HAL
open science

Non-clairvoyant reduction algorithms for heterogeneous platforms

Anne Benoit, Louis-Claude Canon, Loris Marchal

To cite this version:

Anne Benoit, Louis-Claude Canon, Loris Marchal. Non-clairvoyant reduction algorithms for heterogeneous platforms. HeteroPar'2013, in conjunction with Euro-Par 2013, Aug 2013, Aachen, Germany. hal-00926093

HAL Id: hal-00926093

https://inria.hal.science/hal-00926093

Submitted on 9 Jan 2014

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Non-Clairvoyant Reduction Algorithms for Heterogeneous Platforms

Anne Benoit[1], Louis-Claude Canon[2], and Loris Marchal[1]

[1] École Normale Supérieure de Lyon, CNRS & INRIA, France
[anne.benoit,loris.marchal]@ens-lyon.fr
[2] FEMTO-ST, Université de Franche-Comté, Besançon, France
louis-claude.canon@univ-fcomte.fr

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 with constant or asymptotic ratios. 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 through a set of simulations.

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 (see [12,14] for experimental comparisons). Many algorithms have been introduced to optimize this operation on various platforms, with homogeneous [13] or heterogeneous communication costs [10,9]. Recently, this operation has received more attention due to the success of the MapReduce framework [7,15], 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 this framework scales easily to very large clusters of inexpensive commodity computers. We review similar algorithms and use cases in the related work section in the companion research report [1].

Our objective in this paper is to compare the performance of various algorithms for the reduce operations in a non-clairvoyant setting, i.e., when the algorithms are oblivious to the communication and computation costs (the time required to communicate or compute). This models well the fact that communication times cannot usually be perfectly predicted, and may vary significantly over time. 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 design and the analysis of algorithms in a dynamic context has already received some attention. The closest related work is probably [5], 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. [2] studies the problem of 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, [4] studies and compares different robustness metrics for makespan/reliability optimization on task-graph scheduling.

The rest of the paper is organized as follows. Section 2 describes four algorithms and shows that they are approximation algorithms with identical computation costs. In Section 3, we provide more involved probabilistic analysis of their expected performance. Section 4 presents simulated executions of the previous algorithms and compares their respective performance. Finally, we conclude and discuss future research directions in Section 5.

2 Model and Algorithms

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 $v = v_0 \oplus v_1 \oplus \cdots \oplus v_{n-1}$ as fast as possible, i.e., to minimize the total execution time to compute the reduction. 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 deal with both versions of the problem.

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 $[v_i, v_j]$; the processor 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.

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 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 $i2^k + 2^{k-1}$ $(i = 0, \ldots, 2^{\lceil \log_2 n \rceil - k} - 1)$ sends its value to processor $i2^k$, which reduces it with its own value: at most $2^{\lceil \log_2 n \rceil - k + 1}$ processors are involved in round k. 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 [6, 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

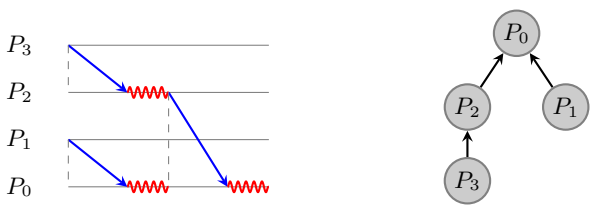


Figure 1. Schedule and communication graph for reducing four values. Blue arrows represent communications while red springs stand for computations.

($k > 0$) 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 [3], where their optimality is proved for special homogeneous cases: **Binomial-stat** is optimal both when the computations are negligible in front of communications ($c_i = 0$ and $d_{i,j} = d$) and 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

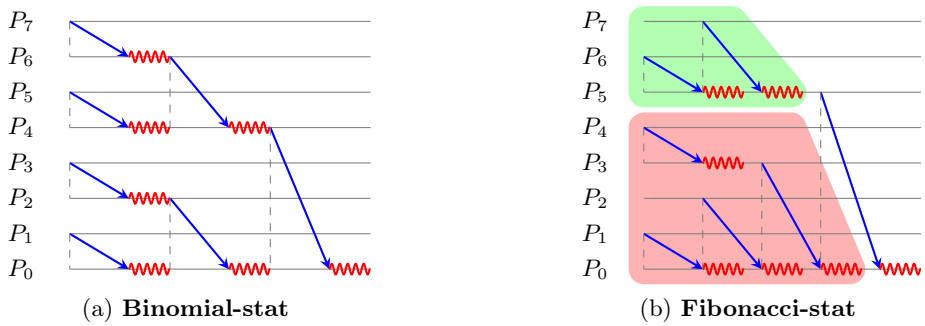


Figure 2. Schedules for **Binomial-stat** of order 3 and **Fibonacci-stat** of order 4, both using 8 processors. For **Fibonacci-stat**, the two schedules of order 2 and 3 used in the recursive construction are highlighted in green and red.

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.

Worst-Case Analysis. We analyze the commutative algorithms in the worst case, and we provide some approximation ratios, focusing on communication times. A λ-approximation algorithm is a polynomial-time algorithm that returns a solution whose execution time is at most λ times the optimal execution time. 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. Results are summarized in Table 1. Due to lack of space, the complete set of theorems and proofs is available in the companion research report [1]. We discuss a subset of the results below.

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

The proof is in [1], and we exhibit here 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 $d\lceil \log_2(n) \rceil$).

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 [3]. 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,

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

	$c = 0$	any 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. Theorem numbers in parenthesis refer to the theorems in [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 $\frac{\Delta}{\log_2 \varphi} + \frac{\Delta}{\lceil \log_2 n \rceil}$. □

3 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,

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

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:

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

4 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 been advocated for modeling job runtimes [11,8]. This distribution 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. Using two distributions with two parameters each is a good compromise between a simulation design with a reasonable size and a general model. Additionally, further simulations with Bernoulli distributions concurred with the following observations, suggesting that our conclusions are not strongly sensitive to the distribution choice. Each simulation was performed with an ad-hoc simulator on a desktop computer.

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 (i.e., simulations have been performed with 1 000 000 distinct seeds).

On a global level, Figure 3 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. 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 CVs 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 CVs greater than 1, both static approaches perform equally with a similar dispersion. For all CVs, **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

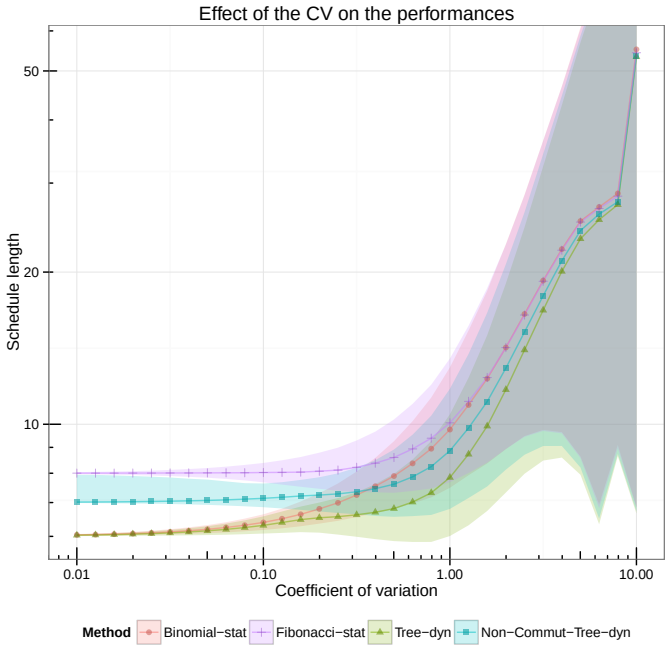


Figure 3. Average schedule length for each method over 1 000 000 MC simulations with $n = 64$, $\mu_d = 1$, $\mu_c = 0$ and varying coefficients of variation for the communication costs. The lower part of the ribbons corresponds to the 10% quantile while the upper part corresponds to the 90% quantile for each method.

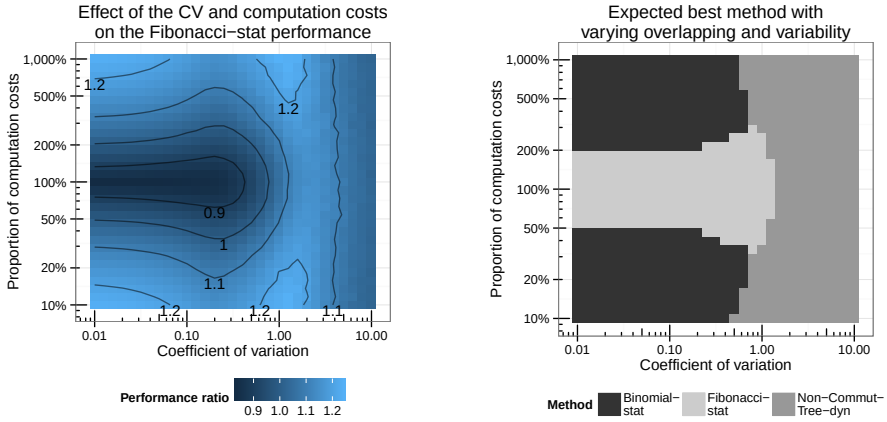
Non-Commut-Tree-dyn are the two best methods for absorbing variations as their expected durations remains stable longer (until the CV reaches 30%). This is due the presence of idleness in their schedules that can be used when required.

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**.

Figure 4(a) 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 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 evolution is is consistent with the previous observations.

Figure 4(a) 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.

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



(a) Non-negligible communication

(b) Non-commutative operation

Figure 4. Ratio of the average performance of **Fibonacci-stat** and **Tree-dyn** (a) and method with the best average performance (b) 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 4(b) shows 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 (as on Figure 4(a)). With low computation costs ($\frac{\mu_c}{\mu_d} = 0.1$), the results are also consistent with Figure 3.

5 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. Benoit, L.-C. Canon, and L. Marchal. Non-clairvoyant reduction algorithms for heterogeneous platforms. Research report RR-8315, INRIA, June 2013.
2. 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.
3. 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.
4. 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.
5. 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.
6. T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms*. The MIT Press, 3rd edition, 2009.
7. J. Dean and S. Ghemawat. Mapreduce: Simplified data processing on large clusters. *Communications of the ACM*, 51(1):107–113, 2008.
8. D. Feitelson. Workload modeling for computer systems performance evaluation. *Book Draft, Version 0.38*, 2013.
9. 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.
10. 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.
11. 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.
12. 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.
13. 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.
14. 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.
15. 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.