A Multi-Round Algorithm for Scheduling Divisible Workload Applications: Analysis and Experimental Evaluation

Yang Yang[1] Henri Casanova [1,2]

[1] Department of Computer Science and Engineering
[2] San Diego Supercomputer Center
University of California at San Diego

Abstract

In this paper we present UMR, an algorithm for scheduling parallel applications that consist of a divisible workload. Our algorithm uses multiple rounds to overlap communication and computation between a master and a number of workers. Multi-round scheduling has been used for divisible workloads in previous work and our contributions in this paper are as follows. UMR uses "uniform" rounds, i.e. a fixed amount of work is sent out to all workers at each round. This restriction makes it possible to compute an approximatively optimal number of rounds, which was not possible for previously proposed algorithms. In addition, we use more realistic platform models than those used in previous works. We provide an analysis of our algorithm both for homogeneous and heterogeneous platforms and present simulation results to quantify the benefits of our approach.

1 Introduction

Applications that consist of many independent computational tasks arise in many fields of science and engineering [1, 2, 3, 4, 5, 6]. These applications often require large amounts of compute resources as scientists wish to tackle increasingly complex problems. Fortunately, advances in commodity technology (CPU, network, RAM) have made clusters of PCs cost-effective parallel computing platforms. In this paper we address the problem of scheduling the aforementioned applications on such platforms with the goal of reducing the execution time, or *makespan*. This problem has been studied for two different application models: *fixed-sized tasks* and *divisible workload* (sometimes referred to as "partitionable"). In the first scenario, the application's workload consists of a number of tasks whose size are pre-determined. In this case, the scheduling problem is a variation of bin-packing and a number of scheduling heuristics have been developed [7, 8, 9, 10, 11]. In the divisible workload scenario, the scheduler can partition the workload in arbitrary tasks, or "chunks". The usual assumption is that the workload is continuous. In practical situations, this often means that the execution time of a base computational unit is orders of magnitudes smaller than the execution time of the entire workload and that all base computational units are the same size. In this paper we focus on the divisible workload scenario, which has been extensively studied [12].

The divisible workload scheduling problem is challenging due to the overhead involved when starting tasks. Overhead is due to: (i) the time to transfer application input/output data to/from a compute resource; (ii) the latency involved in starting a computation. In [12], the divisible workload scheduling problem is identified as: *Given an arbitrarily divisible workload without precedence relations and a multiprocessor/multicomputer system subject to communication delays, in what proportion should the processing workload be partitioned and distributed among the processors so that the entire workload is processed in the shortest possible time?* The trade-off for achieving a good schedule is as follows. On the one hand, dividing the workload into large chunks generally reduces the overhead, and thereby the execution time of the application. On the other hand, dividing the workload into small chunks makes it possible to overlap overhead with useful work more efficiently. In all that follows we consider a traditional *master/worker* paradigm.

Our contributions in this paper are on several fronts. We propose and analyze a new scheduling algorithm: UMR (Uniform Multi-Round). Similarly to previously proposed algorithms, UMR dispatches work to compute resources in multiple rounds. However, we add the restriction that rounds must be "uniform", i.e.

within each round the master dispatches identical chunks to all workers. Due to this restriction, we are able to derive an approximately optimal number of rounds, both for homogeneous and heterogeneous platforms. We evaluate our algorithm with models that are more realistic than those used in previous work. We compare our algorithm with a previously proposed multi-round algorithm and a one-round algorithm. Our simulation results demonstrate the benefits of our approach for wide ranges of scenarios. We also analyze the impact of various system parameters on the behavior and effectiveness of UMR.

This paper is organized as follows. In Section 2 we discuss relevant related work in detail. Section 3 describes our models for the application and the computing platform. Section 4 presents our scheduling algorithm for both homogeneous and heterogeneous platforms. Section 6 gives simulations results and discusses the benefits of our approach. Section 7 concludes the paper and discusses future work.

2 Related Work

There is a large literature on the problem of scheduling fixed-size independent tasks onto a set of processors. A number of heuristics have been proposed and studied for several application and platform models [7, 8, 9, 10, 11]. In this work we focus solely on divisible workloads.

The works in [13, 14, 15, 5, 16] study scenarios in which the workload is divided in as many chunks as processors. Therefore, the entire application is performed in a single round of work allocation. By contrast, our algorithm uses multiple rounds. During each round a portion of the entire workload is partitioned among the available processors. Therefore, our work is most related to the ”multi-installment” algorithm presented in [17]. In both our work and the work in [17] it is assumed that the amount of data to be sent for a chunk is proportional to the chunk size. The key idea is that using many small chunks allows for overlapping of communication and computation. The chunk size can then be gradually increased throughout the application run in order to reduce communication overhead. Our approach differs from [17] in the following way. Whereas [17] allocates decreasing chunks of the workload to processors within a round, we keep the chunk size *fixed within a round*. This has one major benefit: our algorithm is amenable to analysis, which allows us to compute a near-optimal number of rounds, both for homogeneous and heterogeneous platforms. Furthermore, this analysis is possible even though our platform model is more realistic and generic than the one used in [17]. We provide quantitative comparison between our work and [17] in Section 6.1. Other works, such as [6], use multiple rounds to schedule divisible workloads. However, they focus on *steady-state* application performance rather than makespan. Therefore they use identical rounds.

Scheduling divisible workload in multiple rounds has also been studied in [18, 19]. Instead of increasing chunk size throughout application execution, the algorithms in [18, 19] start with large chunks and *decrease* chunk size throughout application execution. Assuming uncertainties on task execution times, reducing the chunk size ensures that the last chunk to complete will not be large, and therefore will not have large (absolute) uncertainty. Therefore, reducing chunk size and using a greedy scheduling algorithm allows for good time-balancing of the execution (i.e. all processors finish computing approximatively at the same time). The work in [18, 19] assumes a fixed network overhead to dispatch chunks of any sizes. In this work, as in [17], we assume that the amount of data to be sent for a chunk is proportional to the chunk size, which is more realistic for most applications. With this assumption, starting by sending a large chunk to the first worker would cause all the remaining workers to be idle during this potentially long initial data transfer. Furthermore, in this paper we do not consider task execution time uncertainties. Consequently, we do not provide a quantitative comparison between our algorithm and the ones in [18, 19].

The work in [15] assumes a network that allows some pipelining of communications in order to model real TCP connections. A number of works model network latencies by adding a fixed overhead to each chunk transfers. Rather than choosing one of those proposed models, we us a generic network model that allows communication pipelining and models several latencies. This model can be instantiated to conform to a variety of network models, including the ones used in [13, 14, 15, 7, 18, 19, 17, 6, 5, 16].

3 Models

3.1 Workload

We consider applications that consist of a workload, W_{total}, that is *continuously divisible*: the scheduler can decide how big a portion, or "chunk", of the workload to give out to a processor. We assume that the amount of application data needed for processing a chunk is *proportional* to the amount of computation for that chunk. As done in most previous work, we only consider transfer of application input data. The works in [15, 16] takes into account output data transfers but uses a single round of work allocation. Similarly, the work in [6] models output but considers only steady-state performance.

3.2 Computing Platform

We assume a *master/worker* model with N worker processes running on N processors. The master sends out chunks to workers over a network. We assume that the master uses its network connection in an sequential fashion: it does not send chunks to workers simultaneously, even though some pipelining of communication can occur [15]. This is a common assumption and is justified either by the master's implementation, or by the properties of the network links (e.g. a LAN). In some cases, for instance on a WAN, it would be beneficial for the master to send data to workers simultaneously in order to achieve better throughput. Section 6.4 presents preliminary results for that scenario. Note that we do not require that the speeds of network communications to each worker be identical. Therefore, the platform topology consists of network links with various characteristics to clusters of heterogeneous processors, as depicted in Figure 1. Finally, we assume that workers can receive data from the network and perform computation simultaneously (as for the "with front-end" model in [12]).

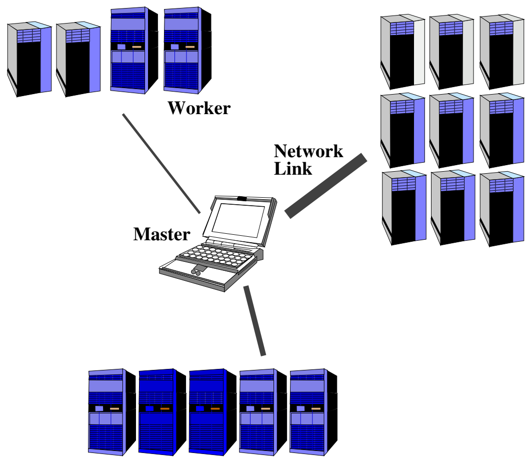

Figure 1: Computing platform model.

Let us formalize our model. Consider a portion of the total workload, $chunk \leq W_{total}$, which is to be processed on worker $i, 1 \leq i \leq N$. We model the time required for worker i to perform the computation, $Tcomp_i$, as

$$Tcomp_i = cLat_i + \frac{chunk_i}{S_i}, \tag{1}$$

where $cLat_i$ is a fixed overhead, in seconds, for starting a computation (e.g. for starting a remote process), and S_i is the computational speed of the worker in units of workload performed per seconds. Computation, including the $cLat_i$ overhead, can be overlapped with communication.

We model the time spent for the master to send $chunk$ units of workload to worker i, $Tcomm_i$, as:

$$Tcomm_i = nLat_i + \frac{chunk_i}{B_i} + tLat_i, \tag{2}$$

where $nLat_i$ is the overhead, in seconds, incurred by the master to initiate a data transfer to worker i (e.g. pre-process application input data and/or initiate a TCP connection); B_i is the data transfer rate to worker i, in units of workload per second; $tLat_i$ is the time interval between when the master finishes pushing data on the network to worker i and the time when worker i receives the last byte of data. We assume that the $nLat_i + chunk/B_i$ portion of the transfer is not overlappable with other data transfer. However, $tLat_i$ is overlappable (to model pipelined networking as in [15]). This model is depicted on Figure 2 for data transfers from the master to 3 workers.

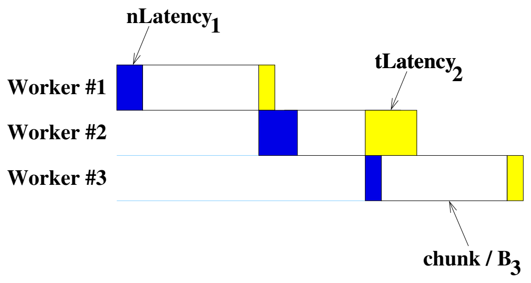

Figure 2: Illustration of the network communication model for 3 identical chunks sent to 3 workers with different values of $nLat_i$, B_i, and $tLat_i$.

This model is flexible enough that it can be instantiated to model several types of network connections. For instance, setting the $nLat$ values to 0 models a pipelined network such as the one used in [15]. In that case, the $tLat$ values represent the network latency between the master and the workers. On Figure 2 we show a large $tLat$ value to worker 2, indicating a long-distance network, and a short transfer time, indicating a large available bandwidth. The model can also be instantiated with non-zero $nLat$ values and zero $tLat$ values as in [9]. This is representative of distinct connections being established for each individual transfer, with no pipelining. Zero $nLat$ and zero $tLat$ corresponds to the work in [17]. To the best of our knowledge, no other work models computation latency, $cLat$. Based on our experience with actual software [20], we deem $cLat$ to be fundamental for realistic modeling. We provide an analysis of our scheduling algorithm using this generic platform model, and thereby validate our approach for various platforms.

4 The UMR Algorithm

4.1 Basic Principle

Similar to the algorithm presented in [17], the UMR algorithm dispatches chunks of the workload in rounds. The chunk size is increased between rounds in order to reduce the overhead of starting computation ($cLat$) and communication ($nLat$). Unlike [17], we keep rounds the chunk size is fixed within each round. We are able to compute near-optimal number of rounds, and a near-optimal chunk size at each round, as demonstrated in the following analysis. In all that follows, M denotes the number of rounds used by UMR.

4.2 Analysis

4.2.1 Homogeneous Platform

We first describe and analyze the UMR algorithm for a platform that consists of N *identical* workers accessible via a network link. Therefore we set:

$$\forall i = 1, .., N \qquad \begin{aligned} S_i &= S, \\ nLat_i &= nLat, \\ tLat_i &= tLat, \\ B_i &= B. \end{aligned} \tag{3}$$

Induction relation for chunk sizes – Let $chunk_j$, for $j = 0, .., M - 1$, be the chunk size at each round. We illustrate the operation of UMR in Figure 3. At time T_A, the master starts dispatching chunks of size $chunk_{j+1}$ for round $(j+1)$. The workers perform computations of sizes $chunk_j$ for round j concurrently. To maximize bandwidth utilization, the master must finish sending work for round $(j+1)$ to all workers before worker N finished its computation for round j, which is shown at time T_B. Therefore, perfect bandwidth utilization is achieved when:

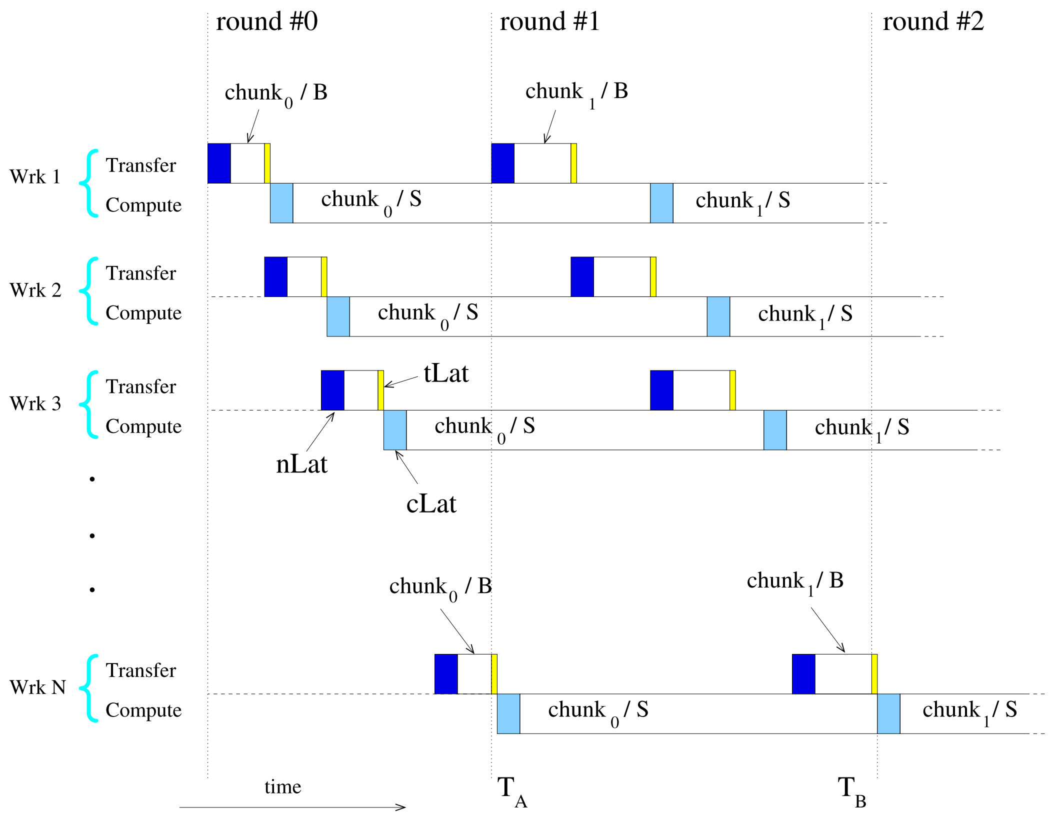

Figure 3: UMR dispatches the workload in rounds, where the chunk size if fixed within a round, and increases between rounds.

$$tLat + cLat + \frac{chunk_j}{S} = N(\frac{chunk_{j+1}}{B} + nLat) + tLat. \qquad (4)$$

The left term is the time worker N spends receiving the last bytes of data, initiating a computation, and computing a chunk during round j. The right term is the time it takes for the master to send data to all N workers during round $j+1$. Eq. 4 defines a simple induction for $chunk_j$, and one can then compute:

$$\begin{aligned} &\forall j \;\; chunk_{j+1} - \alpha = \frac{B}{NS}(chunk_j - \alpha) \\ \Rightarrow \;\; &\forall j \;\; chunk_j = (\frac{B}{NS})^j (chunk_0 - \alpha) + \alpha, \end{aligned} \tag{5}$$

where

$$\alpha = \frac{BS}{B - NS}(N \times nLat - cLat). \tag{6}$$

We have thus obtained a geometric series of chunk sizes, where $chunk_0$ is an unknown.

Necessary conditions for full platform utilization – Let us determine the conditions under which all N workers can be utilized. To utilize all workers, the master must be able to send out all work for round j and work for round $j+1$ to worker 1 before this work becomes idle. This can be written formally as:

$$\begin{aligned} \forall j \quad & \left[N(nLat + \tfrac{chunk_j}{B}) + nLat\right] + \left[\tfrac{chunk_{j+1}}{B} + tLat\right] \leq \\ & \left[nLat + \tfrac{chunk_j}{B} + tLat\right] + \left[cLat + \tfrac{chunk_j}{S}\right], \end{aligned} \tag{7}$$

where the left side is the time needed by the master to send all data for round j and data to worker 1 for round $j+1$, and the right side is the time worker 1 spends receiving data and computing for round j. Replacing $chunk_{j+1}$ by its expression given in Eq. 5, we obtain:

$$\forall j \quad (NS - B)chunk_j \leq (NS - B)\alpha,$$

which, using Eq. 4, is equivalent to:

$$(NS - B)chunk_0 \leq (NS - B)\alpha. \tag{8}$$

If this constraint it not met, then at least one worker can not be used and N should be reduced. We will see in what follows how $chunk_0$ is computed by our algorithm. We give here two necessary conditions for Eq. 8, which are analogous to the "$m\sigma \leq 1$" constraint in [17].

1. If $NS - B > 0$, then Eq. 8 reduces to $chunk_0 < \alpha$. Since $chunk_0$ must be positive, a necessary condition for Eq. 8 is:
$$\alpha > 0. \tag{9}$$
2. If $NB - B < 0$, then Eq. 8 reduces to $chunk_0 > \alpha$. For all workers to be used, $chunk_0$ must be smaller than W_{total}/N. Therefore, a necessary condition is for Eq. 8 is:
$$\alpha < W_{total}/N. \tag{10}$$

Constrained minimization problem – The objective of our algorithm is to minimize $Ex(M, chunk_0)$, the makespan of the application:

$$Ex(M, chunk_0) = \frac{W_{total}}{N} + M \times cLat + \frac{1}{2} \times N(nLat + \frac{chunk_0}{B}) + tLat.$$

The first term is the time for worker N to perform its computation. The second term is the overhead incurred at each round to initiate a computation. The third term correspond to the time for the master to send all the data for round 0. The $\frac{1}{2}$ factor is due to an optimization that is described in detail in Section 5. Finally, the fourth term, $tLat$, can be seen on Figure 3 just after time T_A for worker N.

We also have the constraint that the amount of work sent out by the master during the execution sums up to the entire workload:

$$G(M, chunk_0) = \sum_{j=0}^{M-1} N \times chunk_j - W_{total} = 0.$$

This constrained minimization problem, with M and $chunk_0$ as unknowns, can be solved by using the Lagrange Multiplier method [21]. The multiplier, $L(chunk_0, M, \lambda)$, is defined as:

$$L(chunk_0, M, \lambda) = Ex(M, chunk_0) + \lambda \times G(M, chunk_0),$$

and we must solve:

$$\begin{cases} \frac{\partial L}{\partial \lambda} = G = 0 \\ \frac{\partial L}{\partial M} = \frac{\partial Ex}{\partial M} + \lambda \times \frac{\partial G}{\partial M} = 0 \\ \frac{\partial L}{\partial chunk_0} = \frac{\partial Ex}{\partial chunk_0} + \lambda \times \frac{\partial G}{\partial chunk_0} = 0. \end{cases} \tag{11}$$

This system reduces to the following equation for M:

$$N\alpha - \frac{W_{total} - NM\alpha}{1-(\frac{B}{NS})^M}(\frac{B}{NS})^M \ln(\frac{B}{NS}) - 2cLat \times B\frac{1-(\frac{B}{NS})^M}{1-\frac{B}{NS}} = 0 \quad (12)$$

This equation can be solved numerically by bisection. The solve is fast (on the order of 0.07 seconds on a 400MHz PIII) and can thus be implemented in a runtime scheduler with negligible overhead.

Once we have computed M^*, the solution to the equation above, $chunk_0$, follows as:

$$chunk_0 = \frac{(1-\frac{B}{NS})(W_{total} - NM^*\alpha)}{N \times (1-(\frac{B}{NS})^{M^*})} + \alpha, \quad (13)$$

and $chunk_j (j > 0)$ can be computed with Eq. 5. Complete details on these derivations are provided in Appendix A.

4.2.2 Heterogeneous Platform

In this section, we remove the simplifying assumptions in Eq. 3. Let $chunk_{ji}$ be the chunk sent to worker i in round j. In the homogeneous case, we fixed the *size* of chunks for an round. Here, we fix the *time* it takes for each worker to perform computation during a round. For round j, we define that time as $const_j$ and we have:

$$cLat_i + \frac{chunk_{ji}}{S_i} = const_j. \quad (14)$$

In addition, we denote by $round_j$ the fraction of the workload dispatched during round j, so that:

$$\sum_{i=1}^{N} chunk_{ji} = round_j.$$

By combining these last two equations, one easily obtains:

$$chunk_{ji} = \alpha_i \times round_j + \beta_i, \quad (15)$$

where

$$\begin{aligned} \alpha_i &= \frac{S_i}{\sum_{k=1}^{N} S_k}, \\ \beta_i &= \frac{S_i}{\sum_{k=1}^{N} S_k} \sum_{k=1}^{N} (S_k \times cLat_k) - S_i \times cLat_i. \end{aligned}$$

Induction relation for chunk sizes – Similarly to Eq. 4 for the homogeneous case, one can write:

$$\sum_{i=1}^{N}\left(\frac{chunk_{j+1,i}}{B_i} + nLat_i\right) + tLat_{N} = tLat_{N} + const_j.$$

The left side is the time worker N spends computing during $round_j$. The right side is the time it takes for the master to send data to all N workers during round $(j+1)$. Replacing $const_j$ by its expression, and using Eq. 15, we can compute an induction relation on $round_j$:

$$round_{j+1} = \frac{round_j}{\sum_{i=1}^{N}\frac{S_i}{B_i}} + \frac{\frac{\sum_{i=1}^{N}(S_i \times cLat_i)}{\sum_{i=1}^{N}S_i} - \sum_{i=1}^{N}\left(\frac{\beta_i}{B_i} + nLat_i\right)}{\sum_{i=1}^{N}\frac{\alpha_i}{B_i}},$$

which can be reduced to:

$$round_j = \theta^j \times (round_0 - \eta) + \eta, \tag{16}$$

where

$$\begin{aligned}
\theta &= \left(\sum_{i=1}^{N}\frac{S_i}{B_i}\right)^{-1}, \\
\eta &= \frac{\sum_{i=1}^{N}(S_i \times cLat_i) - \sum_{i=1}^{N}S_i \times \sum_{i=1}^{N}\left(\frac{\beta_i}{B_i} + nLat_i\right)}{\sum_{i=1}^{N}\frac{S_i}{B_i} - 1}.
\end{aligned}$$

Necessary conditions for full platform utilization – As in the homogeneous case, it is possible to derive necessary conditions for all workers to be utilized. All details are presented in Appendix B. We just summarize the two constraint here:

1. If $\gamma > 1$, then a necessary condition for full platform utilization is $\eta > 0$,
2. If $\gamma < 1$, then a necessary condition for full platform utilization is $\eta < W_{total}$,

where γ is defined in Appendix B. An important difference with the homogeneous case is that a resource selection procedure is required to reduce the number of workers (when the platform is homogeneous, one can just reduce the value of N). We address this issue with a simple and effective resource selection scheme which is presented in Section 6.3.

Constrained minimization problem – Similarly to the homogeneous case, we define a constrained optimization problem that can be solved with the Lagrange Multiplier method. The corresponding Ex and G functions are as follows:

$$Ex(M, round_0) = \sum_{j=0}^{M-1} const_j + \frac{1}{2} \times \sum_{i=1}^{N} \left(\frac{chunk_{0i}}{B_i} + nLat_i \right) + tLat_{_N}, \quad (17)$$

$$G(M, round_0) = M \times \eta + \frac{round_0 - \eta}{1 - \theta} \times (1 - \theta^M) - W_{total}.$$

The unknowns are M and $round_0$, and the solution is derived in Appendix C.

5 Practical Implementation of UMR

Before presenting experimental results, we give here a few details about our implementation of the UMR algorithm.

Bounding M^* values for bisection – We need a minimum and maximum value for M^* when using bisection to solve Eq. 12 or Eq. 22. Eq. 23 in Appendix C shows that the maximum value of M^* is $\frac{W_{total}}{\eta}$ in order to have an increasing sequence of rounds. This is for the most general, heterogeneous case and the definition of η is given in Section 4.2.2. This upper bound is very large (resp. infinite) when $(cLat{+}nLat)$ is small (resp. zero). Although this is rare in practical situations, we introduce an artificial bound to handle these cases. In cases where $(cLat + nLat)$ is small or zero, there is little relative makespan improvement to be gained by using extremely large number of rounds and we limit M^* to 50. Therefore, we constrain M^* to be between 1 and $min(50, \frac{W_{total}}{\eta})$ for the bisection.

Rounding M^* to an integer – The Lagrange method produces a real value for M^*. Say we were to use this value to calculate the series of chunk sizes. We would then have $\lceil M^* \rceil$ rounds. The last round would consist in dispatching a potentially small amount of work equal to $chunk_{\lceil M^* \rceil}$, while still incurring a full $cLat$ overhead. Instead, we use the value $\lfloor M^* + \frac{1}{2} \rfloor$ as the number of rounds. This value is no more guaranteed to lead to an optimal schedule than $\lceil M^* \rceil$. However, based on our experiments, it works very well in practice.

Last round optimization – The work in [17] shows that, in an optimal divisible workload schedule, all workers finish computing at the same time. In the UMR algorithm, as it described in Section 4, the finishing time of all N workers has the same "slope" as the starting of the compute times in the first round (as seen in Figure 3). When communications are relatively slow, i.e. when B/S is low, worker 1 finishes computation much earlier than worker N, leading to idle time. To alleviate this limitation, we modify the implementation of UMR for the last round. The main idea is to give a decreasing amount of work to workers during the last round in order to have them all finish at the same time (note that this is similar to what is done in [17] and is different from the uniform round approach we use for all other rounds). The straightforward computation of the modified chunk sizes for the last round is presented in Appendix D.

The three technical modifications described above do not change the main principles of the UMR algorithm, are easy to implement, and improve performance in practice. All the results presented hereafter use the modified version of UMR.

6 Simulation Results

In order to evaluate our approach, we developed a simulator with the Simgrid [22, 23] toolkit. We used the simulator for three sets of experiments. First, we compared UMR to previously proposed algorithms: the multi-round algorithm in [17] and the one-round algorithm in [15]. Second, we study the impact of system parameters (S, B, $cLat$, and $nLat$) on UMR's choice for the optimal number of rounds. Third, we evaluate UMR's robustness to heterogeneity. Fourth, we present a preliminary study of the effects of bandwidth sharing on UMR.

6.1 Comparison with Previous Algorithms

Using our simulator, we compared UMR with the multi-installment algorithm proposed in [17], heretofore referred to as MI. Since a closed form solution for MI is not available on heterogeneous platforms, we only present results for homogeneous platforms in this section. Furthermore, unlike UMR, the MI algorithm does no compute an optimal number of rounds. In fact, since the work in [17] does not model latencies, it would seem that the best scheduling strategy is to use as many rounds as possible. Of course, the authors state that in a practical scenario

Parameter	Values
Number of processors	$N = 10, 15, 20, \ldots, 50$
Workload (unit)	$W_{total} = 1000$
Compute rate (unit/s)	$S = 1$
Transfer rate (unit/s)	$B = 1.1 \times N, 1.1 \times N + 1, \ldots, 5.0 \times N$
Computation latency (s)	$cLat = 0.00, 0.03, \ldots, 0.99$
Communication latency (s)	$nLat = 0.00, 0.03, \ldots, 0.99$

Table 1: Parameter values for the experiments presented in Section 6.1.

it would not be beneficial to use large numbers of rounds (due to latencies). Consequently, we present results for the MI algorithm with 1 to 8 installments. We denote each version my MI-x with $x = 1, \ldots, 8$. We also compare UMR against a simplified version of the one-round algorithm in [15], which we denote as One-Batch. We model only transfer of input data to the workers. However, our version of One-Batch takes into account all the latencies in our model.

We performed experiments for wide ranges of values for parameters defining the platform and the application. We first present aggregate results averaged over large numbers of experiments. We then present results for sub-sets of the results to illuminate the behaviors of the different scheduling algorithms.

6.1.1 Aggregate Results

We evaluated UMR, MI-x, and One-Batch for the parameter values in Table 6.1.1. Note that we chose $S = 1$ to limit the number of parameters. In these conditions, the computation/communication ratio for all workers is exactly equal to the numerical value of B. Since the effect of $tLat$ is just to shift the running time by $tLat$, we use $tLat = 0$ in these experiments. Finally, note that we choose values of B that make it possible to use all workers given the constraints developed in Section 4.2.1.

For each instantiation of these parameters we did the following. We simulated all 10 scheduling algorithms, and computed three metrics for each algorithm: (i) its *makespan* normalized to that achieved by UMR in this experiment; (ii) its *rank* which goes from 0 (best) to 10 (worst); (iii) its *degradation from best* which measures the percent relative difference between the makespan achieved by this algorithm and the makespan achieved by the best algorithm for this experiment. These three metrics are commonly used in the literature for comparing scheduling

Alg.	norm.	rank	deg.	Percentage Wins								
	makespan		from best	MI-1	MI-2	MI-3	MI-4	MI-5	MI-6	MI-7	MI-8	One-Batch
UMR	1.00	0.09	0.09	100.0	97.6	97.1	98.4	99.1	99.4	99.6	99.7	100.0
MI-1	1.21	2.75	21.50		77.2	84.8	88.7	91.1	92.7	93.9	94.7	1.6
MI-2	1.48	2.73	48.33			94.8	96.7	97.6	98.2	98.5	98.8	17.3
MI-3	1.84	3.68	84.48				98.5	99.0	99.2	99.4	99.5	12.9
MI-4	2.22	4.74	122.09					99.4	99.5	99.6	99.7	10.0
MI-5	2.60	5.79	160.04						99.7	99.7	99.8	8.1
MI-6	2.98	6.83	198.11							99.8	99.8	6.7
MI-7	3.36	7.85	236.22								99.9	5.7
MI-8	3.74	8.87	274.35									4.9
One-Batch	1.04	1.67	4.11									

Table 2: Aggregate comparison of MI-x, One-Batch, and UMR. Parameters values from Table 6.1.1. For a total of 1,229,984 experiments.

Alg.	norm.	rank	deg.	Percentage Wins								
	makespan		from best	MI-1	MI-2	MI-3	MI-4	MI-5	MI-6	MI-7	MI-8	One-Batch
UMR	1.00	0.58	0.29	99.9	86.9	80.0	88.8	94.0	96.4	97.6	98.3	99.9
MI-1	1.16	5.76	15.95		11.9	27.8	39.9	48.9	55.6	60.9	65.3	13.7
MI-2	1.06	2.25	6.58			67.5	77.8	83.4	86.8	89.2	90.8	77.5
MI-3	1.09	2.42	9.61				89.6	92.7	94.3	95.4	96.1	65.4
MI-4	1.14	3.53	14.62					95.6	96.5	97.2	97.6	55.8
MI-5	1.20	4.72	20.08						97.6	98.0	98.3	48.5
MI-6	1.25	5.87	25.70							98.5	98.8	42.6
MI-7	1.31	7.00	31.38								99.1	37.5
MI-8	1.37	8.11	37.09									33.6
One-Batch	1.13	4.75	13.52									

Table 3: Aggregate comparison of MI-xx, One-Batch, and UMR. Parameters values for a subset of the values in Table 6.1.1: $cLat < 0.1$, $nLat < 0.1$. For a total of 144,704 experiments.

algorithms. We present averages of these 3 metrics for each algorithm over all parameter configurations in Table 2.

We also compute a metric for pair-wise comparisons: *percentage wins*. For each pair of algorithms we compute the fraction of the experiments for which one algorithm outperforms the other (i.e. leads to a shorter makespan). This is shown on the right-hand side of Table 2: The numbers are the percentage of the experiments for which the algorithm in the rows outperforms the one in the columns. For instance, in Table 2, MI-2 outperforms MI-5 for 97.6% of the experiments over the entire parameter space. The main observation from Table 2 is that UMR outperforms competing algorithms in most cases. We also see that the One-Batch strategy outperforms the MI-x algorithm in the majority of the cases. On average it leads to schedules 4% longer than UMR.

Over all instantiations of system parameters UMR is not the best algorithm

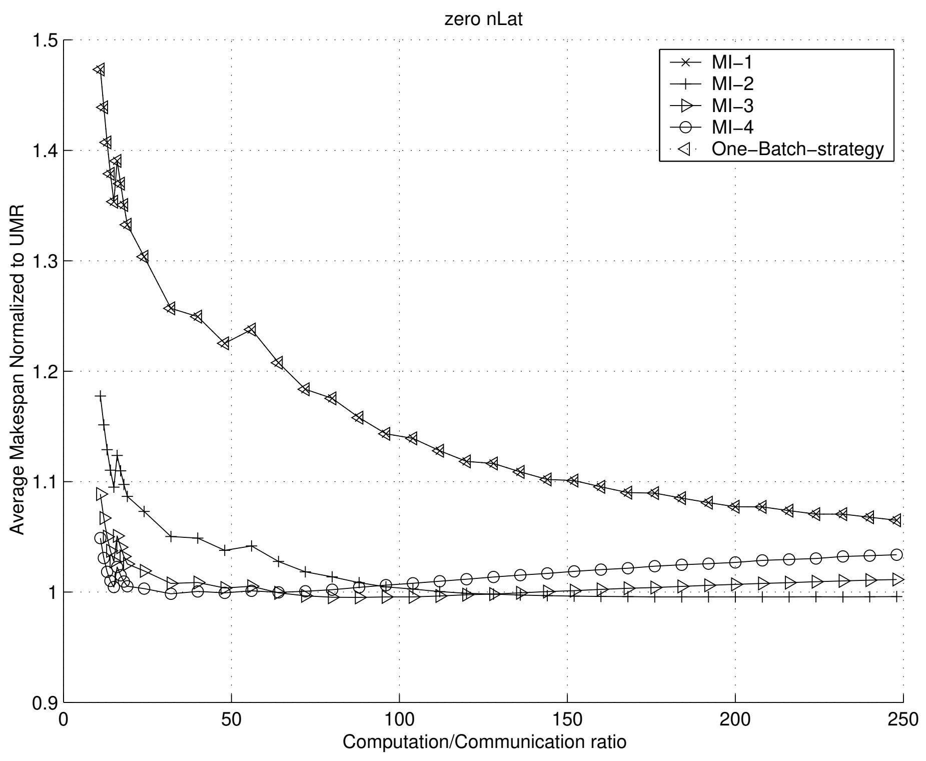

Figure 4: Average makespan of MI-x and One-Batch relative to UMR vs. computation/communication ratio, $W_{total} = 1000\ nLat = 0$.

in only 4.46% of the cases. When UMR is outperformed, it is on average within 2.04% of competing algorithms with a standard deviation of 0.035.

MI-x does not take into account latencies, which explains why its performance is rather poor in our experiments. Consequently, we show results for a subset of the parameter space in Table 3 (which is similar to Table 2). In this table we limit $nLat$ and $cLat$ to be below 0.1 seconds. We see that MI-2 and MI-3 perform better on average, but UMR still leads to better performance in 80% of the time. Note that the One-Batch algorithm performs relatively worse because it cannot overlap communication and computation, which is more critical when latencies are low.

6.1.2 Specific Case Studies

In order to provide more insight, we project the results for the parameter space on the B axis. More specifically, for each value of B we compute the makespan of MI-x and One-Batch normalized to that achieved by UMR, for specific and extreme values of $cLat$ and $nLat$. Note that B is really the computation/communication ration of the system, which is key to determining a good schedule. Finally, we plot MI-x results only for MI-1, MI-2, MI-3, and MI-4 as trends are identical for $x \geq 3$.

No network latency – Figure 4 plots average normalized makespans versus the computation/communication ratio for $nLat = 0$. We can see that when the communication rate is slow (left end of the x-axis), using multiple rounds with MI-x is beneficial as it leads to better overlap. At the other extreme, using smaller values of x is better as the computation latency become predominant. The One-Batch strategy improves with faster communications as overlap of communication and computation becomes less critical. We see that UMR is very close to the best MI-x across the board. This is an important result because it means that our algorithm performs well and consistently for wide ranges of communication/computation ratios.

No computation latency – Figure 5 is similar to Figure 4 but is for $cLat = 0$. As expected, the performance of MI-x degrades because the algorithm does not take network latency into account. This latency becomes more significant when the computation/communication ratio increases. Also, the performance degradation increases with x as more rounds mean more overhead. The One-Batch strategy performs almost as well as UMR in this case because it uses takes into account the network latency to compute chunk sizes. The "step" effects in Figure 5 are due to the rounding of M^* to an integer value (see Section 5).

Small latencies – Figure 6 is for values of $nLat$ and $cLat$ that are lower than 0.1. For larger values we have seen in Table 2 that the MI-x algorithm is heavily outperformed by UMR and One-Batch. The One-Batch strategy gets relatively good performance only for large values of the computation/communication ratio (because in this case overlap of computation and communication with multiple rounds is not critical). MI-1 has a similar behavior, but is not as good as One-Batch because it does not take latencies into account. We can see that no MI-x algorithm is effective across the board relatively to UMR.

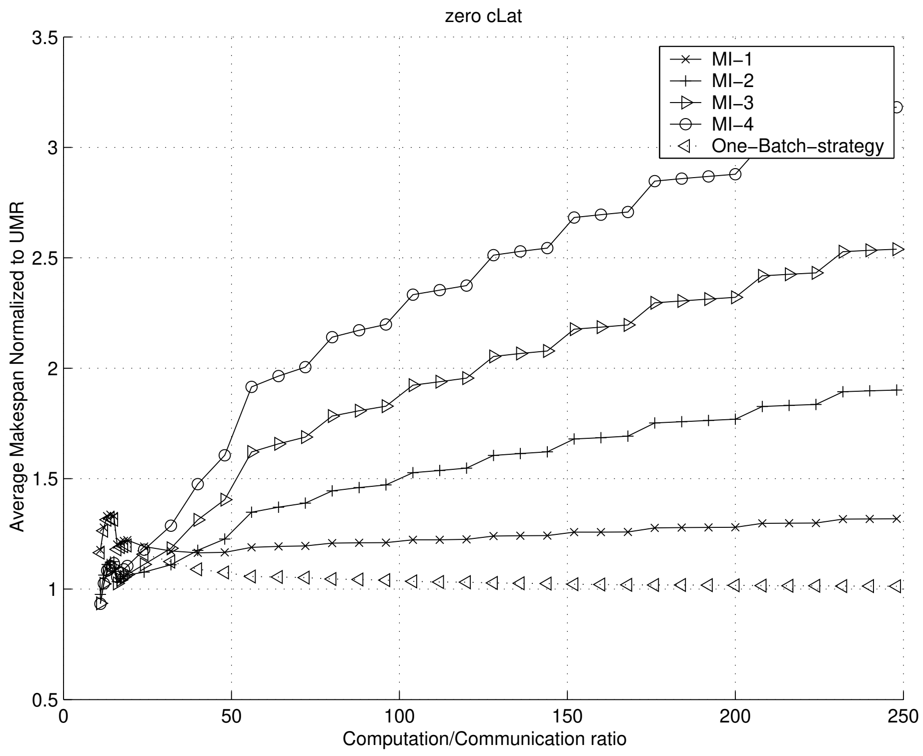

Figure 5: Makespan of MI-x and One-Batch relative to UMR, vs. Computation/Communication ratio,$W_{total} = 1000 \;\; cLat = 0$.

No latencies – Figure 7 shows the Average Makespan for $nLat = 0$ and $cLat = 0$. In such cases the more rounds the better, and M^* should be infinite. Instead, UMR uses an upper limit of 50 rounds (see Section 5). Therefore, UMR outperforms than MI-x since we limit x to be lower than 8. For a more meaningful comparison, we arbitrarily force UMR to use the same number of rounds as MI-x. The results are plotted in in Figure 8. As expected, MI-x outperforms UMR because it is not restricted to using uniform rounds and can therefore achieve better overlap between computation and communication. We note that, on average, UMR is only within 2.1% of MI-x. In these experiments, One-Batch is identical to MI-1.

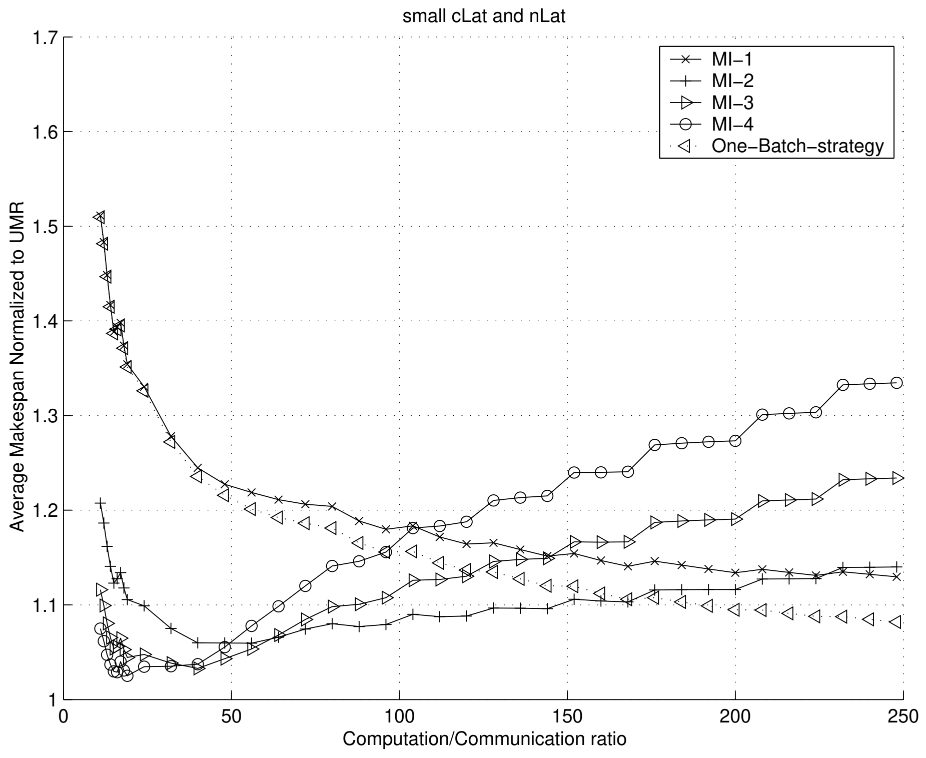

Figure 6: Average makespan of MI-x and One-Batch relative to UMR vs. the computation/communication ratio,$W_{total} = 1000\ nLat < 0.1$, $cLat < 0.1$.

6.1.3 Summary

The conclusions from our result are:

1. UMR leads to better schedules than MI-x and One-Batch in an overwhelming majority of the cases (>95%),

2. Even when UMR is outperformed, it is close to the competing algorithms (on average within 2.04% with a standard deviation of 0.035) .

3. Neither MI-x nor One-Batch ever outperform UMR "across the board" (i.e. for a wide range of computation/communication ratios),

UMR is able to achieve such improvement over previous work in spite of the "uniform" round restriction, an precisely because this restriction makes it possible

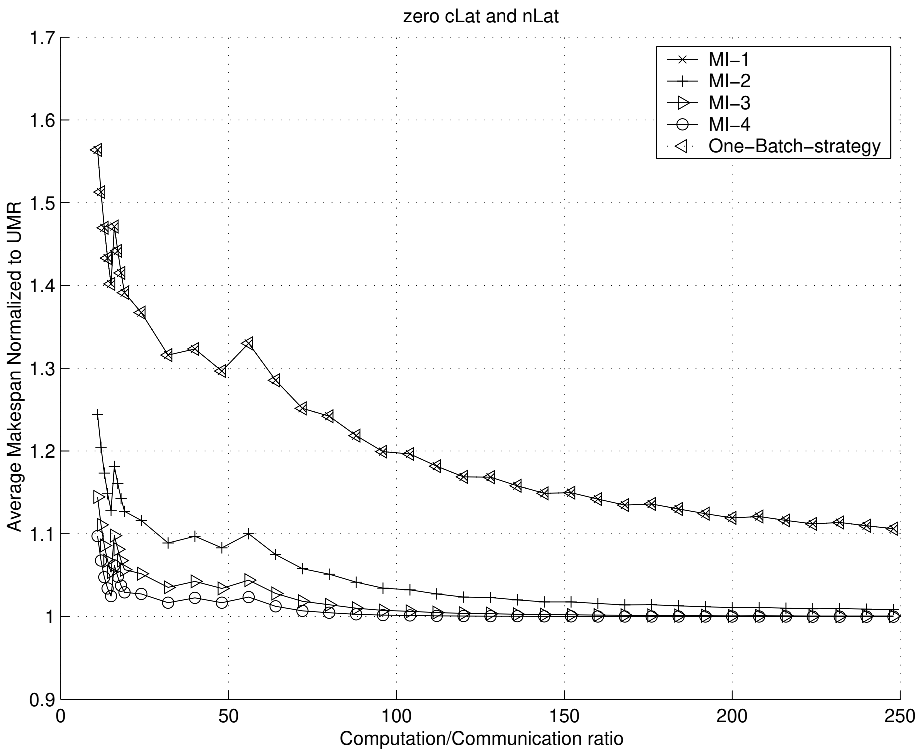

Figure 7: Average makespan of MI-x and One-Batch relative to UMR vs. the computation/communication ratio,$W_{total} = 1000$ $nLat = cLat = 0$.

to compute an optimal number of rounds. In the next section we study how the choice of this number, M^*, varies with system parameters.

6.2 Impact of System Parameters on M^*

In this section, we present experimental results that demonstrate how characteristics of the platform and the application impact the behavior of UMR.

6.2.1 Impact of Latencies on M^*

Figure 9 plots the M^* value chosen by UMR versus both $cLat$ and $nLat$ when they vary between 0 and 0.2. The other parameters are fixed and set to $N = 10$, $W_{total} = 1000$, and $B = 17$. The computing platform is homogeneous. These

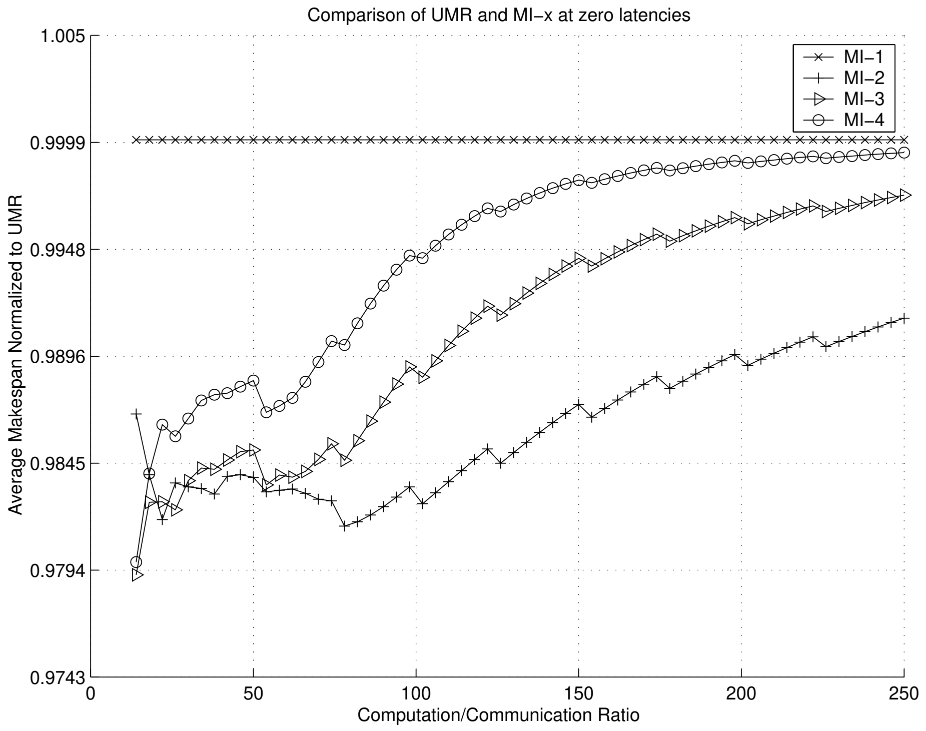

Figure 8: Average makespan of MI-x and One-Batch normalized to that of UMR vs. the computation/communication ratio,we force UMR use x round as MI-x. $W_{total} = 1000\, nLat = 0$, $cLat = 0$.

results are representative of what we observed for other values of these last three parameters. For these experiments the makespan increases with both latencies and varies from 100.5 seconds to 104.5 seconds. Figure 9 demonstrates that UMR chooses different values of M^* for different scenarios, in this case between 7 and 13 rounds. M^* decreases when either latency increases, which is expected as fewer rounds lead to less overhead.

An interesting phenomenon is that $cLat$ has a bigger impact on M^* than $nLat$. In other words, UMR reacts more significantly to increases in $cLat$ by decreasing the number of rounds aggressively. This is due to a fundamental difference in the two latencies, as seen in Figure 3. The effect of the network latency, $nLat$, is to modify the “slope” of the initial round, that is the respective times at which workers start computing. After the first round, the network latency is hidden as it is overlapped with computation. By contrast, the computation latency, $cLat$, is

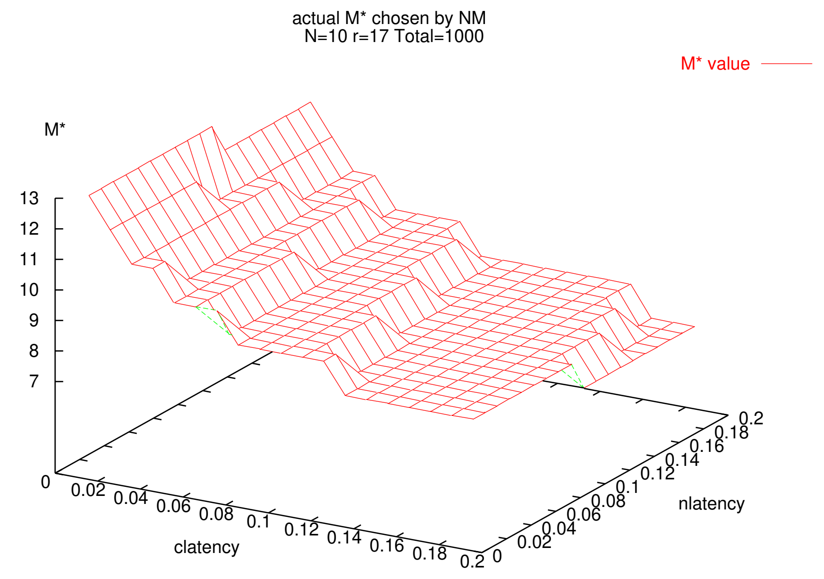

Figure 9: M^* versus $nLat$ and $cLat$.

never hidden and is experienced at each round. In summary, large $nLat$ leads to more idle time in the **first** round, and large $cLat$ leads to longer compute times at **each** round.

The effect of $cLat$ on M^* is then clear and Figure 9 shows that UMR reduces M^* to reduce the overall $cLat$ cost. However, one may wonder why UMR also reduces M^* when $nLat$ increases. After all, its effect is only on the idle time experienced in the first round (and also in the last round if the "last round optimization" described in Section 5 is not used). The reason is that UMR reduced the number of rounds not to reduce overhead, but to allow overlap of computation and communication. As $nLat$ increases, larger chunks need to be sent during the first round so that the first worker finishes computing after the communication to the last worker has completed. This is seen more formally in Eq. 8 since α increases with $nLat$. Therefore, since $chunk_0$ is large, and the series of chunk sizes is increasing, fewer numbers of rounds are necessary to dispatch the entire workload. Hence the moderate decrease in M^* when $nLat$ increases.

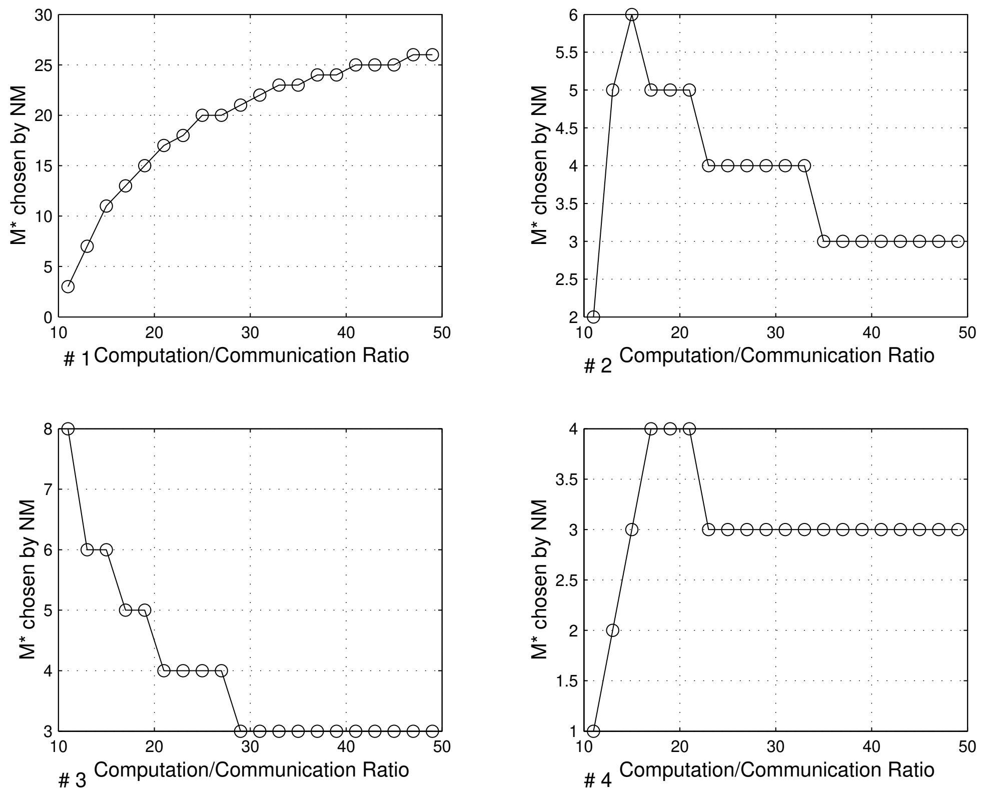

Figure 10: Effect of the computation/communication ratio (B) on M^*. $W_{total} = 1000$, $N = 10$. For subplots #1, #2, #3, and #4, $(cLat, nLat)$ are respectively: $(0.00, 0.30)$, $(0.49, 0.49)$, $(0.97, 0.01)$, $(0.97, 0.97)$.

6.2.2 Impact of the Computation/Communication Ratio on M^*

Figure 10 plots the effect of the computation/communication ratio (equal to B in our experiments) on $M*$, given different 4 different values of $(cLat, nLat)$. Intuitively, two different mechanisms are are play here and are seen on Subplots #3 and #1:

1. Subplot #3: When $cLat$ is significant, UMR tries to use as few rounds as possible, while not spending too much time for the transfer in the first round. As B increases, the penalty for the first round transfer drops, and UMR can use bigger chunks and fewer rounds. This explains the decrease in M^*.

2. Subplot #1: When $cLat$ is insignificant, UMR tries to use a as many rounds as possible. However, this is limited by the constraint in Eq.8: $chunk_0$ must be higher than α. With these values of $(cLat, nLat)$, it is easy to see that α is large when B is small (see Eq. 6) and that α decreases with B. Therefore, as B increases, UMR uses smaller $chunk_0$ values. This explains the increase in M^*.

Subplots #2 and #4 in Figure 10 show the combination of these two effects with first increasing and then decreasing number of rounds as the computation/communication ratio increases.

6.2.3 Summary

The results in this section show that UMR chooses different values of M^* in different situations in order to reduce the overall makespan. We have explained in detail how $cLat$, $nLat$, and the computation/communication ratio impact these choices. The conclusion is that UMR balances communication and computation overlap with overhead to choose the most effective value for M^*.

6.3 Impact of Heterogeneity on Makespan

All the results we have presented so far were for homogeneous platforms. Although the derivation of the UMR algorithm in the heterogeneous case is substantially more complex, the same general trends apply. Nevertheless, we wish to demonstrate that UMR adequately handles heterogeneous platforms. Therefore we present results for the following experiment. We simulated UMR on a platform consisting of 10 processors with random S_i, $cLat_i$, $nLat_i$ and B_i values sampled from a uniform distributions on the interval $((1 - \frac{het-1}{1+het})mean, (1 + \frac{het-1}{1+het})mean)$, where the $mean$'s are : $S = 1$, $cLat = 1$, $nlat = 0.1$, $B = 20$. The parameter het denotes that processor/link characteristics can differ by as much as a factor het.

Figure 11 plots the normalized makespan achieved vs. UMR versus het. The normalized makespan is computed as the ratio of the makespan versus the "ideal" makespan which is achieved when all communication costs are zero: $W_{total}/\sum S_i$. Every data point in the figure is obtained as an average over 100 samples.

Looking at the normalized makespan, one can see that UMR is robust and handles heterogeneous platforms well. For extreme cases in which processor or link performances differ by a factor up to 1000, UMR still managed to achieve a makespan which is within 20% of the ideal, impossible to achieve, makespan.

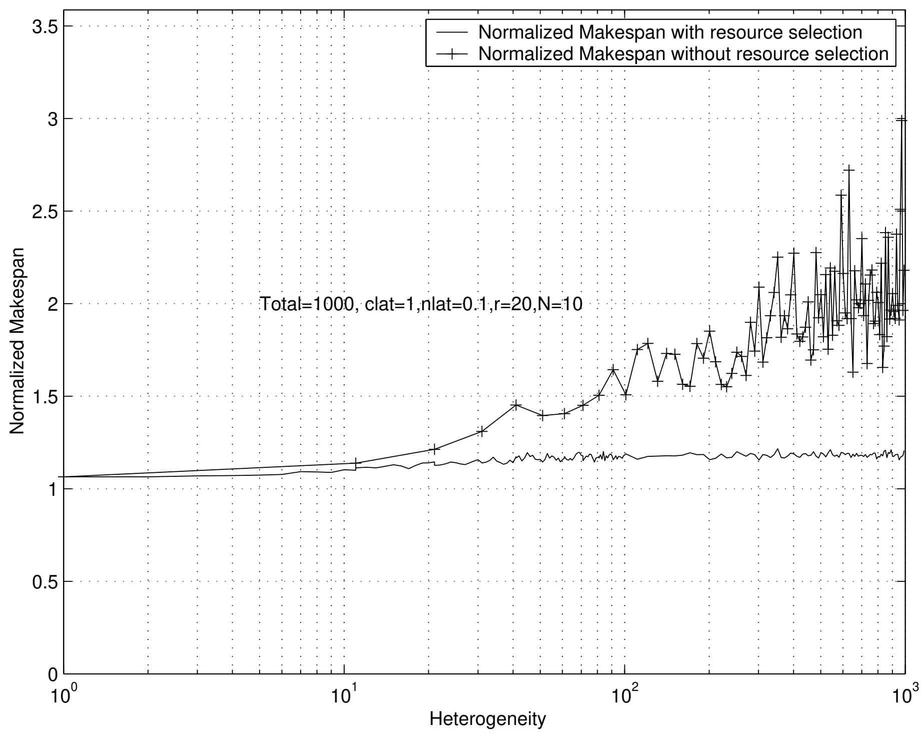

Figure 11: Normalized makespan versus het, with and without resource selection.

In this experiments we had to perform resource selection. Indeed, when generating random values for the system parameters, the conditions given in section 4.2.2 are not satisfied. We must then use fewer resources than available. The idea is that there may be very slow links connecting the master to very fast processors. For such a processor, the data transfer to that processor during a round completes after other processors have finished computing for the same round, which is detrimental to performance. Our resource selection criteria is inspired by an approximate version of the constraints given in Section 4.2.2 and derived in Appendix B. We sort workers according to $\frac{S_i}{B_i}$ in increasing order. We then select N' processors our of the original N such that:

$$\sum_{k=0}^{N'} \frac{S_k}{B_k} < 1.$$

By ordering the resources, we are giving priority to faster links rather than to faster nodes. This is very reminiscent of the bandwidth-centric results in [11].

In order to show the benefits of our resource selection method, Figure 11 also plots the normalize makespan versus het when no resource selection is used. One can see that without resource selection, UMR is not able to maintain a low normalized makespan for $het > 50$.

6.4 Effects of Bandwidth-Sharing

We have assumed that the master uses its network connection sequential: it never dispatches work to two processors simultaneously. This assumption is justified on a shared LAN where there is little benefit to using multiple concurrent connections. In fact, it would be detrimental in cases when collisions occur. However, in other cases such as wide-area links, using multiple connections can lead to higher throughput. This phenomenon is due to specifics of TCP (e.g. slow-start) and of Internet backbone links (i.e. many flows going over a single link). Using multiple TCP connections is a well-known technique that is used by Web browsers and by bulk data transfer tools [24].

Let B_N denote the throughput achieved by each connection on a link when N simultaneous connection share that link. Based on experiments conducted by other researchers in our group, we use the following model for B_N:

$$B_N = \frac{1}{N^\alpha} B_1, \quad \text{where} \;\; 0 \leq \alpha \leq 1.$$

The case $\alpha = 1$ is when there is no benefit to using multiple connections (LAN), and the case $\alpha = 0$ is when connections beyond the first one get bandwidth "for free". We have observed both behaviors in experiments conducted on real networks.

A straightforward scheduling algorithm to take advantage of multiple concurrent connections (when $\alpha < 1$) is to dispatch the entire workload in one single round, sending work to all processors simultaneously. In this preliminary evaluation, we consider a homogeneous platform with N processors (e.g. a homogeneous cluster accessible over a wide-area link). The makespan of the one-round strategy is then given by:

$$Makespan = nLat + \frac{W_{total}}{B} N^{\alpha - 1} + cLat + \frac{W_{total}}{SN},$$

assuming that the network latency, $nLat$, is overlappable among concurrent transfers (which is realistic given the behavior of TCP in practice).

Not surprisingly, our experiments show that UMR is outperformed by the one-round strategy when α is low. Given B, it is easy to calculate the α value above which UMR outperforms the one-round strategy. We show this value as in Figure 6.4. We currently have no explanation for the shape of the curve, but the important point is that α needs to be above the curve for UMR to outperform a one-strategy. In practice, we have observed α values under 0.80 for many wide-area networks. This means that UMR would most likely be outperformed by a one-round strategy if the platform were connected by such links.

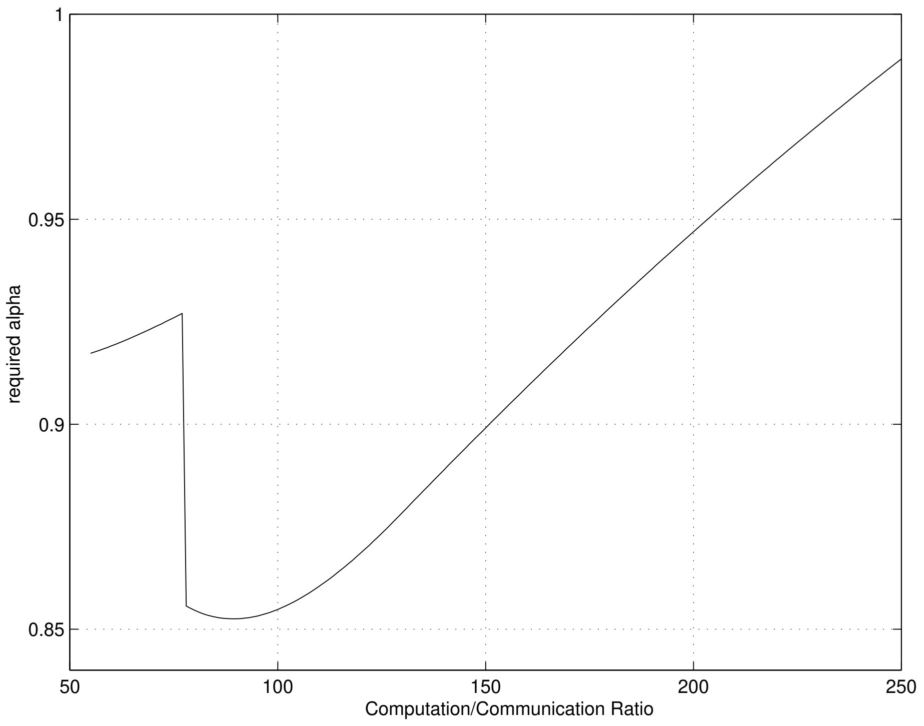

Figure 12: α value for which UMR is equivalent to a one-round strategy in the presence of bandwidth-sharing. $W_{total} = 1000$, $clat = 0.99$, $nlat = 0.09$, $N = 50$.

There are many ways in which multiple, concurrent connections could be exploited by a scheduling algorithm. For instance, data could be sent out simultaneously to subsets of the processors in order to achieve higher throughput as well as overlap of communication and computation. This could be done also in multiple rounds. In fact, it is conceivable that UMR could be extended to decide on the number of multiple connections that should be used. The goal would be to achieve an optimal trade-off between throughput and computation/communication overlap.

7 Conclusion

In this paper we have presented UMR, an algorithm for minimizing the makespan of divisible workload applications on homogeneous and heterogeneous networks of clusters. Like previously proposed algorithms, UMR uses a master/worker paradigm and dispatches work to workers in multiple rounds, which makes it possible to overlap communication and computation. The main question is: how many rounds should be used, and how much work is sent to each worker during each round? The trade-off is that using many rounds to send small amounts of work to workers allow for good overlapping of communication and computation, but incurs costly overheads. One of our contributions is that we use "uniform rounds": during each round a fixed amount of work is sent to each worker. Our main result is that this restriction enables us to analytically compute an approximately optimal number of rounds, which was not possible for previously proposed algorithms. We validated our approach in simulation with a realistic platform model and compared it with the multi-round algorithm in [17] and the one-round algorithm in [15]. We have seen that UMR leads to better schedules than competing algorithms in the overwhelming majority of the cases (>95%). Neither competing algorithm outperforms UMR "across the board" (i.e. for a large range of computation/communication ratios). Even when UMR is outperformed, it is close to the competing algorithms (within 2.04% with a standard deviation of 0.035). We presented results to demonstrate and explain how UMR choose the optimal number of rounds. We showed that thanks to a resource selection strategy, UMR can tolerate highly heterogeneous platforms. Finally, we have presented a preliminary study of the effect of bandwidth-sharing among concurrent network connections on the effectiveness of UMR.

In future work we will study the impact of performance prediction errors on the scheduling of divisible workloads. In this paper we have assumed that the scheduler has perfect knowledge of the performance that can be delivered by networks and CPUs. In realistic platforms, this assumption does not hold and one must develop scheduling algorithms that can tolerate uncertainties in predicted network transfer times and computation times. Some of our previous work has made contributions to that issue in the case of fixed-size task applications [25].

In the realm on divisible workload applications, the work in [18] addresses uncertainty by reducing the chunk size at each round. The UMR algorithm on the other hand increases chunk size at each round for better performance. We will investigate an approach that initially increases chunk size for better overlapping of communication and computation, but decreases chunk size towards the end of the application run in order to reduce uncertainties. We believe that the UMR algorithm is ideally suited to such extensions. We will also investigate how UMR can exploit the bandwidth-sharing properties discussed in Section 6.4. Our ultimate goal is to implement the resulting scheduling algorithm as part of the APST software [20, 26], an environment for deploying scientific applications on real platforms.

References

[1] Tim Davis, Alan Chalmers, and Henrik Wann Jensen. *Practical parallel processing for realistic rendering*. ACM SIGGRAPH, July 2000.

[2] MCell Webpage. `http://www.mcell.cnl.salk.edu/`.

[3] HMMER Webpage. `http://hmmer.wustl.edu/hmmer-html/`.

[4] BLAST Webpage. `http://http://www.ncbi.nlm.nih.gov/BLAST/`.

[5] C. Lee and M. Hamdi. Parallel Image Processing Applications on a Network of Workstations. *Parallel Computing*, 21:137–160, 1995.

[6] D. Altilar and Y. Paker. An Optimal Scheduling Algorithm for Parallel Video Processing. In *Proceedings of the IEEE International Conference on Multimedia Computing and Systems*, 1998.

[7] T. Hsu. Task Allocation on a Network of Procesors. *IEEE Transactions on Computers*, 49(12):1339–1353, december 2000.

[8] T. Braun, H. Siegel, and N. Beck. A Comparison of Eleven Static Heuristics for Mapping a Class of Independent Tasks onto Heterogeneous Distributed Computing Systems. *Journal of Parallel and Distributed Computing*, 61:810–837, 2001.

[9] O. Beaumont, A. Legrand, and Y. Robert. The Master-Slave Paradigm with Heterogeneous Processors. In *Proceedings of Cluster'2001*, pages 419–426. IEEE Press, 2001.

[10] H. Casanova, A. Legrand, D. Zagorodnov, and F. Berman. Heuristics for Scheduling Parameter Sweep Applications in Grid Environments. In *Proceedings of the 9th Heterogeneous Computing Workshop (HCW'2000)*, pages 349–363, May 2000.

[11] O. Beaumont, L. Carter, J. Ferrante, A. Legrand, and Y. Robert. Bandwidth-Centric Allocation of Independent Tasks on Heterogeneous Platforms. In *Proceedings of the International Parallel and Distributed Processing Symposium (IPDPS)*, June 2002.

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

[13] J. Blazewica, M. Drozdowski, and M. Markiewicz. Divisible Task Scheduling - Concept and Verification. *Parallel Computing*, 25:87–98, 1999.

[14] M. Drozdowski and P. Wolniewicz. Experiments with Scheduling Divisible Tasks in Clusters of Workstations. In *Proceedings of Europar'2000*, pages 311–319, 2000.

[15] A. L. Rosenberg. Sharing Partitionable Workloads in Heterogeneous NOWs: Greedier Is Not Better. In *Proceedings of the 3rd IEEE International Conference on Cluster Computing (Cluster 2001)*, pages 124–131, 2001.

[16] D. Altilar and Y. Paker. Optimal Scheduling algorithms for Communication Constrained Parallel Processing. In *In Proceedings of Europar'02*, pages 197–206, 2002.

[17] V. Bharadwaj, D. Ghose, V. Mani, and T. G. Robertazzi. *Scheduling Divisible Loads in Parallel and Distributed Systems*, chapter 10. IEEE Computer Society Press, 1996.

[18] S. Flynn Hummel. Factoring : a Method for Scheduling Parallel Loops. *Communications of the ACM*, 35(8):90–101, August 1992.

[19] T. Hagerup. Allocating Independent Tasks to Parallel Processors: An Experimental Study. *Journal of Parallel and Distributed Computing*, 47:185–197, 1997.

[20] H. Casanova and F. Berman. *Parameter Sweeps on the Grid with APST*, chapter 26. Wiley Publisher, Inc., 2002. F. Berman, G. Fox, and T. Hey, editors.

[21] D. Bertsekas. *Constrained Optimization and Lagrange Multiplier Methods*. Athena Scientific, Belmont, Mass., 1996.

[22] Henri Casanova. Simgrid: A Toolkit for the Simulation of Application Scheduling. In *Proceedings of the IEEE/ACM International Symposium on Cluster Computing and the Grid*, May 2001.

[23] Simgrid Webpage. `http://grail.sdsc.edu/projects/simgrid`.

[24] W. Allcock, J. Bester, J. Bresnahan, A. Chervenak, L. Liming, and S. Tuecke. GridFTP: Protocol Extension to FTP for the Grid. Grid Forum Internet-Draft, March 2001.

[25] H. Casanova, A. Legrand, D. Zagorodnov, and F. Berman. Heuristics for Scheduling Parameter Sweep Applications in Grid Environments. In *Proceedings of the 9th Heterogeneous Computing Workshop (HCW'00)*, pages 349–363, May 2000.

[26] APST Webpage. `http://grail.sdsc.edu/projects/apst`.

A Lagrange Multiplier Method for Homogeneous Platforms

The Lagrange system shown in Eq. 11 can be solved as follows. Using Eq. 5, G can be rewritten as:

$$\begin{aligned} G(M, chunk_0) &= \sum_{j=0}^{M-1} \left[N(\tfrac{B}{NS})^j(chunk_0 - \alpha) + N\alpha\right] - W_{total} \\ &= NM\alpha + N(chunk_0 - \alpha)\tfrac{1-(\frac{B}{NS})^M}{1-\frac{B}{NS}} - W_{total}. \end{aligned}$$

Therefore,

$$\frac{\partial L}{\partial \lambda} = 0 \quad \Rightarrow \quad chunk_0 = \alpha + \frac{1 - \frac{B}{NS}}{1 - (\frac{B}{NS})^M}\left[\frac{W_{total} - NM\alpha}{N}\right],$$

which proves Eq. 13. One can compute the differentials of the multiplier with respect to M and $chunk_0$ as:

$$\frac{\partial L}{\partial M} = cLat + \lambda\left[N\alpha - \frac{W_{total} - NM\alpha}{1 - (\frac{B}{NS})^M}\ln(\frac{B}{NS})(\frac{B}{NS})^M\right], \qquad (18)$$

and

$$\frac{\partial L}{\partial chunk_0} = \frac{NS}{2B} + \lambda N \frac{1 - (\frac{B}{NS})^M}{1 - \frac{B}{NS}}.$$

Therefore,

$$\frac{\partial L}{\partial chunk_0} = 0 \quad \Rightarrow \quad \lambda = -\frac{S}{B}\frac{1 - \frac{B}{NS}}{1 - (\frac{B}{NS})^M}.$$

One can then replace λ by its expression in Eq. 18, leading to:

$$\frac{\partial L}{\partial M} = 0 \quad \Rightarrow \quad cLat - \frac{S}{2B}\frac{1 - \frac{B}{NS}}{1 - (\frac{B}{NS})^M}\left[N\alpha - \frac{W_{total} - NM\alpha}{1 - (\frac{B}{NS})^M}\ln(\frac{B}{NS})(\frac{B}{NS})^M\right] = 0,$$

which is equivalent to Eq. 12.

B Parameter Constraint for Heterogeneous Platforms

We derive constraints for all workers to be utilized in a heterogeneous setting. We focus on the chunk sent to worker l in round j. To fully utilize all the nodes, the master must finish sending round j work to all the workers l to N, and round $j+1$ work to all workers 1 to l, while worker l is computing for round j. Similarly to Eq.7 for the homogeneous case, this can be written as:

$$\begin{aligned}\sum_{i=l}^{N}(nLat_i + \frac{chunk_{ji}}{B_i}) + \sum_{i=1}^{l}(nLat_i + \frac{chunk_{j+1,i}}{B_i}) + tLat_l \\ \leq nLat_l + \frac{chunk_{jl}}{B_l} + cLat_l + tLat_l + \frac{chunk_{jl}}{S_l}.\end{aligned}$$

Multiplying by S_l and summing for l from 1 to N, we obtain:

$$\begin{aligned}\sum_{l=1}^{N} S_l \left[\sum_{i=l}^{N}(nLat_i + \frac{chunk_{ji}}{B_i}) + \sum_{i=1}^{l}(nLat_i + \frac{chunk_{j+1,i}}{B_i})\right] \\ \leq \sum_{l=1}^{N} S_l nLat_l + \sum_{l=1}^{N} S_l cLat_l + \sum_{l=1}^{N} S_l \frac{chunk_{jl}}{B_l} + round_j\end{aligned}$$

$$\begin{aligned}\iff \quad \sum_{l=1}^{N} S_l \left[\sum_{i=l+1}^{N}(nLat_i + \frac{chunk_{ji}}{B_i}) + \sum_{i=1}^{l}(nLat_i + \frac{chunk_{j+1,i}}{B_i})\right] + \\ \sum_{l=1}^{N} S_l(nLat_l + \frac{chunk_{j,l}}{B_l}) \leq \sum_{l=1}^{N} S_l nLat_l + \sum_{l=1}^{N} S_l cLat_l + \sum_{l=1}^{N} S_l \frac{chunk_{jl}}{B_l} + round_j\end{aligned}$$

$$\begin{aligned}\iff \quad \sum_{l=1}^{N} S_l \left[\sum_{i=l+1}^{N}(nLat_i + \frac{\alpha_i round_j + \beta_i}{B_i}) + \sum_{l=1}^{l} S_l(nLat_i + \frac{\alpha_i round_{j+1} + \beta_i}{B_i})\right] \\ \leq \sum_{l=1}^{N} S_l cLat_l + round_j\end{aligned}$$

Replacing $round_j$ and $round_{j+1}$ with the expression in Eq. 16 we obtain:

$$\left(\sum_{l=1}^{N} S_l\right)\left(\sum_{i=1}^{N}(nLat_i+\frac{\beta_i}{B_i})\right)+(round_0-\eta)\sum_{l=1}^{N}\left(\sum_{i=1}^{l}\frac{\alpha_i}{B_i}\theta^{j+1}+\sum_{i=l+1}^{N}\frac{\alpha_i}{B_i}\theta^{j}\right)+$$
$$\left(\sum_{l=1}^{N} S_l\right)\left(\sum_{i=1}^{N}\frac{\alpha_i}{B_i}\right)\eta\leq\sum_{l=1}^{N}S_l cLat_l+\theta^j(round_0-\eta)+\eta.$$

$$\iff\quad(round_0-\eta)\left(\sum_{l=1}^{N}S_l(\sum_{i=1}^{l}\frac{\alpha_i}{B_i}\theta^{j+1}+\sum_{i=l+1}^{N}\frac{\alpha_i}{B_i}\theta^{j})-\theta^j\right)\leq$$
$$\sum_{l=1}^{N}S_l cLat_l-\left(\sum_{l=1}^{N} S_l\right)\left(\sum_{i=1}^{N}(nLat_i+\frac{\beta_i}{B_i})\right)-\left(\sum_{l=1}^{N} S_l\right)\left(\sum_{i=1}^{N}\frac{\alpha_i}{B_i}\right)\eta+\eta$$

Let γ be defined as:

$$\gamma=\sum_{l=1}^{N}S_l(\sum_{i=1}^{l}\frac{\alpha_i}{B_i}\theta+\sum_{i=l+1}^{N}\frac{\alpha_i}{B_i}).$$

Using the definition of θ and η, we can rewrite the above equation as:

$$\theta^j(round_0-\eta)(\gamma-1)\leq\eta(\frac{1}{\theta}-1)-\frac{\eta}{\theta}+\eta$$

$$\iff\qquad(round_0-\eta)(\gamma-1)\leq 0\tag{19}$$

If the master can utilize all workers, then Eq. 19 holds. We now have two cases depending on the sign of $(\gamma-1)$:

1. If $\gamma>1$, then Eq. 19 reduces to $round_0\leq\eta$. Since $round_0$ must be positive, a necessary condition for Eq. 19 is:

$$\eta>0.\tag{20}$$

2. If $\gamma<1$, then Eq. 19 reduces to $round_0\geq\eta$. Since $round_0$ must be lower than W_{total}, a necessary condition for Eq. 19 is:

$$\eta<W_{total}.\tag{21}$$

C Lagrange Multiplier Method for Heterogeneous Platforms

Using Eq. 14 and Eq. 15, one can rewrite Eq. 17 as:

$$\begin{aligned} Ex(M, round_0) = & \frac{(round_0-\eta)(1-\theta^M)}{(1-\theta)\sum_{i=1}^N S_i} + \frac{1}{2} round_0 \sum_{i=1}^N \frac{\alpha_i}{B_i} + \\ & M\frac{\sum_{i=1}^N (S_i \times cLat_i)+\eta}{\sum_{i=1}^N S_i} + \frac{1}{2}\sum_{i=1}^N (nLat_i + \frac{\beta_i}{B_i}) + tLat_{_N}. \end{aligned}$$

The Lagrange multiplier $L(chunk_0, M, \lambda)$ is then defined as:

$$L(round_0, M, \lambda) = Ex(M, round_0) + \lambda \times G(M, round_0)$$

and the Lagrange system is written as:

$$\begin{cases} \frac{\partial L}{\partial \lambda} = G = 0 \\ \frac{\partial L}{\partial M} = \frac{\partial Ex}{\partial M} + \lambda \times \frac{\partial G}{\partial M} = 0 \\ \frac{\partial L}{\partial round_0} = \frac{\partial Ex}{\partial round_0} + \lambda \times \frac{\partial G}{\partial round_0} = 0 \end{cases}$$

which can be rewritten as:

$$\Longrightarrow \begin{cases} \frac{\partial L}{\partial \lambda} & = G(M, round_0) = M \times \eta + \frac{round_0-\eta}{1-\theta} \times (1-\theta^M) - W_{total} = 0 \\ \frac{\partial L}{\partial M} & = -\frac{round_0-\eta}{(1-\theta)\sum_{i=1}^N S_i}\theta^M \ln\theta + \frac{\sum_{i=1}^N (S_i \times cLat_i)+\eta}{\sum_{i=1}^N S_i} + \lambda\left[\eta - \frac{round_0-\eta}{1-\theta}\theta^M \ln\theta\right] = 0 \\ \frac{\partial L}{\partial round_0} & = \frac{1-\theta^M}{(1-\theta)\sum_{i=1}^N S_i} + \frac{1}{2}\sum_{i=1}^N \frac{\alpha_i}{B_i} + \lambda\left[\frac{1-\theta^M}{1-\theta}\right] = 0 \end{cases}$$

Eliminating λ and $round_0$ from the above equations, one obtains the following equation for M:

$$\begin{aligned} & \frac{(M \times \eta - W_{total}) \times \theta^M \ln\theta \sum_{i=1}^N \frac{\alpha_i}{B_i}}{\times(1-\theta^M)} + \\ & \eta\sum_{i=1}^N \frac{\alpha_i}{B_i} - 2\frac{1-\theta^M}{1-\theta} \times \frac{\sum_{i=1}^N (S_i \times cLat_i)}{\sum_{i=1}^N S_i} \quad = 0. \end{aligned} \tag{22}$$

As in the homogeneous case, this equation can be solved by bisection. Let M^* be the solution. One can then compute $round_0$ as:

$$round_0 = \frac{1-\theta}{1-\theta^{M^*}}(W_{total} - M^* \times \eta) + \eta. \tag{23}$$

Using Eq. 15, one can then compute all the $chunk_{ji}$ values.

D Last Round Modification of the UMR Algorithm

The goal is to compute modified values for $chunk_{M^*-1,i}$, $i = 1, \ldots, N$, that is the chunk sizes for the last round. The objective is to have all workers finish their computation at the same instant For this round, the idea is to give decreasing amounts of work to the workers, which is different from the "uniform round" approach used by UMR for all other rounds. We present developments for the most general, heterogeneous model.

Let Δ_{i+1} denote the time elapsed between the start of computation for worker and i $i+1$ during a round. With the UMR algorithm, this time is the same for all rounds and can be easily computed from round 0 as:

$$\forall i = 1, \ldots, N-1 \quad \Delta_{i+1} = \tfrac{chunk_{0,i+1}}{B_{i+1}} + nLat_{i+1}.$$

When B values are small, these quantities are large, leading to idle time in the last round. We must ensure that, in the last round, worker i computes for Δ_{i+1} time units longer than worker $i+1$. This can be written easily for $i = 1, \ldots, N-1$ as:

$$\Delta_{i+1} = \left(\frac{chunk_{M^*-1,i}}{S_i} + cLat_i\right) - \left(\frac{chunk_{M^*-1,i+1}}{S_{i+1}} + cLat_{i+1}\right). \tag{24}$$

We also have the additional constraint:

$$\sum_{i=1}^{i=N} chunk_{M^*-1,i} = round_{M^*-1}, \tag{25}$$

where $round_{M^*-1}$ is computed according to Eq. 4.2.2. Therefore, we have a linear system of N equations of N unknowns (the $chunk_{M^*,i}$ values), which we solve below.

By summing Eq. 24 from i to $N-1$ we obtain

$$\forall i = 0, \ldots, N-1, \frac{chunk_{M^*-1,i}}{S_i} + cLat_i - \left(\frac{chunk_{M^*-1,N}}{S_N} + cLat_N\right) = \sum_{k=i+1}^{N} \Delta_k,$$

which in turn can be summed for all $i = 1, \ldots, N-1$ to obtain:

$$\sum_{i=1}^{N-1} chunk_{M^*-1,i} + \sum_{i=1}^{N-1} S_i cLat_i - \left(\frac{chunk_{M^*-1,N}}{S_N} + cLat_N\right) \sum_{i=1}^{N-1} S_i = \sum_{i=1}^{N-1} S_i \sum_{k=i+1}^{N-1} \Delta_k.$$

But

$$(25) \Rightarrow \sum_{i=1}^{N-1} chunk_{M^*-1,i} = round_{M^*-1} - chunk_{M^*-1,N},$$

which, when replaced in the previous equation, gives us the following expression for $chunk_{M^*-1,N}$:

$$chunk_{M^*-1,N} = \frac{S_N}{\sum_{i=1}^{N} S_i} \left[round_{M^*-1} + \sum_{i=1}^{N-1} S_i \left(cLat_i - \sum_{k=i+1}^{N} \Delta_k \right) \right] - S_N \times cLat_N.$$

The remaining values $chunk_{M^*-1,i}$ $(1 \leq i \leq N-1)$ can then be easily calculated by induction with Eq 24.