MRRS: A More Efficient Algorithm for Scheduling Divisible Loads of Grid Applications

Nguyen The Loc*, Said Elnaffar**, Takuya Katayama*, Ho Tu Bao*
* *Japan Advanced Institute of Science and Technology (JAIST).*
1-1 Asahidai, Nomi, Ishikawa 923-1292 Japan.
{lnguyen,katayama,bao}@jaist.ac.jp
** *College of IT, UAE University, Al-Ain, UAE .*
elnaffar@uaeu.ac.ae

Abstract

Many algorithms in the literature have been targeting the problem of scheduling divisible workloads (those loads that are amenable to partitioning in any number of chunks). Unfortunately, such algorithms have a number of shortcomings such as the sole reliance in their computations on CPU speed, and the assumption that a definite set of workers are available and must participate in processing the load. These constraints limit the utility of such algorithms and make them impractical for a computing platform such as the Grid. In this paper, we propose an algorithm, MRRS, that overcomes these limitations and adopts a worker selection policy that aims at minimizing the execution time. The MRRS has been evaluated against other scheduling algorithms such as UMR and LP and showed better results.

Key words

Divisible load, task scheduling, resource selection.

1. Introduction

By definition, a divisible load is "a load that can be partitioned into any arbitrary number of load fractions" [1]. This kind of workload arises in many domains of Grid computing [3] such as protein sequence analysis and the simulation of cellular micro physiology. Per the Divisible Load Theory [1], the scheduling problem is identified as: "Given an arbitrary divisible workload, in what proportion should the workload be partitioned and distributed among the workers so that the entire workload is processed in the shortest possible time."

The first multi-round (MI) algorithm, introduced by Bharadwaj [1], utilizes the overlapping between communication and computation processes at workers. In the MI algorithm the number of rounds is predefined and fixed. It overlooks communication and computation latencies. The studies in [4] focus on Affine model in which computation and communication latencies are different from zero. Yang et al. [2] through their UMR (Uniform Multi-Round) algorithm, designed a better algorithm that extends the MI by considering latencies. However, in the UMR, the size of workload chunks delivered to workers is solely calculated based on worker's CPU power; the other key system parameters, such as network bandwidth, are not factored in. Beaumont [4] proposes another multi round scheduling algorithm that fixes the execution time for each round. This enabled the author to give analytical proof of the algorithm's asymptotic optimality. But the influence of this assumption on the utilization of transfer-execution overlap is questionable.

One apparent shortcoming in many scheduling algorithms [1], [2], [4] is the abandon of a selection policy for the best subset of available workers. The selection of the best workers is a chief task especially in a large platform such as the Grid that agglomerates hundreds or thousands of workers. The present scheduling algorithms rely on the assumption that all workers are available and must participate in processing load partitions. This assumption is not always realistic and may not lead to the minimum execution time (henceforth it is called *makespan*).

In this paper, we propose a new scheduling algorithm, MRRS (Multi-round Scheduling with Resource Selection), which is inspired by the UMR algorithm. MRRS is superior to UMR with respect to two aspects. First, unlike the UMR, which relies primarily in its computation on the CPU speed, MRRS factors in several other parameters such as bandwidth capacity and all types of latencies (CPU and Bandwidth) which renders the MRRS a more realistic model. Second, the UMR assumes that all workers are available and should participate in the workload processing,

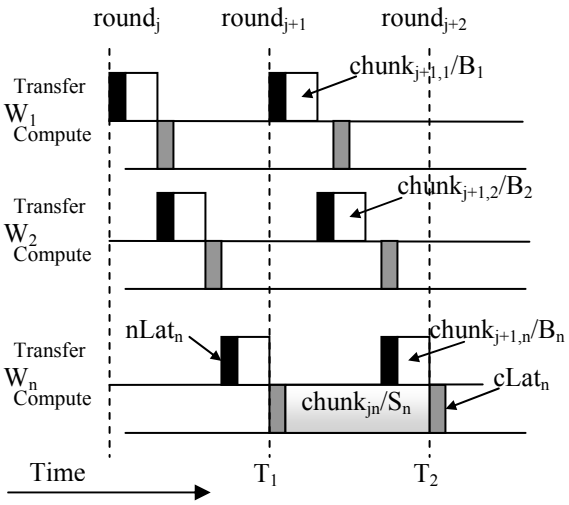

Figure 1: Dispatching Load Chunks Using the MRRS Algorithm

which is not practical especially in a computing environment such as the Grid. The MRRS, on the other hand, is equipped with a worker selection policy that works on selecting the best workers that can produce shorter makespan. As a result, our experiments show that our MRRS algorithm outperforms previously proposed algorithm including the UMR.

The rest of this paper is organized as follows. Section 2 briefly describes the heterogeneous computation platform. Section 3 and Section 4 explain the workload partitioning mechanism and the worker selection strategy, respectively, that are adopted by the MRRS algorithm. Section 5 describes the simulation experiments we have conducted in order to evaluate our work. Section 6 concludes the paper.

2. Heterogeneous Computing Platform

Let us consider a computation Grid, in that, a master process has access to N worker processes and each process runs in a particular computer. The master can divide the total workload into arbitrary chunks and delivers them to appropriate workers. We assume that the master uses its network connection in a sequential fashion, i.e., it does not send chunks to some workers simultaneously. The communication and computation platforms of our system are heterogeneous. Workers can receive data from network and perform computation simultaneously.

The following notations will be used throughout this paper:

- W_i: worker number i
- N: total number of available workers
- n: total number of workers that are actually selected to process the workload
- m: the number of rounds.
- L_{total}: the total amount of workload (flop).
- $chunk_{j,i}$: the fraction of total workload that the master delivers to worker i in round j ($i=1...n$, $j=1...m$).
- S_i: computation speed of the worker i measured by the number of units of workload performed per second (flop/s).
- B_i: the data transfer rate of the connection link between the master and W_i (flop/s).
- $Tcomp_{j,i}$: computation time required for W_i to process $chunk_{j,i}$.
- $cLat_i$: the fixed overhead time (second) needed by W_i to start computation.
- $nLat_i$: the overhead time (second) incurred by the master to initiate a data transfer to W_i. We denote total latencies by $Lat_i = cLat_i + nLat_i$.
- $Tcomm_{j,i}$: communication time required for master to send $chunk_{j,i}$ to W_i.

$$Tcomm_{j,i} = nLat_i + chunk_{j,i} / B_i ;$$
$$Tcomp_{j,i} = cLat_i + chunk_{j,i} / S_i ;$$

- $round_j$: the fraction of workload dispatched during round j

$$round_j = chunk_{j,1} + chunk_{j,2} + ... + chunk_{j,n}$$

We fix the time required for each worker to perform communication and computation during each round

$$cLat_i + \frac{chunk_{j,i}}{S_i} + nLat_i + \frac{chunk_{j,i}}{B_i} = const_j$$

We set: $A_i = B_i S_i / (B_i + S_i)$

so we have

$$chunk_{j,i} = \alpha_i \times round_j + \beta_i \qquad (1)$$

where

$$\alpha_i = \frac{A_i}{\sum_{i=1}^{n} A_i} \qquad \beta_i = A_i \frac{\sum_{k=1}^{n} A_k \left(Lat_k - Lat_i\right)}{\sum_{k=1}^{n} A_k}$$

3. Workload partitioning

In the following subsections, we induce chunk sizes (Section 3.1) and determine the parameters of the initial scheduling round (Section 3.2). We refer the reader to [2] for more information and detailed derivations.

3.1. Induction Relation for Chunk Sizes

Figure 1 depicts the operation of our algorithm, where the computation and communication have been overlapped. At time T_1, the master starts sending round (j+1) to all workers and the last worker W_n starts computation chunk j concurrently. To fully utilize the network bandwidth, the dispatching of the master and the computation of W_n should finish at the same time T_2:

$$\sum_{i=1}^{n} \left(nLat_i + \frac{chunk_{j+1,i}}{B_i} \right) = \frac{chunk_{j,n}}{S_n} + cLat_n$$

If we replace $chunk_{j+1,i}$ and $chunk_{j,n}$ by their expression in (1) we derive:

$$round_{j+1} = round_j \times \theta + \mu \quad (2)$$

where

$$\theta = B_n \Big/ \left((B_n + S_n) \sum_{i=1}^{n} \frac{S_i}{B_i + S_i} \right)$$

$$\mu = \left(\frac{\beta_n}{S_n} + cLat_n - \sum_{i=1}^{n} \left(nLat_i + \frac{\beta_i}{B_i} \right) \right) \Big/ \sum_{i=1}^{n} \frac{\alpha_i}{B_i}$$

From induction equation (2) we can compute:

$$round_j = \theta^j (round_0 - \eta) + \eta \quad (3)$$

where $$\eta = \frac{\beta_n + cLat_n - \sum_{i=1}^{n} \left(nLat_i + \frac{\beta_i}{B_i} \right)}{\sum_{i=1}^{n} \frac{\alpha_i}{B_i} - \frac{\alpha_n}{S_n}}$$

3.2. Determining the Parameters of the Initial Round

In this section we compute the optimal number of rounds, m, and the size of the initial load fragment that should be distributed to workers in the first round, $round_0$. If we let $F(m,round_0)$ denote the makespan, then from Figure 1 we can seen that

$$F(m, round_0) = \quad (4)$$

$$= \sum_{i=1}^{n} \left(\frac{chunk_{0,i}}{B_i} + nLat_i \right) + \sum_{j=0}^{m-1} \left(\frac{chunk_{j,n}}{S_n} + cLat_n \right)$$

$$= round_0 \left(\sum_{i=1}^{n} \frac{\alpha_i}{B_i} + \frac{\alpha_n (1 - \theta^m)}{S_n (1 - \theta)} \right) + \sum_{i=1}^{n} \left(\frac{\beta_j}{B_i} + nLat_i \right)$$

$$+ m \left(cLat_n + \frac{\alpha_n \eta + \beta_n}{S_n} \right) - \frac{\alpha_n \eta (1 - \theta^m)}{1 - \theta}$$

Our objective is to minimize the makespan $F(m,round_0)$, subject to:

$$\sum_{j=0}^{m-1} round_j = L_{total}$$

or (5)

$$G(m, round_0) = m\eta + (round_0 - \eta) \frac{1 - \theta^m}{1 - \theta} - L_{total} = 0$$

We use Lagrangian method [5] to solve this constrained minimization problem. The minimum value of function $F(m,round_0)$ can be found by solving the following equation system:

$$\left\{ \frac{\partial L}{\partial \lambda} = 0; \frac{\partial L}{\partial m} = 0; \frac{\partial L}{\partial round_0} = 0 \right\}$$

where:

- λ: Lagrange multiplier
- $L(m, round_0)$: Lagrangian function which is defined as: $L(m, round_0, \lambda) = F(m, round_0) + \lambda G(m, round_0)$

After solving this equation system we obtain m. Using (5) one can then compute $round_0$. At last, using (3) and (1) we will obtain the value of $round_j$ and $chunk_{j,i}$ respectively ($i=1..n, j=1..m$).

Algorithm 1: Worker_Selection(V)
Begin
Search $W_n \in V$ such that:
$B_n/(B_n+S_n) \le B_i/(B_i+S_i) \quad \forall W_i \in V$
V^*_1 = Brand_and_bound(V);
V^*_2 = Greedy(V, "θ<1"); V^*_3 = Greedy(V, "θ=1");
select $V^* \in \{V^*_1, V^*_2, V^*_3\}$ such that
$m(V^*) = \min\{m_1(V^*_1), m_2(V^*_2), m_3(V^*_3)\}$;
return (V^*);
End

4. Worker Selection Policy

Let V denote the original set of N available workers ($|V|=N$). In this section we explain our worker selection policy that aims at finding the best subset V^* ($V^* \subseteq V$, $|V^*|=n$) that minimizes the makespan.

If W_i denotes worker i, then W_n denotes the last worker receives load chunks in a round, and W_1 denotes the first worker that receives chunks in a round. Algorithm 1 outlines our selection algorithm. It starts with finding the last worker (W_n) that should receive chunks in a round. V^* is initialized by $\{W_n\}$. Afterwards, the selection algorithm, depending on θ, examines three cases using different search algorithms aiming at finding the best algorithm that adds more workers to V^*. After obtaining the three candidate V^* sets, the algorithm chooses the V^* set that produces the minimum makespan.

From (4) we compute the makespan as follows

if θ=1: makespan =

$$= \frac{L_{total}}{\sum_{i \in V^*} A_i} \left(\frac{1}{m} \sum_{i \in FS} \frac{S_i}{B_i + S_i} + \frac{B_n}{B_n + S_n} \right) + C \quad (6)$$

where C is a constant

$$C = \sum_{i \in V^*} nLat_i + m.cLat_n$$

if $\theta \neq 1$: makespan =

$$= \frac{L_{total}}{\sum_{i \in V^*} A_i} \left(\frac{1 - \theta}{1 - \theta^m} \sum_{i \in FS} \frac{S_i}{B_i + S_i} + \frac{B_n}{B_n + S_n} \right) + C \quad (7)$$

Now, since

$$\lim_{m \to \infty} \frac{1 - \theta}{1 - \theta^m} = \begin{cases} 0 & \text{if } \theta > 1 \\ 1 - \theta & \text{if } \theta < 1 \end{cases}$$

and since m (the number of rounds) is usually large (in our experiments, m is in hundreds), we can write:

$$\frac{1 - \theta}{1 - \theta^m} \approx \begin{cases} 0 & \text{if } \theta > 1 \\ 1 - \theta & \text{if } \theta < 1 \end{cases}$$

when θ>1 and by substituting this term into (7) we get

$$makespan = L_{total} \frac{B_n}{(B_n + S_n)\sum_{i \in V^*} \frac{B_i S_i}{B_i + S_i}} \quad (8)$$

when θ <1 and by substituting the above term into (7) we get makespan =

$$= L_{total} \sum_{i \in V^*} \frac{S_i}{B_i + S_i} \Big/ \sum_{i \in V^*} \frac{B_i S_i}{B_i + S_i} + C \quad (9)$$

Based on the above analysis, we have three selection policies for generating V^*:

1. Policy I (θ>1): this policy aims at reducing the total idle time by progressively increasing the load processed in each round (i.e., $round_{j+1} > round_j$).
2. Policy II (θ<1): this policy aims at maximizing the number of workers that can participate by progressively decreasing the load processed in each round (i.e., $round_{j+1} < round_j$).
3. Policy III (θ =1): this policy keeps the load processed in each round constant (i.e., $round_{j+1} = round_j$)

As shown in Algorithm 1, the three policies will be examined in order to choose the one that produces the minimum makespan. In the coming subsections, we will explain the search algorithm adopted by each policy.

4.1. Policy I (θ >1)

From (8), we can see that under this policy, V^* is the subset that maximizes the sum

$$m_1(V^*) = \sum_{i \in V^*} \frac{B_i S_i}{B_i + S_i}$$

subject to θ>1 or

$$\sum_{i \in V^*} \frac{S_i}{B_i + S_i} < \frac{B_n}{B_n + S_n} \quad (10)$$

One can observe that this is Binary Knapsack [7] problem that can be solved using the Brand-and-bound algorithm [7].

4.2. Policy II (θ<1)

From (9), we can see that under this policy, V^* is the subset that minimizes

$$m_2(V^*) = \sum_{i \in V^*} \frac{S_i}{B_i + S_i} \Big/ \sum_{i \in V^*} \frac{B_i S_i}{B_i + S_i}$$

subject to θ *<1 or*

$$\sum_{i \in V^*} \frac{S_i}{B_i + S_i} > \frac{B_n}{B_n + S_n} \quad (11)$$

To start with, we should initiate V^* with the first worker, W_0, that minimizes $m_2()$.

Lemma 1. *$m_2(V^*)$ is minimum if $V^*=\{Wo\}$ such that $B_0 \geq B_i \ \forall P_i \in V$.*

Proof. Consider an arbitrary subset $X \subseteq V$, $X = \{P_1, P_2, \ldots P_r\}$. We have:

$$B_0 > B_i \Rightarrow \sum_{i=1}^{r} \frac{B_0 S_i}{B_i + S_i} > \sum_{i=1}^{r} \frac{B_i S_i}{B_i + S_i} \qquad \forall W_i \in V$$

$$\Rightarrow B_0 \sum_{i=1}^{r} \frac{S_i}{B_i + S_i} > \sum_{i=1}^{r} \frac{B_i S_i}{B_i + S_i}$$

$$\Rightarrow \frac{\frac{S_0}{B_0 + S_0}}{\frac{B_0 S_0}{B_0 + S_0}} < \frac{\sum_{i=1}^{r} \frac{S_i}{B_i + S_i}}{\sum_{i=1}^{r} \frac{B_i S_i}{B_i + S_i}} \Rightarrow m_2(V^*) < m_2(X)$$

□

After adding W_0 to V^*, we should keep conservatively adding more workers until constraint (11) is satisfied. In fact, the next W_k that should be added to V^* is the one that satisfies the following inequality:

$$m_2(V^* \cup \{W_k\}) \leq m_2(V^* \cup \{W_j\}) \ \ \forall W_j \in V\text{-}V^*$$

The *Greedy* algorithm described below progressively adds more P_k until V^* satisfies (11), i.e. until (θ<1). The run time complexity of this search is $O(n)$.

Algorithm Greedy (V, thetaCondition**)**
Begin
  Search $W_n \in V$: $B_n/(B_n+S_n) \leq B_i/(B_i+S_i)$ $\forall W_i \in V$
  Search $W_0 \in V$: $B_0 \geq B_i$ $\forall W_i \in V$
  $V^* = \{W_n, W_0\}$; $V = V\text{-}V^*$;
  Repeat
    Search worker W_k *satisfy*
      $m_2(V^* \cup \{W_k\}) \leq m_2(V^* \cup \{W_j\})$ $\forall W_j \in V$
    $V^* = V^* \cup \{W_k\}$; $V = V\text{-}\{W_k\}$;
  Until thetaCondition*;*
  return (V^*);
End

4.3. Policy III (θ=1)

Under this policy, we need to find V^* that minimizes the following makespan function

$$m_3(V^*) = \sum_{i \in V^*} \frac{S_i}{B_i + S_i} \Big/ \sum_{i \in V^*} \frac{B_i S_i}{B_i + S_i}$$

subject to θ=1 or

$$\frac{B_n}{(B_n + S_n)\sum_{i \in V^*} \frac{S_i}{B_i + S_i}} = 1$$

It is noticeable that $m_3()$ is the same as $m_2()$ (Policy II). However, the two objective functions differ with respect to their constraints. Therefore, we can use the same *Greedy* search algorithm explained earlier with the

Table 2. Experiment Parameters

Parameter	*Value*
Number of workers	$N = 50$
Total workload (flop)	10^6
Computation speed (flop/s)	Randomly selected from $[S_{min}, 1.5 \times S_{min}]$, where $S_{min} = 50$
Communication rate (flop/s)	Randomly selected from $[0.5 \times N \times S_{min}, 1.5 \times N \times S_{min}]$

exception that the termination condition should be $\theta = 1$ (instead of $\theta < 1$).

5. Experimental Results

In order to evaluate our new algorithm, MRRS, we developed a simulator using the SIMGRID toolkit [6], which has been used to evaluate the UMR algorithm. We conducted a number of experiments that aim at i) showing the validity of our approximation assumptions discussed in Section 4, and ii) showing that the MRRS algorithm is superior to its predecessor multi-round algorithms, namely LP and UMR.

5.1. Validity of Approximation Assumptions

The experiments we conducted show that the absolute deviation between theoretically computed makespan, as analyzed in Section 4, and the makespan observed through the simulation experiments is negligible. This confirms that the approximation assumptions adopted in our analysis are plausible.

Table 2 outlines the parameters that we used in our experiments. Let us denote:

- MK_e is the makespan obtained from the experiments.
- MK_1, MK_2, MK_3 are the makespans computed by formula (6), (8) and (9) respectively.
- D_i (i=1, 2, 3) is the absolute deviation between the theoretical makespan, MK_i, and the experimental makespan MK_e. Therefore:

$$D_i = 100.\frac{|MK_i - MK_e|}{MK_e}(\%) \quad i = 1,2,3$$

Table 3. The Absolute Deviation between the Experimental and Theortical Makespans

nLat, cLat (s)	D_1 (%)	D_2 (%)	D_3 (%)
1	3.15	2.42	3.34
10^{-1}	2.23	1.75	2.27
10^{-2}	1.51	0.92	1.94
10^{-3}	0.82	0.51	1.25

Table 3 summarizes the absolute deviations computed for different latencies. From these results we can make the following remarks:

- The absolute deviation between the theoretical and the experimental makespans ranges from 0.5% to 3.1%, which is negligible.
- We notice that $D_2 < D_1 < D_3$. The justification is that the absolute deviation (D) is propotional to the number of participating workers in a given selection policy. The more workers participate, the larger D becomes. As we recall that D_2 represents the deviation caused by policy II ($\theta > 1$), which is the most conservative policy with respect to the number of workers allowed to participate. D_3 represents the deviation caused by policy III ($\theta < 1$), which is the most relaxed policy with respect to the number of participating workers. D_1 of policy I ($\theta = 1$) falls in the middle with respect to the number of participating workers and according the observed deviation.

5.2. Comparison with Previous Algorithms

We compare MRRS with the most powerful scheduling algorithm, namely UMR [2], [8] and LP [4]. Table 4 outlines the configuration parameters used in the simulation experiments. The performances of these algorithms have been compared with respect to three metrics:

- The normalized makespan, that is normalized to the run time achieved by the best algorithm in a given experiment;
- The rank which ranges from 0 (best) to 2 (worst);
- The degradation from the best, which measures the relative difference, as a percentage, between the makespan achieved by a given algorithm and the makespan achieved by the best one.

These metrics are commonly used in the literature for comparing scheduling algorithms [2].

Table 4. Simulation parameters

Parameters	*Values*
N: Number of available workers	10, 12,...,50
Total workload (flop)	5.10^5
Computation speed (flop/s)	Randomly selected from the range $[S_{min}, 1.5S_{min}]$, where S_{min} = 5, 10, 15, 20
Communication rate (flop/s)	Randomly selected from the range $[0.5NS_{min}, 1.5NS_{min}]$
Computation and communication latencies (s)	10,1, 10^{-1},10^{-2}

Table 5. Performance comparisons among MRRS, UMR and LP Algorithms

Algorithm	Normalized Makespan	Rank	Degradation from the best
MRRS	1	0.12	0.65
UMR	1.21	0.88	21.4
LP	1.59	2	59.8

The results summarized in Table 5 suggests that MRRS can outperform its competitors in most of the cases. MRRS's rank drop from the best to the second in 12% of the cases with 5.4% performance degradation in comparision with the UMR.

Figure 2 shows that UMR has chances of outperforming MRRS only if the number of workers is small ($n \leq 20$), because in those cases, the worker selection module of MRRS does not have enough workers, which denies the MRRS from adopting one of the worker selection policies, namely Policy II. LP has almost no chance to win. This is due to the fact that LP does not have any effective strategy of reducing the idle time of workers at the end of each round.

6. Conclusion

The ultimate goal of any scheduling algoritm is to minimize the makespan. UMR and LP are among these algorithms that have been designed to schedule divisible loads in heterogeneous environments where workers have different CPU speeds connected to links with different bandwidths. However, these algorithms do not take into account a number of the chief parameters such as bandwidths and the inevitable latencies of communication and computation. Furthermore, present algorithms are not equipped with a resource selection mechanism as they assume that all available workers will participate in processing the workload. In this work, we presented the MRRS algorithm that divides the workload into chunks in light of more realistic parameters mentioned earlier.

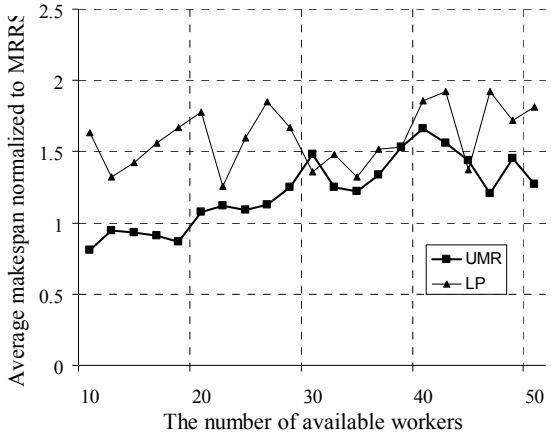

Figure 2: The relation between *n* and the makespan

We explained how the MRRS has a worker selection with an asymptotic optimality. To the best of our knowledge, MRRS is the first scheduling algorithm that addresses the worker selection problem. The simulation experiments show that MRRS is superior to its predecessors especially when it is put into operation in a colossal computing platform such as the Grid, which agglomerate an abundant pool of heterogeneous workers

Acknowledgements

Our research is conducted as a program for the "Fostering Talent in Emergent Research Fields" in Special Coordination Funds for promoting Science and Technology by Ministry of Education, Culture, Sports, Science and Technology, Japan. Our research has also been supported by research grant \#02-06-9-11/06 from the Scientific Research Council of the UAE University.

References

[1] V. Bharadwaj, D.Ghose, V.Mani, and T. G. Robertazzi, *Scheduling Divisible Loads in Parallel and Distributed Systems* (IEEE Computer Society Press, 1996).

[2] Y. Yang, K.V. Raart, & H. Casanova, Multiround Algorithms for Scheduling Divisible Loads, *IEEE Transaction on Parallel and Distributed Systems*, 16(11) 2005, 1092-1104.

[3] I. Foster and C. Kesselman, *Grid2: Blueprint for a New Computing Infrastructure* (Second ed. San Francisco, Morgan Kaufmann Publisher , 2003).

[4] O. Beaumont, H. Casanova, A. Legrand, Y. Robert, & Y. Yang, Scheduling Divisible Loads on Star and Tree Networks: Results and Open Problems, *IEEE Transactions on Parallel and Distributed Systems* , 16(3), 2005, 207-218.

[5] D. P. Bertsekas, *Constrained Optimization and Lagrange Multiplier Methods* (Belmont, Mass. : Athena Scientific , 1996).

[6] H.Casanova, Simgrid: a Toolkit for the Simulation of Application Scheduling, *Proc. of the IEEE International Symposium on Cluster Computing and the Grid (CCGrid'01)*, Australia, 2001, 430-437.

[7] S. Martello and P. Toth, *Knapsack problems : algorithms and computer implementations* (Chichester, West Sussex, England : Wiley , 1990)

[8] Y. Yang, & H. Casanova, UMR: A Multi-Round Algorithm for Scheduling Divisible Workloads, *Proc. of the International Parallel and Distributed Processing Symposium (IPDPS'03)*, Nice, France, 2003.