Scheduling Associative Reductions with Homogeneous Costs when Overlapping Communications and Computations

Louis-Claude Canon
FEMTO-ST, Université de Franche-Comté
Besançon, France
louis-claude.canon@univ-fcomte.fr

***Abstract*—Reduction is a core operation in parallel computing that combines distributed elements into a single result. Optimizing its cost may greatly reduce the application execution time, notably in MPI and MapReduce computations. In this paper, we propose an algorithm for scheduling associative reductions. We focus on the case where communications and computations can be overlapped to fully exploit resources. Our algorithm greedily builds a spanning tree by starting from the root and by adding a child at each iteration. Bounds on the completion time of optimal schedules are then characterized. To show the algorithm extensibility, we adapt it to model variations in which either communication or computation resources are limited. Moreover, we study two specific spanning trees: while the binomial tree is optimal when there is either no transfer or no computation, the k-ary Fibonacci tree is optimal when the transfer cost is equal to the computation cost. Finally, approximation ratios of strategies based on those trees are derived.**

***Keywords*—Reduction, spanning tree, scheduling.**

I. Introduction

Reduction is a generic operation that commonly occurs in parallel computations: it consists in combining several distributed elements to produce a single result, either as an intermediate or a final computational step. This operation arises, for instance, in two major parallel programming paradigms: MPI (Message Passing Interface) and MapReduce [1]. In the context of the MPI programming model, the `MPI_Reduce` collective function produces a single array by combining several arrays of the same size using an element-wise operation. More recently, the reduction problem has been brought forward again with the emergence of the MapReduce programming model: a first step (Map) filters some initial data sets into intermediate data that are then aggregated in a second step (Reduce).

When reducing several elements that are stored on separate machines, elements are first transfered before being processed. With associative reductions, processing data consists in performing binary operations on those elements. Hence, it is possible to reduce subsets of elements in parallel for minimizing the overall completion time. In this case, the execution relies on a spanning tree that determines the transfers between each pair of machines: each leaf first sends its element to its parent, which reduces then the received element with its own. This process is then repeated by discarding all leaves at each step until the root contains the final result.

Existing work on this topic has mainly focused on scheduling communications without accounting for computation costs. In this work, we consider that neither transfer nor computation costs are negligible and that each of those costs are homogeneous. It is the case, for example, when multiplying distributed square matrices: each machine has a matrix and the objective is to obtain the product of all matrices on an arbitrary machine. Depending on the matrix size and the characteristics of the platform, computation and communication costs may be comparable. Therefore, we allow communications to overlap with computations to fully exploit the resources. This overlapping occurs when any parent has several children in the spanning tree: after having received an element from a first child, the parent may start receiving a second element while processing the first one. As the communication cost may differ from the computation cost, the degree of overlapping may vary, which makes this problem difficult.

We propose an algorithm that schedules optimally transfers and reductions by relying on a spanning tree. This algorithm starts by scheduling the root that will contain the final result. Other transfers are then scheduled greedily. Moreover, this algorithm is adapted when the number of concurrent transfers or the number of machines processing data is limited. This shows the extensibility of the greedy principle that consists in choosing the parent that minimizes the completion time of each machine. Two specific tree structures are also investigated. The binomial tree [2] is a spanning tree that is known to be optimal when there is either no transfer or no computation. It is used in two major MPI implementations: MPICH2[1] and Open MPI[2]. We introduce a similar tree, the k-ary Fibonacci tree, which is optimal when all transfer and computation costs are equal. While the binomial tree minimizes the number of steps (at each step, any machine performs at most one transfer and one reduction), the k-ary Fibonacci tree optimizes the pipelining of computations by prefetching data whenever it is possible, which enables some machines to sequentially perform several computations without waiting. For each tree, a corresponding strategy is derived from the main algorithm and its approximation ratio is analytically and empirically studied. Some of these results are also available in the companion research report [3].

[1]For broadcast, reduction, allreduce and barrier operations: http://www.mcs.anl.gov/research/projects/mpich2/ (version 1.4.1p).

[2]For broadcast, reduction, gather and scatter operations: http://www.open-mpi.org/ (version 1.4.5).

The paper is organized as follows. Section II discusses the related work. The model is then detailed in Section III. Section IV presents the main algorithm, its correctness proof and bounds on the completion time of optimal schedules. Sections V and VI covers a similar analysis for an extension of this algorithm. Finally, Section VII describes the specific spanning trees.

II. Related Work

The literature has first focused on a variation of the reduction problem, the (global) combine problem [4], [5], [6]. Algorithmic contributions have then been proposed to improve MPI implementations and existing methods have been empirically studied in this context [7], [8]. Recent works concerning MapReduce either exhibit the reduction problem or highlight the relations with MPI collective functions. We describe below the most significant contributions.

Bar-Noy et al. [9] propose a solution to the global combine problem: similarly to allreduce, all machines must know the final result of the reduction. They consider the postal model with a constraint on the number of concurrent transfers to the same node (multi-port model). However, the postal model does not capture varying degree of overlapping between computations and communications.

Rabenseifner [10] introduces the *butterfly algorithm* for the same problem, with arbitrary array sizes. Several vectors must be combined into a single one by applying an element-wise reduction. The algorithm solves the reduction problem in its first phase. Then, it broadcasts the result to all machines. The main principle lies in halving arrays successively. At each step, machines exchange data pairwise (each machine first contacts its closest neighbor, then its second closest neighbor, etc). The first half of the array is sent from one machine to the other while the second half is transfered in the opposite direction. Each machine performs reductions on the half for which they both possess data. Since this half only is considered for subsequent operations, transfer sizes are divided by two at each step. When all pairwise interactions have been done, the final result is scattered across all machines. To gather it, a similar algorithm is used. Another solution has also been proposed when the number of machines is not a power of two [11]. Those approaches are specifically adapted for element-wise reduction of arrays. Van de Geijn [12] also proposes a method with a similar cost. In our case, the reduction is not applied on an array and the computation is assumed to be indivisible.

Sanders et al. [13] exploit in and out bandwidths. Although the reduction does not require to be applied on arrays, the operation is split in at least two parts. This improves the approach based on a binary tree by a factor of two.

Legrand et al. [14] study steady-state situations where a series of reductions are performed. As in our work, the reduction operation is assumed to be indivisible, transfers and computations can overlap and the full-duplex 1-port model is considered. The solution is based on a linear program and produces asymptotically optimal schedules with heterogeneous costs. On the other hand, our solution has a lower complexity, but requires homogeneous costs.

Liu et al. [15] propose a 2-approximation for heterogeneous costs and non-overlapping transfers and computations. Additionally, they solve the problem when there are only two possible speeds or when any communication time is a multiple of any shorter communication time. In the homogeneous case, their solution builds binomial trees, which are covered in Section VII-C.

In the MPI context, Kielmann et al. [16] design algorithms for collective communications, including `MPI_Reduce`, in hierarchical platforms. They propose three heuristics: flat tree for short messages, binomial tree for long messages and a specific procedure for associative reductions in which data are first reduced locally on each cluster before the results are sent to the root process. Pjesivac-Grbovic et al. [17] conduct an empirical and analytical comparison of existing heuristics for several collective communications. The analytical costs of those algorithms are first determined using different classical point-to-point communication models, such as Hockney, LogP/LogGP and PLogP. The compared solutions are: flat tree, pipeline, binomial tree, binary tree and k-ary tree. Thakur et al. [18] perform a similar study for several MPI collective operations and compare the binomial tree with the butterfly algorithm [10] for `MPI_Reduce`. These works, however, do not provide any guarantee on the performance.

Finally, this problem has also been addressed for MapReduce applications. Agarwal et al. [19] present an implementation of allreduce on top of Hadoop based on spanning trees. Moreover, some MapReduce infrastructures, such as MapReduce-MPI[3], are based on MPI implementations and benefits from the improvements done on `MPI_Reduce`. Hoefler et al. [20] further discuss how anticipated MPI-2.2 and MPI-3 features can optimize the Reduction phase.

III. Model

The model is divided into two parts. First, we characterize the operations that are performed and their costs. Then, we specify how those operations are planned and what constitutes the output of the problem. We conclude with the problem definition and a discussion of the main assumptions.

A. Platform and Application Model

Let n denote the number of elements and machines. Each element is available at time zero on a different machine. Any machine (numbered from 1 to n) may reduce two elements and produces a single element of the same nature in time c with an associative operation. Let d be the time to transfer an element between any pair of machines. As we assume that transfers overlap with computations, a machine can fetch new data for future operations while reducing two elements. However, each machine is involved in no more than one transfer at any given time (1-port model [21]). Then, the time required to process $k+1$ elements on a single machine is $d+(k-1)\max(d,c)+c$ (d for retrieving the first element with which the local element is reduced, followed by k reductions). If $d<c$, this simplifies as $d+kc$, otherwise, it is $kd+c$.

[3]http://www.sandia.gov/ sjplimp/mapreduce.html

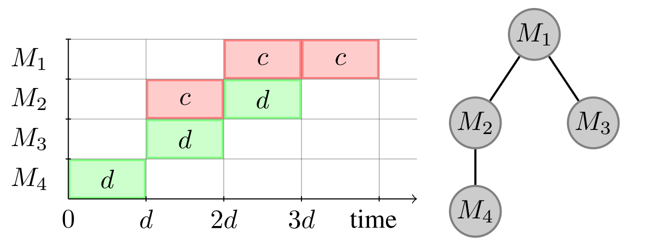

Figure 1. Schedule and corresponding spanning tree for reducing four elements ($n = 4$) when the transfer cost is equal to the computation cost ($d = c$). The transfer times are: $t_4 = 0$, $t_3 = d$ and $t_2 = 2d$. For instance, machine M_2 receives an element from M_4 between time 0 and time d, reduces it with its own between time d and time $2d$ and sends the result to M_1 between time $2d$ and time $3d$. In the case of the matrix multiplication $A \times B \times C \times D$ with A on M_1, B on M_2, C on M_3 and D on M_4, the following circular permutation of the machines produces a valid schedule regarding the commutativity: $M_1 \to M_2$, $M_2 \to M_4$, $M_4 \to M_3$ and $M_3 \to M_1$ (each operation consists then in multiplying the local element with the received one in the correct order).

B. Scheduling Model

All the transfers are characterized by a rooted tree, called *spanning tree*, in which each vertex represents a machine and each edge (i, p_i) represents a transfer from machine i to its parent p_i. Moreover, the transfer of the final element computed by i (or its initial element if i is a leaf) starts at time t_i. Any machine reduces every element it receives with its own pairwise as soon as possible. The *root* of the spanning tree, which contains the final result, is arbitrary (its parent and its transfer time are both left undefined). Finally, the *schedule* characterizing the succession of transfers and reduction operations (parent and transfer time of each machine) is noted $\{p_i, t_i\}_{1 \le i \le n}$.

As machines are assumed to be homogeneous, their order has no impact on any schedule structure. The machines can thus be permuted in any optimal schedule such that the non-commutative property of the reduction operation is respected (see Figure 1 for an example). In particular, if the root has two children which are the roots of two subtrees of sizes S_1 and S_2 (with $n = S_1 + S_2 + 1$), then the first S_1 elements of an expression with a non-commutative operator can be assigned recursively to the machines of the first subtree, while the last S_2 elements are assigned to the second subtree. This permutation can thus be found in linear time.

C. Problem Definition

The problem consists in determining a schedule $\{p_i, t_i\}_{1 \le i \le n}$ with an optimal *length*, i.e., with the shortest completion time. Figure 1 illustrates an optimal schedule and its corresponding spanning tree for reducing four elements.

D. Discussion

While each machine starts with an initial element in the MPI reduce collective operation, it is not necessarily the case in MapReduce applications. This assumption holds when each mapper becomes a reducer because reducers have then access to the data resulting from the mappers. On the contrary, when reductions are performed on distinct machines, initial elements need to be retrieved locally on reducers. However, once this retrieval phase is complete, our assumption holds because each reducer possesses one initial element.

We also assume that all elements are available simultaneously. This is valid for the `MPI_Reduce` collective because it is blocking. However, for MapReduce applications, mappers can be executed in waves. Our model does still apply when the time taken by each wave of mappers is sufficient for reducing all the available results, in which case the following algorithms may be used.

Transfer and computation costs are furthermore supposed to be constant. While this is application-specific for the computation costs, transfer costs on non-dedicated platforms such as clouds may be impacted by high variability even with constant message lengths. The strength of this hypothesis depends thus on the application and on the platform. Exploring situations with high variability and heterogeneity is outside the scope of this article but may be explored in future work.

As schedules are static, their robustness to dynamic changes and faults is limited. Those methods, however, require no synchronisation between machines since the reduction tree can be built identically on each machine (only the identifier of each machine needs to be globally known). Moreover, this study allows to highlight the structure of the problem such as the two specific reduction trees that are characterized in Section VII and constitutes a first step for further work.

IV. Greedy Algorithm

This section presents the main greedy algorithm. Before explaining it, we analyse the transfer times of any feasible schedule and we remark that the problem can be simplified. In order to prove its correctness, we outline the general structure of any greedy algorithm. Finally, bounds on the length of optimal schedules are proposed.

A. Default Transfer Times

We first characterize a lower bound on each transfer time, i.e., the time at which any machine finishes its reductions and starts sending the resulting data. Let m_i be the number of children of machine i (i.e., $m_i = |\{j : p_j = i\}|$) and f_i be the time at which machine i finishes its reductions and is ready to transfer its data to p_i (if p_i is already receiving an element at time f_i, the transfer is postponed and $f_i < t_i$). For any machine i that is a leaf, f_i is set to zero. Moreover, t_i^j is the jth shortest transfer time among all children of machine i.

Lemma 1: Any machine i that is not a leaf finishes its reductions at time $f_i = \max_{j \in [1, m_i]}(t_i^j + d + (m_i - j)\max(d, c) + c)$.

Proof: We prove by induction on k that the reduction of the first k elements received by machine i finishes at time $f_i^k = \max_{j \in [1,k]}(t_i^j + d + (k - j)\max(d, c) + c)$ and, hence, that machine i is ready to transfer its data at time $f_i = f_i^{m_i}$.

Induction basis: the case for $k = 1$ is trivial since the reduction requires a single transfer and a single reduction, and thus finishes at time $f_i^1 = t_i^1 + d + c$.

Induction step: there are two cases. If the transfer of the $(k+1)$th element starts before $f_i^k - \min(d, c)$, then the re-

Algorithm 1 Setting default transfer times

```
inputs
    $d$                                          {communication costs}
    $c$                                          {computation costs}
    $n$                                          {number of elements}
    $\{p_i\}_{1\leq i\leq n}$                    {spanning tree}
do
    $M \leftarrow \{i : 1 \leq i \leq n\}$
    while $M \neq \emptyset$ do
        $L \leftarrow \{i : i \in M, \nexists j \in M, p_j = i\}$    {select the leaves in $M$}
        for all $i \in L$ do
            $P \leftarrow \{j : 1 \leq j \leq n, p_j = i\}$    {select children of machine $i$}
            sort $P$ by data availability dates $f_j$ in ascending order
            $t_{\text{prev}} \leftarrow -\infty$    {longest defined transfer time among children of machine $i$}
            for all $j \in P$ do
                $t_j \leftarrow \max(f_j, t_{\text{prev}} + d)$    {$f_j$ is determined using Lemma 1}
                $t_{\text{prev}} \leftarrow t_j$
            end for
        end for
        $M \leftarrow M \setminus L$
    end while
    return $\{t_i\}_{1\leq i\leq n}$
```

duction can proceed without interruption. The computation finishes at time $f_i^{k+1} = f_i^k - \min(d,c) + d + c = f_i^k + \max(d,c)$. Using the induction hypothesis, $f_i^{k+1} = \max_{j\in[1,k]}(t_i^j + d + (k+1-j)\max(d,c) + c)$. As $t_i^{k+1} \leq f_i^k - \min(d,c)$, $f_i^{k+1} = \max_{j\in[1,k+1]}(t_i^j + d + (k+1-j)\max(d,c) + c)$ and the induction hypothesis holds for $k+1$.

Otherwise, the data transfer of the $(k+1)$th element starts after $f_i^k - \min(d,c)$ and the reduction finishing time depends only on t_i^{k+1}, i.e., $f_i^{k+1} = t_i^{k+1} + d + c$. As $t_i^{k+1} \geq f_i^k - \min(d,c)$, $f_i^{k+1} \geq f_i^k - \min(d,c) + d + c$. Using the induction hypothesis, $f_i^{k+1} \geq \max_{j\in[1,k]}(t_i^j + d + (k+1-j)\max(d,c) + c)$ and the induction hypothesis holds for $k+1$. ■

Transfer times can optimally be set to their lowest possible values with Algorithm 1. This algorithm works by peeling all the leaves at each iteration (on Line 19) until only the root remains. At each iteration, the transfer times of the previous leaves are set to their lowest values on Line 15. As the transfers to a given machine must not overlap due to the 1-port model, every t_i must be determined in a specific order. The algorithm starts then by assigning transfer times to machines whose data are available the earliest (children are sorted accordingly on Line 12). The time complexity of Algorithm 1 is $\Theta(n\log(n))$.

Proposition 1: For any spanning tree, setting the transfer times with Algorithm 1 leads to the shortest length.

Proof: Lemma 1 presents a tight lower bound for each transfer time, i.e., it corresponds exactly to the time at which the data are available. Therefore, it provides the minimum feasible value for t_j on Line 15. The additional $t_{\text{prev}} + d$ term avoids $p_j = i$ to be involved in two concurrent transfers. This term depends on the order (determined on Line 12) on transfers from children to machine i. We can show by contradiction that any other order cannot decrease f_i, the time at which machine i is ready to transfer data, and thus cannot reduce the schedule length. If it was the case, then the data on one of the child j would have to be available before f_j which contradicts Lemma 1. ■

Algorithm 2 Greedy scheduling algorithm

```
inputs
    $d$                                          {communication costs}
    $c$                                          {computation costs}
    $n$                                          {number of elements}
do
    $s_1 \leftarrow 0$
    for $i \leftarrow 2$ to $n$ do
        $M \leftarrow \text{argmin}_{j\in[1,i-1]} s_j$
        $s_i \leftarrow s_M + c + d$
        $s_M \leftarrow s_M + \max(d,c)$
        $p_i \leftarrow M$
    end for
    compute $\{t_i\}_{1<i\leq n}$ with Algorithm 1
    return $\{p_i, t_i\}_{1<i\leq n}$
```

Thus, this algorithm associates a schedule to each possible spanning tree in the best possible way. In the following, we focus on building an efficient spanning tree (i.e., a tree whose corresponding schedule is efficient).

B. Algorithm Description

The main solution is given by Algorithm 2 and is explained below.

This algorithm builds a schedule by considering a *reverse* reduction operation that would first compute an element before transferring it to another machine (instead of receiving an element before processing it). This resembles a broadcast of distinct elements (or scatter) with intermediate computations. When time is reversed, s_i is the earliest time at which an element can be computed on machine i and then be transfered (machine i is thus available for a transfer at time $s_i + c$).

More specifically, the algorithm relies on a greedy principle: machines are successively inserted into a tree in an arbitrary order and the greedy strategy performed on Line 8 selects the machine M that provides an element to machine i the earliest (i.e., the machine with the lowest s_j). Machine i is ready for a new computation as soon as the previous transfer completes (Line 9). The algorithm updates the end date of machine M on Line 10: machine M must have completed its computation and its transfers must be finished when the next computation occurs. The spanning tree is finally updated on Line 11. As machine 1 stores the final reduced data, its parent is left undefined. Before returning the schedule, transfer times are set using Algorithm 1 on Line 13.

The schedule depicted on Figure 1 may be obtained using this algorithm starting from time $4d$ (time flows then from right to left). In the first iteration, M_2 is connected to M_1 with $s_1 = d$ and $s_2 = 2d$. In the second iteration, M_3 is also connected to M_1 with $s_1 = 2d$ and $s_3 = 3d$. In the last iteration, M_4 may finally be connected to M_2 (M_1 is also eligible) with $s_2 = 3d$ and $s_4 = 4d$.

If we assume that an appropriate data structure is used for storing each value s_i (e.g., a self-balancing binary search tree), then Algorithm 2 requires $\Theta(n \log(n))$ steps (n iterations with a logarithmic search on Line 8).

C. Correctness

The scheduling algorithm proposed in this paper relies on a *greedy* strategy for building the spanning tree, i.e., the parent of each machine is assigned once according to some optimisation criterion and this decision is never taken back. Its correctness proof is based on Lemma 2, which states that any tree (including the optimal ones, that is the trees of optimal schedules) can be built greedily by iterating over the machines and by selecting the parent of each machine among the set of visited machines.

We generalize the structure of this greedy approach as follow. Let $O \in \mathcal{O}$ denote a specific order in which machines are visited. $G \in \mathcal{G}$ is a greedy strategy, i.e., a function whose inputs are the set of visited machines and the currently visited one while its output is the parent of the currently visited machine (among the previously visited machines). The general algorithmic procedure can be modeled as a function A whose arguments are an order and a greedy strategy and whose image is a tree. Lastly, two trees $(T, T') \in \mathcal{T}^2$ are considered to be structurally equivalent, i.e., $T \equiv T'$, if and only if their respective spanning trees are isomorphic.

Lemma 2: For any spanning tree, there exists a greedy strategy that builds a structurally equivalent spanning tree using any arbitrary order for the machines (i.e., $\forall (T, O) \in \mathcal{T} \times \mathcal{O}, \exists G \in \mathcal{G}, T \equiv A(G, O)$).

Proof: Any spanning tree can be built by adding each machine in a reverse topological order by specifying the parent of each inserted machine among the set of visited machines. Thus, any tree can be built with a specific greedy strategy and a specific order (i.e., $\forall T \in \mathcal{T}, \exists (G, O) \in \mathcal{G} \times \mathcal{O}, T = A(G, O)$).

As machines are undistinguishable, there is no distinction between machines and the vertices in any resulting spanning tree can be renumbered in any arbitrary order. Thus, any tree built using a given order is structurally equivalent to all the trees obtained with the same greedy strategy, but considering all the possible orders (i.e., $\forall (G, O, O') \in \mathcal{G} \times \mathcal{O}^2, A(G, O) \equiv A(G, O')$).

Therefore, for any tree, there exists a greedy strategy allowing the construction of a structurally equivalent tree using any arbitrary order. ■

Therefore, there exists a greedy algorithm that builds optimal schedules and the following theorem states that Algorithm 2 is such an algorithm.

Theorem 1: Algorithm 2 builds optimal schedules.

Proof: Lemma 2 implies that all structurally distinct schedules can be built by iterating over the machines in any order and by inserting a child at each iteration. Therefore, optimal schedules can be obtained using any order as it is done by Algorithm 2.

By considering a given order, we show by contradiction that there is no other greedy strategy for selecting the parent of any added node that leads to a shorter schedule length. Let $\{p'_i\}_{1<i\leq n}$ be the schedule obtained with another greedy strategy such that its length is lower than the length of the schedule $\{p_i\}_{1<i\leq n}$ built with Algorithm 2. Let k be the lowest index for which both strategies differ (i.e., $p'_k \neq p_k$). It is also assumed that $\{p'_i\}_{1<i\leq n}$ is such that there is no schedule with a better or equal length and such that the first k parents are identical to $\{p_i\}_{1<i\leq k}$ (otherwise the divergence at index k is not significant and we consider this other schedule as being $\{p'_i\}_{1<i\leq n}$ instead).

Let k' be the lowest index (necessarily greater than k) such that machine k' sends its data to p_k in schedule $\{p'_i\}_{1<i\leq n}$ ($p'_{k'} = p_k$). If this index exists, the indexes of machines k and k' can be permuted without increasing the length. If there is no such index k', then machine k can send its data to p_k instead of p'_k as Line 8 guarantees that this is the best choice. In both cases, it contradicts the assumption that there is no schedule with a better or equal length and such that the first k parents are identical to $\{p_i\}_{1<i\leq k}$. ■

D. Bounds on the Schedule Length

The following two propositions characterize lower and upper bounds on the length of any optimal schedule.

Proposition 2: The optimal length for reducing n elements is greater than or equal to $\lceil \log_2(n) \rceil \max(d, c) + \min(d, c)$.

Proof: The proof is by induction on the number of elements. The induction hypothesis H_k is that the optimal length for reducing $n = 2^k + 1$ elements is greater than or equal to $\lceil \log_2(n) \rceil \max(d, c) + \min(d, c) = (k+1) \max(d, c) + \min(d, c)$.

Induction basis: for $k = 0$, there is only one possible schedule with two elements and it is thus optimal. Its length is $d + c$, which is equal to $\max(d, c) + \min(d, c)$.

We show by contradiction that H_{k+1} is true if H_k is true. Assume that the length of the optimal schedule S for reducing $2^{k+1} + 1$ is lower than $(k+2) \max(d, c) + \min(d, c)$. From schedule S, we can build two schedules by ignoring the last reduction that is related to the last data sent to the root. Let S_1 be the schedule having the same root as S without the sub-tree whose root sends the last data to S (the schedule corresponding to this last sub-tree is denoted by S_2). The length of S_1 is lower than $(k + 2) \max(d, c) + \min(d, c) - \max(d, c)$ (for the case where $d > c$, we can remark that each pipelined reduction waits for an element to be received) and the length of S_2 is lower than $(k + 2) \max(d, c) + \min(d, c) - (d + c)$. Both lengths are thus lower than $(k + 1) \max(d, c) + \min(d, c)$. As the number of elements reduced by S is $2^{k+1} + 1$, one of the size of S_1 or S_2 must be greater than or equal to $2^k + 1$. Hence, the optimal length for reducing $2^k + 1$ elements is lower than $(k + 1) \max(d, c) + \min(d, c)$, which contradicts H_k.

The proof is completed by remarking that the optimal length is monotonically non-decreasing when the number of elements to reduce increases. Let $n' = 2^{\lfloor \log_2(n-1) \rfloor} + 1$ be the highest value not greater than a given number of elements n and for which the previous induction provides a lower bound. Therefore, the optimal length for reducing n elements is greater than or equal to $\lceil \log_2(n') \rceil \max(d, c) =$

Algorithm 3 Greedy scheduling algorithm with a limited number of concurrent transfers

```
1: inputs
2:   $d$                        {communication costs}
3:   $c$                          {computation costs}
4:   $n$                  {number of elements to reduce}
5:   $K$       {maximum number of concurrent transfers}
6: do
7:   $t_1 \leftarrow 0$
8:   $s_1 \leftarrow 0$
9:   for $i \leftarrow 2$ to $n$ do
10:    $M \leftarrow \mathrm{argmin}_{j\in[1,i-1]} s_j$
11:    $t_i \leftarrow \max(s_M + c, t_{\max(1,i-K)}) + d$
12:    $s_i \leftarrow t_i$
13:    $s_M \leftarrow \max(s_M + c, t_i - c)$
14:    $p_i \leftarrow M$
15:  end for
16:  for all $i \leftarrow 2$ to $n$ do
17:    $t_i \leftarrow t_n - t_i$
18:  end for
19:  return $\{p_i, t_i\}_{1<i\leq n}$
```

$\lceil\log_2(2^{\lfloor\log_2(n-1)\rfloor} + 1)\rceil \max(d, c) = (\lfloor\log_2(n-1)\rfloor + 1) \max(d, c) = \lceil\log_2(n)\rceil \max(d, c)$. ■

Proposition 3: The optimal length for reducing n elements is lower than or equal to $\lceil\log_2(n)\rceil(d + c)$.

Proof: Consider a schedule that consists of several steps of length $d+c$. At each step, half of the machines that have data transfer them to the other half, which perform the reductions. Such a schedule takes $\lceil\log_2(n)\rceil$ steps and its length is greater than or equal to the optimal length. ■

The closer $\min(d, c)$ is to zero, the tighter those bounds are (they are tight when $d = 0$ or $c = 0$).

Additional weaker lower bounds that depends on the structure of the spanning tree can be derived. They provide an intuition on the structure of optimal schedules. Let $\deg$ be the maximum arity of any reduction in a schedule and depth be the depth of this tree. Then, the length of this schedule is greater than or equal to $d + (\deg -1) \max(d, c) + c$ and to $(\mathrm{depth} -1)(d + c)$. This suggests that optimal schedules have a maximum degree and a depth in $O(\log(n))$. This is actually the case for the trees presented in Section VII.

V. Limited Concurrent Transfers

This section and the next one show that Algorithm 2 can be adjusted when resources are constrained. The following algorithm assumes that there is a limit K on the number of concurrent transfers (see Algorithm 3). This limits the contention in platforms where several machines are interconnected through a network equipment that has a limited aggregated bandwidth. This algorithm is explained below, its optimality is then proved and the length of each generated schedule is characterized.

This algorithm relies on the same principle as Algorithm 2, with which it shares the same complexity, $\Theta(n \log(n))$. Transfer times must however be defined. When time is reversed, t_i corresponds to the time at which the transfer from p_i to i is completed. The time taken to complete a data transfer is computed on Line 11. The maximum operation controls the contention by delaying the current transfer if the number of concurrent transfer reaches the limit K. The final schedule is obtained by reversing the transfer times from Line 16 to Line 18 (p_1 and t_1 are left undefined as machine 1 stores the final reduced data).

Theorem 2: When no more than K concurrent transfers are allowed, Algorithm 3 builds an optimal schedule for reducing n elements.

Proof: Proving this theorem follows the same structure as the proof of Theorem 1. Transfer times must, however, be considered. The arbitrary order in which machines are visited in Algorithm 3 is supported by Lemma 2.

We consider a given order and we show by contradiction that there is no other greedy strategy for selecting the parent of any added node and for setting its transfer time that leads to a shorter schedule length. Let $\{p'_i, t'_i\}_{1<i\leq n}$ be the schedule obtained with another greedy strategy such that its length is lower than the length of the schedule $\{p_i, t_i\}_{1<i\leq n}$ built with Algorithm 3. Let k be the smallest index for which both strategies differ (i.e., $p'_k \neq p_k$ or $t'_k \neq t_k$). It is also assumed that $\{p'_i, t'_i\}_{1<i\leq n}$ is such that there is no schedule with a better or equal length and such that the first k parents and transfer times are identical to $\{p_i, t_i\}_{1<i\leq k}$ (otherwise the divergence at index k is not significant and we consider this last schedule as $\{p'_i, t'_i\}_{1<i\leq n}$ instead). In the following, we consider that time is reversed (before Line 16). There are three cases.

The case where $t'_k < t_k$ can be eliminated because the transfer between machine k and machine p'_k cannot complete before t_k. This can be proved by remarking that transfer times are monotonically non-decreasing in any schedule built by Algorithm 3 (at each iteration, the minimum value s_M of the set $\{s_i\}_{1\leq i<k}$ is increased and t_k is greater than s_M). On Line 11, there are two initialization choices. If $t_k \leftarrow t_{k-K} + d$, then a value $t'_k < t_k$ would violate the contention limit. Otherwise, $t_k \leftarrow s_M + c + d$ and the schedule $\{p'_i, t'_i\}_{1<i\leq n}$ is also invalid by definition of s_M, which is the earliest time at which a computation can occur on machine k.

The second case is when $t'_k > t_k$ and $p'_k = p_k$. Let k' be the smallest index greater than k such that $t_k \leq t_{k'} < t_k + d$. If there is no such machine, the transfer can directly be advanced to t_k without increasing the length. Otherwise, two steps need to be performed before. First, the transfer times t_k and $t_{k'}$ are exchanged. Then, the children of machines k are connected to machine k' and vice versa. This leads to a schedule with the same length as $\{p'_i, t'_i\}_{1<i\leq n}$ and such that the first k parents and transfer times are identical to $\{p_i, t_i\}_{1<i\leq k}$, which contradicts the assumption that there is no such schedule.

When $t'_k \geq t_k$ and $p'_k \neq p_k$, the permutation presented in the proof of Theorem 1 can be performed, which leads to the same previous contradiction.

On the one hand, we have shown that it is not possible to built a better schedule while respecting the contention limit. On the other hand, having more than K concurrent transfers is impossible because transfer times are monotonically non-decreasing when time is reversed. Thus, generated schedules respect the contention limit. ■

The length of any generated schedule depends on the limit K on the number of concurrent transfers. As each machine is assumed to be involved in at most one transfer at any time, there is no more than $\left\lfloor \frac{n}{2} \right\rfloor$ concurrent transfers. Thus, we consider that the limit K is lower than or equal to this hard limit.

Proposition 4: When no more than K concurrent transfers are allowed, the optimal length for reducing n elements is lower than or equal to $\lfloor \log_2(K) \rfloor (d+c) + \left\lceil \frac{n}{K} \right\rceil d + \min\left(\left\lceil \frac{n}{K} \right\rceil, \left\lceil \frac{c}{d} + 1 \right\rceil\right) c$ with $K \le \left\lfloor \frac{n}{2} \right\rfloor$.

Proof: As in the proof of Proposition 3, we consider a sub-optimal schedule that consists of several steps of length $d + c$ and that respects the constraint on the number of concurrent transfers. At each step, some machines compute and transfer data to other machines (considering that time is reversed).

The behavior of this schedule has three modes. In the first mode, the number of concurrent transfers increases from 1 to the largest power of two that is not greater than K (i.e., $2^{\lfloor \log_2(K) \rfloor}$). This first mode takes $\lfloor \log_2(K) + 1 \rfloor$ steps and results in $2^{\lfloor \log_2(K)+1 \rfloor}$ machines having an element.

During the second mode, K of those machines compute and transfer new data to K other machines at each step until $\left\lceil \frac{c}{d} \right\rceil K$ machines have data. This mode takes $\left\lceil \frac{c}{d} - 1 \right\rceil$ steps because there are between K and $2K - 1$ machines that have an element when it starts.

The third mode consists in saturating the communication. First, the previous machines compute new elements. While they proceed to the transfers, delaying them when required, they prepared new data. There is always K concurrent transfers because the machines can provide new data after a time c whereas it takes $\left\lceil \frac{c}{d} \right\rceil d$ units of time to transfer them. While the first step costs c, K new machines receive data each d units of time. As there is $n - 2^{\lfloor \log_2(K)+1 \rfloor} - \left\lceil \frac{c}{d} - 1 \right\rceil K < n - \left\lceil \frac{c}{d} \right\rceil K$ remaining machines, the third mode takes less than $c + \left(\left\lceil \frac{n}{K} \right\rceil - \left\lceil \frac{c}{d} \right\rceil\right) d$ units of time.

The proof is completed by summing the durations of those modes. The minimum term comes from a modified schedule in which the second mode expands until every machine has data. In this case, the second mode takes less than $\left\lceil \frac{n}{K} - 1 \right\rceil$ steps because $n - 2^{\lfloor \log_2(K)+1 \rfloor} < n - K$ machines require data when the second mode starts. ■

VI. Limited Reducers

In MapReduce frameworks, there may be a predefined amount of *reducers*, i.e., machines that perform reductions [1]. In this case, operations must be scheduled only on a subset of machines of size K. The other machines only transfer their data to the reducers, as the mappers do in the execution of MapReduce applications. Algorithm 4 is similar to Algorithm 3 and they both share several properties. However, the transfer times are set to their smallest values (Proposition 1) as with Algorithm 2.

In this algorithm, once a machine has been selected for a reduction on Line 9, it belongs to the subset of K reducers. In this case, this set comprises the first K visited machines. The rest of the algorithm is identical to Algorithm 2. Its cost is also $\Theta(n \log(n))$.

Algorithm 4 Greedy scheduling algorithm with a limited number of reducers

```
1: inputs
2:   $d$                              {communication costs}
3:   $c$                              {computation costs}
4:   $n$                    {number of elements to reduce}
5:   $K$        {maximum number of computing machines}
6: do
7:   $s_1 \leftarrow 0$
8:   for $i \leftarrow 2$ to $n$ do
9:     $M \leftarrow \text{argmin}_{j \in [1, \min(i-1, K)]} s_j$
10:    $s_i \leftarrow s_M + c + d$
11:    $s_M \leftarrow s_M + \max(d, c)$
12:    $p_i \leftarrow M$
13:  end for
14:  compute $\{t_i\}_{1<i\le n}$ with Algorithm 1
15:  return $\{p_i, t_i\}_{1<i\le n}$
```

Theorem 3: When no more than K reducers are available, Algorithm 4 builds an optimal schedule for reducing n elements.

Proof: Before proving the optimality of the algorithm, we first prove that the limit on the number of reducers is respected.

The parent of any machine i is selected among the set of the first K machines (Line 9). Thus, the last $n - K$ machines have no child. As only machines that receive data perform computations, no more than K reducers are used.

As in the proof of Theorem 1, we consider the smallest index k for which a strategy leading to a better schedule differs. Let $M = p_k$ be the machine selected by Algorithm 4 and $M' = p'_k$ the machine in this hypothetical better strategy. There are three cases, the first two being straightforward. If $M' \le K$, then machine M provides an element earlier or at the same time (when time is reversed) and should be selected instead. The same situation occurs when $M' > K$ and $s_{M'} \ge s_M$.

The final case is when $M' > K$ and $s_{M'} < s_M$. We show that the limit on the number of reducers is not respected because the first K machines are reducers. We first prove that when the machine with smallest $s_i + c + d$ does not belong to the first K machines, then each of the first K machines has a child. By construction, the values that are assigned to s_i at each iteration are monotonically non-decreasing. Hence, when $s_{i'} < s_i$ with $i' > i$, then s_i has been incremented at Line 11 and machine i is a reducer (it performs a computation). Thus, the expression $\min(i-1, K)$ on Line 9 is equal to K only when the first K machines are all reducers. If $M' > K$ and $s_{M'} < s_M$, then there are $K + 1$ reducers, which leads to a contradiction.

Therefore, the algorithm does not use more than K reducers and there is no other schedule with a lower length. ■

Algorithm 4 also constitutes an optimal and simpler algorithm for the case covered in Section V when the transfer cost is greater than or equal to the computation cost.

Theorem 4: When no more than K concurrent transfers are allowed and when $d \ge c$, Algorithm 4 builds an optimal schedule for reducing n elements.

Proof: As any machine that is receiving data is a reducer, we prove that the number of concurrent transfers is no more than K because Algorithm 4 limits the number of reducers to K.

The proof is completed by following the same steps as the proof of Theorem 3. In the last case, which occurs when there are already K reducers, there are also K concurrent transfers. This is due to the fact that any reducer is continuously receiving data with the computations being completely overlapped when $d \geq c$. ■

The length of the generated schedules is finally bounded by Proposition 5.

Proposition 5: When no more than K reducers are available, the optimal length for reducing n elements is lower than or equal to $\left(\lfloor \log_2(K) \rfloor + \lceil \frac{n}{K} \rceil\right)(d+c)$ with $K \leq \lfloor \frac{n}{2} \rfloor$.

Proof: When $d \geq c$, it is a direct corollary of Proposition 4 and Theorem 4. Otherwise, the proof is analogous to the proof of Proposition 4 in which the second mode expands until the end. ■

VII. Specific Spanning Trees

This section covers two strategies that exhibit specific spanning trees: binomial and k-ary Fibonacci trees. Those trees are first defined and the lengths of the corresponding schedules are characterized. Then, the situations in which they are optimal are identified. Finally, the two proposed strategies are described and their approximation ratios are studied analytically and empirically.

A. Binomial Tree

The binomial tree is a spanning tree that is already known to be optimal for broadcast operations [2].

Definition 1: A *binomial tree* of order $k > 0$ is a binomial tree of order $k-1$ whose root is the child of the root of another binomial tree of order $k-1$. A binomial tree of order 0 is a single node.

Proposition 6: The length of a schedule whose spanning tree is a binomial tree of order $k \geq 0$ is $k(d+c)$ and the number of reduced elements is 2^k.

Proof: The proof is by induction on the order k of the binomial tree.

Induction basis: for $k = 0$, the cost to reduce a single element is zero.

Induction step: the length for order $k \geq 0$ is assumed to be $k(d+c)$. By definition, the last step with a binomial tree of order $k+1$ consists in reducing two intermediate results of two binomial trees of order k. By induction hypothesis, both elements are available at time $k(d+c)$. The proof is completed by remarking that the time to transfer one element to the root and to compute it takes $d+c$. ■

The length of a binomial tree, characterized by the previous proposition, indicates that the reduction involves several steps during which data are reduced by half. Since the length of each step is $d+c$, data are first transfered before being processed. Thus, transfers do not overlap with computations in binomial trees. Moreover, since all transfers start at the same time at each step, machines send data as soon as possible. This eager strategy, however, builds optimal schedules only in the cases described by the following theorem.

Theorem 5: For any $k \geq 0$ and when $\min(d,c) = 0$, no more than 2^k elements can be reduced in $k(c+d)$ time units and a binomial tree of order k is the unique solution.

Proof: The proof is by induction on the order k of the binomial tree.

Induction basis: for $k = 0$, there is a single element. A binomial tree of order zero is thus the unique solution.

Induction step: the theorem is assumed to be true for a given order $k \geq 0$. We show by contradiction that it is also the case for $k+1$. Consider a schedule that reduces at least 2^{k+1} elements in $(k+1)(c+d)$ time units and that is not a binomial tree. As in the proof of Proposition 2, we can build two schedules from this one, both with lengths lower than or equal to $k(c+d)$ (because $d = 0$ or $c = 0$). By induction hypothesis, each of them is binomial and reduces at most 2^k elements, which leads to a contradiction. ■

As a corollary, binomial trees are optimal when either transfers or computations have negligible costs and Algorithm 2 builds such trees when the number of elements is a power of two and when $\min(d,c) = 0$.

B. k-ary Fibonacci Tree

Although the Fibonacci tree, which is binary, has already been presented [22, Section 6.2.1], we propose an alternate definition analogous to Definition 1. A similar structure is also obtained by Algorithm 1-WAY-GOSSIP-K_n [23].

Definition 2: A *k-ary Fibonacci tree* of order $k > 0$ is a k-ary Fibonacci tree of order $k-2$ whose root is the child of the root of another k-ary Fibonacci tree of order $k-1$. For order -1 and 0, the trees consist each of a single node.

Proposition 7: The length of a schedule whose spanning tree is a k-ary Fibonacci tree of order $k > 0$ is $d + (k-1)\max(d,c) + c$ and the number of reduced elements is F_{k+2}, the $(k+2)$th Fibonacci number. For order -1 and 0, no data is reduced.

Proof: The proof is by induction on the order k of the Fibonacci tree.

Induction basis: for $k = 1$, the cost to reduce two elements is $d + c$. For $k = 2$, two elements are sent successively to the root (the second transfer overlaps with the first computation). The cost is thus $d + \max(d,c) + c$.

Induction step: the proposition is assumed to be true for order k and $k+1$, with $k > 0$. By definition, the last step with a Fibonacci tree of order $k+2$ consists in reducing two intermediate results of two Fibonacci trees of orders k and $k+1$. By induction hypothesis, the first element is available at time $d + (k-1)\max(d,c) + c$ and the second at time $d + k\max(d,c) + c$. The first element can then be sent to the root at time $d + k\max(d,c) + c - \min(d,c)$ in order to overlap the transfer with the penultimate computation of the root. Transferring one element to the root and computing it takes $d + c$, which concludes the proof. ■

In contrast to the binomial tree, the length of the k-ary Fibonacci tree implies that reductions are pipelined: any machine receiving data overlaps transfers and computations such that transfers (resp., computations) can be performed successively without idle period when $d \geq c$ (resp., $d \leq c$).

Theorem 6: For any $k > 0$ and when $d = c$, no more than F_{k+2} elements can be reduced in $d+(k-1)\max(d,c)+c$ time units and a k-ary Fibonacci tree of order k is the unique solution.

Proof: The proof is by induction on the order k of the Fibonacci tree.

Induction basis: for $k = 1$, the only solution for reducing $F_3 = 2$ elements is a Fibonacci tree of order 1 with length $d + c$. Three elements can be reduced either with a chain or a Fibonacci tree of order 2. The length of the chain is $2(d + c)$ whereas the length of the Fibonacci tree is $d+\max(d,c)+c$. As there is no schedule for reducing four elements with a lower length, the theorem also holds for $k = 2$.

Induction step: the theorem is assumed to be true for order k and $k + 1$, with $k > 0$. We show by contradiction that it is also the case for $k + 2$. Consider a schedule that reduces at least F_{k+4} elements in $d + (k + 1)\max(d,c) + c$ time units and that is not a Fibonacci tree. As in the proof of Proposition 2, we can build two schedules from this one, one with length at most $d + (k + 1)\max(d,c) + c - (d + c) = d + (k - 1)\max(d,c) + c$ and another with length at most $d+(k+1)\max(d,c)+c-\max(d,c) = d+k\max(d,c)+c$ (because $d = c$). By induction hypothesis, each of them is a Fibonacci tree and the first reduces at most F_{k+2} elements while the second reduces at most F_{k+3} elements, which leads to a contradiction. ■

C. Approximation Ratios

A direct consequence of Theorems 5 and 6 is that Algorithm 2 builds binomial and k-ary Fibonacci trees for specific input values. To determine the efficiency of those trees with arbitrary costs, we propose two simplifications of the greedy algorithm, one for each tree, and we determine their approximation ratios.

The *binomial strategy* is an algorithm derived from Algorithm 2 where one of the cost (d or c) is set to zero such that $\min(d,c) = 0$. It builds binomial trees when the number of elements is a power of two, independently of the actual costs.

Theorem 7: The binomial strategy is a $\left(1 + \frac{\min(d,c)}{\max(d,c)}\right)$-approximation.

Proof: As a corollary of Proposition 6, the binomial strategy builds schedules with lengths lower than or equal to $\lceil\log_2(n)\rceil(d + c)$. The approximation ratio can be directly derived from Proposition 2. ■

As a corollary, the binomial strategy is a 2-approximation in the worst-case, i.e., when $d = c$.

The *Fibonacci strategy* is an algorithm derived from Algorithm 2 where costs are homogeneous ($d = c$). It builds k-ary Fibonacci trees when the number of elements is a Fibonacci number, independently of the actual costs.

Theorem 8: The Fibonacci strategy is a 2-approximation. When $n \to \infty$, the Fibonacci strategy is a $\log_\phi(2)$-approximation, where $\phi = \frac{1+\sqrt{5}}{2}$ is the golden ratio.

Proof: Proposition 7 characterizes the length of any schedule built with the Fibonacci strategy. When the number of elements n is the $(k + 2)$th Fibonacci number, the Fibonacci strategy builds a schedule with length $d + (k - 1)\max(d,c) + c$. Let $F_k^{(-1)}$ denote the inverse Fibonacci function that associates the number of elements to the order of a Fibonacci tree. We assume that this function is monotonically non-decreasing. Then, the previous length can be expressed as $d + (\lceil F_k^{(-1)}(n)\rceil - 3)\max(d,c) + c$, which is lower than or equal to $(\lceil F_k^{(-1)}(n)\rceil - 1)\max(d,c)$.

We then need to prove that this length is lower than or equal to twice the lower bound given by Proposition 2, i.e., to $2\lceil\log_2(n)\rceil\max(d,c)$. We show that it is the case when $F_k^{(-1)}(n) - 1 \leq 2\log_2(n)$:

$$F_k^{(-1)}(n) - 1 \leq 2\log_2(n)$$
$$\lceil F_k^{(-1)}(n)\rceil - 1 \leq \lceil 2\log_2(n)\rceil \leq 2\lceil\log_2(n)\rceil$$
$$(\lceil F_k^{(-1)}(n)\rceil - 1)\max(d,c) \leq 2\lceil\log_2(n)\rceil\max(d,c)$$

To determine the number of elements n for which the inequality $F_k^{(-1)}(n) - 1 \leq 2\log_2(n)$ holds, we establish a bound on the inverse Fibonacci function. By definition, $F_k \geq \frac{\phi^k-(1-\phi)^2}{\sqrt{5}}$. It follows that $F_k^{(-1)}(n) \leq \log_\phi(\sqrt{5}n + (1 - \phi)^2)$. The inequality $\log_\phi(\sqrt{5}n + (1 - \phi)^2) - 1 \leq 2\log_2(n)$ holds for any $n > 2$. The Fibonacci strategy is furthermore trivially optimal for a single element and for two elements.

The asymptotic approximation ratio is obtained by remarking that $F_k^{(-1)}(n) = \log_\phi(n) + \Theta(1)$. Thus, the limit of the ratio $\frac{\lceil F_k^{(-1)}(n)\rceil-1}{\lceil\log_2(n)\rceil}$ is $\frac{\log_\phi(n)}{\log_2(n)} = \log_\phi(2) \approx 1.44$ as $n \to \infty$. ■

We further study the approximation ratios for specific values of n empirically. For each number of elements, the length obtained with the binomial (resp., Fibonacci) strategy is compared to the length obtained with Algorithm 2 when $d = c$ (resp., $\min(d,c) = 0$). This is conjectured to provide the worst-case ratios for both strategies. Figure 2 depicts those ratios for $2 \leq n \leq 10000$. The worst measured ratio is 1.4 for both strategies.

VIII. Conclusion

This paper covers the reduction problem when communications overlap with computations. This is a general problem related to two major paradigms in distributed systems: MPI and MapReduce. An algorithm is introduced and we show how to extend it to two model variations. Moreover, the structure of the problem is investigated by characterizing two specific spanning trees: the well-known binomial tree and the k-ary Fibonacci tree. Strategies are derived from those trees by fixing cost parameters in the main algorithm. This leads to two strategies, which are within a factor of two from the optimal solution, and asymptotically within a factor $\log_\phi(2) \approx 1.44$ for the Fibonacci strategy. The final empirical study suggests that using an inappropriate method can lead to a performance overhead that is close to 40%.

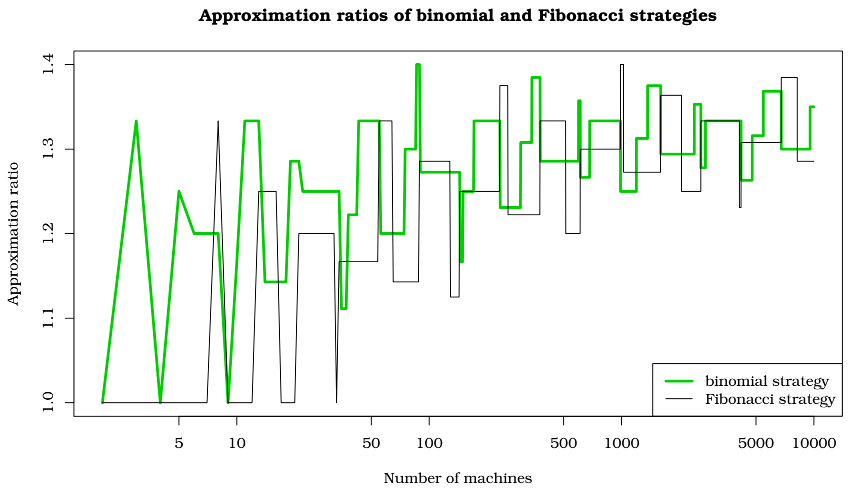

Figure 2. Ratios between binomial (resp., Fibonacci) schedule lengths and optimal lengths when $d = c$ (resp., $\min(d, c) = 0$).

As perspectives, more complex settings in the model could be explored. Although considering homogeneous costs is a preliminary step exhibiting structural properties of optimal solutions, it becomes necessary to consider heterogeneous costs when accounting for more general operations such as concatenation. Moreover, this problem frequently occurs in contexts where the topology of the network is constrained or where elements have arrival dates. A last direction would consist in designing dynamic scheduling strategies that are robust to cost variability.

References

[1] J. Dean and S. Ghemawat, "Mapreduce: simplified data processing on large clusters," *Communications of the ACM*, vol. 51, no. 1, pp. 107–113, Jan. 2008. [Online]. Available: http://doi.acm.org/10.1145/1327452.1327492

[2] S. L. Johnsson and C.-T. Ho, "Optimum broadcasting and personalized communication in hypercubes," *IEEE Transactions on Computers*, vol. 38, no. 9, pp. 1249–1268, Sep. 1989.

[3] L.-C. Canon and G. Antoniu, "Scheduling Associative Reductions with Homogeneous Costs when Overlapping Communications and Computations," INRIA, Rapport de recherche RR-7898, Mar. 2012. [Online]. Available: http://hal.inria.fr/hal-00675964

[4] A. Bar-Noy, S. Kipnis, and B. Schieber, "An optimal algorithm for computing census functions in message-passing systems," *Parallel Processing Letters*, vol. 3, no. 1, pp. 19–23, 1993.

[5] J. Bruck and C.-T. Ho, "Efficient global combine operations in multi-port message-passing systems," *Parallel Processing Letters*, vol. 3, no. 4, pp. 335–346, 1993.

[6] R. A. van de Geijn, "On global combine operations," *Journal of Parallel and Distributed Computing*, vol. 22, no. 2, pp. 324–328, 1994.

[7] Q. Ali, V. S. Pai, and S. P. Midkiff, "Advanced collective communication in Aspen," in *International Conference for High Performance Computing, Networking, Storage and Analysis*, ser. SC '08, New York, NY, USA, 2008, pp. 83–93.

[8] H. Ritzdorf and J. L. Träff, "Collective operations in NEC's high-performance MPI libraries," in *IEEE International Parallel and Distributed Processing Symposium*, ser. IPDPS, Apr. 2006.

[9] A. Bar-Noy, J. Bruck, C.-T. Ho, S. Kipnis, and B. Schieber, "Computing global combine operations in the multiport postal model," *IEEE Transactions on Parallel and Distributed Systems*, vol. 6, no. 8, pp. 896–900, Aug. 1995.

[10] R. Rabenseifner, "Optimization of Collective Reduction Operations," in *Computational Science - ICCS 2004*, ser. Lecture Notes in Computer Science, M. Bubak, G. van Albada, P. Sloot, and J. Dongarra, Eds. Springer Berlin / Heidelberg, 2004, vol. 3036, pp. 1–9.

[11] R. Rabenseifner and J. L. Träff, "More Efficient Reduction Algorithms for Non-Power-of-Two Number of Processors in Message-Passing Parallel Systems," in *Recent Advances in Parallel Virtual Machine and Message Passing Interface*, ser. Lecture Notes in Computer Science. Springer Berlin, 2004.

[12] E. W. Chan, M. F. Heimlich, A. Purkayastha, and R. A. van de Geijn, "Collective communication: theory, practice, and experience," *Concurrency and Computation: Practice and Experience*, vol. 19, no. 13, pp. 1749–1783, 2007.

[13] P. Sanders, J. Speck, and J. L. Träff, "Two-tree algorithms for full bandwidth broadcast, reduction and scan," *Parallel Computing*, vol. 35, no. 12, pp. 581–594, 2009.

[14] A. Legrand, L. Marchal, and Y. Robert, "Optimizing the steady-state throughput of scatter and reduce operations on heterogeneous platforms," *Journal of Parallel and Distributed Computing*, vol. 65, no. 12, pp. 1497–1514, 2005.

[15] P. Liu, M.-C. Kuo, and D.-W. Wang, "An Approximation Algorithm and Dynamic Programming for Reduction in Heterogeneous Environments," *Algorithmica*, vol. 53, no. 3, pp. 425–453, Feb. 2009.

[16] T. Kielmann, R. F. H. Hofman, H. E. Bal, A. Plaat, and R. A. F. Bhoedjang, "MPI's reduction operations in clustered wide area systems," 1999.

[17] J. Pjesivac-Grbovic, T. Angskun, G. Bosilca, G. Fagg, E. Gabriel, and J. Dongarra, "Performance analysis of MPI collective operations," in *IEEE International Parallel and Distributed Processing Symposium*, ser. IPDPS, Apr. 2005.

[18] R. Thakur, R. Rabenseifner, and W. Gropp, "Optimization of Collective communication operations in MPICH," *International Journal of High Performance Computing Applications*, vol. 19, pp. 49–66, 2005.

[19] A. Agarwal, O. Chapelle, M. Dudík, and J. Langford, "A Reliable Effective Terascale Linear Learning System," *CoRR*, vol. abs/1110.4198, 2011.

[20] T. Hoefler, A. Lumsdaine, and J. Dongarra, "Towards Efficient MapReduce Using MPI," in *Recent Advances in Parallel Virtual Machine and Message Passing Interface*, ser. Lecture Notes in Computer Science, M. Ropo, J. Westerholm, and J. Dongarra, Eds. Springer Berlin / Heidelberg, 2009, vol. 5759, pp. 240–249.

[21] C. Banino, O. Beaumont, L. Carter, J. Ferrante, A. Legrand, and Y. Robert, "Scheduling strategies for master-slave tasking on heterogeneous processor platforms," *IEEE Transactions on Parallel and Distributed Systems*, vol. 15, no. 4, pp. 319–330, Apr. 2004. [Online]. Available: http://dx.doi.org/10.1109/TPDS.2004.1271181

[22] D. E. Knuth, *The art of computer programming, volume 3: (2nd ed.) sorting and searching*. Addison Wesley Longman Publishing Co., Inc., 1998.

[23] J. Hromkovič, R. Klasing, B. Monien, and R. Peine, "Dissemination of information in interconnection networks (broadcasting & gossiping)," in *Combinatorial network theory*. Springer, 1996, pp. 125–212.