Extensions to the Multi-Installment Algorithm: Affine Costs and Output Data Transfers

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

Divisible workload applications occur in many fields of science and engineering. Although these application can be easily parallelized in a master-worker fashion, they pose several scheduling challenges. Previously proposed scheduling algorithms either distribute work to processors in a single round *of work allocation or in* multiple rounds*. Multi-round algorithms can achieve better overlap of computation and communication but are more difficult to analyze. Consequently, a number of open questions still remain for multi-round scheduling. In this paper we improve on the seminal "multi-installment" algorithm proposed by Bharadwaj. et al. for homogeneous star networks. Their work suffers from two important limitations: (i) communication and computation latencies are assumed to be negligible; and (ii) size of application output data is assumed to be negligible. These two limitations strongly restrict the applicability of multi-installment to real-world platforms and applications. In this paper we remove both limitations.*

1 Introduction

Scheduling the tasks of a parallel application onto a distributed computing platform is key for achieving high performance and is a well-known challenging problem. Application that consist of of independent tasks with no synchronizations and no inter-task communications arise in many fields of science and engineering. A possible model for independent tasks is one for which the number of tasks and the task sizes, i.e. their computational costs, are pre-determined. In this case, the scheduling problem is akin to bin packing and several heuristics have been proposed [17, 3]. Another model is one in which the number of tasks and the task sizes can be chosen arbitrarily, that is the application workload can be arbitrarily divided. In practice, this model is an approximation of an application that consists of large numbers of identical, low-granularity computations.

This *divisible workload* scenario arises in many application domains [19, 12, 27, 13, 25]. Divisible workload applications are amenable to straightforward master-worker parallel execution and can thus be easily deployed on computing platforms ranging from commodity clusters to computation grids [28]. Divisible workload scheduling is challenging due to the overhead involved when starting tasks. This overhead is due to: (i) the time to transfer application input/output data to/from each compute resource; and (ii) the possible latency involved when initiating a computation and/or a transfer. There are two types of approaches for scheduling a divisible workload. One can divide the workload in as many chunks as

the number of processors and dispatch these chunks in a *single round* of allocation. This scheme is simple to design and implement, but suffers from poor overlap of communication and computation. The alternative to alleviate this problem is to dispatch the workload in *multiple rounds*, with each worker being allocated a chunk of the workload at each round. While single-round approaches have been studied throughly [34, 32, 19, 31, 28, 2, 22, 8], multi-round algorithms are significantly more difficult to analyze and thus fewer results are available. In this paper we focus on multi-round algorithms.

The "multi-installment" algorithm in [9] is essentially the only previously proposed multi-round algorithm. It operates only on homogeneous platforms, which are the focus of this paper (we proposed a multi-round algorithm for heterogeneous platforms in [33]). A severe limitation of the multi-installment algorithm is that it assumes a *linear cost model*, by which a computation or communication task takes a time exactly proportional to the task size. In other words, the multi-installment approach does not consider latencies associated with communication or computation, which is unrealistic for most real-world platforms and applications. In this paper, we consider an *affine cost model* that incorporates latencies. Although this model has been used for single-round algorithms [15, 11, 8], it is an open question to determine a closed-form schedule analogous to the "multi-installment" schedule for affine costs.

Another limitation of previous work on multi-round divisible workload scheduling is that only input data transfers are considered. Although solutions for single-round scheduling with output data transfer have been proposed [31], the incorporation of output data transfers in multi-round scheduling is still an open question.

Our contribution in this paper addresses these two questions and is as follows:

1. We obtain closed-form solutions to the multi-installment scheduling problem on homogeneous platforms with affine cost models both for communication and computation. Our solutions are analogous to but more general than those in [10]: we give a new form of the recursion on chunk sizes, which enables the use of generating functions to solve the scheduling problem even in the presence of latencies.

2. Building on our first solution, we obtain solutions to multi-installment scheduling for a homogeneous platform when both input and output data transfers are considered, while still using affine cost models for communication and computation.

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 closed-form solutions for multi-round divisible load scheduling on homogeneous platforms with affine cost models. Section 5 extends the result in Section 4 to incorporate output data transfers. Section 6 concludes the paper.

2 Related Work

Divisible workload scheduling algorithms fall into two categories: single-round algorithms and multi-round algorithm. We first describe previous work on single-round algorithms as many results are available and are indicative of the kind of result that one could hope to achieve in the context of multi-round algorithms. Early work on single-round algorithms focused on developing the fundamental recursion relations that make it possible to compute chunk sizes inductively [34, 35, 5]. These works used a purely linear cost model (i.e. no latencies) and studied a variety of *homogeneous* platform topologies including linear arrays, buses, stars, and trees. Building on these initial accomplishments closed-form solutions to the scheduling problems for homogeneous platforms have been developed for buses and trees [4], linear arrays [30], 3-D mesh [18], and hypercubes [29]. A number of works have also studied the asymptotic

performance behavior as the number of processors tends to infinity on linear networks [22, 30], buses and trees [4], rings and 2-D meshes [14], and 3-D meshes [18]. A few works have focused on developing optimal schedules for *heterogeneous* platforms, namely computing an optimal ordering of compute resources [26, 9, 8]. In this work we focus on star topologies as they are the most relevant to current practice. We only consider homogeneous platforms in this paper (see our work in [33] on multi-round scheduling for heterogeneous platforms).

While these works all used a purely linear model, the work in [15] was one of the first to model a fixed latency associated to network communication via an affine model, which is more realistic and has since then been used in [31, 24]. The work in [11] models an affine cost for computation, which is also more realistic. The introduction of affine costs renders the problem of scheduling on heterogeneous platforms much more complex. Results are available for special cases in which the platform is only partially heterogeneous [8]. For the most general heterogeneous platform, one must resort to Linear Programming [20]. In this work we consider affine costs both for computation and communication.

Building on these numerous results, authors have reported on practical implementation and experimental results obtained with one-round algorithms [19, 12, 28]. However, in spite of the known limitation of one-round algorithms, namely poor overlap of computation with communication, work on multi-round algorithms is rather scarce. Proposed approaches belong in three categories: (i) those that focus on minimizing application makespan by improving overlap of communication with computation; (ii) those that focus on minimizing application makespan in the presence of performance prediction errors; and (iii) those that focus on maximizing steady-state application performance. This work belongs to the first category, but we review all three categories below.

The first multi-round algorithm for minimizing application makespan with no performance prediction errors is the "multi-installment" approach proposed in [10] and little progress has been made in that area since then. Multi-installment consists in dispatching chunks of workload to compute resources in multiple rounds. The algorithm starts with small chunks and *increases* the chunk size to achieve effective overlap of communication and computation. In this paper we directly improve on the results in [10] for homogeneous platforms by considering latencies associated to computation and communication and by accounting for transfer of output data. Considering latencies raises the question of the optimal number of rounds, which we have addressed in [33]. In this paper we assume, as in [10], that the number of rounds is fixed (i.e. provided as an input parameter to the scheduling algorithm).

Multi-round algorithms for divisible workload that account for significant performance prediction errors, either due to shared computing platforms or to non-deterministic applications, were proposed in [21, 24]. Instead of increasing chunk size throughout application execution, these approaches start with large chunks and *decrease* chunk size throughout application execution, and dispatch chunks to compute resources in a greedy fashion, in order to avoid the "wait for the last task" problem. The major disadvantage is that these algorithms can lead to very poor overall of computation with communication. In [36] we have proposed an approach that first increases and then decreases chunk size throughout application execution to achieve both effective overlap of computation with communication and robustness to performance prediction errors. In this paper we assume zero performance prediction errors.

Finally, multi-round algorithms have also been developed to maximize *steady-state* application performance, that is the asymptotic amounts of computation performed per time unit [1, 6, 7]. Consequently these algorithms use identical rounds and the schedules are periodic. In this work we are solely concerned with minimizing application makespan.

3 Models

3.1 Application

We consider applications that consist of a workload W_{total} (i.e. an amount of computation to perform) that is *continuously divisible*: the scheduler can decide how big a chunk of the workload to give out to a processor. We assume that the amount of input application data needed for processing a chunk is *proportional* to the amount of workload for that chunk. It is a common assumption in the divisible workload literature to ignore transfer of output application data. The works in [31, 2] takes into account output data transfers but uses a single-round approach. The multi-round algorithm in [1] models output but only considers steady-state application performance. We ignore transfers of output data in Section 4 and incorporate them in Section 5.

3.2 Computing Platform

We assume a star topology, depicted in Figure 1, used in a master/worker fashion 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. 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). Throughout this paper we assume a homogeneous star topology. We assume that workers can receive data from the network and perform computation simultaneously (corresponding to the "with front-end" model in [10]).

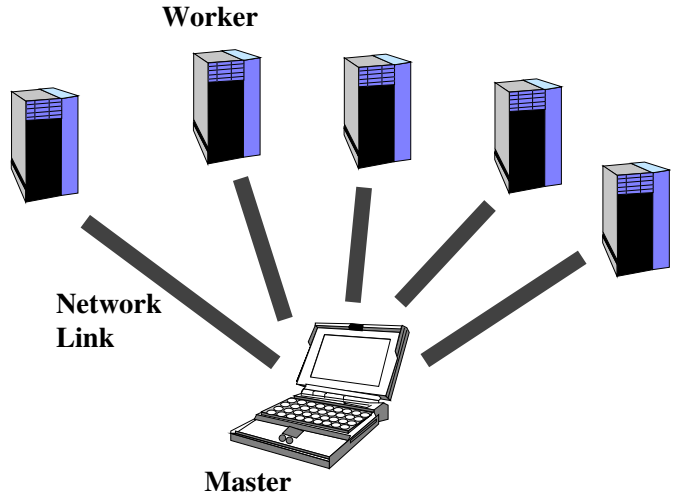


Figure 1. Computing platform model.

Consider a portion of the total workload, $chunk \leq W_{total}$, which is to be processed on a worker. We model the time required for a worker to perform the computation, T_{comp}, as

$$T_{comp} = \alpha + \frac{chunk}{S}, \tag{1}$$

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

We model the time spent for the master to send $chunk$ units of workload to a worker, T_{comm}, as:

$$T_{comm} = \beta + \frac{chunk}{B}, \tag{2}$$

where β is the overhead, in seconds, incurred by the master to initiate a data transfer to a worker (e. g. pre-process application input data and/or initiate a TCP connection); and B is the data transfer rate to workers, in units of workload per second. We assume that the $\beta + chunk/B$ portion of the transfer is not overlappable with other data transfer.

Choosing α and β as zero or non-zero makes the computation and communication cost models linear or affine. Based on our experience with actual software [16], we deem α to be fundamental for realistic modeling. But to the best of our knowledge, only [11] has modeled this latency in the context of divisible load scheduling (only for a one-round algorithm).

4 Affine Cost Models for Multi-installment

The multi-installment algorithm introduced in [10] only considers *linear* cost models. In this section we evolve the multi-installment approach so that it can account for affine cost models for communication and computation. In Section 4.1 we first revisit the multi-installment approach with a purely linear cost model to obtain a new and more uniform recursion on chunk sizes than that given in [10]. In Section 4.2 we then extend this recursion to use affine cost models.

4.1 Linear Cost Models

We denote by T_{total} the amount of time to process W_{total} units of workload on a single worker. Let M be the total number of rounds, which is given as a parameter and not computed by the algorithm. Figure 2 depicts the computation of the workload on 5 workers in 3 rounds, i.e. with 15 chunks of workload. Chunk transfers from the master are shown in Gray boxes whereas chunk computations are shown in white boxes. For technical reasons, as in [10], we number the chunks in the reverse order in which they are allocated to workers: the last chunk is numbered 0, the worker receiving the last chunk is numbered 0, and the last round is also numbered 0. The goal of the algorithm is to ensure that all workers finish computing at the same time and that the workers be as utilized as possible.

With a linear cost model, the time to transfer the i^{th} chunk of workload, of size $chunk_i$, to a worker is given as:

$$T_{comm_i} = chunk_i/B = chunk_i/S \times S/B = g_i/R,$$

where R is the computation-communication ratio of the platform, and g_i is the computation time of the chunk on a worker.

We can now derive a recursion on the chunk computation times. In order for both the network and the workers to be kept as busy as possible, each worker must compute a chunk in exactly the time required for all the next N chunks to be sent to the workers [10]. This can be easily written as:

$$g_i = (g_{i-1} + g_{i-2} + g_{i-3} + \cdots + g_{i-N})/R.$$

For example, in Figure 2, we can write that while worker 2 computes chunk 7 (from time A to time B) chunks 6 to 2 must be sent to workers 1, 0, 4, 3, and 2 as: $g_7 = (g_6 + g_5 + \cdots + g_2)/R$. Note that the above equation is only valid for $i \geq N$. For $i < N$ the same idea works modulo a small modification:

$$g_i \quad = \quad (g_{i-1} + g_{i-2} + g_{i-3} + \cdots + g_{i-N})/R + g_0,$$

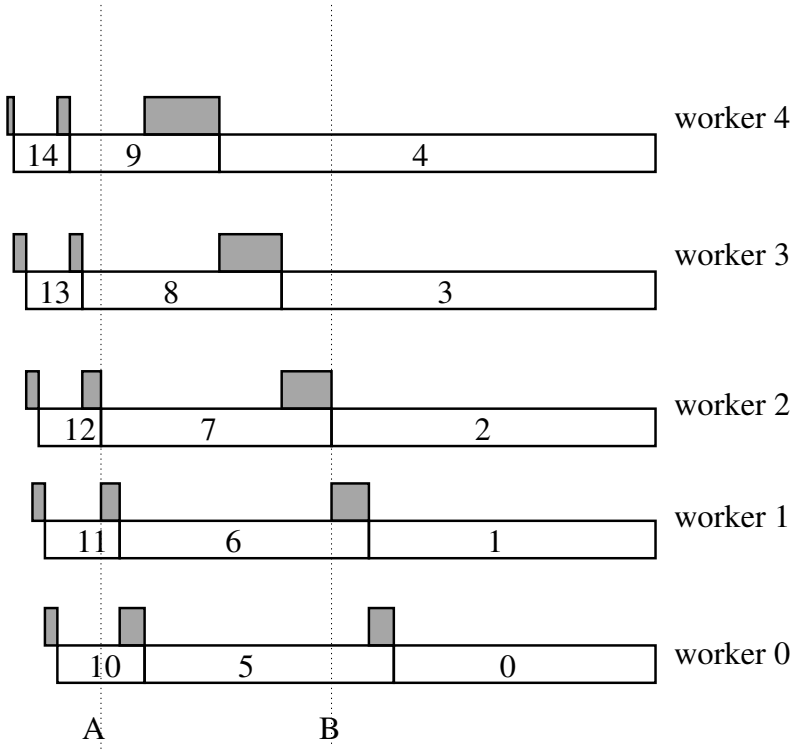


Figure 2. Illustration of the chunk recursion with no latencies.

where we let $g_i = 0$ for $i < 0$. These recursive relations are analogous in spirit to the $g_{i+1} = (1+1/R)g_i$ used in [10] but will allow us to incorporate affine costs in Section 4.2. We summarize our recursion as:

$$\forall\ i \geq N \qquad g_i = (g_{i-1} + g_{i-2} + g_{i-3} + \cdots + g_{i-N})/R, \tag{3}$$

$$\forall\ 0 \leq i < N \quad g_i = (g_{i-1} + g_{i-2} + g_{i-3} + \cdots + g_{i-N})/R + g_0, \tag{4}$$

$$\forall\ i < 0 \qquad g_i = 0. \tag{5}$$

Note that g_i values for $i \geq MN$ are ignored since we use only M rounds. However, we do not set them to 0 in order to keep the g_i series infinite, which allows us to use generating functions to solve the recursion. Let $G(x)$ be the generating function for the g_i series:

$$G(x) = \sum_{i=0}^{\infty} g_i x^i \qquad \text{(also commonly noted as } [x^i]G(x) = g_i\text{)}$$

Multiplying Eq.3 by x^i gives:

$$\forall\ i \geq N \quad g_i x^i = \left(\sum_{j=i-N}^{i-1} g_j x^i\right)/R = \left(x\sum_{j=0}^{i-1} g_j x^{i-1} - x^{N+1}\sum_{j=0}^{i-N-1} g_j x^{i-N-1}\right)/R. \tag{6}$$

Note that items out of range (i.e., with $i < 0$) in the equation above do not matter because we have $g_i = 0$ for $i < 0$. Similarly, multiplying Eq.4 by x^i leads to:

$$\forall\ 0 \leq i < N \quad g_i x^i = \left(x\sum_{j=0}^{i-1} g_j x^{i-1} - x^{N+1}\sum_{j=0}^{i-N-1} g_j x^{i-N-1}\right)/R + g_0 x^i. \tag{7}$$

Summing Eq. 6 and Eq. 7 for all $i \geq 0$ gives:

$$\begin{aligned} G(x) &= \sum_{i=0}^{\infty} g_i x^i = \sum_{i=N}^{\infty} g_i x^i + \sum_{i=0}^{N-1} g_i x^i \\ &= (\sum_{i=0}^{\infty} (x \sum_{j=0}^{i-1} g_j x^{i-1} - x^{N+1} \sum_{j=0}^{i-N-1} g_j x^{i-N-1}))/R + g_0 \sum_{i=0}^{N-1} x^i \\ &= (x \sum_{i=0}^{\infty} \sum_{j=0}^{i-1} g_j x^{i-1} - x^{N+1} \sum_{i=0}^{\infty} \sum_{j=0}^{i-N-1} g_j x^{i-N-1})/R + g_0 \sum_{i=0}^{N-1} x^i \\ &= (\frac{xG(x)}{1-x} - \frac{x^{N+1}G(x)}{1-x})/R + g_0 \frac{1-x^N}{1-x}. \end{aligned}$$

The last step is due to the well-known generating function property:

$$g_i = [x^i]G(x) \Rightarrow [x^n]\frac{G(x)}{1-x} = \sum_{i=0}^{n} g_i.$$

Finally we obtain:

$$G(x) = g_0 \frac{(1-x^N)}{(1-x) - x(1-x^N)/R}. \tag{8}$$

The standard Rational Expansion method can be used here to determine the coefficients of $G(x)$ [23]. One must find the $N+1$ roots ρ_i, $0 \leq i < N$, of $Q(x) = R(1-x) - x(1-x^N)$, the denominator polynomial. $G(x)$ can then be rewritten as:

$$G(x) = g_0 \sum_{i=0}^{N} \frac{\eta_i}{1 - x/\rho_i} = g_0 \sum_{i=0}^{N} \frac{\eta_i}{1 - \theta_i x},$$

where the η_i can be computed as in [23]. Note that in the general case when $Q(x)$ has roots of degree higher than 1 we must resort to the more complex General Rational Expansion theorem [23] to compute the power series for $G(x)$. However, in Appendix A we prove that: (i) when $R \neq N$, $Q(x)$ has only roots of degree 1; and (ii) when $R = N$, $Q(x)$ has only one root of degree higher than 1, which is root $x = 1$ with degree 2. Consequently, when $R \neq N$ we can compute the η_i coefficients above with the simple Rational Expansion method given in [23]. When $R = N$ the solution using the General Rational Expansion theorem is straightforward as there is only one root of degree higher than 1. One can then derive an expression for the g_i series as:

$$g_i = [x^i]G(x) = g_0 \sum_{j=0}^{N} \eta_j \theta_j^i. \tag{9}$$

Each chunk size turns out to be a linear combination of N geometric series. To compute g_0 one can just write that all the g_i's must sum up to T_{total}:

$$T_{total} = g_0 \sum_{i=0}^{NM-1} \sum_{j=0}^{N} \eta_j \theta_j^i, \quad \text{which gives} \quad g_0 = T_{total}/(\sum_{j=0}^{N} \eta_j \frac{1-\theta_j^{NM}}{1-\theta_j}). \tag{10}$$

Note that we do not use the "binomial expansion" method used in [10] because, strictly speaking, it does not provide a closed-form solution.

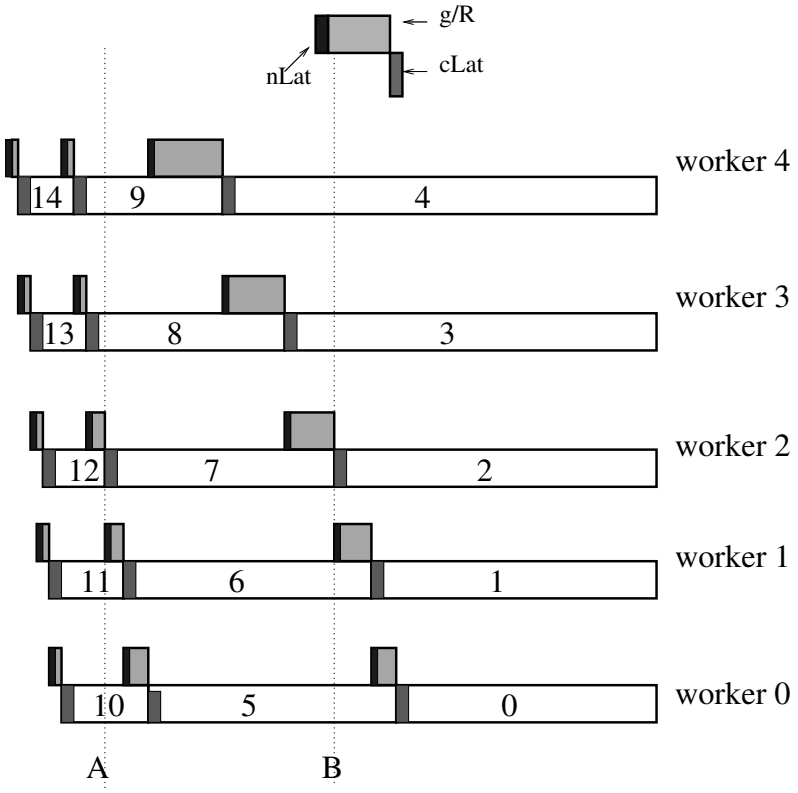


Figure 3. Illustration of the chunk recursion with latencies.

4.2 Affine Cost Model

We now incorporate the α and β latencies defined in Section 3. Figure 3 depicts the same execution as that on Figure 2. Equations 3, 4, and 5 are easily rewritten as:

$$\forall\, i \geq N \qquad \alpha + g_i = (g_{i-1} + g_{i-2} + g_{i-3} + \cdots + g_{i-N})/R + N \times \beta \tag{11}$$

$$\forall\, 0 \leq i < N \quad \alpha + g_i = (g_{i-1} + g_{i-2} + g_{i-3} + \cdots + g_{i-N})/R + i \times \beta + g_0 + \alpha \tag{12}$$

$$\forall\, i < 0 \qquad g_i = 0 \tag{13}$$

For example, in Figure 3, we can write that while worker 2 computes chunk 7 (from time A to time B) chunks 6 to 2 must be sent to workers 1, 0, 4, 3, and 2 as: $\alpha + g_7 = (g_6 + g_5 + \cdots + g_2)/R + N \times \beta$, which is Eq. 11 above.

As in Section 4.1, we can multiply Equations 11 and 12 and sum over all i to compute $G(x)$, the

generating function associated to series g_i:

$$\begin{aligned} G(x) &= \sum_{i=0}^{\infty} g_i x^i = \sum_{i=N}^{\infty} g_i x^i + \sum_{i=0}^{N-1} g_i x^i \\ &= \sum_{i=0}^{\infty} \left(x \sum_{j=0}^{i-1} g_j x^{i-1} - x^{N+1} \sum_{j=0}^{i-N-1} g_j x^{i-N-1} \right) /R \\ &\quad +(N \times \beta - \alpha) \sum_{i=0}^{\infty} x^i + (g_0 + \alpha - N \times \beta) \sum_{i=0}^{N-1} x^i + \beta \sum_{i=0}^{N-1} i x^i \\ &= \frac{x - x^{N+1}}{(1-x)R} G(x) + \frac{N \times \beta - \alpha}{1-x} + (g_0 + \alpha - N \times \beta) \frac{1 - x^N}{1-x} \\ &\quad + \frac{\beta}{1-x} \left(\frac{x(1-x^{N-1})}{1-x} - (N-1)x^N \right), \end{aligned}$$

which leads to the following generating function:

$$G(x) = \frac{(g_0 + \alpha - N \times \beta)(1-x^N) + (N \times \beta - \alpha) + \beta(\frac{x(1-x^{N-1})}{1-x} - (N-1)x^N)}{(1-x) - x(1-x^N)/R}. \tag{14}$$

In fact, this generating function is related to that in the linear cost model case:

$$\begin{aligned} G(x) &= g_0 G^{'}(x) + \frac{\alpha \times x^N + \beta(x + x^2 + x^3 + ... + x^N)}{(1-x) - x(1-x^N)/R} \\ &= g_0 G^{'}(x) + G^{''}(x), \end{aligned}$$

where $G^{'}(x)$ is the generating function derived in Section 4.1, and $G^{''}(x)$ is a generating function with the same denominator as $G^{'}(x)$.

As in Section 4.1, the Rational Expansion method [23] (or the General Rational Expansion theorem in the case $R = N$) can be used to compute the power series for $G(x)$ given the roots of the denominator polynomial $Q(x)$. Let ρ_j, $j = 0, \ldots, N$, be the roots of $Q(x)$ and let $\theta_j = 1/\rho_j$ be their inverses. One can now obtain the series g_i as:

$$g_i = [x^i]G(x) = g_0 \sum_{j=0}^{N} \eta_j \theta_j^i + \sum_{j=0}^{N} \phi_j \theta_j^i, \tag{15}$$

where the η_j and the ϕ_j series can be computed respectively for G' and for G'' as in [23]. Here also each chunk size is a linear combination of N geometric series.

To compute g_0 one can just write that all the g_i's must sum up to T_{total}:

$$T_{total} = \sum_{i=0}^{NM-1} g_i = g_0 \sum_{j=0}^{N} \eta_j \frac{1-\theta_j^{NM}}{1-\theta_j} + \sum_{j=0}^{N} \phi_j \frac{1-\theta_j^{NM}}{1-\theta_j},$$

which gives:

$$g_0 = \frac{T_{total} - \sum_{j=0}^{N} \phi_j \frac{1-\theta_j^{NM}}{1-\theta_j}}{\sum_{j=0}^{N} \eta_j \frac{1-\theta_j^{NM}}{1-\theta_j}}. \tag{16}$$

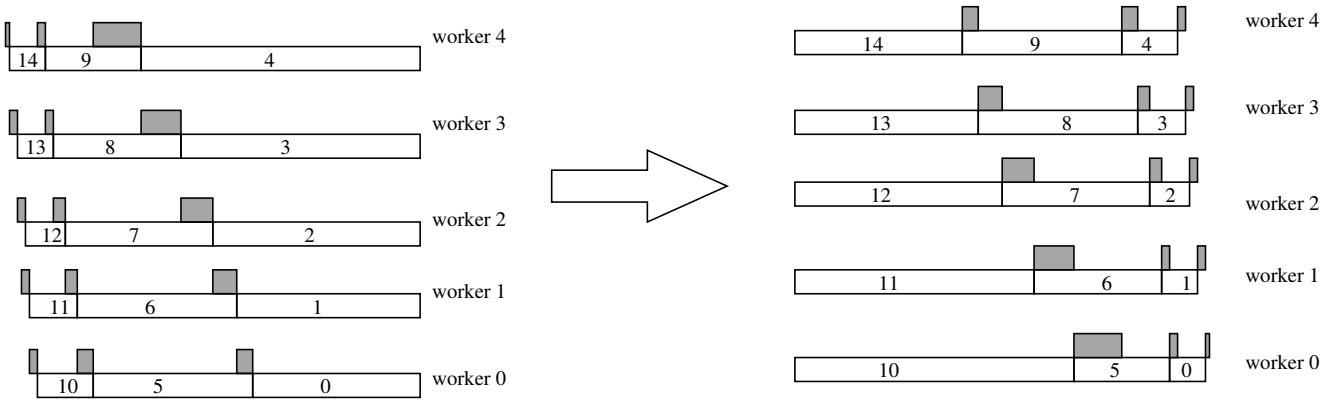


Figure 4. Flipping the input-only schedule to get an output-only schedule.

Based on the g_i series one can then easily compute the chunk sizes $chunk_i$ since $chunk_i = g_i \times S$. This completes our derivation of a closed-form solution for multi-installment scheduling on a homogeneous platform with affine costs for both computation and communication, which is a direct improvement over the work in [10]. This was achieved by rewriting the classical chunk size induction relations in a form that is more uniform and more tractable than those developed in previous work.

5 Incorporating Output Data Transfers

In this section, we derive solutions for multi-installment scheduling when output data transfers are considered. In Section 4 we have discussed situations in which chunks of input data are sent to the workers and no output is received. The other extreme is when no input data is sent but output data is sent by each worker back the the master at each round. The latter case can just be seen as "flipping" the chunk orders in the former case. This is illustrated in Figure 4.

When both input and output data are transfered between the master and the workers, the schedule is a combination of the above two scenarios, as depicted in Figure 5. In the first round, the master sends a chunk of input data to each worker. Then, the master sends a new chunk of input to the first worker just before this worker finishes computing its current chunk. Immediately afterwards the master receives output from the first worker. This ensures that network utilization is maximized. The same process continues for all workers until the last round. In the last round, each worker returns output data to the master. Note that in this scenario we assume that workers are able to compute and transfer at the same time, but only one transfer (either input or output) can take place on the link at one time: we assume a single communication channel.

We first develop our solution with linear cost models in Section 5.1 and extend to affine costs in Section 5.3.

5.1 Linear Cost Models

As in Section 4 we denote by g_i the computation time of the i^{th} chunk of workload on a worker. With a linear cost model, we model the time to transfer the i^{th} chunk of workload to a worker, T_{input_i}, and the time to transfer the output of the computation of this chunk, T_{output_i}, as:

$$T_{input_i} = g_i/R \quad \text{and} \quad T_{output_i} = g_i/R',$$

where R is the computation-communication ratio for the platform (i.e., B/S) and R' is the effective communication ratio for output transfers (i.e., $R' = R \times \delta$ where δ is the number of bytes of output data

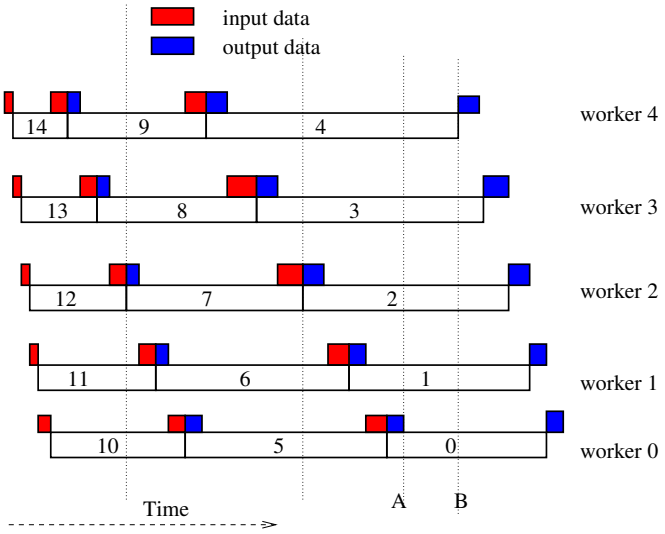


Figure 5. Schedule with both input and output.

generated for each byte of input data for the target application).

We can now derive equations for the recursion on chunk sizes as was done in Section 4.1. The compute time of each chunk is equal to the sum of the input transfer times for the N subsequent workers, plus the the sum of the output transfer times for the N previous workers. This can be written as:

$$g_i = \sum_{j=i-N}^{i-1} g_j/R + \sum_{j=i+1}^{i+N} g_j/R'$$

For example, one can see in Figure 5 that the compute time for chunk 7 is equal to the sum of the input transfer times for chunks 2 to 6, plus the sum of the output transfer times for chunks 8 to 12. The above equation is true for $N \leq i < MN$. As in Section 4.1 a modification is needed for the last round. In the last round, the network channel between the master and the workers remains idle between the transfer of output back to the master for the N^{th} chunk and for the $(N-1)^{\text{th}}$ chunk. Let Δ denote the duration of this idle time, which is illustrated as the time interval between times A and times B in Figure 5. For the last round, i.e. for $0 \leq i < N$ one can then write:

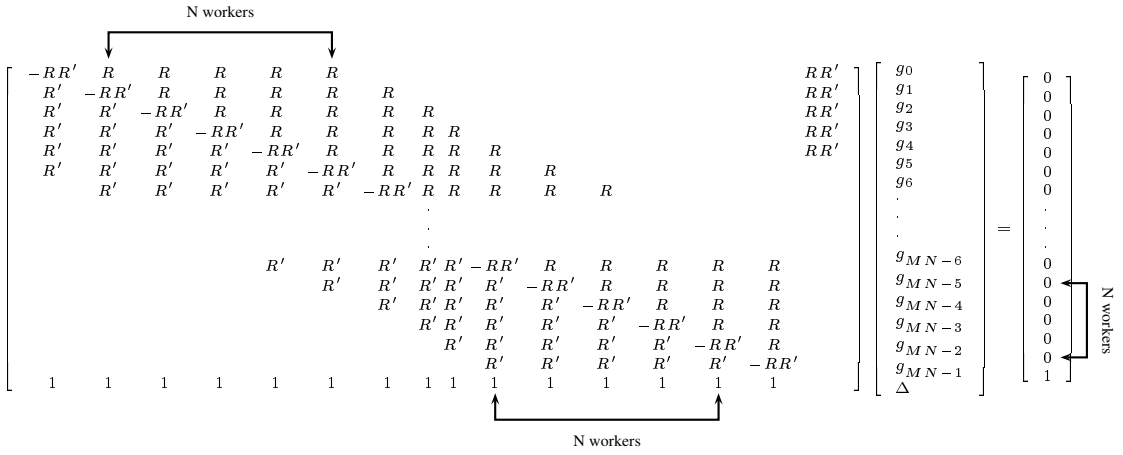
$$g_i = \sum_{j=i-N}^{i-1} g_j/R + \sum_{j=i+1}^{i+N} g_j/R' + \Delta.$$

For out-of range i, we just let $g_i = 0$. We can summarize our recursion as:

$$\begin{cases} \forall\ N \leq i < MN & g_i = \displaystyle\sum_{j=i-N}^{i-1} g_j/R + \sum_{j=i+1}^{i+N} g_j/R' \\ \forall\ 0 \leq i < N & g_i = \displaystyle\sum_{j=i-N}^{i-1} g_j/R + \sum_{j=i+1}^{i+N} g_j/R' + \Delta \\ \forall\ i \geq MN \text{ or } i < 0 & g_i = 0. \end{cases} \tag{17}$$

Our goal is to solve this recursion. Although we have employed generating functions to solve the recursion in Section 4, there is a key difference here. The g_i series in Section 4 was an infinite series, with the computation of g_i requiring only values of g_j for $j < i$, which made it possible to ignore g_i

values for $i \geq MN$. For the above recursion however, the computation of g_i involves g_j values for $j < i$ **and** for $j > i$. This requires to set g_i values to 0 for $i \geq MN$, which makes the g_i series finite and thus precludes the use of generating functions. The alternative, given that the g_i series is finite, is to express the above recursion as a set of MN linear equations:



or

$$\mathcal{C}\mathcal{G} = \mathcal{A} \tag{18}$$

with

$$\mathcal{G} = [g_0, g_1, g_2, \cdots, g_{MN-1}, \Delta]'.$$

The chunk sizes can then be computed as the solution of the linear system in Eq. 18. Note that the last row of matrix $\mathcal{C}$ corresponds to the normalization condition that the chunks must add up to the total workload.

5.2 Analysis of Results

We wish to gain some understanding of the property of the solution to the linear system identified in the previous section. More specifically, we wish to obtain at least empirical answers to these two questions: (i) under which conditions are all workers utilized? and (ii) what is the asymptotic application makespan when $N \to \infty$?

The first question is equivalent to: under which conditions are all components of $\mathcal{G}$ in Eq. 18 positive? Or, equivalently, under which conditions is the right-most column of $\mathcal{C}^{-1}$ strictly positive? Providing an analytical answer seems intractable. Instead, we turn to trying to determine the condition for full platform utilization empirically. For instance, in a simplified situation in which $R = R'$, for $N = 10$ and $M = 5$, if $R > 18.4955$ then $\mathcal{G} > 0$, whereas if $R \leq 18.4955$, $\mathcal{G} \not> 0$. In this specific instance, $R^* = 18.4955$ seems to be a threshold for full platform utilization. We plotted such empirical R^* values for $N = 5 \sim 10$ and $M = 1 \sim 20$ in Figure 6.

It is interesting to note that the threshold value R^* depends on the number of rounds M. This is different from multi-installment with no output data transfers, where the threshold is strictly $R > N$ [10]. From Figure 6 we also see that for each N, the threshold gradually approaches $2N$ as M increases. Consequently, $R > 2N$ seems to be a valid condition for full platform utilization for applications with $R = R'$. In other experiments not reported here in which $R \neq R'$, we found a similar empirical condition: $R + R' > 4N$. We leave an analytical development of these bounds for future work.

We also take an empirical approach to answer question (ii) above. Figure 7 plots the components of $\mathcal{G}$ for $N = 10$, $M = 50$, and $R = 20$. These results are similar to those obtained for other values. One can see that as M increases the application makespan decreases exponentially and tends to T_{total}/N as $M \to \infty$. This is due to the fact that for large M, the first N and the last N chunks make up

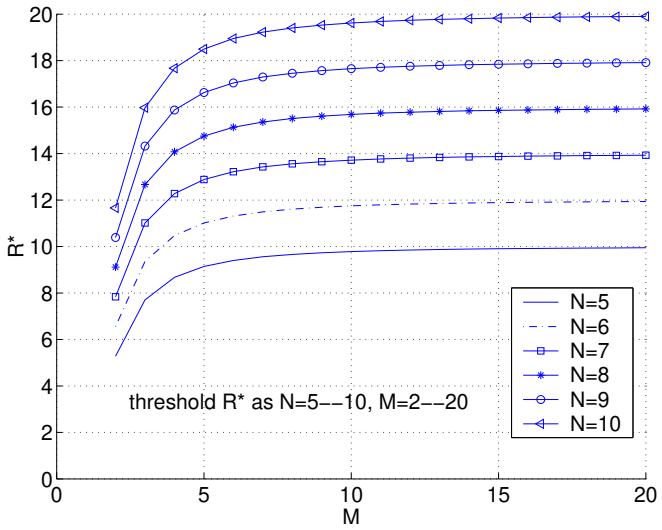


Figure 6. R^* **threshold values for full platform utilization,** $N = 5 \sim 10$**,** $M = 1 \sim 20$**.**

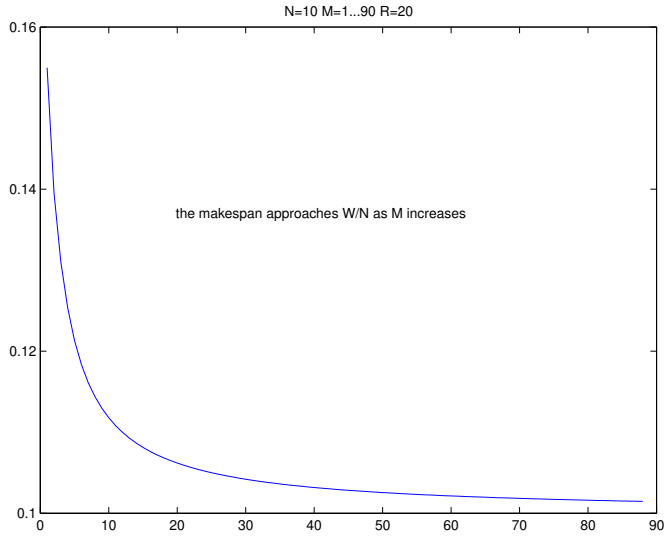


Figure 7. Asymptotic trend of makespan as $M \to \infty$

an insignificant fraction of the total workload, meaning that the time spent transferring these chunks approaches 0.

5.3 Affine Cost Models

We use exactly the same approach as in Section 5.1. With an affine cost model the time to transfer the i^{th} chunk of workload to a worker, T_{input_i}, and the time to transfer the output of the computation of this chunk, T_{output_i}, as computed as:

$$T_{input_i} = \beta + g_i/R \quad \text{and} \quad T_{output_i} = \beta' + g_i/R',$$

where β' is the network latency incurred for transfer output data (which may conceivably be different from β). Also, the time to compute the i^{th} chunk, T_{comp_i}, is:

$$T_{comp_i} = \alpha + g_i.$$

One can then rewrite Eq. 17 as:

$$
\begin{cases}
\forall\ 0 \le i < N & g_i + \alpha = \displaystyle\sum_{j=i-N}^{i-1} g_j/R + \sum_{j=i+1}^{i+N} g_j/R' + N\beta' + i\beta \\
\forall\ N \le i < (M-1)N & g_i + \alpha = \displaystyle\sum_{j=i-N}^{i-1} g_j/R + \sum_{j=i+1}^{i+N} g_j/R' + N(\beta + \beta') \\
\forall\ (M-1)N \le i < MN & g_i + \alpha = \displaystyle\sum_{j=i-N}^{i-1} g_j/R + \sum_{j=i+1}^{i+N} g_j/R' + \Delta + N\beta + (MN-1-i)\beta' \\
\forall\ i \ge MN \ \text{ or } \ i < 0 & g_i = 0
\end{cases}
$$

Δ is exactly as it was defined in Section 5.1. As in Section 5.1 we can solve the above recursion with a set of linear equations:

N workers

$$
\begin{bmatrix}
-RR' & R & R & R & R & R & & & & & & & & & & RR' \\
R' & -RR' & R & R & R & R & R & & & & & & & & & RR' \\
R' & R' & -RR' & R & R & R & R & R & & & & & & & & RR' \\
R' & R' & R' & -RR' & R & R & R & R & R & & & & & & & RR' \\
R' & R' & R' & R' & -RR' & R & R & R & R & R & & & & & & RR' \\
R' & R' & R' & R' & R' & -RR & R & R & R & R & & R & & & & \\
 & R' & R' & R' & R' & R' & -RR' & R & R & R & R & R & & & & \\
 & & & & & & & \cdot & & & & & & & & \\
 & & & & & & & \cdot & & & & & & & & \\
 & & & & & & & \cdot & & & & & & & & \\
 & & & & R' & R' & R' & R' & R' & -RR' & R & R & R & R & R & \\
 & & & & & R' & R' & R' & R' & R' & -RR' & R & R & R & R & \\
 & & & & & & R' & R' & R' & R' & R' & -RR' & R & R & R & \\
 & & & & & & & R & R' & R' & R' & R' & -RR' & R & R & \\
 & & & & & & & & R' & R' & R' & R' & R' & -RR' & R & \\
 & & & & & & & & & R' & R' & R' & R' & R' & -RR' & \\
1 & 1 & 1 & \cdots & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 &
\end{bmatrix}
\begin{bmatrix}
g_0 \\ g_1 \\ g_2 \\ g_3 \\ g_4 \\ g_5 \\ g_6 \\ \cdot \\ \cdot \\ \cdot \\ g_{MN-6} \\ g_{MN-5} \\ g_{MN-4} \\ g_{MN-3} \\ g_{MN-2} \\ g_{MN-1} \\ \Delta
\end{bmatrix}
= -RR'
\begin{bmatrix}
N\beta' - \alpha \\ N\beta' + \beta - \alpha \\ N\beta' + 2\beta - \alpha \\ N\beta' + 3\beta - \alpha \\ N\beta' + 4\beta - \alpha \\ \cdot \\ \cdot \\ \cdot \\ \cdot \\ \cdot \\ 4\beta' + N\beta - \alpha \\ 3\beta' + N\beta - \alpha \\ 2\beta' + N\beta - \alpha \\ \beta' + N\beta - \alpha \\ N\beta - \alpha \\ T_{total}
\end{bmatrix}
$$

N workers

N workers

N workers

or

$$\mathcal{C}'\mathcal{G}' = \mathcal{A}',$$

where

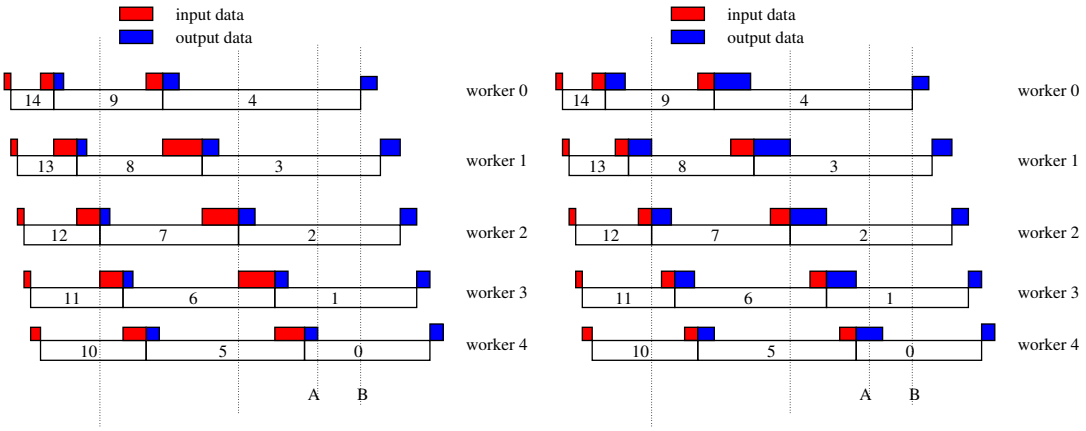
$$\mathcal{G}' = [g_0, g_1, g_2, \cdots, g_{MN-1}, \Delta]'.$$

5.4 An Open Question: Dual Channels

Consider the case when we allow the sending channel and receiving channel to work independently, that is, a worker sends back results at the same time that the next worker receives its input. The final schedule depends on which activity occupies the link longer, sending or receiving. The schedule is then determined by this activity and is similar to one of the two extreme cases mentioned in Section 5.1. When transfers of input data take longer, the schedule is approximately determined by Eq. 3 and is depicted on Figure 8(a); when transfers of output data take longer, the schedule is approximately determined by Eq. 17 and depicted on Figure 8(b). However, in both cases, the beginning and the end of the application execution are more complicated. A question worthy of investigation would be that of the division between the two cases. We leave this for future work.

6 Conclusion

In this paper we have presented multi-round algorithms for scheduling divisible workload on homogeneous star networks. We set out to answer two open questions: (i) Given a number of rounds, is it possible to obtained a schedule analogous to the ones given in [9] for homogeneous platform, but with



(a) When input data transfers take longer, the schedule is determined by the input recursion equations – Eq. 3.

(b) When output data transfers take longer, the schedule is determined by the output recursion equations – Eq. 17.

Figure 8. Two possible schedules for a dual-channel network.

more realistic affine cost models for computation and communication? and (ii) is it possible to obtain such closed-form solution while accounting for non-negligible output transfer time? Our contributions were as follows. First, building on the work in [9], we have developed new closed-form solution for multi-installment scheduling on homogeneous platform with affine cost models. This was achieved via a new formulation of the well-known chunk size recursion and using generating functions. Second, we gave chunk size recursions for the scheduling problem with output data transfers. We solved these recursions via sets of linear equations. We also gave an empirical analysis of the conditions for full platform utilization.

A corollary of using affine cost models is that there is a clear trade-off for multi-round scheduling: on the one hand dividing the workload into small chunks (i.e. many rounds) makes it possible to overlap communication with computation effectively; on the other hand dividing the workload into large chunks (i.e. few rounds) reduces the overhead due to latencies, and thus the overall makespan. This implies that there is an optimal number of rounds for multi-round scheduling, but determining this number of rounds is an open question. In [33] we have proposed UMR, a heuristical multi-round algorithm that tolerates latencies and computes an approximately optimal number of rounds under a some restrictions. In future work we will compare the efficacy of the new multi-installment algorithm developed in this paper with that of UMR on homogeneous platforms.

Acknowledgments

The authors would like to thank Dr. Mo from the Dept. of Mathematical and Statistical Sciences, University of Alberta, Canada for his help with the proof in Appendix A.

A On the degree of the roots of $Q(x)$

In Section 4.1 and 4.2 we have mentioned that a condition for using the partial fraction and rational expansion method for computing the generating functions was that the denominator polynomial, $Q(x)$, has only roots of degree 1. If it were not the case we would have to use the more involved General

Rational Expansion Theorem [23] to convert the generating function to power series. We prove here that:

(i) if $R \neq N$ all roots of $Q(x)$ are of degree 1; and

(ii) if $R = N$ the *only* root of $Q(x)$ with degree higher than 1 is $x = 1$, with degree 2.

Since $x = 1$ is clearly a root we can rewrite $Q(x)$ as:

$$Q(x) = (1 - x)P(x), \quad \text{where} \quad P(x) = (R - x - x^2 - x^3 - ... - x^N). \tag{19}$$

Proof of (i) – Let us assume that $R \neq N$. In this case, the root $x = 1$ has degree 1 since it is not a root of $P(x)$. We now focus on proving that $P(x)$ has only roots of degree 1. Assume that $P(x)$ has a root ρ (which is $\neq 1$) of degree $d > 1$. Then one can write:

$$P(x) = (x - \rho)^d F(x)$$

One can then compute the derivative of $P(x)$ as:

$$P'(x) = (x - \rho)^{d-1}(F'(x) + (x - \rho)F(x)),$$

which shows that ρ must also be a root of $P'(x)$. One can compute $P'(x)$ with Eq. 19 as:

$$P'(x) = -(1 + 2x + 3x^2 + 4x^3.... + Nx^{N-1}). \tag{20}$$

Therefore, ρ satisfies the following two equations:

$$R - \rho - \rho^2 - \rho^3 - ... - \rho^N = 0 \tag{21}$$
$$1 + 2\rho + 3\rho^2 + 4\rho^3.... + N\rho^{N-1} = 0 \tag{22}$$

Multiplying Eq. 22 by $(1 - \rho)$ gives:

$$1 + \rho + \rho^2 + ... + \rho^{N-1} - N\rho^N = 0, \tag{23}$$

which one can then add to Eq. 21 to obtain:

$$\rho = \left(\frac{R+1}{N+1}\right)^{\frac{1}{N}}. \tag{24}$$

Multiplying Eq. 23 by ρ, and add to Eq. 21, we have:

$$\rho = \left(\frac{R}{N}\right)^{(\frac{1}{N+1})} \tag{25}$$

From Eqs. 24, 25, we get:

$$\left(\frac{R}{N}\right)^N = \left(\frac{R+1}{N+1}\right)^{N+1}. \tag{26}$$

We now show that Eq. 26 implies $R = N$. Consider the function $f(x)$ defined as:

$$f(x) = (N\ln(x) - (N+1)\ln(x+1)) - (N\ln(N) - (N+1)\ln(N+1)).$$

Clearly Eq. 26 is equivalent to $f(R) = 0$. The derivative of $f(x)$ is:

$$f'(x) = \frac{N - x}{x(x+1)},$$

which is positive on the interval $(-\infty, N)$, zero at $x = N$, and negative on the interval $(N, +\infty)$. Therefore, $f(x)$ reaches its maximum at $x = N$. Since $f(N) = 0$, the *only* real root of $f(x)$ is $x = N$. Consequently, $f(R) = 0$ implies that $R = N$, which is a contradiction. As a result Eq. 21 and Eq. 22 cannot hold simultaneously, showing that $P(x)$ cannot have a root of degree higher than 1. This completes the proof that all roots of $Q(x)$ are or degree 1.

Proof of (ii) – Assume that $R = N$. As in the proof for (i) let us assume that $P(x)$ has a root ρ of degree $d > 1$. Then we know that ρ is a root of $P(x)$ and of $P'(x)$. From Eq. 24 one obtains $\rho = 1^{(\frac{1}{N})}$. Now, if $\rho = 1$ then $P'(\rho) \neq 0$. And thus ρ is not a root of $P'(x)$, which is a contradiction. If $\rho \neq 1$ then

$$P(\rho) = R - \frac{\rho(1-\rho^N)}{1-\rho} = R - \frac{1^{(\frac{1}{N})}(1 - 1^{(\frac{1}{N})^N})}{1 - 1^{(\frac{1}{N})}} = R \neq 0,$$

which shows that ρ is not a root of $P(x)$, which is a contradiction. Therefore, $P(x)$ has no root of degree higher than 1. Therefore, the only root of $Q(x)$ that has a degree higher than 1 is the root $x = 1$ and its degree is 2.

Based on (i) and (ii) we conclude that when $R \neq N$, $Q(x)$ cannot have roots with degree higher than 1; and that when $R = N$, the only root of $Q(x)$ with degree higher than 1 is $x = 1$, with degree 2.

References

[1] 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.

[2] D. Altilar and Y. Paker. Optimal scheduling algorithms for communication constrained parallel processing. In *Proceedings of Europar'02*, pages 197–206, 2002.

[3] D. Applegate and W. Cook. A computational study of the job-shop scheduling problem. 3:149–156, 1991.

[4] Bataineh, Hsiung, and Robertazzi. Closed form solutions for bus and tree networks of processors load sharing a divisible job. *IEEE Transations on Computers*, 43(10), 1994.

[5] S. Bataineh and T. Robertazzi. Bus-oriented load sharing for a network of sensor-driven processors. *IEEE transactions on systems,man and cyternetics*, 21(5), 1991.

[6] 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.

[7] 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.

[8] O. Beaumont, A. Legrand, and Y. Robert. Optimal algorithms for scheduling divisible workloads on heterogeneous systems. Technical Report 2002-36, Ecole Normale Superieure de Lyon, October 2002.

[9] Bharadwaj, Ghose, and Mani. Optimal sequencing and arrangement in single-level tree networks with communication delays. *IEEE transactions on parallel and distributed systems*, 5(9), 1994.

[10] Bharadwaj, Ghose, and Mani. Multi-installment load distribution in tree networks with delays,. *IEEE Trans. on Aerospace and Electronc Systems*, 31(2):555–567, 1995.

[11] Bharadwaj, Li, and Ko. On the influence of start-up costs in scheduling divisible loads on bus networks. *IEEE TPDS*, 11(12):1288–1305, 2000.

[12] Bharadwaj and Ranganath. Theoretical and experimental study on large size image processing applications using divisible load paradigm on distributed bus networks. *Image and Vision Computing*, 20(13-14):917–1034, 2002.

[13] Blast webpage. `http://http://www.ncbi.nlm.nih.gov/BLAST/`.

[14] Blazewicz. Performance limits of a two-dimensional network of load sharing processors. *Foundations of computing and decision sciences*, 21(1):3–15, 1996.

[15] Blazewicz and Drozdowski. Distributed processing of divisible jobs with communication startup costs. *Discrete Applied Mathematics*, 76:21–41, 1997.

[16] 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.

[17] E. Coffman. *Computer and Job shop scheduling*. Wiley, 1976.

[18] Drozdowski and Glazek. Scheduling divisible loads in a three-dimensional mesh of processors. *Parallel Computing*, 25(4), 1999.

[19] Drozdowski and Wolniewicz. Experiments with scheduling divisible tasks in clusters of workstations. In *Europar*, pages 311–319, 2000.

[20] Drozdowski and Wolniewicz. Divisible load scheduling in systems with limited memory. *Cluster Computing*, 6(1):19–29, 2003.

[21] S. Flynn Hummel. Factoring : a method for scheduling parallel loops. *Communications of the ACM*, 35(8):90–101, August 1992.

[22] Ghose and Mani. Distributed computationa with communication delays: Asymptotic performance analysis. *JPDC*, 23(3), 1994.

[23] P. Graham, Knuth. *Concrete Mathmatics*. Wiley, 1994.

[24] T. Hagerup. Allocating independent tasks to parallel processors: An experimental study. *Journal of Parallel and Distributed Computing*, 47:185–197, 1997.

[25] Hmmer webpage. `http://hmmer.wustl.edu/hmmer-html/`.

[26] Kim, Jee, and Lee. Optimal load distribution for tree network processors. *IEEE Transactions on Aerospace and Electronic Systems*, 32(2):607–611, 1996.

[27] Lee and Hamdia. Parallel image processing applications on a network of workstation. *Parallel Computing*, 21:137–160, 1995.

[28] C. Lee and M. Hamdi. Parallel image processing applications on a network of workstations. *Parallel Computing*, 21:137–160, 1995.

[29] Li. Parallel processing of divisible loads on partitionable static interconnection networks. *Cluster Computing*, 6(1):47–55, 2003.

[30] K. Li. Scheduling divisible tasks on heterogeneous linear arrays with applications to layered networks. In *IPDPS*, 2002.

[31] 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.

[32] J. Sohn and T. Robertazzi. Optimal divisible job load sharing for bus networks. *IEEE transactions on Aerospace and Electronic systems*, 32(1), 1996.

[33] Y. Yang and H. Casanova. Umr: A multi-round algorithm for scheduling divisible workloads. In *Proceedings of the International Parallel and Distributed Processing Symposium (IPDPS'03), Nice, France*, April 2003. to appear.

[34] yuan-chien Cheng and Robertazzi. Distributed computation with communication delay. *IEEE transactions on aerospace and electronic systems*, 24(6), 1988.

[35] yuan-chien Cheng and Robertazzi. Distributed computation for a tree-network with communication delay. *IEEE transactions on aerospace and electronic systems*, 26(3), 1990.

[36] Y.Yang and H. Casanova. Rumr: Robust scheduling for divisible workloads. In *Proceedings of HPDC 2003*, pages 114–123, 2003.