Non-clairvoyant reduction algorithms for heterogeneous platforms

Anne Benoit, Louis-Claude Canon, Loris Marchal

Non-clairvoyant reduction algorithms for heterogeneous platforms

Anne Benoit, Louis-Claude Canon, Loris Marchal

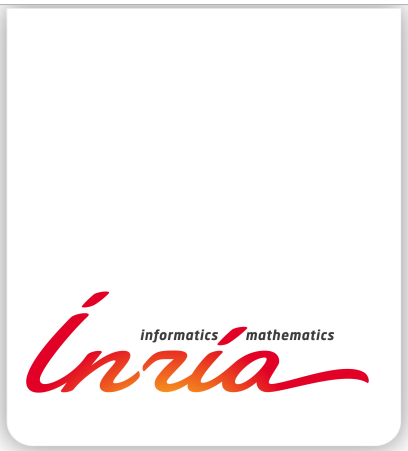
RESEARCH REPORT

N° 8315

June 2013

Project-Team ROMA

ISSN 0249-6399 ISRN INRIA/RR--8315--FR+ENG



Non-clairvoyant reduction algorithms for heterogeneous platforms

Anne Benoit*, Louis-Claude Canon†, Loris Marchal‡

Project-Team ROMA

Research Report n° 8315 — June 2013 — 19 pages

Abstract: We revisit the classical problem of the reduction collective operation in a heterogeneous environment. We discuss and evaluate four algorithms that are non-clairvoyant, i.e., they do not know in advance the computation and communication costs. On the one hand, **Binomial-stat** and **Fibonacci-stat** are static algorithms that decide in advance which operations will be reduced, without adapting to the environment; they were originally defined for homogeneous settings. On the other hand, **Tree-dyn** and **Non-Commut-Tree-dyn** are fully dynamic algorithms, for commutative or non-commutative reductions. With identical computation costs, we show that these algorithms are approximation algorithms. When costs are exponentially distributed, we perform an analysis of **Tree-dyn** based on Markov chains. Finally, we assess the relative performance of all four non-clairvoyant algorithms with heterogeneous costs though a set of simulations.

Key-words: scheduling, reduction, approximation algorithms, non-clairvoyant algorithms

* École Normale Supérieure de Lyon and IUF, Lyon, France
† FEMTO-ST, Université de Franche-Comté, Besançon, France
‡ CNRS, École Normale Supérieure de Lyon, Lyon, France

RESEARCH CENTRE GRENOBLE – RHÔNE-ALPES

Inovallée
655 avenue de l'Europe Montbonnot
38334 Saint Ismier Cedex

Algorithmes de réductions non-clairvoyants pour plates-formes hétérogènes

Résumé : Nous revisitons le problème classique de la primitive de communication collective de réduction dans le cadre d'un environnement hétérogène. Nous présentons et analysons quatre algorithmes non-clairvoyants, c'est-à-dire qui ne connaissent pas les coûts de calcul et de communication. D'un côté, **Binomial-stat** et **Fibonacci-stat** sont des algorithmes statiques qui décident a priori de l'arbre de réduction, sans pouvoir s'adapter à l'environnement au cours de la réduction. D'un autre côté, **Tree-dyn** et **Non-Commut-Tree-dyn** sont des algorithmes complètement dynamiques pour les réductions commutatives ou non-commutatives. Lorsque les coûts de calcul sont constants, nous montrons que les algorithmes commutatifs admettent des facteurs d'approximation. Lorsque les coûts de calcul et/ou de communication sont distribués selon une loi exponentielle, nous analysons **Tree-dyn** à l'aide des chaînes de Markov. Enfin, nous comparons les performances des quatre algorithmes dans un environnement hétérogène à l'aide de simulations.

Mots-clés : ordonnancement, réduction, algorithmes d'approximation, algorithmes non-clairvoyants

1 Introduction

Reduction is one of the most common collective operations, together with the broadcast operation. Contrarily to a broadcast, it consists in gathering and summarizing information scattered at different locations. A classical example is when one wants to compute the sum of (integer) values distributed over a network: each node owns a single value and can communicate with other nodes and perform additions to compute partial sums. The goal is to compute the sum of all values. Reductions have been used in distributed programs for years, and standards such as MPI usually include a "reduce" function together with other collective communications. Many algorithms have been introduced to optimize this operation on various platforms, with homogeneous or heterogeneous communication costs. We review some of them in the related work section below.

Recently, this operation has received more attention due to the success of the MapReduce framework [13, 27], which has been popularized by Google. The idea of MapReduce is to break large workloads into small tasks that run in parallel on multiple machines, and scales easily to very large clusters of inexpensive commodity computers. Hadoop [26] is the most popular open-source implementation of the MapReduce framework, originally developed by Yahoo! to manage jobs that produce hundreds of terabytes of data on thousands of cores. Examples of applications implemented with Hadoop can be found at `http://wiki.apache.org/hadoop/PoweredBy`. A crucial feature of MapReduce is to hide the complexity of distributed computing to the programmer, and to rely on robust dynamic algorithms to allocate jobs to computing nodes, detect nodes that perform poorly or have failed and re-assign jobs that slow down the process.

Our objective in this paper is to compare the performance of various algorithms for the reduce operations on dynamic environments, i.e., when the communication and computation times cannot be perfectly predicted and may vary significantly. We would like to assess how classical static algorithms perform in such settings, and to quantify the advantage of dynamic algorithms (if any). We use various techniques and models, ranging from worst-case analysis to probabilistic methods such as Markov chains. The rest of the paper is organized as follows. Section 2 reviews existing reductions algorithms and other related work. Sections 3 and 4 describe four algorithms and a simple analysis of their worst-case performance. In Section 5, we provide more involved probabilistic analysis of their expected performance. Section 6 presents simulated executions of the previous algorithms and compares their respective performance. Finally, we conclude and discuss future research directions in Section 7.

2 Related work

The literature has first focused on a variation of the reduction problem, the (global) combine problem [4, 6, 25]. Algorithmic contributions have then been proposed to improve MPI implementations and existing methods have been empirically studied in this context [2, 22]. 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. [3] propose a solution to the global combine problem: sim-

ilarly 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 [20] 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. Another solution has also been proposed when the number of machines is not a power of two [21]. Those approaches are specifically adapted for element-wise reduction of arrays. Van de Geijn [10] 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. [23] 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. [16] 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.

Liu et al. [17] propose a 2-approximation algorithm 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 3.

In the MPI context, Kielmann et al. [15] 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. [19] 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. [24] perform a similar study for several MPI collective operations and compare the binomial tree with the butterfly algorithm [20] 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. [1] present an implementation of allreduce on top of Hadoop based on spanning trees. Moreover, some MapReduce infrastructures, such as MapReduce-MPI[1], are based on MPI implementations and benefits from the improvements done on `MPI_Reduce`.

The design and the analysis of algorithms in dynamic context has already received some attention. The closest related work is probably [9], in which the authors study the robustness of several task-graph scheduling heuristics for building static schedules. The schedules are built with deterministic costs and the performance is measured using random costs. [5] studies the problem of

[1] `http://www.sandia.gov/~sjplimp/mapreduce.html`

computing the average performance of a given class of applications (streaming applications) in a probabilistic environment. With dynamic environments comes the need for robustness to guarantee that a given schedule will behave well in a disturbed environment. Among others, [8] studies and compares different robustness metrics for makespan/reliability optimization on task-graph scheduling. Optimizing the performance for task-graph scheduling in dynamic environments is a natural follow-up, and has been tackled notably using the concepts of IC and AREA-maximizing schedules [11].

3 Model and algorithms

In this section, we present the platform model and the algorithms studied throughout the paper.

3.1 Platform model

We consider a set of n processors (or nodes) $P_0, \ldots, P_{n-1}$. Each processor P_i owns a value v_i. We consider an associative operation $\oplus$. Our goal is to compute the value $v = v_0 \oplus v_2 \oplus \cdots \oplus v_{n-1}$. We do not enforce a particular location for the result: at the end of the reduction, it may be present on any node.

There are two versions of the problem, depending on whether the $\oplus$ operation is commutative or not. For example, when dealing with numbers, the reduction operation (sum, product, etc.) is usually commutative while some operations on matrices (such as the product) are not. The algorithms proposed and studied below may handle only the commutative reductions or both versions.

We denote by $d_{i,j}$ the time needed to send one value from processor P_i to processor P_j. A value may be an initial value or a partial result. When a processor P_i receives a value from another processor, it immediately computes the reduction with its current value. We assume that each processor can receive at most one result at a time. The communication costs are heterogeneous, that is we may well have different communication costs depending on the receiver ($d_{i,j} \neq d_{i,j'}$), on the sender ($d_{i,j} \neq d_{i',j}$) and non-symmetric costs ($d_{i,j} \neq d_{j,i}$). Even though these costs are fixed, we consider non-clairvoyant algorithms that make decisions without any knowledge of these costs.

The computation time of the atomic reduction on processor P_i is denoted by c_i. In the case of a non-commutative operation, we ensure that a processor sends its value only to a processor that is able to perform a reduction with its own value. Formally, assume that at a given time, a processor owns a value that is the reduction of $v_i \oplus \cdots \oplus v_j$, which we denote by $[v_i, v_j]$. It may only send this value to a processor owning a value $[v_k, v_{i-1}]$ or $[v_{j+1}, v_k]$, which is called a *neighbor value* in the following.

Note that in the following, we usually assume that we do not know the values of communication or computation costs beforehand, but only some information on them, such as lower and upper bounds or their distribution.

3.2 Reduction algorithms

During a reduction operation, a processor sends its value at most once, but may receive several values. It computes a partial reduction each time it receives a

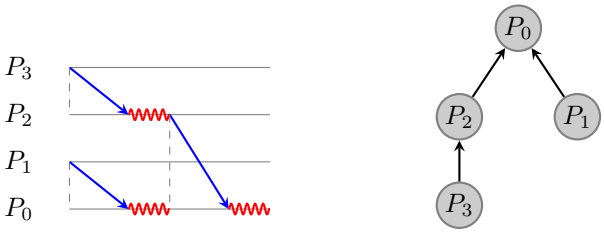
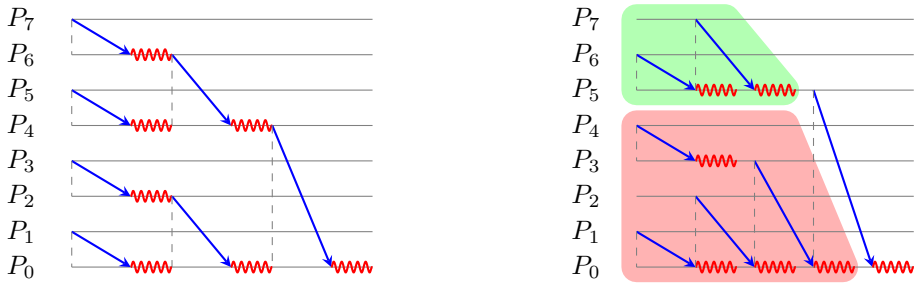


Figure 1: Schedule and communication graph for reducing four values. Blue arrows represent communications while red springs stand for computations.



(a) **Binomial-stat** (b) **Fibonacci-stat**

Figure 2: Schedules for **Binomial-stat** of order 3 and **Fibonacci-stat** of order 4, both using 8 processors. For **Fibonacci-stat**, the two schedules of order 2 and 3 used in the recursive construction are highlighted in green and red.

value. Thus, the communication graph of a reduction is a tree (see Figure 1): the vertices of the tree are the processors and its edges are the communications of values (initial or partially reduced values). In the example, P_0 receives the initial value from P_1, and then a partially reduced value from P_2. In the following, we sometimes identify a reduction algorithm with the tree it produces.

We now present the four algorithms that are studied in this paper. The first two algorithms are static algorithms, i.e., the tree is built before the actual reduction. Thus, they may be applied for commutative or non-commutative reductions. The last two algorithms are dynamic: the tree is built at run-time and depends on the durations of the operations.

The first algorithm, called **Binomial-stat**, is organized with $\lceil \log_2 n \rceil$ rounds. Each round consists in reducing a pair of processors that own a temporary or initial data using a communication and a computation. During round $k = 1, \ldots, \lceil \log_2 n \rceil$, each processor $i2^k + 2^{k-1}$ $(i = 0, \ldots)$ sends its value to processor $i2^k$, which reduces it with its own value. Note that rounds are not synchronized throughout the platform: each communication starts as soon as the involved processors are available and have terminated the previous round. We can notice that the communication graph induced by this strategy is a binomial tree [12, Chapter 19], hence the name of the algorithm. This strategy is illustrated on Figure 2(a).

The second algorithm, called **Fibonacci-stat**, is constructed in a way similar to Fibonacci numbers. The schedule constructed for order k, denoted by FS_k $(k > 0)$ first consists in two smaller order schedules FS_{k-1} and FS_{k-2} put in

parallel. Then, during the last computation of FS_{k-1}, the root of FS_{k-2} (that is, the processor that owns its final value) sends its value to the root of FS_{k-1}, which then computes the last reduction. A schedule of order -1 or 0 contains a single processor and no operation. This process is illustrated on Figure 2(b). Obviously, the number of processors involved in such a schedule of order k is F_{k+2}, the $(k+2)$th Fibonacci number. When used with another number n of processors, we compute the smallest order k such that $F_{k+2} \geq n$ and use only the operations corresponding to the first n processors in the schedule of order k.

The previous two schedules were proposed in [7], where their optimality is proved for special homogeneous cases: **Binomial-stat** is optimal either when the computations are negligible in front of communications ($c_i = 0$ and $d_{i,j} = d$) or either when the communications are negligible in front of computations ($c_i = c$ and $d_{i,j} = 0$). **Fibonacci-stat** is optimal when computations and communications are equivalent ($c_i = c = d_{i,j} = d$). In the non-commutative case, both algorithms build a tree such that only neighboring partial values are reduced. In the commutative case, any permutation of processors can be chosen.

Then, we move to the design of dynamic reduction algorithms, i.e., algorithms that take communication decisions at runtime. The first dynamic algorithm, called **Tree-dyn**, is a simple greedy algorithm. It keeps a slot (initially empty), and when a processor is idle, it looks into the slot. If the slot is empty, the processor adds its index in the slot, otherwise it empties the slot and starts a reduction with the processor that was in the slot (i.e., it sends its value to the processor that was in the slot, and the latter then computes the reduced value). It means that a reduction is started as soon as two processors are available. Since in the obtained reduction tree, any two processors may be paired by a communication, this can only be applied to commutative reductions.

Finally, **Non-Commut-Tree-dyn**, is an adaptation of the previous dynamic algorithm to non-commutative reductions. In this algorithm, when a processor is idle, it looks for another idle processor with a neighbor value (as described above). Now, we keep an array of idle processors rather than a single slot. If there is an idle neighbor processor, a communication is started between them, otherwise the processor waits for another processor to become idle.

4 Worst-case analysis for commutative reductions

We analyze the commutative algorithms in the worst case, and we provide some approximation ratios, focusing on communication times. We let $\Delta = \frac{D}{d}$, where $d = \min_{i,j} d_{i,j}$ and $D = \max_{i,j} d_{i,j}$. We consider that $c_i = c$, i.e., all computation costs are identical.

Let us first recall results from [7], in the context of identical communication costs ($d = D$) and identical computation costs (c). The duration of the schedule built by **Fibonacci-stat** at order k is $d + (k-1)\max(d, c) + c$, and the number of reduced elements n is such that $F_{k+1} \leq n \leq F_{k+2}$, where F_k is the kth Fibonacci number.

4.1 No computation cost

First, we consider that $c = 0$, i.e., communications are negligible in front of computations.

Theorem 1. *Without computation cost,* **Binomial-stat** *and* **Tree-dyn** *are* Δ*-approximation algorithms, and this ratio can be achieved.*

Proof. The proof is in two steps: we first prove the approximation ratio by showing an upper bound of the results of **Binomial-stat** and **Tree-dyn** and a lower bound of the optimal solution before showing a case that achieves this ratio.

Let us consider an algorithm that would perform reductions through $\lceil \log_2(n) \rceil$ steps. At each step, all processors possessing an element group themselves in pair for sending and reducing their elements (they perform their reductions pairwise). At each step, half the elements are processed. This algorithm takes at most a time $D\lceil \log_2(n) \rceil$, because one communication at each step may be maximum. **Binomial-stat** is better than this algorithm because it has the same structure for the reductions (the source and destination of each communication is the same), except that no synchronization is required between each global step. Indeed, communications happen as soon as possible. **Tree-dyn** is also better than this algorithm because two available processors may start as soon as possible without delaying the remaining steps. Therefore, the time taken by **Binomial-stat** and **Tree-dyn** is not greater than $D\lceil \log_2(n) \rceil$.

On the other hand, consider the minimum time taken to perform those reductions: because there are n elements to reduce, there are at least $\lceil \log_2(n) \rceil$ successive reductions. If we assume that the duration of each communication is minimum (equal to d), we obtain the lower bound $d\lceil \log_2(n) \rceil$ to complete all reductions.

Finally, the approximation ratio is $\frac{D\lceil \log_2(n) \rceil}{d\lceil \log_2(n) \rceil} = \Delta$.

Let us exhibit an instance on which the ratio is achieved. Let $d_{i,j} = d$ for $1 \leq i < j \leq n$ and $d_{i,j} = D$ for all remaining $1 \leq j < i \leq n$. With both **Binomial-stat** and **Tree-dyn**, we consider that any processor P_i sends its element to a processor P_j such that $i > j$, which takes a time $D\lceil \log_2(n) \rceil$. The optimal solution, however, consists in avoiding any communication of size D (with total time of $d\lceil \log_2(n) \rceil$). □

However, this result does not hold for **Fibonacci-stat** and **Non-Commut-Tree-dyn**:

- For **Fibonacci-stat**, consider the schedule in Figure 2(b). Even without computation costs, the number of communication steps is 4 because P_0 has to receive four messages and these communications must be sequential. If these four communications take a time D (and all other $d_{i,j}$ are d), the optimal solution may complete in a time $3d$, hence a ratio of $\frac{4}{3}\Delta$. We prove below (see Theorem 2) that **Fibonacci-stat** is in fact a $(\Delta/\log_2 \varphi + \Delta/\lceil \log_2 n \rceil)$-approximation algorithm, where $\varphi = \frac{1+\sqrt{5}}{2}$ is the golden ratio.
- For **Non-Commut-Tree-dyn**, the number of steps may well exceed $\lceil \log_2(n) \rceil$ as well. Consider for instance that for a reduction on 8 processors, in the first step neighboring processors communicate together two by two (as depicted on Figure 3). However, P_4 and P_2 terminate slightly before P_6 and P_0, and initiate a communication. Then P_6 and P_0 must wait until completion of this partial reduction, and then two more steps are required to complete the reduction. Here again, the worst-case ratio is $\frac{4}{3}\Delta$ (with $d_{i,j} = D$ if $i > j$, and $d_{i,j} = d$ otherwise). We do not prove any approximation ratio for this algorithm, because it does not seem very fair

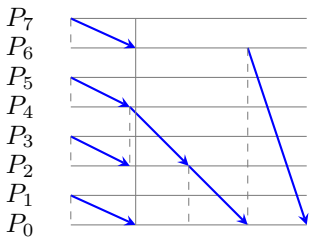


Figure 3: Schedule of a worst-case scenario for **Non-Commut-Tree-dyn**.

to compare a non-commutative version of the algorithm with the optimal commutative solution.

Theorem 2. *Without computation cost,* **Fibonacci-stat** *is a* $(\Delta/\log_2\varphi + \Delta/\lceil\log_2 n\rceil)$*-approximation algorithm, where* $\varphi = \frac{1+\sqrt{5}}{2}$ *is the golden ratio* $(1/\log_2\varphi \approx 1.44)$.

Proof. Since computation costs are negligible, the makespan of **Fibonacci-stat** schedule in the worst case is kD, with k the order of the Fibonacci schedule [7]. We know that $n > F_{k+1}$ and, by definition of the Fibonacci numbers, we have $F_{k+1} = \frac{1}{\sqrt{5}}(\varphi^{k+1} - (1-\varphi)^{k+1})$. Since $-1 < 1-\varphi < 0$, it follows that $n > F_{k+1} \geq \frac{1}{\sqrt{5}}(\varphi^{k+1} - (1-\varphi)^2)$ (as soon as $k \geq 1$). We therefore have $\varphi^{k+1} \leq \sqrt{5}n + (1-\varphi)^2$, and

$$(k+1)\log_2\varphi \leq \log_2 n + \log_2\left(\sqrt{5} + \frac{(1-\varphi)^2}{n}\right).$$

Thus

$$k \leq \frac{\log_2 n}{\log_2\varphi} + \underbrace{\frac{\log_2\left(\sqrt{5} + \frac{(1-\varphi)^2}{n}\right)}{\log_2\varphi} - 1}_{\leq 1 \text{ when } n \geq 1}$$

Thus, $k \leq \frac{\log_2 n}{\log_2\varphi} + 1 \leq \frac{\lceil\log_2 n\rceil}{\log_2\varphi} + 1$. Recall that the lower bound is $d\lceil\log_2 n\rceil$, hence the approximation ratio of $\frac{\Delta}{\log_2\varphi} + \frac{\Delta}{\lceil\log_2 n\rceil}$. □

4.2 Identical computation costs

Next, we consider arbitrary c and we derive approximation ratios with heterogeneous communication costs.

Theorem 3. *With identical computation costs* c*,* **Binomial-stat** *and* **Tree-dyn** *are* $(\Delta+1)$*-approximation algorithms.*

Proof. Similarly to the proof of Theorem 1, it is easy to see that **Binomial-stat** and **Tree-dyn** are not slower than an algorithm with $\lceil\log_2(n)\rceil$ synchronized steps, each of duration $(c+D)$ (they work with this number of steps but with communication times $d_{i,j} \leq D$ and without synchronization between steps). Thus, their execution time is upper bounded by $(c+D) \times \lceil\log_2(n)\rceil$.

The minimum time taken to perform the reductions can also bounded by $\max(c,d) \times \lceil\log_2(n)\rceil$, because there are at least $\lceil\log_2(n)\rceil$ steps, and even though communication and computation times may overlap, each step takes at least a

time $\max(c,d)$. Using this bound, we can derive an upper bound on the ratio, $R = \frac{c+D}{\max(c,d)}$. If $c \leq d$, then $R = \frac{c+D}{d} \leq \frac{d+D}{d} = \Delta + 1$. Otherwise, $d \leq c$ and $R = \frac{c+D}{c} = \frac{c+d\Delta}{c} \leq \frac{c+c\Delta}{c} = \Delta + 1$, hence the $(\Delta+1)$-approximation for **Binomial-stat** and **Tree-dyn**. □

We did not succeed to achieve the previous ratios. Indeed, strong assumptions are made on the optimal solution, and unless $c = 0$, the number of steps of an optimal solution is in fact strictly greater than $\lceil \log_2(n) \rceil$, that we used for comparison. We prove below a better approximation ratio in the case $c = d$.

Indeed, when $c = d$, if **Fibonacci-stat** uses only communications of size d, then it is optimal and it reduces n elements in time $(k+1)d$ [7]. By definition of the Fibonacci numbers, we have $F_{k+1} \leq n \leq F_{k+2}$ with $F_k = \frac{1}{\sqrt{5}}(\varphi^k + (1-\varphi)^k)$, where $\varphi = \frac{1+\sqrt{5}}{2}$ is the golden ratio ($\log_2 \varphi \approx 0.69$). Therefore, asymptotically, $k = \frac{\log_2 n}{\log_2 \varphi} + O(1)$. The design of **Fibonacci-stat** is such that communications and computations overlap, hence the good performance with identical costs.

Theorem 4. *With identical computation costs $c = d$,* **Binomial-stat** *and* **Tree-dyn** *are $(\Delta+1)(1+1/\log_2 n)\log_2 \varphi$-approximation algorithms, and this ratio can be achieved ($\log_2 \varphi \approx 0.69$).*

Proof. If $c = d$, **Fibonacci-stat** takes at least a time $(k+1)d$, assuming that it uses only communications of cost d. Because it is optimal in that case, the optimal time is at least $(k+1)d$. We have

$$n \leq F_{k+2} = \frac{1}{\sqrt{5}}(\varphi^{k+2} - (1-\varphi)^{k+2}) \leq \frac{1}{\sqrt{5}}(\varphi^{k+2} - (1-\varphi)^3)$$

as soon as $k \geq 1$. We easily derive the following inequalities:

$$\varphi^{k+2} \geq \sqrt{5}n + (1-\varphi)^3$$

$$(k+2)\log_2 \varphi \geq \log_2 n + \log_2\left(\sqrt{5} + \frac{(1-\varphi)^3}{n}\right)$$

$$k \geq \frac{\log_2 n}{\log_2 \varphi} + \underbrace{\frac{\log_2\left(\sqrt{5} + \frac{(1-\varphi)^3}{n}\right)}{\log_2 \varphi} - 2}_{\geq -1 \text{ when } n \geq 1}$$

Thus $k+1 \geq \log_2 n / \log_2 \varphi$. We still have an execution time for **Binomial-stat** and **Tree-dyn** upper bounded by $(c+D)\lceil \log_2 n \rceil$ (see proof of Theorem 3), and therefore the ratio writes $R \leq \frac{(D+c)\lceil \log_2 n \rceil}{d \log_2 n / \log_2 \varphi} \leq (\frac{D+d}{d}\log_2 \varphi)(1 + \frac{1}{\log_2 n}) = (\Delta+1)(1+\frac{1}{\log_2 n})\log_2 \varphi$.

Finally, we exhibit an example where an asymptotic ratio of $\log_2 \varphi(\Delta+1)$ is achieved. Consider that $d_{i,j} = d$ and $d_{j,i} = D$, for all $i < j$. The optimal solution performs only communications of size d (from P_i to P_j, with $j > i$), and in this case with $c = d$, a solution with a Fibonacci tree using only small communications is optimal; furthermore, it completes in time $(k+1)d$.

On the other hand, **Binomial-stat** and **Tree-dyn** perform only communications of size D (from P_j to P_i, with $j > i$), and complete in exactly a time $\lceil \log_2 n \rceil (D+c)$. Because asymptotically, $k = \frac{\log_2 n}{\log_2 \varphi} + O(1)$, the ratio is $\frac{D+c}{d/\log_2 \varphi} + O(1) = \log_2 \varphi(\Delta+1) + O(1)$. □

	$c=0$	c	$c=d$
Binomial-stat	Δ (Th. 1)	$\Delta+1$ (Th. 3)	$(\Delta+1)(1+\frac{1}{\log_2 n})\log_2\varphi$ (Th. 4)
Tree-dyn	Δ (Th. 1)	$\Delta+1$ (Th. 3)	$(\Delta+1)(1+\frac{1}{\log_2 n})\log_2\varphi$ (Th. 4)
Fibonacci-stat	$\frac{\Delta}{\log_2\varphi}+\frac{\Delta}{\lceil\log_2 n\rceil}$ (Th. 2)	$\frac{\Delta}{\log_2\varphi}+\frac{2\Delta}{\lceil\log_2 n\rceil}$ (Th. 5)	Δ (Th. 5)

Table 1: Approximation ratios for commutative algorithms with corresponding theorems.

Now consider **Fibonacci-stat** in the case of arbitrary c. We have the following theorem:

Theorem 5. *With identical computation costs c,* **Fibonacci-stat** *is a $(\Delta/\log_2\varphi + 2\Delta/\lceil\log_2 n\rceil)$-approximation algorithm $(1/\log_2\varphi \approx 1.44)$. When we further have $c=d$, the ratio becomes Δ.*

Proof. The length of the **Fibonacci-stat** schedule in the worst case is bounded by $D+(k-1)\max(D,c)+c$, with k the order of the Fibonacci schedule. We know that $n > F_{k+1}$ and, by definition of the Fibonacci numbers, we have $F_{k+1} = \frac{1}{\sqrt{5}}(\varphi^{k+1} - (1-\varphi)^{k+1})$. Since $-1 < 1-\varphi < 0$, it follows that $n > F_{k+1} \geq \frac{1}{\sqrt{5}}(\varphi^{k+1} - (1-\varphi)^2)$ (as soon as $k \geq 1$).

We therefore have $\varphi^{k+1} \leq \sqrt{5}n + (1-\varphi)^2$, and

$$(k+1)\log_2\varphi \leq \log_2 n + \log_2\left(\sqrt{5} + \frac{(1-\varphi)^2}{n}\right)$$

Thus

$$k \leq \frac{\log_2 n}{\log_2\varphi} + \underbrace{\frac{\log_2\left(\sqrt{5}+\frac{(1-\varphi)^2}{n}\right)}{\log_2\varphi} - 1}_{\leq 1 \text{ when } n\geq 1}$$

Thus, $k+1 \leq \frac{\log_2 n}{\log_2\varphi} + 2 \leq \frac{\lceil\log_2 n\rceil}{\log_2\varphi} + 2$.

Recall that the lower bound is $\max(c,d)\lceil\log_2 n\rceil \geq d\lceil\log_2 n\rceil$. If $c < D$, the length of the schedule is bounded by $(k+1)D$, hence a ratio of $\frac{\Delta}{\log_2\varphi} + \frac{2\Delta}{\lceil\log_2 n\rceil}$. If $c \geq D$, the lower bound is $c\lceil\log_2 n\rceil$ and the length of the schedule is upper bounded by $(k+1)c$, hence an even better approximation ratio of $\frac{1}{\log_2\varphi} + \frac{2}{\lceil\log_2 n\rceil}$. In the particular case where $c=d$, the optimal cannot be better than **Fibonacci-stat** using only communications of size d, because **Fibonacci-stat** is optimal when $c=d=D$. The length of the schedule is bounded by $kD+c$, and the optimal is at least $kd+c$, hence a ratio of Δ. □

Results are summarized in Table 1.

5 Markov chain analysis

In this section, we assume that communication and computation costs are exponentially distributed (i.e., each $d_{i,j}$ for $1 \leq i,j \leq n$ follows an exponential law with rate λ_d and each c_i for $1 \leq i \leq n$ follows an exponential law with rate λ_c).

With this model, both dynamic approaches, **Tree-dyn** and **Non-Commut-Tree-dyn**, may be analysed using memoryless stochastic processes. Intuitively, each state of those processes is characterized by the number of concurrent communications and computations. A state in which there is neither communication nor computation is an absorbing state that corresponds to the termination of a reduction algorithm. In the initial state, there are $\lfloor \frac{n}{2} \rfloor$ concurrent communications (and one idle machine that is ready to send its value if n is odd). The completion time of an algorithm is then the time to reach the final state. To determine this duration, we use the first-step analysis.

Formally, let P be the transition rate matrix of a continuous-time Markov chain and $s \in S$ be each state. Let $\phi(s)$ be a function on S taking real values (it will be the expected duration spent in state s or its variance). Let w_i be the sum of the application of ϕ to each state taken by the process until the final state is reached starting from state s_i. Then, w_i is determined using the first-state analysis:

$$w_i = \begin{cases} \phi(s_i) & \text{if } s_i \text{ is an absorbing state} \\ \phi(s_i) + \sum_{j=1}^{n} p_{i,j} w_j & \text{otherwise} \end{cases}$$

We apply this analysis to determine the expected duration and the variance of **Tree-dyn** with $\lambda_c = 0$. For clarity, n is assumed to be even (the following analysis can be performed to the cases where n is odd by adapting the initial state).

In this case, each state $s_{i,j}$ is characterized by the number of concurrent communications i and whether the slot containing a ready processor is empty ($j = 0$) or not ($j = 1$). The initial state is $s_{\frac{n}{2},0}$ and the final state is $s_{0,1}$.

There are two kinds of transitions: from state $s_{i,0}$ to state $s_{i-1,1}$ (a communication terminates and the intermediate result of the local reduction is ready to be sent to the next available processor) and from state $s_{i,1}$ to state $s_{i,0}$ (a communication terminates and a new one is initiated with the available processor identified by the slot) for $1 \leq i \leq \frac{n}{2}$. In both cases, the rate of the transition is determined by the number of concurrent communications, that is $i\lambda_d$.

In order to determine the expected completion time of **Tree-dyn** (noted $C_{\mathbf{Tree-dyn}}$), we define $\phi(s_{i,j})$ as $\frac{1}{i\lambda_d}$, the expected time spent is state $s_{i,j}$ (zero for state $s_{0,1}$). Therefore,

$$\begin{aligned} C_{\mathbf{Tree-dyn}} = w_{\frac{n}{2},0} &= \sum_{i=1}^{\frac{n}{2}} \phi(s_{i,0}) + \phi(s_{i-1,1}) \\ &= \sum_{i=1}^{\frac{n}{2}} \frac{1}{i\lambda_d} + \sum_{i=1}^{\frac{n}{2}-1} \frac{1}{i\lambda_d} = \frac{1}{\lambda_d}\left(2H\left(\frac{n}{2}-1\right) + \frac{2}{n}\right) \end{aligned}$$

where $H(n)$ is the nth harmonic number.

Similarly, we compute the variance of the completion time (noted $V_{\mathbf{Tree-dyn}}$) by defining $\phi(s_{i,j})$ as zero for state $s_{0,1}$ and $\frac{1}{i^2\lambda_d^2}$ otherwise:

$$V_{\mathbf{Tree-dyn}} = \frac{1}{\lambda_d^2}\left(2\sum_{i=1}^{\frac{n}{2}-1} \frac{1}{i^2} + \frac{4}{n^2}\right)$$

6 Simulation results

In this section, we consider that the d_{ij} and c_i costs are distributed according to a gamma distribution, which is a generalization of exponential and Erlang distributions. This distribution has advocated for modeling job runtimes [18, 14]. It is positive and it is possible to specify its expected value (μ_d or μ_c) and standard deviation (σ_d or σ_c) by adjusting its parameters.

When comparing several methods, the same seed is used for each of them such that the same costs are obtained in the same order (e.g., the kth started communication will have the same cost for all methods). This preserves the average behavior, which is our primary study object, but allows for a more stable comparison of the methods especially when the distribution variance (or the platform heterogeneity) is high.

6.1 Cost dispersion effect

In this first simulation, we are interested in characterizing how the dispersion of the communication costs affects the performance of all methods. In order to simplify this study, no computation cost is considered ($\mu_c = 0$). The dispersion is defined through the coefficient of variation (CV), which is defined as the ratio of the standard deviation over the expected value (this latter is set to 1). The number of processors is $n = 64$ and the time taken by each method is measured over 1 000 000 Monte Carlo (MC) simulations.

On a global level, Figure 4 shows the expected performance with distinct CVs. When the heterogeneity is noticeable (CV greater than 1), the performance decreases significantly. In those cases, schedules are mostly affected by a few extreme costs whose importance depends on the CV (even though the average cost remains the same, extreme values tend to increase with the CV). Additionally, the variability in the schedule durations is also impacted by the CV (i.e., two successive executions with the same settings may lead to radically different performance depending on the schedule).

Several observations can be made relatively to each method. As expected, **Binomial-stat** is similar to **Tree-dyn** for CV lower than 10%. In this case, the improvement offered by **Tree-dyn** may not outweigh the advantage of following a static plan in terms of synchronization. For CV greater than 1, both static approaches (**Binomial-stat** and **Fibonacci-stat**) perform equally with a similar dispersion. For all CV, **Tree-dyn** has the best expected performance while **Fibonacci-stat** has the worst, and **Non-Commut-Tree-dyn** has the second best expected performance when the CV is greater than 30%. Finally, when the CV is close to 10, all methods are equivalent as a single communication with a large cost may impact the entire schedule duration. In terms of robustness, we can see that **Fibonacci-stat** and **Non-Commut-Tree-dyn** are the two best methods for absorbing variations as their expected durations remains stable longer (until the CV reaches 30%). This is due the presence of idleness in their schedules that can be used when required. Finally, the ribbons for **Non-Commut-Tree-dyn** are large even for low CV values due to its multi-modal behavior (an additional step is sometimes required depending on the scheduling decisions).

Note that by reusing the same seed, we can check that all methods behave extremely similarly with the same costs when the dispersion is high. Given

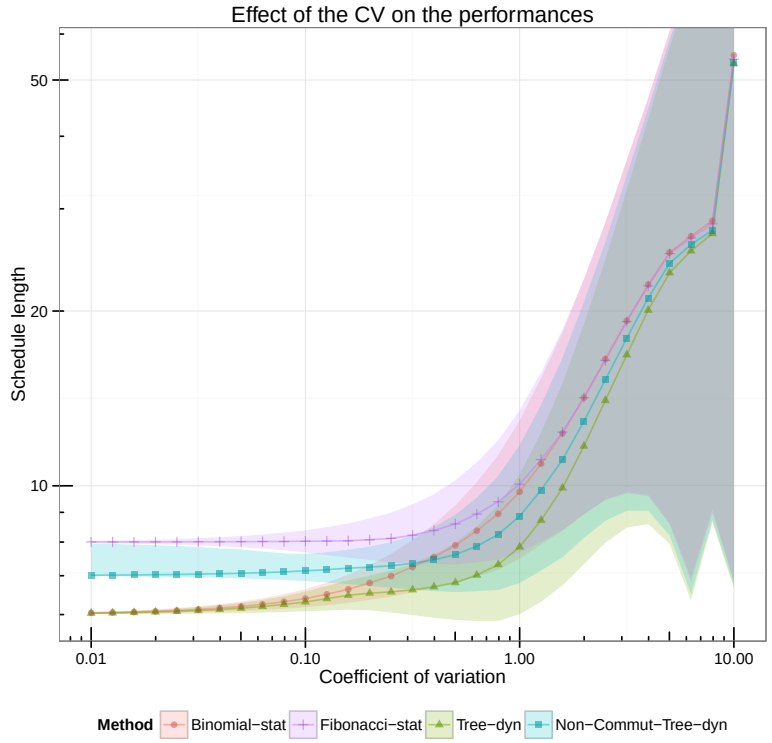


Figure 4: Average schedule length for each method over 1 000 000 MC simulations with $n = 64$, $\mu_d = 1$, $\mu_c = 0$ and varying coefficients of variation for the communication costs. The lower part of the ribbons corresponds to the 10% quantile while the upper part corresponds to the 90% quantile for each method.

the large dispersion, a much larger number of MC simulations would have been required to reach this conclusion without reusing the same seed (the right-most part of the curves shows indeed greater variations than the rest).

6.2 Non-negligible computation

When considering nonzero computation costs, we reduce the number of parameters by applying the same CV to the computation and to the communication costs (i.e., $\frac{\sigma_c}{\mu_c} = \frac{\sigma_d}{\mu_d}$). As **Fibonacci-stat** is designed for overlapping computations and communications, we characterize the cases when this approach outperforms **Tree-dyn** (i.e., the cases for which exploiting this overlapping compensates for ignoring the cost dispersion). The ratio $\frac{\mu_c}{\mu_d}$ controls the amount of possible overlapping between the computation and the communication costs.

Figure 5 shows the improvement of **Tree-dyn** over **Fibonacci-stat** when varying the CV and the ratio $\frac{\mu_c}{\mu_d}$ (the overlapping degree between computations and communications). The contour line with value 1 delimits the area for which **Fibonacci-stat** is better than **Tree-dyn** on average. This occurs when the

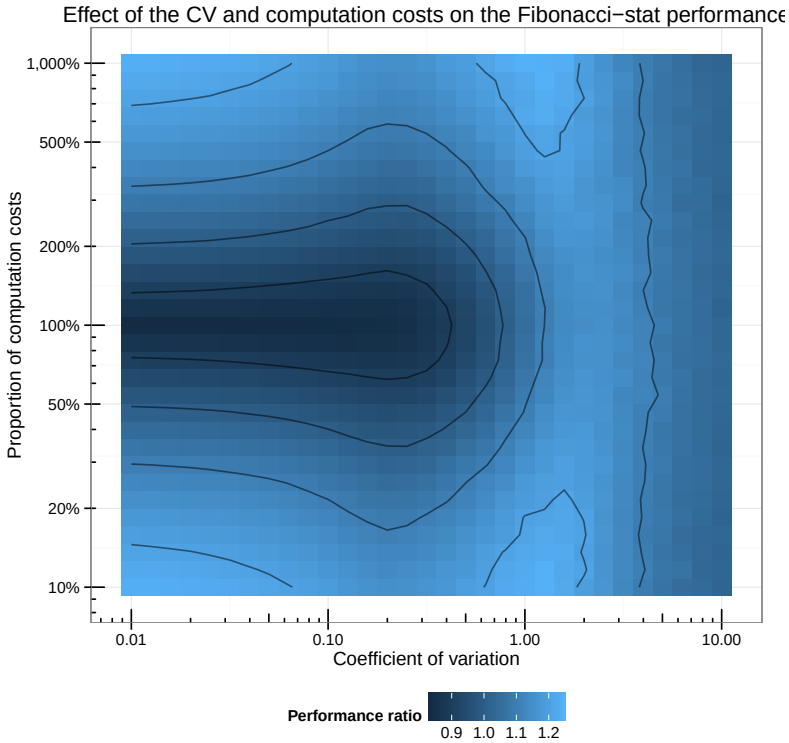


Figure 5: Ratio of the average performance of **Fibonacci-stat** and **Tree-dyn** over 1 000 MC simulations for each square with $n = 64$, $\mu_d = 1$, $\frac{\sigma_c}{\mu_c} = \frac{\sigma_d}{\mu_d}$, varying coefficients of variation for the costs and varying $\frac{\mu_c}{\mu_d}$. **Fibonacci-stat** outperforms **Tree-dyn** when the ratio is lower than 1 (on the center left side).

computation cost is greater than around half the communication cost and when the variability is limited. When the computation costs are low ($\frac{\mu_c}{\mu_d} = 0.1$), the ratio decreases when the CV increases until this later reaches 0.2. Then, the ratio increases until the CV is around 1, and decreases again for larger CV. This is consistent with the previous observations.

Note that this figure only shows the average performance ratio. When the variability increases, **Fibonacci-stat** may outperform **Tree-dyn** depending on the actual costs (as seen on Figure 4) even though the later is expected to be better on average.

Figure 5 is horizontally symmetrical as any case such that $\frac{\mu_c}{\mu_d} > 1$ is equivalent to the situation where the communication and the computation costs are swapped (and for which $\frac{\mu_c}{\mu_d} < 1$). These costs can be exchanged because a communication is always followed by a reduction operation.

6.3 Non-commutative reduction

Finally, we assess the performance of **Non-Commut-Tree-dyn** by comparing it to all other methods that support a non-commutative operation (i.e., **Binomial-stat** and **Tree-dyn**) when varying the dispersion and the overlapping degree as in the previous study.

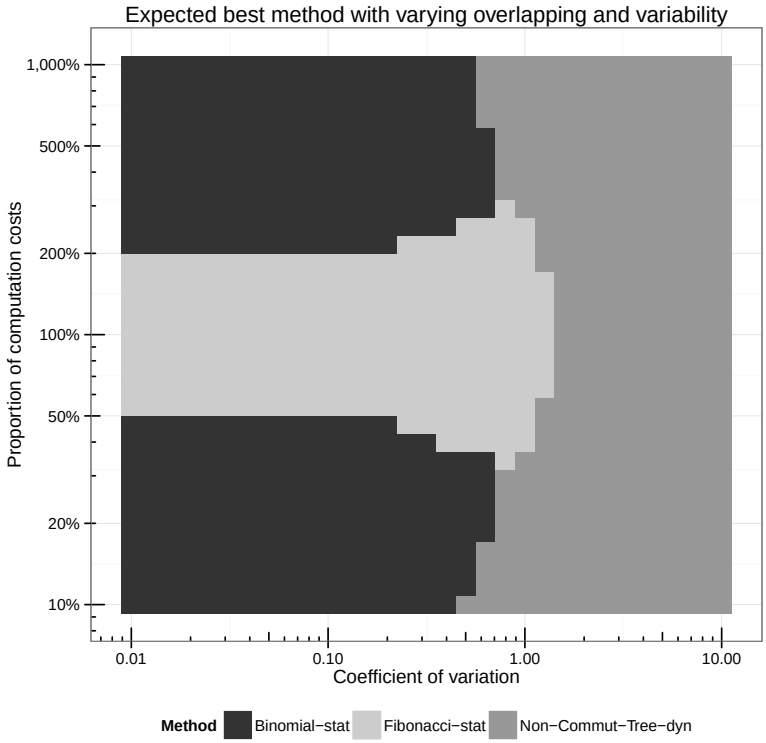


Figure 6: Method with the best average performance over 1 000 MC simulations for each square with $n = 64$, $\mu_d = 1$, $\frac{\sigma_c}{\mu_c} = \frac{\sigma_d}{\mu_d}$, varying coefficients of variation for the costs and varying $\frac{\mu_c}{\mu_d}$.

Figure 6 shows the expected best method (the method with the best average performance) when varying the CV and the ratio $\frac{\mu_c}{\mu_d}$. We see that **Non-Commut-Tree-dyn** has the best performance when the cost dispersion is large. Additionally, the transition from **Binomial-stat** to **Fibonacci-stat** is when the computation cost reaches half the communication cost. Lastly, **Fibonacci-stat** is more robust to cost dispersion when $\mu_c = \mu_d$ than **Binomial-stat** when $\frac{\mu_c}{\mu_d} = 0.1$ even though the corresponding schedules have no idleness. This can be explained by the fact that in **Fibonacci-stat**, operations are pipelined on processors and thus, the costs on the same processor may counterbalance themselves (a higher cost for a reduction may be absorbed by a subsequent lower cost).

Again, with low computation costs ($\frac{\mu_c}{\mu_d} = 0.1$), the results are consistent with

Figure 4. Moreover, with low CV (lower than 0.1), the transition between the best method among **Binomial-stat** and **Fibonacci-stat** occurs when $\frac{\mu_c}{\mu_d} = 0.5$ as on Figure 5 (with **Tree-dyn** and **Fibonacci-stat**, but with low CV, **Tree-dyn** and **Binomial-stat** are equivalent).

Note that Figure 6 has the same symmetry as Figure 5 and that the variability also increases with the CV (as before).

7 Conclusion

In this paper, we have studied the problem of performing a non-clairvoyant reduction on a distributed heterogeneous platform. Specifically, we have compared the performance of traditional static algorithms, which build an optimized reduction tree beforehand, against dynamic algorithms, which organize the reduction at runtime. Our study includes both commutative and non-commutative reductions. We have first proposed approximation ratios for all commutative algorithms using a worst-case analysis. Then, we have proposed a Markov chain analysis for dynamic algorithms. Finally, we have evaluated all algorithms through extensive simulations to show when dynamic algorithms become more interesting than static ones. We have outlined that dynamic algorithms generally achieve better makespan, except when the heterogeneity is limited and for specific communication costs (no communication cost for **Binomial-stat**, communication costs equivalent to computation costs for **Fibonacci-stat**). The worst-case analysis has also confirmed this last observation.

As future work, we plan to investigate more complex communication models, such as specific network topologies. It would also be interesting to design a better dynamic algorithm for non-commutative reductions, which avoids the situation when many processors are idle but cannot initiate a communication since no neighboring processors are free.

References

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

[2] 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*, SC '08, pages 83–93, New York, NY, USA, 2008.

[3] 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*, 6(8):896–900, Aug. 1995.

[4] A. Bar-Noy, S. Kipnis, and B. Schieber. An optimal algorithm for computing census functions in message-passing systems. *Parallel Processing Letters*, 3(1):19–23, 1993.

[5] A. Benoit, F. Dufossé, M. Gallet, Y. Robert, and B. Gaujal. Computing the throughput of probabilistic and replicated streaming applications. In *Proc.*

of SPAA, Symp. on Parallelism in Algorithms and Architectures, pages 166–175, 2010.

[6] J. Bruck and C.-T. Ho. Efficient global combine operations in multi-port message-passing systems. *Parallel Processing Letters*, 3(4):335–346, 1993.

[7] L.-C. Canon and G. Antoniu. Scheduling Associative Reductions with Homogeneous Costs when Overlapping Communications and Computations. Rapport de recherche RR-7898, INRIA, Mar. 2012.

[8] L.-C. Canon and E. Jeannot. Evaluation and optimization of the robustness of dag schedules in heterogeneous environments. *IEEE Trans. Parallel Distrib. Syst.*, 21(4):532–546, 2010.

[9] L.-C. Canon, E. Jeannot, R. Sakellariou, and W. Zheng. Comparative Evaluation of the Robustness of DAG Scheduling Heuristics. In *Proceedings of CoreGRID Integration Workshop*, Heraklion-Crete, Greece, Apr. 2008.

[10] 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*, 19(13):1749–1783, 2007.

[11] G. Cordasco, R. D. Chiara, and A. L. Rosenberg. On scheduling dags for volatile computing platforms: Area-maximizing schedules. *J. Parallel Distrib. Comput.*, 72(10):1347–1360, 2012.

[12] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms*. The MIT Press, 3rd edition, 2009.

[13] J. Dean and S. Ghemawat. Mapreduce: Simplified data processing on large clusters. *Communications of the ACM*, 51(1):107–113, 2008.

[14] D. Feitelson. Workload modeling for computer systems performance evaluation. *Book Draft, Version 0.38*, 2013.

[15] 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.

[16] 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*, 65(12):1497–1514, 2005.

[17] P. Liu, M.-C. Kuo, and D.-W. Wang. An Approximation Algorithm and Dynamic Programming for Reduction in Heterogeneous Environments. *Algorithmica*, 53(3):425–453, Feb. 2009.

[18] U. Lublin and D. G. Feitelson. The workload on parallel supercomputers: modeling the characteristics of rigid jobs. *J. Parallel Distrib. Comp.*, 63(11):1105–1122, 2003.

[19] 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*, IPDPS, Apr. 2005.

[20] R. Rabenseifner. Optimization of Collective Reduction Operations. In M. Bubak, G. van Albada, P. Sloot, and J. Dongarra, editors, *Computational Science - ICCS 2004*, volume 3036 of *Lecture Notes in Computer Science*, pages 1–9. 2004.

[21] 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*, Lecture Notes in Computer Science. Springer Berlin, 2004.

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

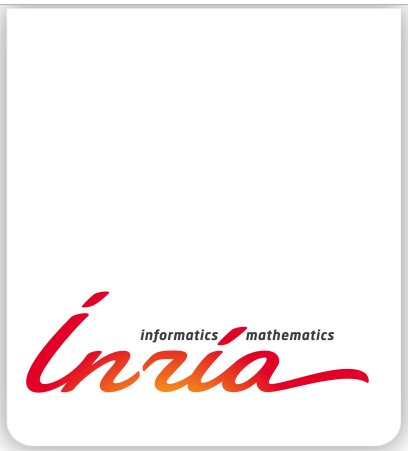
[23] P. Sanders, J. Speck, and J. L. Träff. Two-tree algorithms for full bandwidth broadcast, reduction and scan. *Parallel Computing*, 35(12):581–594, 2009.

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

[25] R. A. van de Geijn. On global combine operations. *Journal of Parallel and Distributed Computing*, 22(2):324–328, 1994.

[26] T. White. *Hadoop: The definitive guide*. Yahoo Press, 2010.

[27] M. Zaharia, A. Konwinski, A. Joseph, R. Katz, and I. Stoica. Improving mapreduce performance in heterogeneous environments. In *Proc. of the 8th USENIX conf. on Operating systems design and implementation*, pages 29–42, 2008.



RESEARCH CENTRE
GRENOBLE – RHÔNE-ALPES

Inovallée
655 avenue de l'Europe Montbonnot
38334 Saint Ismier Cedex

Publisher
Inria
Domaine de Voluceau - Rocquencourt
BP 105 - 78153 Le Chesnay Cedex
inria.fr

ISSN 0249-6399