On-line Task Granularity Adaptation for Dynamic Grid Applications [⋆]

Nithiapidary Muthuvelu[1], Ian Chai[1], Eswaran Chikkannan[1], and Rajkumar Buyya[2]

[1] Multimedia University, Persiaran Multimedia, 63100 Cyberjaya, Selangor, Malaysia,
{nithiapidary, ianchai, eswaran }@mmu.edu.my

[2] Dept. of Computer Science and Software Engineering, The University of Melbourne, 3053 Carlton, Victoria, Australia,
raj@csse.unimelb.edu.au

Abstract. Deploying lightweight tasks on grid resources would let the communication overhead dominate the overall application processing time. Our aim is to increase the resulting computation-communication ratio by adjusting the task granularity at the grid scheduler. We propose an on-line scheduling algorithm which performs task grouping to support an unlimited number of user tasks, arriving at the scheduler at runtime. The algorithm decides the task granularity based on the dynamic nature of a grid environment: task processing requirements; resource-network utilisation constraints; and users QoS requirements. Simulation results reveal that our algorithm reduces the overall application processing time and communication overhead significantly while satisfying the runtime constraints set by the users and the resources.

1 Introduction

A grid application contains a large number of tasks [1] and a scheduler at the user site transmits each task file to a grid resource for further execution and retrieves the processed output file(s) [2][3]. A lightweight or fine-grain task requires minimal execution time (i.e. 15 seconds to one minute). Executing a computation-intensive application with a large number of lightweight tasks on a grid would result in a low computation-communication-ratio due to the overhead involved in handling each task [4]; the term *computation* refers to the task execution time, whereas *communication* refers to the user authentication, and task and output file transmission time. There are two issues involved in this matter:

1. The communication overhead increases proportionally with the number of tasks.
2. A resource's processing capability and the network capacity may not be optimally utilised when dealing with fine-grain tasks. For example:
   (a) Assume that a high-speed machine allows a user to use the CPU power for x seconds. Executing lightweight tasks one at a time would not utilise the full processing speed (i.e. $x\times$Million Instructions per Second) of the machine within x seconds due to the overhead involved in invoking and executing each task.

⋆ This research is partially supported by e-ScienceFund, Ministry of Science, Technology and Innovation, Malaysia, and Endeavour Awards, Austraining International.

(b) Transmitting task/output files (of very minimal sizes) one by one between the user and the resources would underutilise the relevant achievable bandwidth.

In short, deploying lightweight tasks on grid would lead to inefficient resource-network utilisation, and unfavourable application throughput. This statement is proven with experiments in Sec. 5 of this paper. In our previous work [5], we showed that task grouping reduces the overall application processing time significantly. In our current work, we present an on-line scheduling algorithm for deciding the task granularity. The scheduler has no knowledge on the total number of tasks in the application as the tasks come on a real-time basis, arriving at the scheduler at runtime (e.g. processing data arriving from sensors).

Upon receiving the tasks, the scheduler selects and groups a number of tasks into a batch, and deploys the grouped task on a grid resource. The task granularity is determined as to maximise the resource-network utilisation and minimise the overall application processing time. Hence, the decision highly depends on the dynamic nature of a grid environment:

1. The processing requirements of each task in a grid application.
2. The utilisation constraints imposed by the resource providers to control the resource usage [6].
3. The varying bandwidths of the networks interconnecting the scheduler and the resources [7].
4. The quality of service (QoS) requirements for executing an application [8].

Our proposed scheduling algorithm focuses on computation-intensive, bag-of-tasks applications. It assumes that all the tasks in an application are independent and have similar compilation platform. The algorithm considers the latest information from the grid resources, decides the task granularity, and proceeds with task grouping and deployment. Our ultimate goal is to reduce the overall application processing 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. The factors and issues involved in task grouping in grid are described in Sec. 3. Section 4 explains the strategies and the process flow of the proposed scheduler system which is followed by the relevant performance analysis in Sec. 5. Finally, Sec. 6 concludes the paper by suggesting future work.

2 Related Work

Here, we focus on the work related to batch processing in distributed computing which involve task granularity adaptation. James et al [9] grouped and scheduled equal numbers of independent jobs using various scheduling algorithms to a cluster of nodes. This induced an overhead as the nodes were required to be synchronised after each job group execution iteration. Simulations were conducted to optimise the number of jobs in a batch for a parallel environment by Sodan et al [10]. The batch size is computed based on average runtime of the jobs, machine size, number of running jobs in the machine, and minimum/maximum node utilisation. However, these 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.

Maghraoui et al [11] adapted the task granularity to support process migration (upon resource unavailability) by merging and splitting the atomic computational units of the application jobs. The jobs are created using a specific API; special constructs are used to indicate the atomic jobs in a job file which are used as the splitting or merging points during job migration.

A number of simulations had been conducted to prove that task grouping reduces the overall grid application processing time. The authors in [5][12] 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. MIPS or MI are not the preferred benchmark matrices as the execution times for two programs of similar MI but with different program locality (e.g. program compilation platform) can differ [13]. Moreover, a resource's full processing capacity may not be available all the time because of the I/O interrupt signals.

In our previous work [14], task grouping was simulated according to 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). The simulations show that the grouping algorithm performs better than the conventional task scheduling algorithm by 20.05% in terms of overall application throughput when processing 500 tasks. However, it was assumed that the task file size is similar to the task length which is an oversimplification as the tasks may contain massive computation loops.

In this paper, we treat the file size of a task separately from its length or processing needs. We also consider two additional constraints: space availability at the resource; and output file transmission time. In addition, the algorithm is designed to support an unlimited number of user tasks arriving at the scheduler at runtime.

3 Factors Influencing the Task Granularity

Figure 1 shows the implementation focus of our scheduling algorithm in a grid environment. The factors influencing the task granularity are passed to the scheduler from the (1) user application, (2) grid resources, and the (3) scheduler. The user application is a bag-of-tasks (BOT) with QoS requirements. Each task is associated with three requirement properties: size of the task file (TFSize); estimated size of the output file (OFSize); and the estimated CPU time (ETCPUTime). The QoS includes the budget (UBudget) and deadline (UDeadline) allocated for executing all the user tasks.

Each resource (R) from a set of participating grid resources (GR) provides its utilisation constraints to the scheduler:

1. Maximum CPU time (MaxCPUTime) allowed for executing a task.
2. Maximum wall-clock time (MaxWCTime) a task can spend at the resource. This encompasses the CPU time and the processing overhead (waiting time and task packing/unpacking overhead) at the resource.

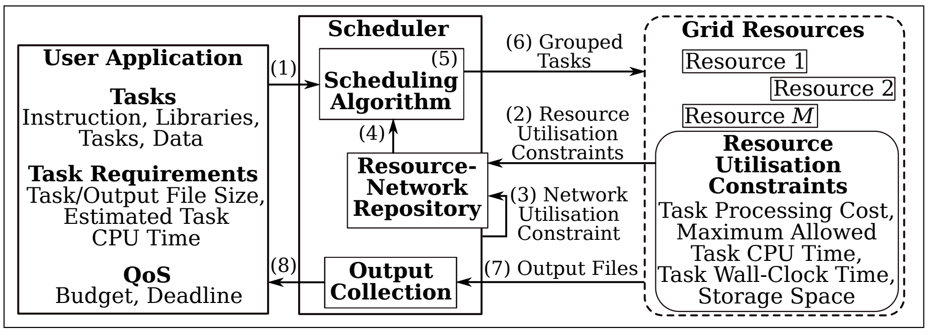

User Application
Tasks
Instruction, Libraries, Tasks, Data
Task Requirements
Task/Output File Size, Estimated Task CPU Time
QoS
Budget, Deadline
(1)
(8)
Scheduler
(5)
Scheduling Algorithm
(4)
Resource-Network Repository
Output Collection
(6) Grouped Tasks
(2) Resource Utilisation Constraints
(3) Network Utilisation Constraint
(7) Output Files
Grid Resources
Resource 1
Resource 2
Resource M
Resource Utilisation Constraints
Task Processing Cost, Maximum Allowed Task CPU Time, Task Wall-Clock Time, Storage Space

Fig. 1. Scheduler Components and their Information Flow

3. Maximum storage space (MaxSpace) that a task or a set of tasks (including the relevant output files) can occupy at a time.
4. Task processing cost (PCost) per unit time charged by a resource.

Finally, the network utilisation constraint is the maximum time that a scheduler can wait for the task/output files to be transmitted to/from the resources (MaxTransTime). It is the tolerance threshold that a scheduler can accept in terms of file transmission time.

Having these information, we derived the seven objective functions for determining the granularity of a task group, TG, for a resource, Ri, as follows:

Objective 1: TG CPU time $\leq MaxCPUTime_{Ri}$
Objective 2: TG wall-clock time $\leq MaxWCTime_{Ri}$
Objective 3: TG and output transmission time $\leq MaxTransTime_{Ri}$
Objective 4: TG and output file size $\leq MaxSpace_{Ri}$
Objective 5: TG turnaround time $\leq$ Remaining $UDeadline$
Objective 6: TG processing cost $\leq$ Remaining $UBudget$
Objective 7: Number of tasks in $TG \leq$ Remaining BOT_{TOTAL}
where, BOT_{TOTAL} = total number of tasks waiting at the scheduler.

However, there are three issues that affect the granularity according to these seven objective functions.

ISSUE I: A resource can be a single node or a cluster. The task wall-clock time is affected by the speed of the resource's local job scheduler and the current processing load. In order to obey the objectives 2 and 5, one should know the overheads of the resources' queuing systems in advance.

ISSUE II: The task CPU time differs according to the resources' processing capabilities. For example, a group of five tasks can 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; the processing speed of a resource cannot be estimated in advance based on the hardware specification only.

Moreover, the task CPU time highly depends on the programming model or compilation platform. Hence, we should learn the resource speed and the processing need of the tasks prior to the application deployment.

ISSUE III: Task grouping increases the resulting file size to be transmitted to a resource, leading to an overloaded network. Moreover, the achievable bandwidth and latency [7][15] of the interconnecting network are not static; e.g. the 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 $x < y$). Hence, we should determine the appropriate batch size that can be transferred at a particular time.

4 Scheduler Implementation

In our scheduling algorithm, the issues mentioned in Sec. 3 are tackled using three approaches in the following order: Task Categorisation; Task Category-Resource Benchmarking; and Average Analysis. The following subsections explain the three approaches respectively and present the process flow of the entire scheduler system.

4.1 Task Categorisation

The tasks in a BOT vary in terms of TFSize (e.g. a non-parametric sweep application), ETCPUTime, and OFSize. When adding a task into a group, the resulting total TFSize, ETCPUTime, and OFSize of the group get accumulated. Hence, the scheduler should select the most appropriate tasks from the BOT (without significant delay) and ensure that the resulting group satisfies all the seven objective functions.

We suggest a task categorisation approach to arrange the tasks in a tree structure based on certain class interval thresholds applied to the TFSize, ETCPUTime, and OFSize. The tasks are divided into categories according to TFSize class interval ($TFSize_{CI}$), followed by ETCPUTime class interval ($ETCPUTime_{CI}$), and then OFSize class interval ($OFSize_{CI}$).

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

$TCat_0$: 0 to $(TFSize_{CI} + TFSize_{CI}/2)$
$TCat_1$: $(TFSize_{CI} + TFSize_{CI}/2)$ to $(2 \times TFSize_{CI} + TFSize_{CI}/2)$
$TCat_2$: $(2 \times TFSize_{CI} + TFSize_{CI}/2)$ to $(3 \times TFSize_{CI} + TFSize_{CI}/2)$

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 units, then a task with,
$TFSize = 12$ belongs to $TCat_0$ as $TCat_0(0 < TFSize < 15)$
$TFSize = 15$ belongs to $TCat_1$ as $TCat_1(15 \leq TFSize < 25)$
$TFSize = 30$ belongs to $TCat_2$ as $TCat_2(25 \leq TFSize < 35)$

Algorithm 1: Level 1 Task Categorisation.

```
Data: Requires $TFSize$ of each $T$ and $TFSize_{CI}$
1  for $i \leftarrow 0$ to $BOT_{TOTAL}$ do
2      if $T_{i-TFSize} < TFSize_{CI}$ then
3          $TCatID \leftarrow 0$
4      else
5          $ModValue \leftarrow T_{i-TFSize} \bmod TFSize_{CI}$
6          $BaseValue \leftarrow T_{i-TFSize} - ModValue$
7          if $ModValue < TFSize_{CI}/2$ then
8              $TCatID \leftarrow (BaseValue/TFSize_{CI}) - 1$
9          else
10             $TCatID \leftarrow ((BaseValue + TFSize_{CI})/TFSize_{CI}) - 1$
11     $T_i$ belongs to $TCat$ of ID $TCatID$
```

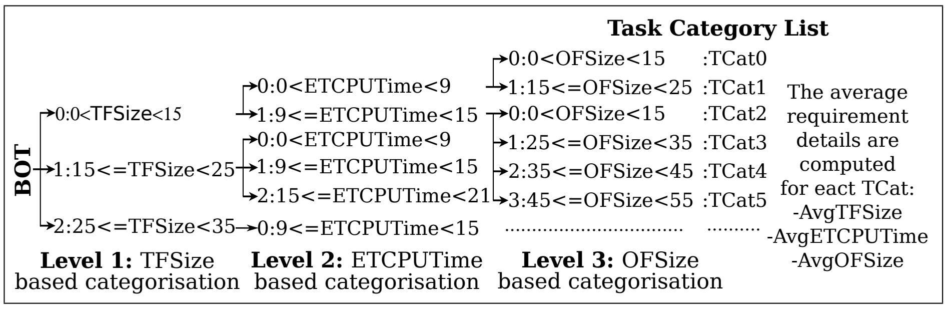


Fig. 2. Task Categorisation

This is followed by level 2 categorisation; TCat(s) from level 1 is further divided into sub-categories according to ETCPUTime and $ETCPUTime_{CI}$. The similar categorisation algorithm is applied with ETCPUTime of each task and $ETCPUTime_{CI}$. Subsequently, level 3 categorisation divides the TCat(s) from level 2 into sub-categories based on OFSize and $OFSize_{CI}$.

Figure 2 shows an instance of categorisation with $TFSize_{CI} = 10$, $ETCPUTime_{CI} = 6$, $OFSize_{CI} = 10$. The category(s) at each level is created when there is at least one task belonging to the particular category. For each resulting TCat, the average requirements are computed: average TFSize (AvgTFSize); average ETCPUTime (AvgETCPUTime); and average OFSize (AvgOFSize).

When a new set of tasks arrives at the scheduler, each task is checked for its requirements and assigned to the appropriate TCat; new categories with certain ranges are created as needed. Having this organisation, the scheduler can easily locate the task files (for a group) that obey the utilisation constraints and QoS requirements.

4.2 Task Category-Resource Benchmarking

In this benchmark phase, the scheduler selects a few tasks from the categories for further deployment on the resources before scheduling the entire user application. This helps the scheduler to study the capacity, performance, and overhead of the resources and the interconnecting network over the user tasks. It selects p tasks from the first m dominating categories (based on the total number of tasks in each category) and sends to each resource. The total number of benchmark tasks, $BTasks_{TOTAL}$, can be expressed as:

$$BTasks_{TOTAL} = m \times p \times GR_{TOTAL} \tag{1}$$

Upon retrieving the processed output files, the remaining UBudget and UDeadline are updated accordingly, and the following seven actual deployment matrices of each task are computed:

> task file transmission time (scheduler-to-resource); CPU time; wall-clock time; processing cost; output file transmission time (resource-to-scheduler); processing overhead; and turnaround time.

Finally, the average of each deployment matrix is computed for each task category-resource pair. For a category k, the average deployment matrices on a resource j are expressed as average deployment matrices of $TCat_k - R_j$, which consist of:

> average task file transmission time ($AvgSTRTime_{k,j}$); average CPU time ($AvgCPUTime_{k,j}$); average wall-clock time ($AvgWCTime_{k,j}$); average processing cost ($AvgPCost_{k,j}$); average output file transmission time ($AvgRTSTime_{k,j}$); average processing overhead ($AvgOverhead_{k,j}$); and average turnaround time ($AvgTRTime_{k,j}$).

The average deployment matrices of those categories which did not participate in the benchmark phase are then updated based on the average ratio of the other categories. Assume that m categories have participated in the benchmark phase, then the average matrices of a category can be formulated in the following order:

$$AvgSTRTime_{i,j} = (\textstyle\sum_{k=0}^{m-1}(AvgTFSize_i \times AvgSTRTime_{k,j}/AvgTFSize_k))/m$$
$$AvgCPUTime_{i,j} = (\textstyle\sum_{k=0}^{m-1}(AvgETCPUTime_i \times AvgCPUTime_{k,j}/AvgETCPUTime_k))/m$$
$$AvgRTSTime_{i,j} = (\textstyle\sum_{k=0}^{m-1}(AvgOFSize_i \times AvgRTSTime_{k,j}/AvgOFSize_k))/m$$
$$AvgPCost_{i,j} = (\textstyle\sum_{k=0}^{m-1}(AvgCPUTime_{i,j} \times AvgPCost_{k,j}/AvgCPUTime_{k,j}))/m$$
$$AvgOverhead_{i,j} = (\textstyle\sum_{k=0}^{m-1} AvgOverhead_{k,j})/m$$
$$AvgWCTime_{i,j} = AvgCPUTime_{i,j} + AvgOverhead_{i,j}$$
$$AvgTRTime_{i,j} = AvgWCTime_{i,j} + AvgSTRTime_{i,j} + AvgRTSTime_{i,j}$$

where,
k = 0,1,2,...,$TCat_{TOTAL} - 1$; TCat ID participated in benchmark.
j = 0,1,2,...,$GR_{TOTAL} - 1$; grid resource ID.
i = 0,1,2,...,$TCat_{TOTAL} - 1$; TCat ID did not participate in benchmark.
m = Total categories participated in benchmark.

In short, the benchmark phase studies the response and performance of the resources and the interconnecting network on each category.

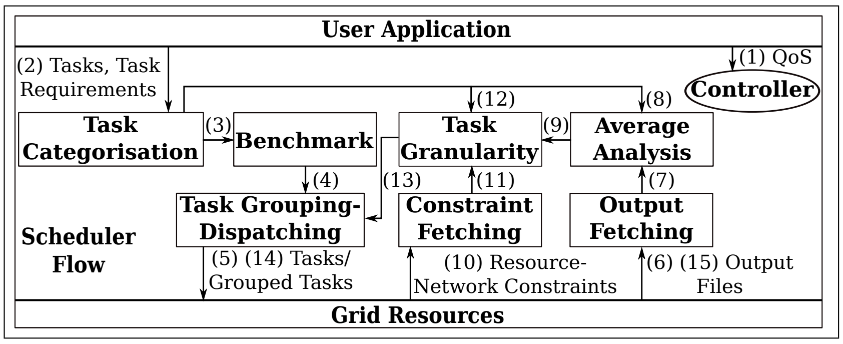


Fig. 3. Process Flow of the Scheduler System

4.3 Average Analysis

Knowing the behaviour of the resources and network, we can group the tasks according to the seven objective functions of task granularity. However, as grid resides in a dynamic environment, the deployment matrices of the categories may not reflect the latest grid status after a time period. Therefore, the scheduler should update the deployment matrices of each $TCat_k - R_j$ pair periodically based on the latest arrived processed task groups.

First, it gets the 'actual' deployment matrices of the latest arrived processed groups. Using the previous $TCat_k - R_j$ average matrices, it computes the deployment matrices that each task group 'supposed' to get. Then, the ratio 'supposed':'actual' of each deployment matrix is computed to estimate and update the $TCat_k - R_j$ average matrices. For those categories which did not participate in the latest task groups, their $TCat_k - R_j$ average matrices get updated based on the ratio of the other categories as explained in Sec. 4.2.

4.4 Scheduler Process Flow

Figure 3 presents the process flow of the entire scheduler system. (1) The *Controller* manages the scheduler activity in terms of the flow and periodic average analysis. It ensures that the QoS requirements are satisfied at runtime. (2) The *Task Categorisation* categorises the user tasks. (3) It then invokes the *Benchmark* which selects $BTasks_{TOTAL}$ from the categorised BOT for (4,5) further task deployment on the grid resources. (6) The *Output Fetching* collects the processed benchmark tasks and (7,8) the *Average Analysis* module studies the task category-resource or $TCat_k - R_j$ average deployment matrices. (10) The *Constraint Fetching* retrieves the resource-network utilisation constraints periodically to set the task granularity objective functions. (9,11,12) Having the categorised tasks, $TCat_k - R_j$ average deployment matrices, and the resource-network utilisation constraints, the *Task Granularity* determines the number of tasks from various categories that can be grouped for a particular resource.

When selecting a task from a category, the expected deployment matrices of the resulting group are accumulated from the average task deployment matrices of the particular category. The final granularity must satisfy all the seven objective functions mentioned in Sec. 3 of this paper. The task categorisation process derives the need for enhancing objective 7 to control the total number of tasks that can be selected from a category *k*:

Objective 7: Total tasks in TG from a $TCat_k \leq size\_of(TCat_k)$

(13,14) Upon setting the granularity, the *Task Grouping-Dispatching* selects and groups the tasks, and transfers the batch to the designated resource. (15) The processed task groups are then collected by the *Output Fetching*; the remaining UBudget and UDeadline are updated accordingly. The cycle (10-15) continues for a certain time period and then the *Controller* signals the *Average Analysis* to update the average deployment matrices of each $TCat_k - R_j$ to be used by the subsequent task group scheduling and deployment iterations.

5 Performance Analysis

The scheduling algorithm is simulated using the GridSim [16]. There are 400-2500 tasks involved in this performance analysis with TFSize (6-40 size units), ETCPUTime (70-130 time units), and OFSize (6-40 size units). The $TFSize_{CI}$, $ECPUTime_{CI}$, $OFSize_{CI}$ are of 10 size units each. The QoS constraints are UDeadline (200K-600K time units) and UBudget (6K-8K cost units).

The grid is configured with eight cluster-based resources, each with three processing elements. The processing capacity of a cluster is 200-800 MIPS and the associated utilisation constraints: MaxCPUTime (30-40 time units), MaxWCTime (400-700 time units), MaxSpace (1K-5K size units), MaxTransTime (8K-9K time units), and PCost (3-10 cost units per a time unit). For benchmarking, two tasks are selected from the first four dominating categories. The user submits 400 tasks to the scheduler at start-up time and periodically submits 200 tasks at intervals set by Poisson distribution with λ=1.0 time unit. Figure 4 depicts the performance table/charts of the scheduler from the following experiments.

EXPERIMENT I: First, we trace the performance of the scheduler with three resources (R_0-R_2), UBudget=6000 cost units, and UDeadline=200K time units. Table 1 depicts the number of remaining tasks in each TCat during the deployment iterations. Initially, 13 categories are created as indicated in Column I. Column II shows the tasks upon the benchmark phase ($BTasks_{TOTAL} = 24$) with remaining UDeadline=190K time units and UBudget=5815 cost units.

After the benchmark, the task granularity is computed for each resource based on $TCat_k - R_j$ average deployment matrices. The resulting task groups for the three resources:

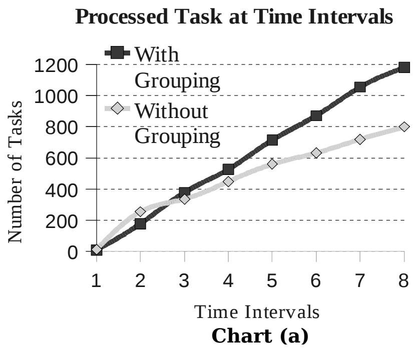
$R_0 : TCat_0(24), R_1 : TCat_0(13) + Tcat_1(56), R_2 : TCat_1(4) + TCat_2(20)$
e.g. $TCat_0(24)$ indicates 24 tasks from $TCat_0$

Table 1. Remaining Category Tasks

TCat	0	1	2	3	4	5	6	7	8	9	10	11	12	Total
I	37	66	80	67	2	62	16	35	29	1	3	1	1	400
II	37	60	74	61	2	56	16	35	29	1	3	1	1	376
III	0	0	54	61	2	56	16	35	29	1	3	1	1	259
IV	0	0	20	9	2	56	16	35	29	1	3	1	1	173
V	15	45	63	38	3	82	23	51	44	2	4	2	1	373
VI	0	0	0	0	0	31	11	83	74	4	6	4	1	214

Table 2. The Validation of Task Granularity

R	Estimated Average Matrices according to the Task Granularity *vs* Objective Functions	Actual Deployment Matrices of the Processed Task Groups (Proposed Scheduler)	Average Deployment Matrices of the Processed Individual Tasks (Conventional Scheduler)
R_0	AvgCPUTime:29 *vs* MaxCPUTime:30 AvgWCTime:31 *vs* MaxWCTime:700 AvgTrantTime:1600 *vs* MaxTransTime:9500 AvgSpace:336 *vs* MaxSpace:5000 AvgPCost:88 *vs* UBudget:5815 AvgTRTime:965 *vs* UDeadline:190K	CPU time:22 Wall-clock time:25 Transmission time:1712 Space:355 Processing cost:66	CPU time:1.6 Wall-clock time:2 Transmission time:634 Space:20 Processing cost:4.8
R_1	AvgCPUTime:49 *vs* MaxCPUTime:50 AvgWCTime:53 *vs* MaxWCTime:700 AvgTrantTime:7310 *vs* MaxTransTime:8000 AvgSpace:1274 *vs* MaxSpace:10000 AvgPCost:299 *vs* UBudget:5727 AvgTRTime:7365 *vs* UDeadline:190K	CPU time:34 Wall-clock time:38 Transmission time:4900 Space:1113 Processing cost:204	CPU time:1.1 Wall-clock time:2 Transmission time:564 Space:20 Processing cost:6.6



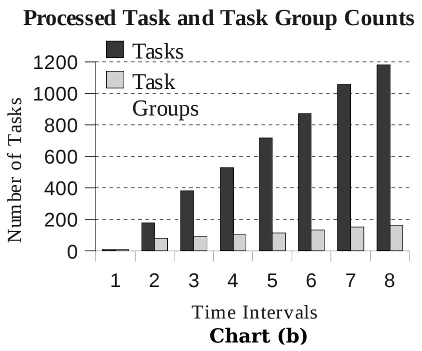


Fig. 4. Performance Tables and Charts of the Proposed Scheduler

Table 2 shows how the estimated granularities for R_0 and R_1 adhere to the constraints in the objective functions. The actual deployment matrices of the relevant processed task groups prove that the scheduling algorithm fulfills all the seven objective functions for deploying 117 tasks (Table 1, Column III) in batches. The next iteration uses the same average deployment matrices, resulting in groups with $R_0 : TCat_1(17), R_1 : TCat_2(17) + Tcat_3(19), R_2 : TCat_3(33)$; Column IV indicates the remaining 173 tasks.

After this point, a new set of 200 tasks arrived at the scheduler (Column V). The subsequent iteration is guided by the average analysis and task groups are formed based on the latest grid status; $R_0 : TCat_0(15) + TCat_1(14), R_1 : TCat_1(31) + TCat_2(19), R_2 : TCat_2(44) + TCat_3(1)$. The scheduler flow continues with average analysis, task granularity, grouping, deployment and new task arrival. At the end, the scheduler managed to complete 786 tasks out of 1000 within the UDeadline (Column VI). For comparison purpose, a similar experiment was conducted with conventional task scheduling (deploying tasks one-by-one). The scheduler deployed only 500 tasks out of 1000 within the UDeadline.

An instance of average deployment matrices of the conventional scheduler is shown in Table 2. R_0 manage to process 24 tasks (in a group) in 1759 time units which can be averaged as 73.29 time units per task. However, the conventional scheduler spent 637.6 time units to process one task; 99.4% of the deployment time is used for file transmission purpose. This indicates that a grid environment is not suitable for lightweight tasks. Hence, there is a strong need for the proposed scheduler which can adaptively resize the batch size for efficient grid utilisation.

EXPERIMENT II: Here, we conduct the simulation in an environment of eight resources with UDeadline=600K time units and UBudget=8000 cost units. The charts in Fig. 4 show the performance based on the observations at eight time intervals upon scheduler start-up. After the second interval, the scheduler produced better outcome throughout the application deployment in terms of total processed tasks as shown in Chart (a). For example, our scheduler successfully executed 1181 tasks by interval 8, whereas the conventional scheduler executed only 800 tasks, resulting in a performance improvement of 47.63%. Chart (b) depicts the task and task group counts processed by the proposed scheduler. For example, 716 tasks are successfully processed by our scheduler at interval 5 (Chart (a)). Interval 5 on Chart (b) indicates that there are only 113 file transmissions needed to process the 716 tasks (24 benchmark tasks and 89 groups). However, the conventional scheduler had 560 file transmissions by this interval (Chart (a)), an additional communication overhead of 20.18%.

6 Conclusion

The proposed scheduling algorithm uses simple statistical computations to decide on the task granularity that satisfies the current resource-network utilisation constraints and user's QoS requirements. The experiments prove that the scheduler leads towards an economic and efficient usage of grid resources and network utilities. The scheduler is currently being implemented for real grid applications. In future, the algorithm will be adapted to support work-flow application models. The scheduler will be improved to deal with unforeseen circumstances such as task failure and migration as well.

References

1. Berman, F., Fox, G.C., Hey, A.J.G., eds.: Grid Computing - Making the Global Infrastructure a Reality. Wiley and Sons (2003)
2. Baker, M., Buyya, R., Laforenza, D.: Grids and grid technologies for wide-area distributed computing. Softw. Pract. Exper. **32** (2002) 1437–1466
3. Jacob, B., Brown, M., Fukui, K., Trivedi, N.: Introduction to Grid Computing. IBM Publication (2005)
4. Buyya, R., Date, S., Mizuno-Matsumoto, Y., Venugopal, S., Abramson, D.: Neuroscience instrumentation and distributed analysis of brain activity data: a case for escience on global grids: Research articles. Concurrency and Computation: Practice and Experience (CCPE) **17** (2005) 1783–1798
5. Muthuvelu, N., Liu, J., Soe, N.L., Venugopal, S., Sulistio, A., Buyya, R.: A dynamic job grouping-based scheduling for deploying applications with fine-grained tasks on global grids. In: Proceedings of the 2005 Australasian workshop on Grid computing and e-research, Australian Computer Society, Inc. (2005) 41–48
6. 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, Washington, DC, USA, IEEE Computer Society (2007) 66–73
7. Arnon, R.G.O.: Fallacies of distributed computing explained. http://www.webperformancematters.com/ (2007)
8. Ranaldo, N., Zimeo, E.: A framework for qos-based resource brokering in grid computing. In: Proceedings of the 5th IEEE European Conference on Web Services, the 2nd Workshop on Emerging Web Services Technology. Volume 313., Halle, Germany, Birkhuser Basel (2007) 159–170
9. James, H., Hawick, K., Coddington, P.: Scheduling independent tasks on metacomputing systems. In: Proceedings of Parallel and Distributed Computing Systems, Fort Lauderdale, US (1999) 156–162
10. Sodan, A.C., Kanavallil, A., Esbaugh, B.: Group-based optimizaton for parallel job scheduling with scojo-pect-o. In: Proceedings of the 2008 22nd International Symposium on High Performance Computing Systems and Applications, Washington, DC, USA, IEEE Computer Society (2008) 102–109
11. 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** (2006) 467–480
12. Ng, W.K., Ang, T., Ling, T., Liew, C.: Scheduling framework for bandwidth-aware job grouping-based scheduling in grid computing. Malaysian Journal of Computer Science **19** (2006) 117–126
13. Stokes, J.H.: Behind the benchmarks: Spec, gflops, mips et al. http://arstechnica.com/cpu/2q99/benchmarking-2.html (2000)
14. Muthuvelu, N., Chai, I., Chikkannan, E.: An adaptive and parameterized job grouping algorithm for scheduling grid jobs. In: Proceedings of the 10th International Conference on Advanced Communication Technology. Volume 2. (2008) 975–980
15. Lowekamp, B., Tierney, B., Cottrell, L., Jones, R.H., Kielmann, T., Swany, M.: A Hierarchy of Network Performance Characteristics for Grid Applications and Services. (2003)
16. Buyya, R., Murshed, M.M.: Gridsim: A toolkit for the modeling and simulation of distributed resource management and scheduling for grid computing. Concurrency and Computation: Practice and Experience (CCPE) **14** (2002)