UMR2: A BETTER AND MORE REALISTIC SCHEDULING ALGORITHM FOR THE GRID

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

Numerous studies have been targeting the problem of scheduling divisible workloads in distributed computing environments. The UMR (Uniform Multi-Round) algorithm stands out from all others by being the first close-form optimal scheduling algorithm. However, present algorithms, including the UMR, do not pay due attention to optimizing the set of workers that get selected to participate in processing workload chunks. In addition to the absence of a good resource selection policy, the UMR relies primarily in its computation on the CPU speed and overlooks the role of other key parameters such as network bandwidth. In this paper, we propose an extended version of UMR, called UMR2, that overcomes these limitations and adopts a worker selection policy that aims at minimizing the makespan. We, theoretically and experimentally, show that UMR2 is superior to UMR, specifically in a WAN computing platform such as the Grid.

KEY WORDS
Divisible loads, multi-round algorithms, Grid computing.

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 science such as protein sequence analysis, simulation of cellular micro physiology, and more [2], [3]. 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." Any scheduling algorithm should address the following issues:

- *Workload Partitioning Problem*. This problem is concerned with the method by which the algorithm should divide the workload in order to dispatch to workers.
- *Resource Selection Problem*. This problem is concerned with how to select the best set of workers that can process the workload partitions such that the makespan is minimal.

First multi-round algorithm MI, introduced by Bharadwaj [1], utilizes the overlapping between communication and computation processes at workers. In MI algorithm the number of rounds is fixed and predefined. It overlooks communication and computation latencies. Beaumont [4] proposes a 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. Yang et al. [2], through their UMR algorithm, designed a better algorithm that extends the MI by considering latencies. However, in 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.

One apparent shortcoming in many scheduling algorithms [1], [2], [4] is the abandon of designing a solid selection policy for generating the best subset of available workers. Part of the reason is that the main focus of these algorithms is confined to the LAN environment, which makes them not perfectly suitable for a WAN environment such as the Grid [3]. In the Grid, resource computing (workers) join and leave the computing platform dynamically. Unlike other algorithms, we cannot assume in the Grid that all available resources, which may be in thousands, must participate in the scheduling process. The more recent algorithms discussed in [2] very tersely allude to this problem by proposing primitive intuitive solutions that are not back up by any analytical model.

In this paper, we propose a new scheduling algorithm, UMR2 (inspired by UMR [2]), which is better and more realistic. UMR2 is superior to UMR with respect to two aspects. First, unlike UMR that relies primarily in its computation on the CPU speed, UMR2 factors in several other parameters, such as bandwidth and all types of latencies which renders the UMR2 a more realistic model. Second, UMR2 is equipped with a worker selection policy that finds out the best workers. As a result, our

experiments show that our UMR2 algorithm outperforms previously proposed algorithm including the UMR. The rest of this paper is organized as follows. Section 2 briefly describes the computation platform. Section 3 and 4 describe the UMR and UMR2 algorithms, respectively. Section 5 analytically shows how UMR2 is superior to UMR. Section 6 experimentally validates our work and shows how UMR2 outperforms UMR and other similar scheduling algorithms. Section 7 concludes the paper.

2. The Heterogeneous Computing Platform

Let us consider a computation Grid in which a master process has access to N worker processes and each process runs in a particular computer. The master can divide the total load L_{total} into arbitrary chunks and delivers them to appropriate workers. The following notation will be used throughout this paper:

- W_i: worker number i.
- N: total number of available workers.
- n: number of workers that are actually selected
- m: the number of rounds.
- $chunk_{ji}$: the fraction of total workload that the master delivers to W_i in round j ($i=1,..,n$; $j=1,..,m$).
- S_i: computation speed of the worker i (flop/s).
- B_i: the data transfer rate of the connection link between the master and W_i (flop/s).
- $Tcomp_{ji}$: computation time required for W_i to process $chunk_{ji}$.
- $cLat_i$: 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 $Lat_i = cLat_i + nLat_i$.
- $Tcomm_{ji}$: communication time required for master to send $chunk_{ji}$ 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 workload dispatched during round j

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

3. The UMR Algorithm: Overview

In the subsequent sections, we sketch the key concepts in the UMR algorithm, referring the reader to [2] for more details. Next, we explain how the UMR partitions its load into chunks (Section 3.1), how it decides on the size of each chunk (Section 3.2), what are the initial parameters for the first round (Section 3.3), and outline the simple worker selection policy adopted by the UMR.

3.1 Load Partitioning Policy

UMR adopts a load partition policy that ensures that each worker spends the equal CPU time like others through a round; network bandwidth is not taken into account:

$$cLat_i + chunk_{ji} / S_i = const_j$$

so we derive

$$chunk_{ji} = \frac{S_i}{\sum_{k=1}^{n} S_k} round_j + \beta_i \qquad (1)$$

where

$$\beta_i = \frac{S_i}{\sum_{k=1}^{n} S_k} \sum_{k=1}^{n} \left(S_k cLat_k\right) - S_i cLat_i \qquad (2)$$

3.2 Induction Relation on Chunk Sizes

To fully utilize the network bandwidth, the dispatching of the master and the computation of W_n should finish at the same time

$$round_j = \phi^j (round_0 - \eta) + \eta \qquad (3)$$

where $\phi = \left(\sum_{i=1}^{n} \frac{S_i}{B_i} \right)^{-1}$ (4)

$$\eta = \frac{\sum_{i=1}^{n} \left(S_i \times cLat_i\right) - \sum_{i=1}^{n} S_i \times \sum_{i=1}^{n} \left(\frac{\beta_i}{B_i} + nLat_i \right)}{\sum_{i=1}^{n} \frac{S_i}{B_i} - 1} \qquad (5)$$

3.3 Determining the First Round Parameters

Since the objective of the UMR is to minimize the makespan of the application, we can write:

$F(m, round_0) =$

$$= \sum_{i=1}^{n} \left(\frac{chunk_{0i}}{B_i} + nLat_i \right) + \sum_{j=0}^{m-1} \left(\frac{chunk_{jn}}{S_n} + cLat_n \right) \qquad (6)$$

At the same time, we also have the constraint that the chunk sizes sum up to the total workload:

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

This optimization problem can be solved by the Lagrangian method [2], [5].

3.4 Worker Selection Policy

UMR sorts workers according to S_i/B_i in increasing order, and selects the first n workers out of the original N workers such that: $S_1 / B_1 + S_2 / B_2 + ... + S_n / B_n < 1$

Furthermore, UMR requires that, the computation-communication ratio B_i/S_i be larger than the number of workers n: $B_i / S_i > n \qquad (\forall i = 1,2,...,N)$ (7)

4. UMR2: The New Algorithm

In this section, we present our UMR2 algorithm. We explain how the UMR2 partitions its load into chunks (Section 4.1), how it decides on the size of each chunk (Section 4.2), and the determination of initial parameters for the first round (Section 4.3). The key concept in the UMR2 algorithm, which is the worker selection policy, is described in the last Section 4.4.

4.1 Load Partitioning Policy

Unlike the UMR, which considers the CPU power only, our algorithm considers both of the CPU power and the network bandwidth when partitioning the load:

$cLat_i + chunk_{ji}/S_i + nLat_i + chunk_{ji}/B_i = const_j$

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

so we have $chunk_{j,i} = \alpha_i round_j + \beta_i$ (8)

where $\alpha_i = A_i / (A_1 + A_2 + ... + A_n)$

$\beta_i = \alpha_i[A_1(Lat_1 - Lat_i) + ... + A_n(Lat_n - Lat_i)]$ (9)

Expressions (8) and (9) show the equal role that CPU power (S_i) and bandwidth (B_i) play. This renders the UMR2 a more realistic algorithm and therefore, a better one with respect to performance.

4.2 Induction Relation on Chunk Sizes

Similar to the induction relation derived in section 3.2 for the UMR, we have:

$$round_j = \theta^j (round_0 - \eta) + \eta \quad (10)$$

where

$\theta = B_n / (B_n + S_n) / [S_1/(B_1 + S_1) + ... + S_n/(B_n + S_n)]$ (11)

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

4.3 Determining the First Round Parameters

To find out $round_0$ and m of UMR2 we have to minimize the $makespan_{UMR2}$: $F(m, round_0) =$

$$= \sum_{i=1}^{n} \left(\frac{chunk_{0i}}{B_i} + nLat_i \right) + \sum_{j=0}^{m-1} \left(\frac{chunk_{jn}}{S_n} + cLat_n \right) \quad (12)$$

subject to:

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

After obtaining m and $round_0$ by using Lagrangian method, we can obtain the value of $round_j$ and $chunk_{ji}$ using (10) and (8), respectively.

4.4 Worker Selection Policy

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

Algorithm 1: Resource_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 = Branch_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

If W_i denotes worker i, then W_n denotes the last worker that receives load chunks in a round, and W_1 denotes the first worker that receives chunks in a round. Our selection, as sketched in Algorithm 1, starts with finding the last worker (W_n) that should receive chunks in a round. Therefore, 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 one that produces the minimum makespan. When θ=1, and by using (12), we compute the makespan as follows: $makespan_{UMR2} =$

$$= \frac{L_{total}}{\sum_{i \in V^*} A_i} \left(\frac{1}{m} \sum_{i \in V^*} \frac{S_i}{B_i + S_i} + \frac{B_n}{B_n + S_n} \right) + C \quad (13)$$

where C is a constant

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

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:

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

We evaluate the accuracy of this approximation by experiments mentioned in sub section 6.1

When θ>1 and by substituting this term into (12) we get

$$makespan_{UMR2} = \frac{L_{total} \times B_n}{(B_n + S_n) \sum_{i \in V^*} \frac{B_i S_i}{B_i + S_i}} + C \quad (14)$$

When θ<1 and by substituting the above term into (12) we get $makespan_{UMR2} =$

$$= L_{total} \sum_{i \in V^*} \frac{S_i}{B_i + S_i} \Bigg/ \sum_{i \in V^*} \frac{B_i S_i}{B_i + S_i} + C \quad (15)$$

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., $round_{j+1} > round_j \ \forall j$ =0,1,...,m-1).
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., $round_{j+1} < round_j \ \forall j$ =0,...,m-1).
3. Policy III (θ=1): this policy keeps the load processed in each round constant (i.e., $round_{j+1} = round_j \ \forall j$ = 0,1,...,m-1). As shown in Algorithm 1, three policies will be examined in order to choose the one that produces the minimum makespan.

Next, we discuss each policy in more detail.

4.4.1. Policy I (θ >1)

From (14), 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 $\theta > 1$ or

$$\sum_{i \in V^*} \frac{S_i}{B_i + S_i} < \frac{B_n}{B_n + S_n} \quad (16)$$

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

4.4.2. Policy II ($\theta < 1$)

From (15), 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 $\theta < 1$ or

$$\sum_{i \in V^*} \frac{S_i}{B_i + S_i} > \frac{B_n}{B_n + S_n} \quad (17)$$

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 $V^*=\{W_0\}$ 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:

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

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

After adding W_0 to V^*, we should keep conservatively adding more workers until constraint (17) is satisfied. In fact, the next W_k that should be added to V^* is the one that satisfies the following inequality:

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

The *Greedy* algorithm described below progressively adds more P_k until V^* satisfies (17), i.e. until ($\theta < 1$). The run time of this search is $O(n)$.

Algorithm 2: Greedy(*V*, thetaCondition**)**

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

4.4.3. Policy III ($\theta = 1$)

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

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

5. UMR2 vs. UMR: Analytical Comparison

In this section we analytically show how UMR2 is always better than UMR through the following lemmas.

LEMMA 2. If the UMR2 and UMR algorithms end up with the same set of selected workers (V*) then $makespan_{UMR2} < makespan_{UMR}$

Proof. If we sort the n workers of V^* by S_i/B_i in an increasing order:

$$S_1 / B_1 < S_2 / B_2 < \ldots < S_n / B_n < 1/n \quad (18)$$

We can write

$$B_n / S_n > n \Rightarrow B_n / (B_n + S_n) > n.S_n / (B_n + S_n) \quad (19)$$

Concurrently, from (18) we derive

$$S_n / (B_n + S_n) > S_i / (B_i + S_i) \quad (\forall i = 1,2,\ldots,n) \quad (20)$$

From (19) and (20) we derive

$$\frac{B_n}{B_n + S_n} > \sum_{i=1}^{n} \frac{S_i}{B_i + S_i} \Rightarrow \theta > 1$$

In the case of $\theta > 1$, $makespan_{UMR2}$ is computed by (14)

$$makespan_{UMR2} = \frac{L_{total} \times B_n}{(B_n + S_n) \sum_{i \in V^*} \frac{B_i S_i}{B_i + S_i}} + C \quad (21)$$

From (6) we derive:

$$makespan_{UMR} = \frac{L_{total}}{\sum_{i \in V^*} S_i} \left(1 + \frac{1 - \phi}{\phi - \phi^{m+1}} \right) + C$$

From (18) we have

$$\sum_{i=1}^{n} \frac{S_i}{B_i} < 1 \Rightarrow \phi > 1 \Rightarrow \lim_{m \to \infty} \frac{1 - \phi}{\phi - \phi^{m+1}} = 0$$

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

$$1 + (1\text{-}\phi) / (\phi - \phi^{m+1}) \approx 1$$

So we have

$$makespan_{UMR} = \left(L_{total} \Big/ \sum_{i \in V^*} S_i \right) + C \quad (22)$$

From (18) we derive

$\Rightarrow B_n/(B_n+S_n) \le B_i/(B_i+S_i) \qquad (\forall i = 1,2,...,n)$

$$\Rightarrow \frac{B_n}{B_n+S_n}\sum_{i=1}^{n} S_i \le \sum_{i=1}^{n}\frac{B_i S_i}{B_i+S_i}$$

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

And by considering (21) and (22) we derive:

$\Rightarrow$ makespan $_{UMR2}$ < makespan $_{UMR}$ □

LEMMA 3. In all cases, $makespan_{UMR2} < makespan_{UMR}$

Proof. If we denote:

- A is the subset which chosen by UMR
- B is the subset which chosen by UMR2
- *V1, V2, V3* is the subsets which chosen by UMR2 in 3 cases: $\theta>1$, $\theta= 1$, $\theta<1$, respectively

Using Lemma 2, we have (23)

$$\text{makespan}_{UMR}(A) > \text{makespan}_{UMR2}(A)$$

As discussed in Section 4.4.1, *V1* is an optimal solution of the Knapsack system produced by the Branch-and-bound algorithm. So we have

$$makespan_{UMR2}(A) \ge makespan_{UMR2}(V1) \quad (24)$$

Because *B* is chosen by UMR2 by comparing V1, V2, V3 so we have

$$makespan_{UMR2}(V1) \ge makespan_{UMR2}(B) \quad (25)$$

From (23) (24) (25) we derive

$$makespan_{UMR}(A) > makespan_{UMR2}(B)$$ □

6. Experimental Results

In order to evaluate UMR2 experimentally, we developed a simulator using the SIMGRID toolkit [6] which has been used to evaluate the original UMR algorithm. To evaluate the UMR2 algorithm, we used the same metrics (Table 4) and the same values of configuration parameters (Table 1) that were used to evaluate UMR. 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 UMR2 algorithm is superior to its predecessor multi-round algorithms, namely LP and UMR.

6.1. 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 as shown

Table 1. 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×S_{min}], where $S_{min} = 50$
Communication rate (flop/s)	Randomly selected from [0.5×N×S_{min} ,1.5×N× S_{min}]

Table 2. The Absolute Deviation between the Experimental and Theoretical

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

in Table 2. This confirms that the approximation assumptions adopted in our analysis are plausible. Table 1 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 (13), (14) and (15) respectively.
- D_i (i=1, 2, 3) is the absolute deviation between the theoretical makespan, MK_i, and the experimental makespan MK_e. Therefore:

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

Table 2 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 makespan 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 proportional 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 (θ >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 (θ <1), which is the most relaxed policy with respect to the number of participating workers. D_1 of policy I (θ =1) falls in the middle with respect to the number of participating workers and according the observed deviation.

6.2. Comparison with Other Algorithms

We compare UMR2 with the most powerful scheduling algorithm, namely UMR [2], [8] and LP [4]. Table 3 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

Table 3. Simulation parameters

Parameters	*Values*
N: Number of workers	10, 12,..,50
Total workload (flop)	5.10^5
CPU speed (flop/s)	Randomly selected from the range [S_{min} ,1.5×S_{min}] S_{min} = 5, 10, 15, 20
Bandwidth (flop/s)	Randomly selected from the range [0.5×NS_{min} ,1.5×N S_{min}]
Latencies (s)	10,1, 10^{-1},10^{-2}

comparing scheduling algorithms [2]. The summarized result in Table 4 shows that UMR2 could outperform its competitors (ranked #1) in most of the cases (98%) with a performance increase of 21.8% over the UMR's. UMR2 was ranked the 2nd in 2% of the cases as it showed 5.4% performance degradation in comparison with the UMR. Fig. 1 helps us understand the 2% of the cases where UMR may outperform our algorithm. As shown, if the number of available workers is small ($N \leq 20$) the performance of UMR2 may fall behind as the lack of workers denies the UMR2 adopting one of the resource selection policies, namely Policy II. This suggests that the UMR2 is better in a WAN environment such as the Grid where thousands of workers are accessible, whereas UMR is more appropriate for LAN settings. 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.

7. Conclusion

The ultimate goal of any scheduling algorithm is to minimize the time needed to process a given workload. UMR and LP have been designed to schedule divisible loads in heterogeneous environments such as the Grid. However, these algorithms suffer from shortcomings that make them less practical for the Grid. For example, these algorithms do not take into account a number of chief parameters such as bandwidths and latencies.

Fig. 1: The effect of *N* on the makespan

Table 4. Performance comparisons among UMR2, UMR and LP Algorithms

Algorithm	Normalized Makespan	Rank	Degradation from the best
UMR2	1	0.02	0.11
UMR	1.21	0.98	21.4
LP	1.59	2	59.8

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 UMR2 algorithm that divides the workload into chunks in light of more realistic parameters mentioned earlier. We explained the UMR2's worker selection policy which is, to the best of our knowledge, the first algorithm that addresses the resource selection problem. Having such policy is indispensable in large commuting platform such as the Grid, where thousands of workers are accessible but the best subset must be chosen. The simulation experiments show that UMR2 is superior to its predecessors especially when it is put into operation in a colossal WAN environment such as the Grid, which agglomerates 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, A. Legrand, & Y. Robert, Scheduling divisible workloads on heterogeneous platform, *Parallel Computing*, 29 (9) 2003, 1121-1152.

[5] D. P. Bertsekas, *Constrained optimization and lagrange multiplier methods* (Belmont, Mass. : Athena Scientific , 1996).

[6] Available at *http://simgrid.gforge.inria.fr*

[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.