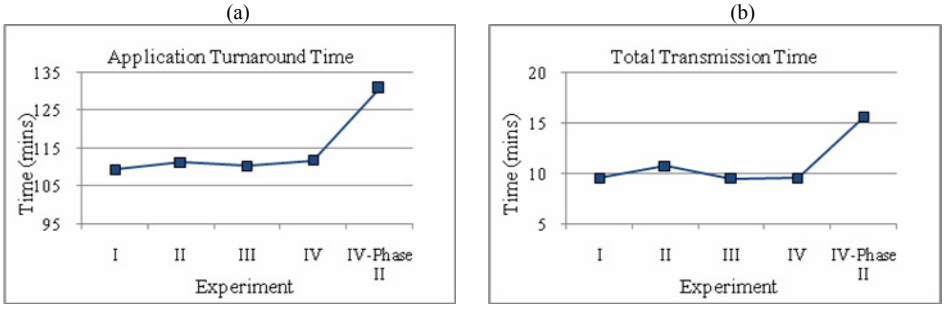

Fig. 10. Performance charts – Experiment Phase III

performance improvement of 16.45% as compared to Experiment Phase II.

We also realize that, for our environmental set-up (Fig. 2), the values of *AALatestTime*, *AAIterations*, and $AATask_{TOTAL}$ do not deliver major variations in terms of turnaround and transmission times. This is due to the Intranet access to four of the resources that are located within the same domain as the meta-scheduler; the network condition was stable throughout the experiments.

Despite this condition, it can be seen that Experiment I with a frequent, average analysis conveys the best performance. Performing an average analysis after every two task group iterations reflects the latest state of the grid. In this case, we select the task groups processed within the last 5 minutes for the analysis purpose. When we increase the *AALatestTime* to 10 minutes, there is a slight decrease in the overall performance. Computing the average deployment metrics from the groups processed within the last 10 minutes misses the accuracy towards reflecting the latest grid status on the average deployment metrics. Similar impacts can be seen when we increase the period size or the *AAIterations*.

10. CONCLUSION

In this paper, we realize the need for task grouping when processing fine-grain tasks on global grids. This leads us towards five policies to decide the task granularity or the size of a batch before the task deployment. Thus, we propose task categorization and benchmarking techniques in order to learn the behaviour of a grid before deciding the size of a batch. Following that, we realize the need for a periodic average analysis to reflect the latest grid status on the task granularity. We develop the GridBatch meta-scheduler to test and analyse the performance of the proposed policies and techniques. We also analyse all the important parameters involved in the proposed techniques that affect the overall application turnaround and communication times.

In the future, we will conduct these experiments in a massive grid environment in order to further analyse the influence of the parameters introduced in our techniques. Meanwhile, the proposed process flow of the entire meta-scheduler will be adapted to handle work-flow grid applications.

REFERENCE

[1] F. Berman, G. C. Fox, A. J. G. Hey, *"Grid Computing - Making the Global Infrastructure a Reality"*, Wiley and Sons, Mar., 2003.

[2] R. Buyya, S. Date, Y. M. Natsumoti, S. Venugopal, "Neuroscience Instrumentation and Distributed Analysis of Brain Activity Data: A Case for e-Science on Global Grids", *Concurrency and Computation: Practice and Experience (CCPE)*, Vol.17, No.15, pp.1783-1798, 2005.

[3] M. Baker, R. Buyya, D. Laforenza, "Grids and Grid Technologies for Wide-Area Distributed Computing", *Software: Practice and Experience (SPE)*, Vol.32, No.15, pp.1437-1466, 2002.

[4] B. Jacob, M. Brown, K. Fukui, N. Trivedi, *"Introduction to Grid Computing"*, IBM Publication, Dec., 2005.

[5] S. Venugopal, R. Buyya, W. Lyle, "A Grid Service Broker for Scheduling e-Science Applications on Global Data Grids", *Concurrency and Computation: Practice and Experience (CCPE)*, Vol. 18, pp.685-699, 2006.

[6] H. James, K. Hawick, P. Coddington, *"Scheduling Independent Tasks on Meta-Computing Systems"*, *In Proceedings of Parallel and Distributed Computing Systems*, Fort Lauderdale, pp. 156-162, 1999.

[7] E. G. Coffman, Jr., M. Yannakakis, M. J. Magazine, C. Santos, "Batch Sizing and Job Sequencing on a Single Machine", *Annals of Operation Research*, Vol.26, No. 1-4, pp.135-147, 1990.

[8] T. Cheng, M. Kovalyov, "Single Machine Batch Scheduling with Sequential Job Processing", *IIE Transactions*, Vol.33, No.5, pp.413-420, 2001.

[9] G. Mosheiov, D. Oron, "A Single Machine Batch Scheduling Problem with Bounded Batch Size", *European Journal of Operational Research*, Vol.187, No.3, pp.1069-1079, 2008.

[10] K. E. Maghraoui, T. J. Desell, B. K. Szymanski, C. A. Varela, "The Internet Operating System: Middleware for Adaptive Distributed Computing", *International Journal of High Performance Computing Applications*, Vol.20, No.4, pp.467-480, 2006.

[11] A. C. Sodan, A. Kanavallil, B. Esbaugh, *"Group-based Optimisation for Parallel Job Scheduling with Scojo-PECT-O"*, *In Proceedings of the 22nd International Symposium on High Performance Computing Systems and Applications*, p.102-109, Washington, DC, USA, 2008. IEEE Computer Society.

[12] N. Muthuvelu, J. Liu, N. L. Soe, S. Venugopal, A. Sulistio, R. Buyya, *"A Dynamic Job Grouping-based Scheduling for Deploying Applications with Fine-Grained Tasks on Global Grids"*, *In Proceedings of the Australasian Workshop on Grid Computing and E-Research*, p. 41-48, Australian Computer Society, Inc., 2005.

[13] W. K. Ng, T. Ang, T. Ling, C. Liew, "Scheduling Framework for Bandwidth-Aware Job Grouping-based Scheduling in Grid Computing", *Malaysian Journal of Computer Science*, Vol.19, No.2,

pp.117-126, 2006.

[14] J. H. Stokes, "Behind the Benchmarks: Spec, Gflops, MIPS et al", http://arstechnica.com/cpu/2q99/benchmarking-2.html, 2000.

[15] N. Muthuvelu, I. Chai, E. Chikkannan, *"An Adaptive and Parameterized Job Grouping Algorithm for Scheduling Grid Jobs"*, In *Proceedings of the 10th International Conference on Advanced Communication Technology*, Vol. 2, pp.975-980, 2008.

[16] N. Muthuvelu, I. Chai, E. Chikkannan, R. Buyya, *"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.

[17] A. Ghobadi, C. Eswaran, N. Muthuvelu, I. K. T. Tan, Y. L. Kee, *"An Adaptive Wrapper Algorithm for File Transfer Applications to Support Optimal Large File Transfers"*, *In Proceedings of the 11th International Conference on Advanced Communication Technology*, p.315-320, Piscataway, NJ, USA, 2009. IEEE Press.

[18] J. Feng, G. Wasson, M. Humphrey, *"Resource Usage Policy Expression and Enforcement in Grid Computing"*, In *Proceedings of the 8th IEEE/ACM International Conference on Grid Computing*, p.66-73, Washington, DC, USA, 2007. IEEE Computer Society.

[19] R. G. O. Arnon, "Fallacies of Distributed Computing Explained", http://www.webperformancematters.com/, 2007.

[20] B. Lowekamp, B. Tierney, L. Cottrell, R. H. Jones, T. Kielmann, M. Swany, *"A Hierarchy of Network Performance Characteristics for Grid Applications and Services"*, *Global Grid Forum*, Jun., 2003.

[21] P. Huang, H. Peng, P. Lin, X. Li, "Static Strategy and Dynamic Adjustment: An Effective Method for Grid Task Scheduling", *Future Generation Computer Systems (FGCS)*, Vol.25, No.8, pp.884-892, 2009.

Nithiapidary Muthuvelu

She received her B.IT degree from Universiti Tenaga Nasional, Malaysia, in 2003 and an M.IT degree from the University of Melbourne, Australia, in 2004. She is teaching at Multimedia University, Malaysia, since 2005. Currently, she is pursuing her Ph.D study in the field of grid computing at Multimedia University. Her research interests include: Distributed and Parallel Processing and Data Communication. She is a member of the IEEE Computer Society.

Dr. Ian Chai

He 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. C. Eswaran

He 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, at Multimedia University, Malaysia. He 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.

Dr. Rajkumar Buyya

He 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-of 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. 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 in 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 for four IEEE conferences (CCGrid, Cluster, Grid, and e-Science). He has presented over 200 invited talks on his vision on IT Futures and advanced computing technologies at international conferences and institutions. These contributions and his international research leadership are recognised through the award of the "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.