Performance Analysis and Optimality Results for Data-Locality Aware Tasks Scheduling with Replicated Inputs

Olivier Beaumont, Thomas Lambert, Loris Marchal, Bastien Thomas

Performance Analysis and Optimality Results for Data-Locality Aware Tasks Scheduling with Replicated Inputs

Olivier Beaumont[1], Thomas Lambert*[2], Loris Marchal[3], and Bastien Thomas[4]

[1]Inria Bordeaux and LaBRI, Talence, France
[2]Inria Rennes and LS2N, Nantes, France
[3]CNRS and LIP, Lyon, France
[4]Inria Rennes and IRISA, Rennes, France

Abstract

Replication of data files, as automatically performed by Distributed File Systems such as HDFS, is known to have a crucial impact on data locality in addition to system fault tolerance. Indeed, intuitively, having more replicas of the same input file gives more opportunities for this task to be processed locally, *i.e.* without any input file transfer. Given the practical importance of this problem, a vast literature has been proposed to schedule tasks, based on a random placement of replicated input files. Our goal in this paper is to study the performance of these algorithms, both in terms of makespan minimization (minimize the completion time of the last task when non-local processing is forbidden) and communication minimization (minimize the number of non-local tasks when no idle time on resources is allowed). In the case of homogenous tasks, we are able to prove, using models based on "balls into bins" and "power of two choices" problems, that the well known good behavior of classical strategies can be theoretically grounded. Going further, we even establish that it is possible, using semi-matchings theory, to find the optimal solution in very small time. We also use known graph-orientation results to prove that this optimal solution is indeed near-perfect with strong probability. In the more general case of heterogeneous tasks, we propose heuristics solutions both in the clairvoyant and non-clairvoyant cases (*i.e.* task length is known in advance or not), and we evaluate them through simulations, using actual traces of a Hadoop cluster.

1 Introduction

Data locality is known to be the key for performance in both big data analysis and high-performance computing: in these distributed systems, it is crucial to limit the amount of data movement by allocating computing tasks close to their input data, as communications lead to large spendings in both processing time, network congestion and energy consumption. Many distributed systems, in particular the ones used in the context of BigData applications, are based on a distributed file system that comes with a built-in replication strategy. This is for instance the case of the Hadoop Distributed File System (HDFS) used in the well-known MapReduce framework, which splits files into large chunks of data and distributes them across nodes. By default, each block is replicated three times: two replicas are stored on the same cluster rack, while the third one is sent on a remote rack.

The original motivation for replication is data resilience, since replication trivially limits the risk of loosing a file both in the case of a node failure and in the case of a rack failure, it also dramatically improves data locality. Indeed, the MapReduce scheduler takes advantage of the replication when allocating map tasks: whenever a node has a slot available for a map task, a local tasks is allocated to it if possible, that is,

*Corresponding author: thomas.lambert@inria.fr

a task whose block is stored locally. While these schedulers have been largely used and studied through experimentations, very little is known on their theoretical performance, as detailed below in the related work section.

Our first goal in the paper is therefore to assess that replication strategies introduced in distributed file systems such as HDFS, together with the dynamic schedule strategies of Hadoop MapReduce, are enough to achieve good performance. To do this, we establish several theoretical results that hold true under simplifying assumptions, such as considering tasks with the same duration.

The first contribution of this paper is to prove, using an analogy with the power of two choices in balls into bins problem, that these simple algorithms can achieve provably good performance. The most general problem is bi-criteria, as one may aim both at minimizing the number of non-local tasks (in order to minimize energy or to save system bandwidth) and at minimizing the maximal completion time (and therefore the end of the job). To cope with this additional difficulty, throughout this paper, we will consider two simple metrics that are of practical interest. The "makespan" metric considers the minimization of the completion of the job, under the constraint that non local tasks (*i.e.* tasks that would involve a communication) are forbidden. The "communication" metric considers the minimization of the number of non local tasks, under the constraint that no processor is kept idle while there remains an unprocessed task. The second main contribution of this paper is to prove that it is in fact possible, knowing the placement of replicas and assuming homogeneous duration of tasks, to find the optimal solution for both metrics defined above. Moreover, in the asymptotic case where the number of tasks is large with respect to the number of processors, we are able to prove that the achieved load balancing is perfect, *i.e.* that all resources process exactly the same number of tasks (up to at most one task), with strong probability. Otherwise, the allocation is near-perfect (perfect plus at most one task on the most loaded processor), also with strong probability. A preliminary version of this work appeared in [1]. With respect to [1], we describe in Section 5 how to compute optimal allocations for the two metrics studied in the paper (makespan and communication) based on semi-matching algorithms for bipartite graphs. We also propose a probabilistic evaluation of the asymptotic behavior of the proposed optimal solution for the makespan metric, based on existing results in the field of graph orientation.

Our second goal is to propose heuristic algorithms for the general problem, that are based on the well founded methods proved in simplified contexts, and to compare them to the best heuristics from the literature. Indeed, above results hold true when all tasks have the same duration, which is not always the case as we can observe in the actual Hadoop trace provided in [2]. There is no hope to design optimal resource allocation algorithms in this context since the problem trivially becomes NP-Complete. Therefore, we adapt the algorithms designed for the homogeneous case to the heterogeneous case, in order to assess their efficiency through simulations. We also propose some clairvoyant strategies (aware of task durations) in order to compare to previous non-clairvoyant ones from the literature. We prove that our heuristics, that are conceptually simple and inspired by optimal algorithms perform at least as well as the best heuristics from the literature.

The rest of the paper is organized as follows. Section 2 states the problem and notations, Section 3 reviews the related work on independent tasks scheduling when input files are replicated. In Section 4, we model the allocation process using balls into bins and power of two choices classical randomized algorithms, which enables us to establish sophisticated and strong bounds on the performance of classical scheduling algorithms such as those proposed in the MapReduce context. In Section 5, we prove that the allocation problem is in fact equivalent to a semi-matching problem for which very efficient strategies have been designed. We also relate this allocation problem to an orientation of graph problem in order to propose a probabilistic evaluation of the optimal allocation for makespan optimization. Then, in the context of tasks with heterogeneous durations, we consider both the clairvoyant case (*i.e.* tasks lengths are known at submission time) and the non-clairvoyant case in Section 6. We compare these heuristics using homogeneous durations of tasks model and actual traces of a Hadoop cluster in Section 7. We end the paper by concluding remarks in Section 8.

2 Problem modeling

2.1 Problems Description

We consider the problem of processing a job consisting of n independent tasks, denoted by the set $T = \{t_1 \dots, t_n\}$, on a distributed computing platform made of m resources (or machines, or processors), denoted by the set $P = \{p_1, \dots, p_m\}$. Each task depends on exactly one input data file, or data chunk, and we assume, as it is the case in distributed file systems such as HDFS, that all chunks belonging to the same job have the same data size S. These n chunks are originally distributed onto the distributed nodes, with replication factor r, *i.e.* there are exactly r identical copies of each input data chunk in the distributed platform. We assume that no two copies of the same data chunk are present on the same node, and that replicas are distributed uniformly at random. We consider that all nodes of the system have the same processing capabilities, and that the processing time of task t_i is d_i on any processor. In the following, we will consider both homogeneous task durations ($d_i = d$ for all tasks) and heterogeneous task durations. In the latter case, the scheduler is said to be *clairvoyant* if task durations are known before the execution, and *non-clairvoyant* otherwise.

As explained in the introduction, the goal is to process all tasks on the available processors and to take advantage of data locality to avoid moving too much data around, by carefully choosing the set of tasks assigned to each processor (task allocation). In most systems, it is possible to overlap communications and computations, so that the communication of data chunks does not necessary induce idle on one resource. Therefore, the problem we address is bi-criteria, as there is a natural trade-off between time and data movements considered as two competing objectives.

The completion time C_j on a processor p_j is the sum of the durations of all tasks assigned to p_j, thus assuming no idle time and a perfect overlap, *i.e.*

$$C_j = \sum_{t_i \text{ on } p_j} d_i.$$

Then, we formally define the first objective as the makespan, *i.e.* the maximal completion time among all processors and we will denote it by $C_{\max} = \max_j C_j$.

The second objective is the amount of data movements or communications, *i.e.* the total volume of data sent between nodes at runtime. Since all tasks have size S, this amount of communications is directly proportional to the number of non local tasks, *i.e.* tasks that are not processed on one of the r processors owning its input chunks,

$$\text{communication amount} = \text{number of non local tasks} \times S.$$

In the case of homogeneous tasks, achieving optimal makespan $C_{\max}*$ is easy, since any solution that never leaves a processor idle achieves optimal makespan, possibly at the price of assigning many non local tasks. Conversely, it is easy to minimize the volume of communications, by performing local tasks only: the strategy then consists in keeping a processor idle if it does not hold unprocessed chunks anymore.

Therefore, to make problems meaningful in the context of both criteria, we will consider each metric under the assumption that the other metric is set to its optimal value.

More precisely, when optimizing the makespan metric, we assume that only local tasks are performed, *i.e.* that the volume of communications is zero. Conversely, when optimizing the communication metric, we assume that the makespan is optimal, *i.e.* that a processor is never kept idle if there exists an unprocessed task. In the case of homogeneous tasks, the optimal makespan is therefore given by $\lceil |T|/|P| \rceil$.

In the case of heterogeneous durations, minimizing the makespan is trivially NP-complete (even on two processors using a reduction to 2-Partition [3]). By analogy, we keep the same constraint on the processing of tasks as in the homogeneous case, and we assume that a processor is never kept idle if there exists an unprocessed task. This restricts the search of communication efficient algorithms to List Schedules, *i.e.* schedules that never leave a resource idle when there is work to do.

Finally, we assume that there exists a centralized coordinator, or scheduler, that takes all allocation decisions. Note that we are interested in algorithms with low runtime complexity, such as greedy schedulers, since they are to be used to allocate tasks at runtime in large scale systems.

2.2 Formal Definitions

To make our model complete, let us remark that we can represent the distribution of chunks onto resources as a bipartite graph $G = (T, P, E)$, where an edge between $t_i \in T$ and $p_j \in P$ means that processor p_j owns one of the r replicas of the data chunk associated to task t_i. This model will be particularly useful in Section 5.2 where we use sophisticated semi-matching algorithms to solve the allocation problems. With these graph notations, the two optimization problems described above can be formalized as follows (in the homogeneous case).

Problem 1 (MAKESPAN-ORIENTED)**.** *Given a bipartite graph $G = (T, P, E)$, find an allocation function $\sigma : T \to P$ such that*

- $\forall t_i \in T$, $(\sigma(t_i), t_i) \in E$.
- $\max_{p_j \in P} |\{t_i,\ \sigma(t_i) = p_j\}|$ *is minimal.*

Problem 2 (COMM-ORIENTED)**.** *Given a bipartite graph $G = (T, P, E)$ find an allocation function $\sigma : T \to P$ such that*

- $\max_{p_j \in P} |\{t_i,\ \sigma(t_i) = p_j\}| \leq \left\lceil \frac{|T|}{|P|} \right\rceil$.
- $|\{t_i,\ (\sigma(t_i), t_i) \notin E\}|$ *is minimal.*

In MAKESPAN-ORIENTED the first condition enforces that all tasks are processed locally, so that $\max_{p_j \in P} |\{t_i,\ \sigma(t_i) = p_j\}|$ denotes the makespan of the allocation. In COMM-ORIENTED, the first condition enforces the optimality of the makespan (if all processors perform no more than $\left\lceil \frac{|T|}{|P|} \right\rceil$ tasks, then load balancing is perfect) and $|\{t_i,\ (\sigma(t_i), t_i) \notin E\}|$ denotes the overall number of non-local tasks.

3 Related Work on Independent Tasks Scheduling

3.1 Data Locality in MapReduce

In MapReduce, minimizing the amount of communications performed at runtime is a crucial issue. The initial distribution of the chunks onto the platform is performed by a distributed filesystem such as HDFS [4]. By default, HDFS replicates randomly data chunks several times onto the nodes (usually 3 times). As already noticed, this replication has two main advantages. First, it improves the reliability of the process, limiting the risk of losing input data. Second, replication tends to minimize the number of communications at runtime. Indeed, by default, each node is associated to a given number of Map and Reduce slots (usually two of each kind). Whenever a Map slot becomes available, the default scheduler first determines which job should be scheduled, given job priorities and history. Then, it checks whether the job has a local unprocessed data chunk on the processor. If yes, such a local task is assigned, and otherwise, a non-local task is assigned and the associated data chunk is sent from a distant node.

A number of papers have studied the data locality in MapReduce, in particular during its "Map" phase. The work of Zaharia et al. [5], Xie and Lu [6] and Isard et al. [7] strive to increase the data locality by proposing better schedulers than the default one. We analyze these works in detail below. They propose efficient heuristic strategies to efficiently schedule map jobs, where our goal if both to theoretically evaluate the performance of the default strategy, to propose an optimal algorithm in the homogeneous case, and then to extend it into a general purpose algorithm. For this last part, the closest algorithm is Maestro, proposed by Ibrahim in [8], and whose detailed description is given in Section 6.

Zaharia et al. [5] first proposed the *Delay* scheduler to improve data locality when several jobs compete on a cluster. In their strategy, if a given job has a free slot for a new task on a processor but owns no local chunk, instead of running a non-local task, what would induce a data movement (as in the classical scheduler), this job waits for a small amount of time, allowing for other jobs to run local tasks instead (if they have some). The authors show that this improves data locality while preserving fairness among jobs.

Ibrahim et al. [8] also outline that, apart from the shuffling phase, another source of excessive network traffic is the high number of non-local map tasks. They propose *Maestro*, an assignment scheme that intends to maximize the number of local tasks. To this end, *Maestro* estimates which processors are expected to be assigned the smallest number of local tasks, given the distribution of the replicas. These nodes are then selected first for local task assignment. They experimentally show that this strategy reduces the number of non-local map tasks. A more precise description of *Maestro* is given in Section 6.

Xie and Lu [6] propose a simple assignment strategy based on the degree of each processor to overcome the problem of non-local map tasks. The idea is to give priority to processors with the smallest non-zero number of unprocessed replicas, so that they could be assigned a local task. They propose a "peeling" algorithm that first serves the processor with a single unprocessed replica (and thus assigns to them their unique local task), and then switches to a classical random assignment for other processors. Using queuing theory and assuming that processing times are given by geometric distribution, they prove that their strategy offers close to optimal assignment for small to medium load. In a similar context, [9] propose a new queuing algorithm to simultaneously maximize throughput and minimize delay in heavily loaded conditions.

Guo et al. [10] consider the locality of map tasks. They propose an algorithm based on the Linear Sum Assignment Problem to compute an assignment with minimal communications. Unfortunately, in the case where there are more tasks than processors, their formulation is obviously wrong: they add fictitious processors to get back to the case with equal number of tasks and processors, solve the problem, and then remove the fictitious processors without taking care of the task reassignment.

Isard et al. [7] propose a flow-based scheduler: Quincy. They consider the case of concurrent jobs and their goal is to ensure that if a job is submitted to the platform, its computation time will be smaller than Jt seconds, where t is its computation time with exclusive access to the platform and J the number of jobs running on the platform. To fulfill this deadline, they propose a sophisticated model with many parameters (size of the input files, bandwidth, data transfer, possible pre-emption of tasks,...) and transform this problem into a flow problem (see Ford and Fulkerson [11]). Recently this algorithm's computation time has been significantly improved by Gog et al. [12] in order to have sub-second scheduling time at each submission of a new job.

Selvitopi et al. [13] propose a static locality-aware scheduling approach for MapReduce. The goal here is to reduce the data transfers between Map and Reduce phases by allocating both map and reduce tasks at the same time. The solution proposed is based on graph and hypergraph models and relies on application-specific knowledge. This approach brings significant speed-ups for the targeted applications (sparse matrix-vector multiplication and generalized sparse matrix-matrix multiplication).

3.2 Balls-into-bins

When considering the makespan metric, processors are only allowed to compute the chunks they own locally. This might generate some load imbalance, since some of the processors may stop their computations early. Such a process is closely related to *balls-into-bins* problems, see Raab and Steger [14] and we will translate the balls-into-bins results from the literature in order to theoretically evaluate the performance of MapReduce default scheduler. More specifically, we prove in Section 4.2 that it is possible to simulate the greedy algorithm proposed for the makespan metric with a variant of a balls-into-bins game. In this randomized process, n balls are placed randomly into m bins and the expected load of the most loaded bin is considered. In a process where data chunks are not replicated, chunks correspond to balls, processing resources correspond to bins, and if tasks must be processed locally, then the maximum load of a bin is equal to the makespan achieved by the greedy assignment. The case of weighted balls, that corresponds to tasks whose lengths are not of unitary length, has been considered by Berenbrink et al. [15]. It is shown that when assigning a large number of small balls with total weight W, one ends up with a smaller expected maximum load than the

assignment of a smaller number of uniform balls with the same total weight. In the case of identical tasks, Raab and Steger [14] provide value on the expected maximum load, depending on the ratio $\frac{m}{n}$. For example, in the case $m = n$, the expected maximum load is $\frac{\log n}{\log \log n}(1 + o(1))$.

Balls-into-bins techniques have been extended to multiple choices algorithms, where r random candidate bins are pre-selected for each ball, and then the balls are dynamically assigned to the candidate bin whose load is minimum. When translated into MapReduce scheduling problem, r corresponds to the number of replicas of a given input file, as guaranteed by the replication algorithm of HDFS. It is well known that having more than one choice strongly improves load balancing. We refer the interested reader to Mitzenmacher [16] and Richa et al. [17] for surveys that illustrate the power of two choices. Typically, combining previous results with those of Berenbrink et al. [18], it can be proved that whatever the expected number of balls per bin, the expected maximum load is of order $n/m + O(\log \log m)$ even in the case $r = 2$, what represents a very strong improvement over the single choice case. We will prove that these results can be translated to the MapReduce context and provide information on the load balancing achieved by the default scheduler when a replication factor r is used. Peres et al. [19] study the case of a non-integer value r. In this case, with a given parameter β, for each ball, with probability β the assignment is made after choosing between two bins or, with probability $(1 - \beta)$, the assignation is made like for a regular balls-into-bins process. In this case, for $0 < \beta < 1$, the expected maximum load is $\frac{n}{m} + O(\frac{\log m}{\beta})$. Thus, the exact $\beta = 1$ (*i.e.* $r = 2$) is needed to reach the $O(\log \log m)$ usual balls-into-bins gap. The combination of multiple choice games with weighted balls has also been considered by Peres et al. [19]. In this case, each ball comes with its weight w_i and is assigned, in the r-choices case, to the bin of minimal weight, where the weight of a bin is the sum of the weights of the balls assigned to it. Both the results for $(1 + \beta)$ and for $r = 2$ have been extended.

3.3 Scheduling Tasks with Replicated Inputs

Several papers have studied a related scheduling problem, where data locality is important: how to allocate tasks which depend on several input data, initially distributed on various repositories [20, 21, 22]. However, contrarily to our model, these studies assume that (i) tasks depend on several input data and (ii) some input data are shared among tasks. This makes the problem much more complicated. These papers propose heuristics for this problem and compare them through simulations.

4 Theoretical Evaluation of a Simple Strategy for Homogeneous Task Durations

In this section, we focus on straightforward allocation policies, such as the default one of MapReduce. We provide theoretical bounds on those greedy strategies, both for the communication and for the makespan metrics.

4.1 Communications of the Greedy Scheduler without Replication

We start with the communication metric, as defined in the COMM-ORIENTED problem: given a set of tasks T and their input chunks, allocate the tasks to the processors from P so as to minimize the data movement, while enforcing a perfect load-balancing: no processor should be kept idle when there are still some unprocessed tasks.

We study the performance of the simple greedy scheduler, called Greedy-Comm, which resembles the default MapReduce scheduler: when a processor gets idle, it is allocated an unprocessed task. If there is a local unprocessed chunk, such a (random) unprocessed task is chosen, otherwise a (random) non-local unprocessed task is allocated to it.

Our analysis below assumes that the average number of tasks per processor is small in comparison to the total number of tasks ($|T|/|P| = o(|T|)$). We estimate the amount of communication induced by such a scheduler in absence of replication.

Theorem 1. *The expected amount of communication of the greedy scheduler without initial data replication is lower bounded by* $\sqrt{|T||P|/2\pi}$ *(provided that* $|P|$ *divides* $|T|$ *and* $|T|/|P| = o(|T|)$*).*

Proof. Note that without replication, our scheduling problem ressembles the *work stealing* scenario: each processor owns a set of data chunks; it first processes its local chunks before stealing chunks from other nodes after finishing its own local work.

We consider here a variant of the greedy scheduler: when a non-local task must be allocated on a processor, it is not taken (stolen) at random among all non-local tasks, but rather chosen randomly among the chunks of a processor that has the highest number of unprocessed chunks. This way, we diminish the risk that this processor (the victim of the theft) will eventually have to process some non-local tasks itself. In the genuine greedy scheduler, a non-local task might remove a task from a processor that has only few of them, leading this processor to eventually process a non-local task itself. The formula obtained below is a lower bound on the amount of communication of the greedy scheduler.

Let us estimate the number of chunks that are sent to each processor during the allocation. Let us denote by N_k the expected number of processors that received exactly k chunks. In the end, since all tasks have the exact same duration, each processor will process exactly $|T|/|P|$ tasks. Thus, in our variant, a processor with k tasks will request $|T|/|P| - k$ tasks if $k < |T|/|P|$ and none otherwise. This allows to express the expected total amount of communications as

$$\begin{aligned} V &= \sum_k \max(0, |T|/|P| - k) N_k \\ &= \sum_{0 \leq k < |T|/|P|} (|T|/|P| - k) N_k \end{aligned}$$

Let us now compute the probability that a processor gets exactly k chunks during the data distribution. This is similar to the probability that a bin receive exactly k balls in the balls-into-bins distribution process [23] and is given by $Pr(k) = \binom{|T|}{k}(1/|P|)^k(1 - 1/|P|)^{|T|-k}$. When $k = o(|T|)$, $k > 0$ and $1/|P| = o(1)$, we approximate this probability at first order by $Pr(k) = e^{-|T|/|P|}(|T|/|P|)^k/k!$. Then, N_k simply becomes $|P| \times Pr(k)$. This allows to compute V as

$$\begin{aligned} V &= \frac{|T|}{|P|} N_0 + \sum_{1 \leq k < |T|/|P|} (|T|/|P| - k)|P| e^{-|T|/|P|} \frac{(|T|/|P|)^k}{k!} \\ &= \frac{|T|}{|P|} N_0 + |P| e^{-|T|/|P|} \sum_{1 \leq k < |T|/|P|} \frac{(|T|/|P|)^{k+1}}{k!} - \frac{(|T|/|P|)^k}{(k-1)!} \\ &= \frac{|T|}{|P|} |P| e^{-|T|/|P|} + |P| e^{-|T|/|P|} \left(\frac{(|T|/|P|)^{|T|/|P|}}{(|T|/|P| - 1)!} - |T|/|P| \right) \\ &\geq |P| e^{-|T|/|P|} \frac{(|T|/|P|)^{|T|/|P|+1}}{\sqrt{2\pi |T|/|P|}} (|P|e/|T|)^{|T|/|P|} \\ &\geq \sqrt{|T||P|/2\pi}. \end{aligned}$$

□

Figure 1 depicts the evolution of above lower bound with $|T|/|P|$, and shows the comparison between it and a simulation of Greedy-Comm with $|P| = 50$ processors and different values of $|T|$. The results show that this lower bound accurately describes the behavior of Greedy-Comm strategy, proving the reliability of the probabilistic model.

4.2 Makespan of the Greedy Scheduler vs. Balls-into-Bins Problems

Let us now consider the MAKESPAN-ORIENTED problem. We aim at minimizing the makespan when only local tasks can be processed. We consider the following basic greedy strategy, called Greedy-Makespan:

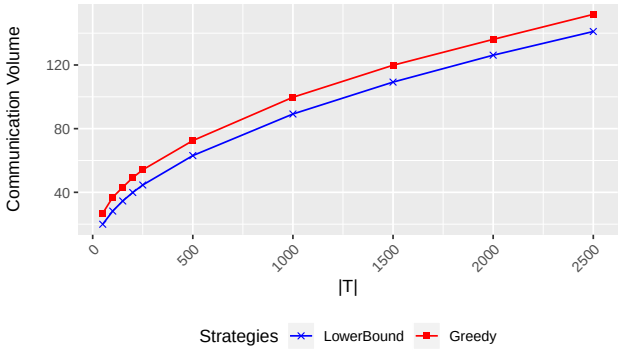


Figure 1: Number of Non-Local Tasks – homogeneous case, no replication. We recall that the communication volume is proportional with the number of non-local tasks in this case (see Section 2.1).

when a processor finishes a task, if it still holds the input chunk of some unprocessed task, it randomly chooses such a task and then process it. Otherwise, it stops its execution.

For the sake of the generality, we move to the case of heterogeneous task durations, where task t_i comes with its duration of d_i. We also consider that processors have slightly different initial availability dates and we denote by ϵ_j the availability date of processor p_j. However, the Greedy-Makespan scheduler ignores task durations before their execution. We assume, as in [15] or [19], that this heterogeneity in task durations and processor availability times allows us to consider that no ties have to be broken. The initial chunk distribution is given by $n = |T|$ random sets of r choices: $\mathcal{C}_1, \ldots, \mathcal{C}_n$, where $\mathcal{C}_i$ is the set of nodes owning the input chunk of task t_i. Together with the initial processor availability times and the task durations, these random choices entirely define the behavior of the scheduler.

We prove that in presence of replication, the expected makespan of the Greedy-Makespan scheduler is closely related to balls-into-bins results with r multiple choices (see Mitzenmacher [16]). The process of distributing n balls $B_1, \ldots, B_n$ of sizes $d_1, \ldots, d_n$ into $m = |P|$ bins whose initial loads are given by $\epsilon_1, \ldots, \epsilon_m$ is done as follows. For each ball, r bins are selected at random (using the random choices $\mathcal{C}_i$) and the ball is placed in the least loaded of these r bins. The following theorem explicits the relation between the simple dynamic scheduler and the balls-into-bins process.

Theorem 2. *Let us denote by* MaxLoad *the maximal load of a bin obtained with the balls-into-bins process and by* $C_{\max}$ *the makespan achieved using* Greedy-Makespan. *Then,*

$$\textsc{MaxLoad}(\mathcal{C}_1, \ldots, \mathcal{C}_n, \epsilon_1, \ldots, \epsilon_m) = C_{\max}(\mathcal{C}_1, \ldots, \mathcal{C}_n, \epsilon_1, \ldots, \epsilon_m).$$

Proof. In order to prove above result, let us prove by induction on i the following lemma.

Lemma 3. *Let* $j_b(i)$ *denotes the index of the bin where the ball* B_i *is placed and let* $j_p(i)$ *denotes the index of the processor where task* t_i *is processed, then* $j_b(i) = j_p(i)$.

Proof. Let us consider ball B_1 and task t_1. B_1 is placed in the bin such that ϵ_k is minimal, where $k \in \mathcal{C}_1$. Conversely, t_1 is replicated onto all the processors p_k, where $k \in \mathcal{C}_1$. Since each processor executes its tasks following their index, t_1 is processed on the first processor owning its input data that is looking for a task, *i.e.* the processor such that ϵ_k is minimal. This achieves the proof in the case $n = 1$.

Let us assume that the lemma holds true for all indices $1, \ldots, i-1$, and let us consider the set of bins B_k and the set of processors p_k such that $k \in \mathcal{C}_i$. By construction, at this instant, processors p_k, $k \in \mathcal{C}_i$ have processed tasks whose index is smaller than i only. Let us denote by $S_i = \{t_{i_1}, \ldots, t_{i_{n_i}}\}$ this set of tasks, whose indices i_k's are smaller than i. These tasks have been processed on the processors whose indices are the same as those of the bins on which balls $\{B_{i_1}, \ldots, B_{i_{n_i}}\}$ have been placed, by induction hypothesis.

Therefore, for each $p_k,\ k \in \mathcal{C}_i$, the time at which p_k ends processing the tasks assigned to it and whose index is smaller than i is exactly the weight of the balls with index smaller than i placed in B_k. Therefore, the processor p_k that first tries to compute t_i is the one such that ϵ_k plus the weight of the balls with index smaller than i placed in B_k is minimal, so that $j_b(i) = j_p(i)$, what achieves the proof of the lemma. □

Therefore, the makespan achieved by Greedy-Makespan on the inputs $(\mathcal{C}_1, \ldots, \mathcal{C}_n, \epsilon_1, \ldots, \epsilon_m)$ is equal to the load of most loaded bin after the balls-into-bins process on the same input, what achieves the proof of the theorem. □

Thanks to this result, we inherit for our allocation problem of all known bounds on the maximum load for balls-into-bins processes derived in the literature (and mentioned in Section 3). In particular, going back to the case of tasks with identical processing times, the expected makespan of Greedy-Makespan when $r \geq 2$ can be proven to be of order $|T|/|P| + O(\log\log|P|)$ (with high probability) [18].

In addition, Berenbrink et al. [15] introduce the number of "holes" in a balls-into-bins process, *i.e.* the number of missing balls in the least loaded bin to reach perfect load balancing, and proves it can be bounded by a linear function of $|P|$. In our case, with the use of Greedy-Comm, the holes can be interpreted as the number of time steps a processor stays idle before the end of the execution, and thus the number of tasks it steals from other resources. We investigate this $O(|P|)$ asymptotic behavior of the communication cost of Greedy-Comm in Section 7.

5 Semi-Matching Based strategies

In this section, we propose a new approach to solve MAKESPAN-ORIENTED and COMM-ORIENTED based on semi-matchings, a concept coming from graph theory. We focus on tasks with homogeneous durations, we prove that $\mathcal{ASM}_1$, a semi-matching-based strategy, is optimal for both problems and we propose a probabilistic evaluation of its performance for MAKESPAN-ORIENTED. These results establish an original link, to the best of our knowledge, between independent tasks scheduling with replication and semi-matching.

In Section 7, we show that $\mathcal{ASM}_1$ can be used as a basis to design efficient strategies in the more general case of heterogeneous tasks durations. Therefore, we dedicate this section to the theoretical results on $\mathcal{ASM}_1$, even if the context of homogeneous tasks durations can appear as too simplistic.

Section 5.1 establishes the relation between semi-matchings, MAKESPAN-ORIENTED and COMM-ORIENTED. Section 5.2 presents $\mathcal{ASM}_1$ and proves its optimality for both problems. Finally, Section 5.3 proposes a probabilistic evaluation of the optimal result for MAKESPAN-ORIENTED.

5.1 Equivalence of MAKESPAN-ORIENTED and COMM-ORIENTED with semi-matching problems

In order to find optimal solutions for both MAKESPAN-ORIENTED and COMM-ORIENTED problems, we rely on the concept of semi-matchings defined by Harvey et al. [24] and pioneered by Horn [25], Bruno et al. [26] and Abraham [27]. A semi-matching is a relaxation of the well-know matching problem on bipartite graph. For the record, a *matching* in a graph is a set of edges such that each vertex has at most one incident edge in this set. A *semi-matching* is a set of edges such that each vertex from the "left" set (the tasks in our case) has at most one incident edge in this set, with no condition for the vertices from the "right" set (the machines in our case), as defined below.

Definition 4 (Semi-matching)**.** *Let $G = (P, T, E)$ be a bipartite graph. A* semi-matching *of G is a subset M of E such that:*

$$\forall t_i \in T,\ \exists \text{ a unique } p_j \in P \text{ such that } \{p_j, t_i\} \in M.$$

Figure 2 depicts two different semi-matchings associated to one particular bipartite graph. The set T is on the left side and the set P is on the right side of each subfigure. Solid edges define the semi-matchings.

Note that vertices on the right can have several of their edges selected (this is the case for vertex 2 and vertex 3 on Figure 2(a) and for vertex 2 and vertex 4 on Figure 2(b)), but that is never the case for vertices on the left.

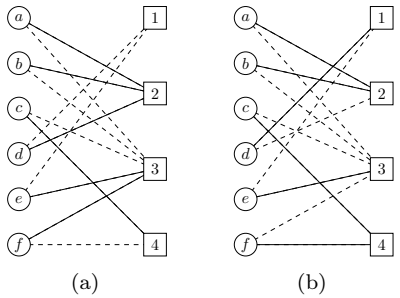


Figure 2: Two examples of semi-matchings for the same input graph.

Let us first focus on the link between MAKESPAN-ORIENTED and semi-matching, as it is more straightforward. As defined in Section 2, the input of MAKESPAN-ORIENTED is a bipartite graph representing tasks, processors and the initial placement of data chunks on these processors (represented by the edges). As tasks can be computed locally only (on processors owning the associated data chunk), there is always an edge between task $t_i \in T$ and the processor p_j on which it is processed. Therefore, MAKESPAN-ORIENTED can be interpreted as the selection for every task of a unique incident edge, that is the definition of a semi-matching. In order to evaluate the quality of semi-matching when solving MAKESPAN-ORIENTED, we propose the following metric.

Definition 5 (Degree in a semi-matching and maximum)**.** *Let M be a semi-matching of $G = (P, T, E)$, then*
(i) the degree in M *of a vertex p_j of P, denoted $d_M(p_j)$, is the degree of p_j in $G' = (P, T, M)$, the sub-graph of G induced by M,*
(ii) the maximum degree $d(M)$ *of M is defined as $D(M) = \max_{p_j \in P} d_M(p_j)$ and*

As an example, on Figure 2(a), the maximum degree of this semi-matching is 3 (and it is reached on processor 2). On Figure 2(b), the maximum degree is 2 (reached on 2 and 4)

The degree of a vertex on a semi-matching represents the number of edges that have been selected in this semi-matching. Therefore, in the context of MAKESPAN-ORIENTED, the degree of a processor is the number of tasks that will be processed on this processor. Thus minimizing the maximal degree of a semi-matching is equivalent to optimizing MAKESPAN-ORIENTED (if $D(M)$ is minimal, then $\max_{p_j \in P} |\{t_i,\ M(t_i) = p_j\}|$ is also minimal).

In the case of COMM-ORIENTED, the link with semi-matching is less straightforward. First, a solution to COMM-ORIENTED problem is not a semi-matching as soon as there is one non-local task (a task processed on a processor that initially does not store the required data chunk). In this case the transformation consists in two steps: 1) a semi-matching is found; 2) this semi-matching is transformed into a solution of COMM-ORIENTED (*i.e.* the number of task per processor should be balanced).

For the first step the question is: what is a good semi-matching to have a good solution for COMM-ORIENTED after the step 2)? To do so, we introduce a new metric for semi-matchings.

Definition 6 (Degree in a semi-matching and maximum)**.** *Let M be a semi-matching of $G = (P, T, E)$, then the* total load imbalance $Imb(M)$ *of M is given by*

$$Imb(M) = \sum_{\substack{p_j \in P \\ d_M(p_j) > \left\lceil \frac{|T|}{|P|} \right\rceil}} \left(d_M(p_j) - \left\lceil \frac{|T|}{|P|} \right\rceil \right).$$

In a more intuitive way, when computing the total load imbalance of a semi-matching M, we determine for every processor from P how many of its edges are in M and then consider the overhead when compared to the average ($|T|/|P|$). The number of these in-excess edges represents total load imbalance. To illustrate these notions, let us consider Figure 2. On Figure 2(a), the total load imbalance is 1 (the vertex 2 is the only one with more than two edges) while on Figure 2(b), the total load imbalance is 0 (every processor holds at most 2 tasks).

As defined in Section 2, the final output of COMM-ORIENTED is a schedule with less than $\lceil |T|/|P| \rceil$ tasks allocated on every processor. Thus, a semi-matching with load imbalance larger than 0 is impossible. However, the total load imbalance provides the number of tasks that have to been reallocated and thus an upper bound on the number of non-local tasks that will be created during the process. Theorem 7 formalizes this intuition and also proves that a solution to COMM-ORIENTED with a volume of communication of C can be transformed into a semi-matching M such that $Imb(M) \leq C$.

Theorem 7. *Given a bipartite graph G, for all semi-matching M, there exists a solution of COMM-ORIENTED with less or equal than $Imb(M)$ communication. Conversely, for all solution of COMM-ORIENTED, with C communications, there exists a semi-matching M such that $Imb(M) \leq C$.*

Proof. From a semi-matching M, let us build a valid schedule for COMM-ORIENTED. For every processor p_j such that $d_M(p_j) > \lceil |T|/|P| \rceil$, we unassign $d_M(p_j) - \lceil |T|/|P| \rceil$ tasks and attribute them to processors $p_{j'}$s such that $d_M(p_{j'}) < \lceil |T|/|P| \rceil$. In the end, we obtain a schedule σ such that $\max_{p_j \in P} |\{t_i,\ \sigma(t_i) = p_j\}| \leq \lceil |T|/|P| \rceil$ and $|\{t_i,\ \{\sigma(t_i), t_i\} \notin E\}| \leq Imb(M)$ as non-local tasks may only be tasks whose allocation has been modified by the previous transformation.

From a schedule σ solution of COMM-ORIENTED with $|\{t_i,\ \{\sigma(t_i), t_i\} \notin E\}| = C$ communications, let us now build a semi-matching as follows. Since σ is a solution of COMM-ORIENTED, then $\max_{p_j \in P} |\{t_i,\ \sigma(t_i) = p_j\}| \leq \lceil |T|/|P| \rceil$. Let us assign every non-local task to an arbitrary processor such that there is an edge between them. With this transformation, we obtain a valid semi-matching M and $Imb(M) \leq C$. Indeed, the set $\{p_j,\ d_M(p_j) > \lceil |T|/|P| \rceil\}$ is empty before the transformation (as $\{t_i,\ \sigma(t_i) = p_j\}| \leq \lceil |T|/|P| \rceil$) and thus during the transformation, $Imb(M)$ only increases by at most 1 at each new assignment of the non-local t_is. □

Theorem 7 states that for a given bipartite graph, if a semi-matching is optimal on the total load imbalance metric, then it can be transformed into an optimal schedule for COMM-ORIENTED (under the assumption that tasks durations are homogeneous). Moreover, the transformation is straightforward: for every processor with more than $\lceil |T|/|P| \rceil$ allocated tasks, reallocate the tasks in excess to processor with the smaller number of allocated tasks.

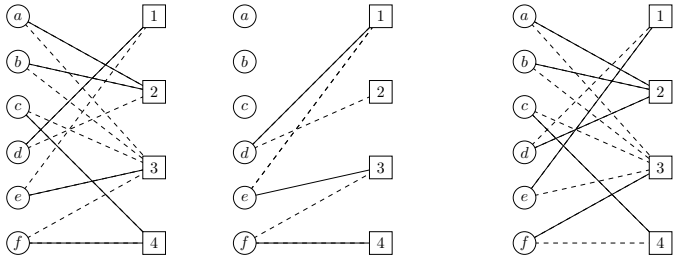
5.2 Optimal Algorithm for Semi-Matching Problems

Several algorithms exist to compute optimal (according to some metrics) semi-matchings. We can cite algorithms from Fakcharoenphol [28] (worst case complexity $O(\sqrt{|T|} \cdot |P| \cdot \log |T|)$) or Galčík et al. [29] (the authors propose two algorithms, one with worst case complexity $O(\sqrt{|T|} \cdot |P| \cdot \log |T|)$ and one with expected complexity $O(n^\omega \log^{1+o(1)} |T|)$ where ω is the best known exponent for matrix multiplication). In this paper, we rely on the the algorithm $\mathcal{ASM}_1$ from Harvey et al. [24] whose complexity is $O(|T| \cdot |P|)$, but that achieves low computation time in practice. $\mathcal{ASM}_1$ is based on the notion of *alternating* and *cost-reducing* paths.

Definition 8. *Let $G = (P, T, E)$ be a bipartite graph and let M be a subset of E.*

- *A* path *of G is a sequence of vertices $(x_1, \ldots, x_k)$ such that $\forall i \in [1, k-1]$, $(x_i, x_{i+1}) \in E$. Note that in a path of a bipartite graph the vertices switch between T and P.*
- *An* alternating path *of G according to M is a path $(x_1, \ldots, x_k)$ of G such that:*
  - *If $x_i \in T$, then $(x_i, x_{i+1}) \in M$.*
  - *If $x_i \in P$, then $(x_i, x_{i+1}) \notin M$.*

An example of alternating path is described in Figure 3. On Figure 3(a), solid edges represent a semi-matching, and dashed ones represent unused edges. On Figure 3(b), the proposed path is an alternating one according to the previous semi-matching (to improve the clarity of the scheme, edges that are not in the path have been removed). In this example, the alternating path is $2 - d - 1 - e - 3 - f - 4$ ($(2, d)$ is not in the semi-matching, $(d, 1)$ is).



(a) M (b) x (c) $M \otimes x$

Figure 3: Example of alternating path.

If M is a semi-matching and x is an alternating path, then $M \otimes x$ (where $\otimes$ denotes the XOR operator) is also an alternating path. More precisely, if the edges that were in x and M are removed from M while the edges that were in x and not in M are added to M, then this new set of edges is also a semi-matching. We illustrate this operation on Figure 3(c) (the initial semi-matching is the one from Figure 3(a) and the alternating path is the one from Figure 3(b)). Thus, using alternating paths can be used to improve semi-matchings. More precisely, if the first and last vertices of an alternating path are in P, then it is possible to build a semi-matching that improves the degree of the last vertex while increasing by one the degree of the first one. Such an alternating path is a cost-reducing path.

Definition 9. *Let $G = (P, T, E)$ be a bipartite graph and M be a semi-matching of G. An* cost-reducing path *according to M is an alternating path $x = (p_{j_1}, t_{i_1}, \ldots, p_{j_k})$ such that $d_M(p_{j_1}) + 1 < d_M(p_{j_k})$.*

Harvey et al. [24] prove that a semi-matching without cost-reducing path is optimal for many metrics, in particular the one that aims at minimizing the maximum or the sum of a convex function of the degree over all vertices of P, as stated in Theorem 10.

Theorem 10 (Harvey et al. [24])**.** *Let $F_G(M)$ be a metric function to minimize, with $G = (P, T, E)$ a bipartite graph and M a semi-matching of G. Let $f : \mathbb{R}^+ \mapsto \mathbb{R}$ be a convex function. Then if $F_G(M) = \max_{p \in P} f_{cost}(d_M(p))$ ou $F_G(M) = \sum_{p \in P} f_{cost}(d_M(p))$, then M is optimal for F_G if and only if there is no cost-reducing path in M.*

The functions $x \mapsto x$ and $x \mapsto \max\left(0, x - \left\lceil \frac{|T|}{|P|} \right\rceil\right)$ are convex and thus D and Imb fulfill the condition of Theorem 10. Therefore, every semi-matching without cost-reducing path minimizes both maximal degree and load imbalance.

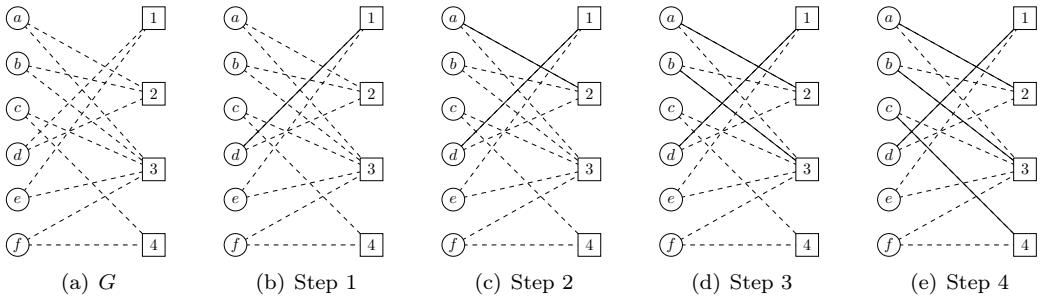
$\mathcal{ASM}_1$ algorithm considers the processors one after another, searching at each time for the smallest alternating path from this processor to a non-assigned task (if any) and applying it (through the XOR operator) to the current partial semi-matching, as described in Algorithm 1. Harvey et al. have proved that there is no cost-reducing path at the end of its execution and thus such a semi-matching optimally solve both MAKESPAN-ORIENTED and COMM-ORIENTED.

To illustrate $\mathcal{ASM}_1$, we depict a step by step execution on Figure 4. The input graph is depicted on Figure 4(a). During Step 1 (Figure 4(b)), processor 1 is considered. As one of its neighbors (task d) has not been allocated yet, the path $1 - d$ is an alternating path to an unallocated task and can therefore be added to the semi-matching. During following steps, the same situation occurs: one of the neighbors of a considered processor is unallocated and thus single-edge alternating paths can be found. This way, edges

Algorithm 1: $\mathcal{ASM}_1\ (G)$

Input: A bipartite graph $G = (P, T, E)$
Output: A semi-matching M of G of minimum maximal degree and minimum total load imbalance
$M = \emptyset$;
while *M is not a semi-matching* **do**
  foreach $p_j \in P$ **do**
    if *There is an alternating path according to M from p_j to an unassigned t_j* **then**
      $x =$ the smallest such path ;
      $M \leftarrow M \otimes x$;
    else
      Remove p_j from P ;
return M

$(2, a)$ (Figure 4(c)), $(3, b)$ (Figure 4(d)), $(4, c)$ (Figure 4(e)) and $(1, e)$ (Figure 4(f)) are added to the semi-matching. However, after Step 5, all the neighbors of the processor 2 (the tasks a, b and d) are already allocated. Thus, a longer alternating path is needed. Such a path is depicted on Figure 4(g): $2 - b - 3 - f$. So, by applying this path to the current semi-matching, edge $(3, b)$ is removed and replaced by edges $(2, b)$ and $(3, f)$ (Figure 4(h)). After this step, there is no remaining unallocated task and $\mathcal{ASM}_1$ terminates its execution and returns semi-matching M.



(a) G (b) Step 1 (c) Step 2 (d) Step 3 (e) Step 4

(f) Step 5 (g) x during Step 6 (h) Returned M

Figure 4: Execution of $\mathcal{ASM}_1\ (G)$

The worst case complexity of $\mathcal{ASM}_1$ is $O((|T|+|P|)|E|)$ ($O(|T|^2+|T||P|)$ in the case $|E| = r|T|$). In this formula, $|E|$ corresponds to the cost of alternating paths search. As illustrated above, there are in practice many one-edge alternating paths that are found during first steps, reducing then the practical complexity of $\mathcal{ASM}_1$. We wrote a C implementation to perform the simulations depicted in Section 7 and we measured its computation time on an Intel® Core™ i7-4600M CPU @ 2.90GHz. The results are depicted in Figure 5. In [12], the objective for a scheduler operating at the level of a cluster is a scheduling time of at most $1s$. The average computation time of $\mathcal{ASM}_1$ is below $1s$, except for very high values of $|T|$, *i.e.* when there are

more than 250,000 tasks, which is much larger than typical number of tasks. For example, it corresponds to 3 times the number of tasks of the largest job in [2]. Anyway, even in this case, the average computation time is $3s$. Note that a cluster of 10,000 machines is not unrealistic, a Google cluster is composed of 12500 machines according to [12].

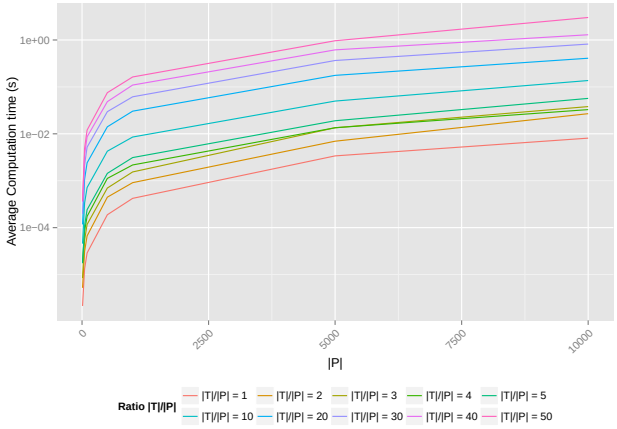


Figure 5: Average computation time in second of $\mathcal{ASM}_1$ for different values of $\frac{|T|}{|P|}$ and $|P|$.

5.3 Probabilistic Evaluation for Makespan Metric

Let us now formalize our allocation problem using the GRAPH-ORIENT problem. In GRAPH-ORIENT, the goal is, for an undirected graph, to orient each edge so as to obtain a directed version of the graph, where the in-degree of each node is minimized. To formally state GRAPH-ORIENT problem, we distinguish (undirected) edges, denoted $\{u, v\}$ from (directed) arcs, denoted (u, v).

Problem 3 (GRAPH-ORIENT)**.** *Given an undirected graph $G = (V, E)$, find a directed version $G' = (V, E')$, with $\{u, v\} \in E \Leftrightarrow (u, v) \in E'$ or $(v, u) \in E'$, such that $\max_{v \in V} |\{u \in V,\ (u, v) \in E'\}|$ is minimized.*

There is a clear correspondence between GRAPH-ORIENT and MAKESPAN-ORIENTED in the case where tasks are replicated twice, *i.e.* $r = 2$ (we consider other values later). Consider such an instance $G = (P, T, E)$ of MAKESPAN-ORIENTED. Let $G' = (P, E')$ be an undirected graph where vertices corresponds to processors in MAKESPAN-ORIENTED, and each edge corresponds to a task, as follows: $\{p_j, p_{j'}\} \in E'$ if and only if $\{p_j, t_i\}$ and $\{p_{j'}, t_i\}$ are in E. Note that multiple edges are allowed between any two vertices, such that $|E'| = |T|$.

From a solution to GRAPH-ORIENT, we can build a solution to MAKESPAN-ORIENTED as follows. We consider a directed version $G'' = (P, E'')$ of G' and we define a semi-matching M: $\{p_j, t_i\} \in M$ if and only if $(p_{j'}, p_j) \in E''$. We can easily check that it is a valid semi-matching and that its maximal degree is the maximal number of incident arc to a vector in G'': an edge represents a task, and the orientation of an edge represents the choice of the processor in charge of this task. Figure 6 illustrates this correspondence: Figure 6(a) presents the original instance of MAKESPAN-ORIENTED and Figure 6(b) shows its transformation into an orientation problem. Figure 6(d) is a solution of the orientation problem and Figure 6(c) is the associated semi-matching in the original problem.

Note that GRAPH-ORIENT can be extended to hypergraphs to model versions of MAKESPAN-ORIENTED with an arbitrary replication factor r.

A major contribution to GRAPH-ORIENT has been proposed by Sanders et al. [30], where polynomial time algorithms are proposed for GRAPH-ORIENT and its hypergraph version, together with a probabilistic

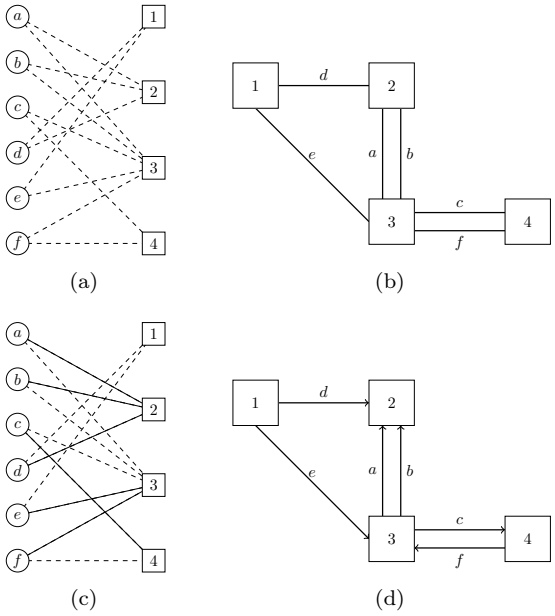


Figure 6: Figure 6(a) represents an input of MAKESPAN-ORIENTED which equivalent version of an input of GRAPH-ORIENT is on Figure 6(b). The semi-matching on Figure 6(c) is equivalent to the orientation of Figure 6(d).

evaluation of the expected value of the optimal value of an input of GRAPH-ORIENT. More precisely, with high probability, the optimal solution is at least *near-perfect* (*i.e.* $\max_{v \in V} |\{u \in V,\ (u,v) \in E'\}| \leq 1 + \left\lceil \frac{|E'|}{|P|} \right\rceil$).

Theorem 11 (Sanders et al. [30])**.** *Let $G = (V, E)$ be an undirected graph. With high probability, there exists a directed version of G such that the vertex with the most in-incident arcs has an in-degree inferior or equal to $1 + \left\lceil \frac{|E|}{|V|} \right\rceil$.*

Therefore, under the assumption that each task has at least two replicas, Theorem 11 says that there exists a near-perfect semi-matching for the MAKESPAN-ORIENTED problem, as stated in the following corollary.

Corollary 12. *Let $G = (P, T, E)$ be a bipartite graph such that the degree of every element of T is at least 2. Then, with high probability, there exists a semi-matching with maximal degree smaller or equal to $1 + \left\lceil \frac{|T|}{|P|} \right\rceil$.*

Proof. By randomly removing excess edges, let us consider the case where for all $t_i \in T$, the degree of t_i is 2. Then, G can be transformed into an undirected graph $G' = (P, E')$ as described above (see Figure 6). Theorem 11 proves that there is a directed version of G' such that the vertex with the most in-incident arcs has an in-degree less or equal to $1 + \left\lceil \frac{|E'|}{|P|} \right\rceil = 1 + \left\lceil \frac{|T|}{|P|} \right\rceil$. This oriented version of G' induces a semi-matching for G with the transformation described in Figure 6, and this semi-matching has the same maximal degree as the maximal in-degree of G'', what achieves proof of Corollary 12. □

Furthermore, Theorem 11 has been extended by Czumaj et al. [31] by proving that when the number of tasks is small, then the optimal solution of GRAPH-ORIENT is exactly $1 + \left\lceil \frac{|T|}{|P|} \right\rceil$ and exactly $\left\lceil \frac{|T|}{|P|} \right\rceil$ when the number of tasks is high.

Theorem 13 (Czumaj et al. [31])**.** *Let $G = (V, E)$ be an undirected graph. There exist two positive constants λ, c such that:*

- *If* $|E| \geq c|V| \log |V|$*, then with high probability there exists a directed version of* G *such that the vertex with the highest number of incident arcs has an in-degree equal to* $\left\lceil \frac{|E|}{|V|} \right\rceil$.
- *If* $|E| \leq \lambda|V| \log |V|$*, then with high probability there exists a directed version of* G *such that the vertex with the highest number of incident arcs has an in-degree equal to* $1 + \left\lceil \frac{|E|}{|V|} \right\rceil$.

Thus, the expected behavior of optimal solutions for MAKESPAN-ORIENTED and COMM-ORIENTED with homogeneous tasks can be summarized as follows.

- For large values of $\frac{|T|}{|P|}$, there exists a perfect semi-matching and thus solutions with optimal makespan and no communication (since the existence of a perfect semi-matching ensures a solution without communications)
- For small values of $\frac{|T|}{|P|}$, the makespan is almost always $\left\lceil \frac{|T|}{|P|} \right\rceil + 1$. Moreover, there is no solution without communications and the number of communications can be bounded by $|P|$ (one at most per processor).

6 Heuristic Strategies for Heterogeneous Tasks

In Section 5, we assume that all the chunks have the same size. However, in practice, the time needed to process one chunk may depend on the data itself. This results in heterogeneity in task durations, and may degrade the performance of both the simple greedy heuristics of Section 4 and the more involved strategies of Section 5. We illustrate such a behavior in the simulation results presented in Section 7.3.

We therefore introduce two strategies that can be used in a scenario where tasks have heterogeneous durations. We consider two cases, depending if the information on task durations is available in advance to the scheduler (denoted as the clairvoyant cases) or not (non-clairvoyant). All the strategies presented in this section have variants for both the MAKESPAN-ORIENTED and the COMM-ORIENTED problem, depending if non-local task processing is allowed or not.

6.1 Non-clairvoyant case

In this setting, we assume that the tasks to be scheduled have possibly heterogeneous durations, but the exact duration of a specific task is unknown to the scheduler prior its execution. Performing load-balancing at runtime seems a natural way of dealing with this situation, and is indeed already a feature of the greedy strategies Greedy-Comm and Greedy-Makespan presented in Section 4). However, more involved heuristics may also rely on data placement to better chose which task should be started on what processor. We present three such heuristics below.

6.1.1 Maestro

The first allocation strategy, called Maestro has been proposed by Ibrahim et al. in [8]. It relies on the following parameters

- For a processor p_j, HCN_j denotes its *host chunk number*, *i.e.* the number of data chunks corresponding to unprocessed tasks and owned by this processor.
- The *chunk weight* of task t_i is given by

$$Cw_i = 1 - \left(\sum_{j=j_1}^{j=r} \frac{1}{HCN_j} \right),$$

where $p_{j_1}, \ldots, p_{j_r}$ are the processors that own the replicas of the data chunk of t_i. The goal of this value is to evaluate how likely the task t_i may be processed non-locally. If the data chunks of t_i are placed on resources with already many other data chunks (thus with high HCN_js), then t_i have less chance to be compute on one of these resources because of the competition from the other potential local tasks (Cw_i is high). Conversely, a task placed on resources with low HCN_js will probably be processed locally (Cw_i is low).

- For two distinct processors $p_j, p_{j'}$, the number of shared chunks between them is denoted by $Sc_j^{j'}$. Using above notations, $Sc_j^{j'}$ denotes the number of tasks t_i for which there exists k and k' with $j = R_i^{(k)}$ and $j' = R_i^{(k')}$.
- The node weight of a processor p_j is defined as

$$NodeW_j = \sum_{i=1}^{HCN_j} \left(1 - \sum_{k=1}^{r} \frac{1}{HCN_{j_{i,k}} + Sc_{j_{i,k}}^{j}}\right),$$

where $(p_{j_1}, \ldots, p_{j_r})$ are the processors that own data chunk of task t_i. For a processor, a large $NodeW_j$ values indicates that it has little chance to process non-local tasks.

Maestro works in two phases. During the first phase, Maestro assigns m tasks, exactly one per processor. It first selects the processor with the smallest node weight $NodeW_j$ (a processor with few hosted chunks or/and many shared chunks with other critical processors) and allocates to it its local task with the largest chunk weight, *i.e.* the task with the largest chance of being stolen. In other words, it gives to the processor with the highest chance to process non-local tasks the local task with highest chance to be processed non-locally. This phase is computationally intensive and its complexity can be estimated to $O(n_{max}|P|^2)$, where n_{max} is the number of chunks owned by the most loaded processor.

During the second phase, which is purely dynamic, each time a processor becomes idle and requests a task, the local task with the highest chunk weight is allocated to this processor and processed. The makespan-oriented version of Maestro stops here. For communication minimization, non-local processing is allowed: if no local task is available on a requesting processor, an arbitrary unprocessed task is stolen from another processor. For a detailed pseudo-code of this two phases see [32].

Finally, note that in both cases, the Maestro strategy slightly differ from [8] as we do not launch speculative tasks to deal with failures. Therefore, the results of Maestro in our simulations (in Section 7) are slightly better for the communications metric.

6.1.2 Locality Aware Greedy Strategies

These heuristics rely on the same idea as Maestro, *i.e.* favor (i) tasks with high estimated risk to be processed non-locally and (ii) processors with high risk of processing non-local tasks. These heuristics have nevertheless a lower computational cost. As for the previous greedy strategies, decisions are taken when a resource p_j becomes idle. Among the unprocessed tasks owned by p_j, a greedy strategy is used to determine which task t_i it should process, given the state of the other resources owning the data chunk associated to t_i. Indeed, p_j should rather choose the task whose makespan is expected to be the largest, what corresponds to the task for which candidate processors still own a large number of unprocessed tasks. In order to estimate the expected completion time of a task, we propose two variant of this heuristic. In both cases, each resource p_j maintains the number n_j of still unprocessed tasks for which it owns the input data chunk. When a resource p_j becomes idle, it chooses the local task t_i such that $f(t_i)$ is maximal (ties are broken arbitrarily). In the LocAwareGreedyMin variant, $f(t_i)$ is chosen as

$$f(t_i) = \min\left(n_{j^{(1)}}, \ldots, n_{j^{(r)}}\right),$$

where $j^{(1)}, \ldots, j^{(r)}$ are the indices of processors owning chunk of t_i. In the LocAwareGreedyAvg variant, $f(t_i)$ is chosen as

$$f(t_i) = \text{mean}\left(n_{j^{(1)}}, \ldots, n_{j^{(r)}}\right).$$

In the case when non-local task processing is allowed (COMM-ORIENTED problem), when a processor has no more local tasks to process, the same criterion is used to choose the candidate task among available non-local tasks.

6.2 Clairvoyant case

Let us now assume that tasks are heterogeneous, but that their duration is known in advance. In this case, this information can be taken into account when deciding the allocation of tasks to processors. The strategies designed in this framework will be used in the simulations presented in Section 7.3, in particular to decide whether the knowledge of task durations is needed to optimize the performance in the heterogeneous case.

A natural strategy consists in using the classical *Earliest Time First* criterion. Let us assume that task durations $t_1, \ldots, t_n$ are sorted so that $t_1 \geq t_2 \geq \ldots \geq t_n$. We allocate the tasks t_1 to t_n as follows. At step i, we allocate t_i on the resource p_j such that p_j owns data chunk of t_i and the load of p_j is minimam, given already allocated tasks t_1 to t_{i-1}. This corresponds to extending the *Longest Processing Time First* criterion widely used in the scheduling and bin-packing literatures to take into account the locality constraint. In the COMM-ORIENTED case, this criterion is applied in priority to local tasks and then, if there is no more such tasks, to all the remaining ones without locality constraint.

Another option is to extend the Locality Aware Greedy to take into account the available information on task durations. Each resource p_j maintains (i) the list of durations Rem_j of still unprocessed tasks for which it owns the input data files and (ii) its ready time RT_j, given already allocated tasks. Then, two options are considered for estimating the processor completion times. When a resource p_j becomes idle, it chooses the task t_i such that $f(t_i)$ is maximal (ties are broken arbitrarily), where

$$f(t_i) = \min_{1 \leq l \leq r} \left(RT_{j^{(l)}} + \text{Rem}_{j^{(l)}} \right) \qquad \text{(LocAwareClairGreedyMin variant), or}$$

$$f(t_i) = \operatorname*{mean}_{1 \leq l \leq r} \left(RT_{j^{(l)}} + \text{Rem}_{j^{(l)}} \right) \qquad \text{(LocAwareClairGreedyAvg variant).}$$

Again, in the case of COMM-ORIENTED problem, the same criterion is applied on non-local tasks when no more local tasks are available.

7 Performance Evaluation through Simulations

In this section, we propose some experimental evaluation of the strategies described in Section 6, 5 and 4. The objectives of these simulations are the following:

- Evaluate the gain of optimal solutions in comparison to the pre-existing greedy ones in the case of homogeneous tasks;
- Assess the performance of the strategies, designed for homogeneous tasks, when confronted to heterogeneous tasks;
- Measure the performance variation between clairvoyant and non-clairvoyant strategies.

7.1 Allocation strategies used in the simulations

We concentrate on 3 classes of non-clairvoyant scheduling strategies, *i.e.* Greedy-based (Section 4), $\mathcal{ASM}_1$-based (Section 5.2) and locality-aware based strategies (Maestro-based, LocAwareGreedyMin-based and LocAwareGreedyAvg-based, Section 6). Clairvoyant versions or these strategies (except $\mathcal{ASM}_1$ and Maestro) have also been proposed (Section 6.2). All these strategies exist with two modes: MAKESPAN-ORIENTED and COMM-ORIENTED. We already present the two versions of Greedy-based and locality-aware based strategies in the previous sections.

Let us now present $\mathcal{ASM}_1$-based strategies:

- Static-$\mathcal{ASM}_1$-Makespan uses the assignment (statically) computed by $\mathcal{ASM}_1$. Note that on uniform settings, any optimal-certified algorithm will achieve the same efficiency.
- $\mathcal{ASM}_1$-Makespan uses the assignment (statically) computed by $\mathcal{ASM}_1$. If a processor is idle with no pre-assigned tasks available, it randomly chooses an unprocessed task among those for which it owns a chunk locally. This strategy is only used in Section 7.3 to adapt $\mathcal{ASM}_1$ to the heterogeneous case.
- $\mathcal{ASM}_1$-Communications uses the assignment (statically) computed by $\mathcal{ASM}_1$. If a processor is idle with no pre-assigned tasks, it first tries to process one of its local tasks. If there is no longer unprocessed local tasks, then the processor steals a task among the local tasks of the processor with the largest number of unprocessed assigned tasks at this instant. This stealing procedure is close to the transformation described in Theorem 7. A processor is considered as idle when its degree is below $\left\lceil \frac{|T|}{|P|} \right\rceil$. As the stolen processor has the highest degree, its degree is above $\left\lceil \frac{|T|}{|P|} \right\rceil$. If the stolen task is local, the degree of the stolen processor may fall below $\left\lceil \frac{|T|}{|P|} \right\rceil$ but in this case, this is equivalent to using an alternating path without changing the load imbalance.

In the following plots, we never mix simulation results for MAKESPAN-ORIENTED and COMM-ORIENTED strategies. Thus, as there is no ambiguity, we do not indicate the version of the heuristics used in order to have shorter names.

7.2 Homogeneous Setting

In this section, we focus on the case where all tasks have the same computation time. For each strategy, we run 250 simulations for $|P| = 50$, for $\frac{|T|}{|P|} \in \{1, 2, 3, 4, 5, 10, 20, 30, 40, 50\}$ and for $r \in \{2, 3, 4, 5\}$ where r is the number of replications of each chunk ($r = 3$ by default HDFS, [4]).

7.2.1 Makespan metric

We first focus on MAKESPAN-ORIENTED, *i.e.* the case where non-local tasks are forbidden. As stated in Section 5.3, the asymptotically expected optimal makespan is $\frac{|T|}{|P|} + 1$ (for a small number of tasks per processor) or $\frac{|T|}{|P|}$ (for a large number of tasks per processor) and, as shown in Section 5.2, $\mathcal{ASM}_1$ is optimal.

The results of the simulations are depicted on Figure 7, where we display the average overhead, *i.e.* the makespan minus $\frac{|T|}{|P|}$ (the perfect load-balancing). We can notice the validity of the asymptotic theoretical results even for relative small values. Indeed, even in the case of small $|P|$ or $|T|$, the makespan is consistently below $\frac{|T|}{|P|} + 1$ (on more than 10^5 runs, the optimal makespan was $\frac{|T|}{|P|} + 2$ in only two cases).

Moreover, we can notice a three phases behavior of the optimal value $\mathcal{ASM}_1$. First, for small $|T|$ (*i.e.* small $\frac{|T|}{|P|}$), the optimal is $\frac{|T|}{|P|} + 1$. Then, during a transition phase, the average of this optimal value is between $\frac{|T|}{|P|}$ and $\frac{|T|}{|P|} + 1$ and finally, for large values of $|T|$ the optimal is $\frac{|T|}{|P|}$. At last, we can observe the behavior predicted by Theorem 13, with $|T| \leq \lambda |P| \log |P|$ at first and $|T| \geq c|P| \log |P|$ at the end. Note that according to our results, constants c and λ depend on the value of r. As for the results of Greedy, LocAwareGreedyMin, LocAwareGreedyAvg and Maestro, we can notice note that all of them are, on average, above $\frac{|T|}{|P|} + 1$ and, except sometimes for $|T| = |P|$ with Maestro, never optimal. In general, there is a clear gain when using an optimal static strategy in place of a dynamic one. Note that Maestro strongly relies on its second wave, that is purely dynamic, when $|T| > |P|$. In such cases, the results of Maestro are undistinguished to the ones of LocAwareGreedyMin and LocAwareGreedyAvg, that have smaller computational costs.

7.2.2 Communication metric

Let us now COMM-ORIENTED, *i.e.* the case where non-local tasks are allowed. We remind that, according to Berenbrink et al. [15] (see Section 4.2), the expected amount of communication is bounded by a linear

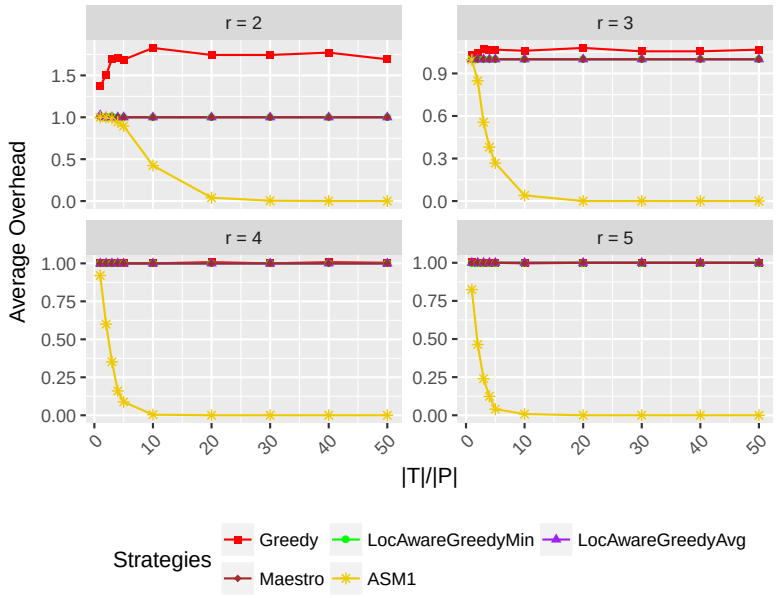


Figure 7: Results for the makespan metric in the homogeneous settings when $|P| = 50$. MAESTRO, LOCAWAREGREEDYMIN and LOCAWAREGREEDYAVG always overlap with each other, and overlap with GREEDY for $r = 4$ and $r = 5$.

function of $|P|$. Let us also notice that an assignment of maximal degree $|T|/|P|$ has a load imbalance of 0. Thus, we know that there exists a constant c such that, with high probability, there exists a solution with no communications if $|T| \geq c|P| \log |P|$.

The results of our simulations are depicted on Figure 8. Like for the makespan metric, there are several phases for the $\mathcal{ASM}_1$ strategy. In a first phase, there is not always a solution without communications, *i.e.* there is not always a solution of MAKESPAN-ORIENTED where the maximal degree is equal to $\frac{|T|}{|P|}$ In a second phase, $\mathcal{ASM}_1$ manages to completely avoid communications if there are enough tasks with respect to the number of processors. Note that even in the first phase, the number of non-local tasks is below 15% in the worst case and in general below 5% as soon as $r \geq 2$.

Like for the makespan metric, there is a clear hierarchy between GREEDY and the locality-aware strategies. MAESTRO performs slightly better, in particular for small number of tasks per processor whereas GREEDY can achieve more than 30% of non-local tasks. However, if greedy strategies are relatively close to $\mathcal{ASM}_1$ when the number of tasks per processor is large, the fraction of non-local tasks is always positive. This can be seen on Figure 9 and on Figure 7 (if there were no communication, then the results would be optimal for the makespan metric). Figure 9 also allows a better view of the relative performances on the locality-aware strategies. We can observe that the first wave of MAESTRO induces a small gain when $|T| = |P|$, but this advantage disappear otherwise. LOCAWAREGREEDYAVG seems slightly more efficient than LOCAWAREGREEDYMIN. One can also note that $\frac{|T|}{|P|}$ value has little impact on the performance of greedy strategies (except for small values) and their performance seems rather related to $|P|$ and r, which is consistent with the claims from Berenbrink et al. [15].

In a more general way, for both metrics, increasing r slightly improves the results and mainly benefits to $\mathcal{ASM}_1$. The value $r = 3$, used by default in HDFS, seems to be a good trade-off between replication cost and communication-avoiding parallel execution.

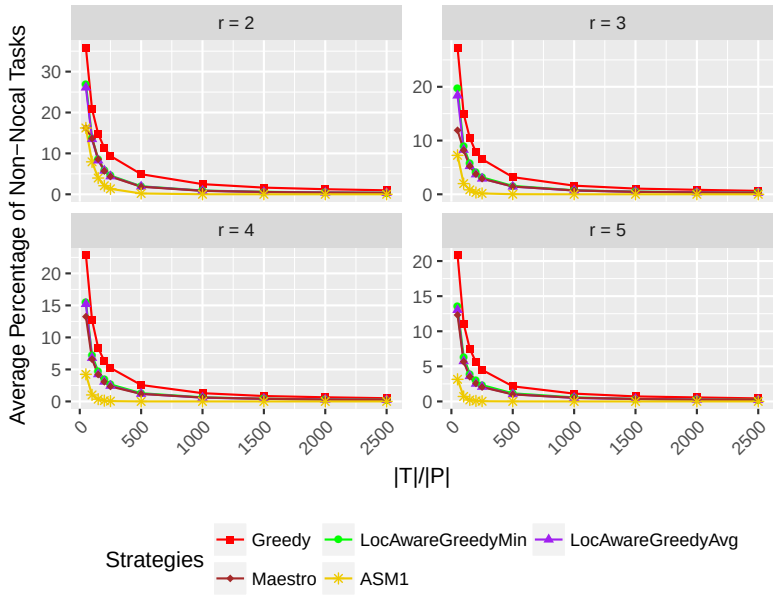


Figure 8: Results for the communication metric and homogeneous settings with $|P| = 50$. The results of LocAwareGreedyMin and LocAwareGreedyAvg overlap and Maestro is slightly below.

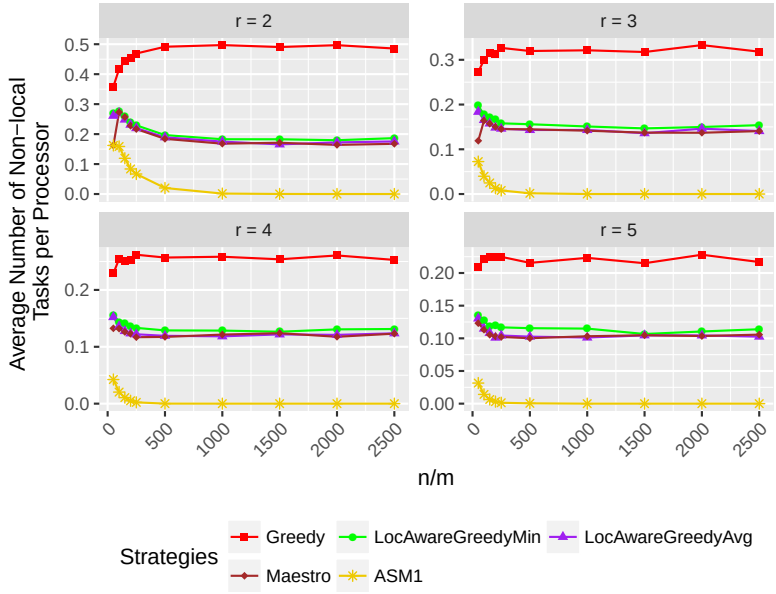
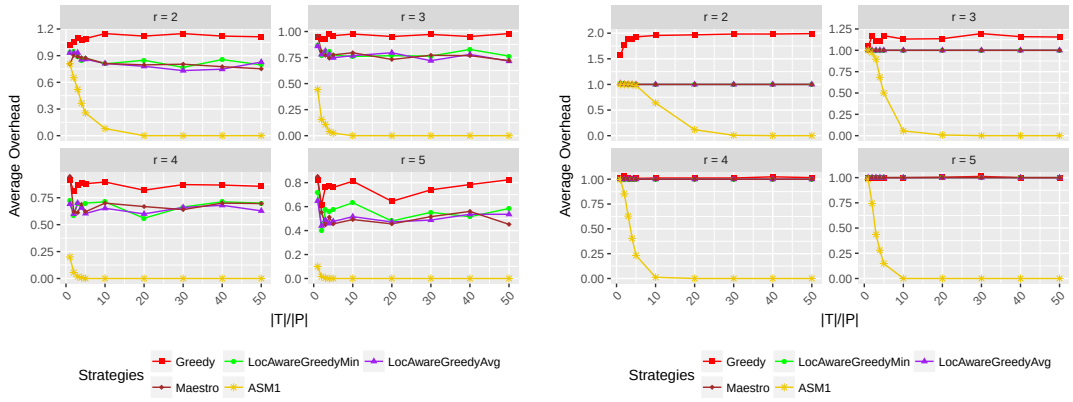


Figure 9: Non-local tasks per processor for homogeneous settings and $|P| = 50$. Maestro and LocAwareGreedyAvg sometimes overlap.

7.2.3 Influence of the number of processors

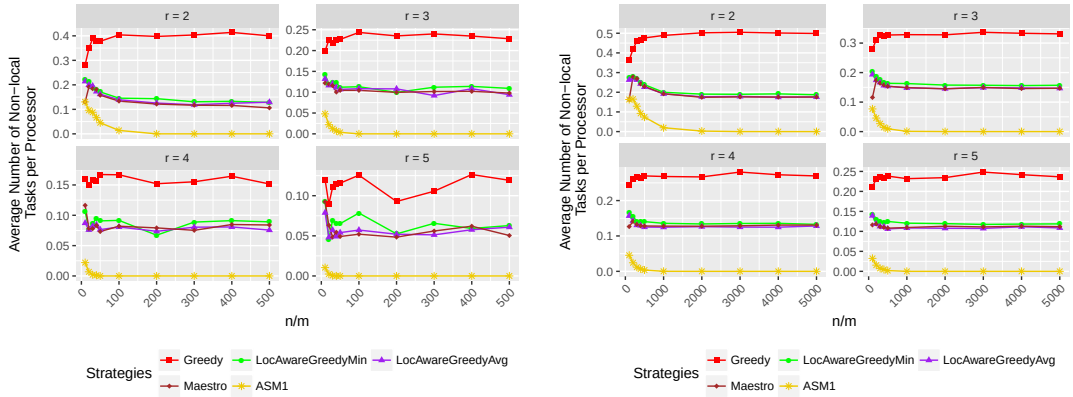
In the previous experiment sets, we focused on a fixed number of processors and then vary the ratio $|T|/|P|$ by changing the number of tasks $|T|$. The first reason is to have similar results than in Section 7.3, when

we use traces where the number of tasks is fixed (and thus the only degree of liberty is the ratio $|T|/|P|$). The second reason is that the general behavior of the different strategies is mainly determined by the ratio $|T|/|P|$. This phenomenon can be observed in Figures 10 and 11.



(a) $|P| = 10$ (b) $|P| = 100$

Figure 10: Results for the makespan metric, homogeneous settings and different number of processors.



(a) $|P| = 10$ (b) $|P| = 100$

Figure 11: Results for the communication metric (average number of non-local tasks per processors), homogeneous settings and different number of processors.

Globally, the results for 100 processors are very close to the one with 50 processors (a slightly larger average makespan for GREEDY strategy is the main difference). In the case of 10 processors, there is more instability. However, the general trend is the same and the difference between $\mathcal{ASM}_1$ and greedy strategies are similar to the ones for larger number of processor. The main difference is the inside hierarchy between the different locality-aware strategy (with no clear winner).

7.3 Heterogeneous Setting

In order to produce realistic scenarios for heterogeneous task durations, we use the traces from [2]. These traces correspond to the record of jobs during 10 months on a Hadoop cluster. In order to emulate non-clairvoyant scheduling, *i.e.* the fact that task durations are unknown when they are submitted, the computation time of each task is randomly picked at the beginning of its simulated execution. This is no longer the

case for clairvoyant strategies. In the following, we focus on jobs with at most 10000 tasks (more than 95% of the jobs in [2]). In addition, we classify jobs according to their Normalized Standard Deviation (NSD), *i.e.* the standard deviation of the computation times divided by the mean of the computation times, see Table 1.

	$NSD < 0.05$	$NSD \in [0.05, 0.1[$	$NSD \in [0.1, 0.25[$	$NSD \in [0.25, 0.5[$	$NSD \in [0.5, 1[$	$NSD \geq 1$	Total
$\|T\| < 50$	49 (2, 41%)	109 (5, 35%)	123 (6, 04%)	60 (2, 95%)	39 (1, 92%)	26 (1, 28%)	406 (19, 94%)
$\|T\| \in [50, 100[$	18 (0, 89%)	50 (2, 46%)	93 (4, 57%)	61 (3, 00%)	34 (1, 67%)	23 (1, 13%)	279 (13, 70%)
$\|T\| \in [100, 250[$	11 (0, 54%)	75 (3, 68%)	205 (10, 07%)	110 (5, 40%)	78 (3, 83%)	25 (1, 23%)	504 (24, 75%)
$\|T\| \in [250, 500[$	10 (0, 49%)	55 (2, 70%)	105 (5, 16%)	68 (3, 34%)	50 (2, 46%)	17 (0, 83%)	305 (14, 98%)
$\|T\| \in [500, 1000[$	1 (0, 05%)	5 (0, 25%)	21 (1, 03%)	44 (2, 16%)	33 (1, 62%)	18 (0, 88%)	122 (5, 99%)
$\|T\| \in [1000, 5000[$	1 (0, 05%)	10 (0, 49%)	43 (2, 11%)	32 (1, 57%)	33 (1, 62%)	76 (3, 73%)	195 (9, 58%)
$\|T\| \in [5000, 10000[$	0 (0, 00%)	16 (0, 79%)	57 (2, 80%)	33 (1, 62%)	16 (0, 79%)	10 (0, 49%)	132 (6, 48%)
$\|T\| \geq 10000$	0 (0, 00%)	9 (0, 44%)	4 (0, 20%)	14 (0, 69%)	39 (1, 91%)	27 (1, 33%)	93 (4, 57%)
Total	90 (4, 42%)	329 (16, 16%)	651 (31, 97%)	422 (20, 73%)	322 (15, 82%)	222 (10, 90%)	2036 (100%)

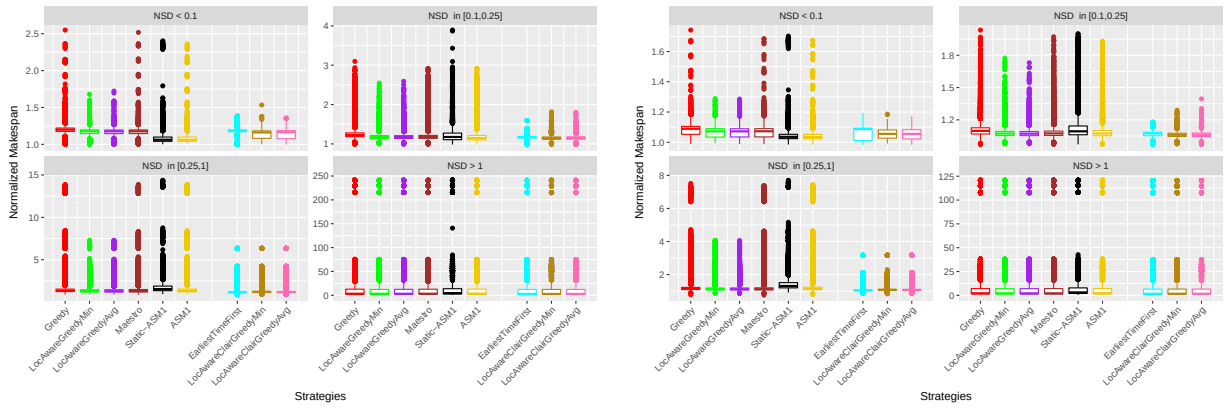
Table 1: Repartition in function of the number of tasks ($|T|$) and the normalized standard deviation (NSD) of the jobs from [2].

The replication ratio r is set to 3. Furthermore, for each job, we perform simulations for different values of $|T|/|P|$, $|T|$ being defined in the traces. At last, for each job, each $|T|/|P| \in \{1, 2, 5, 10\}$ and for each strategy we perform 50 simulations.

In all figures, we use boxplots to represent the results of the different experiments. In our boxplots, the box represents the values of the second and third quartiles with the horizontal bar set at the median. The vertical bars above and below the box goes from the second to the ninth decile. In all figures, non-clairvoyant strategies are on the left hand side, clairvoyant strategies are on the right hand side.

7.3.1 Makespan metric

The results for MAKESPAN-ORIENTED of these trace-based simulations are depicted in Figure 12. We only show results for $|T|/|P| = 5$ and $|T|/|P| = 10$. For smaller values of $|T|/|P|$, the differences between strategies are too small. The reason is rather simple. On the one hand, if the durations of the different tasks are close (small NSD), the results are close to the one presented in Section 4, where all strategies achieve close to $|T|/|P| + 1$ tasks per processor. In this case, clairvoyant strategies are a bit more stable, but return similar results than non-clairvoyant ones on average. This phenomenon can also be observed for larger $|T|/|P|$ values, see Figure 12. On the other hand, if the durations of the different tasks are too different, the makespan is dominated by the duration of the few longest tasks.



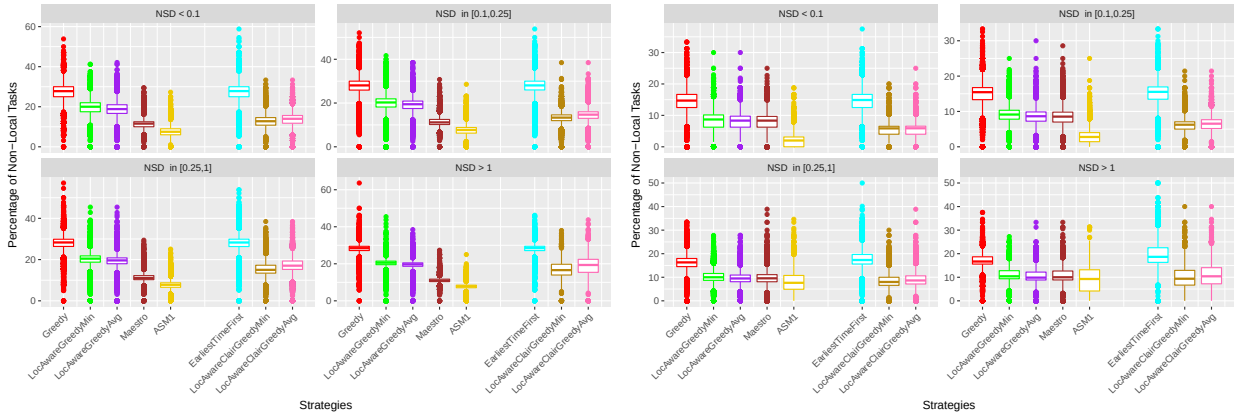
(a) $\frac{|T|}{|P|} = 5$ (b) $\frac{|T|}{|P|} = 10$

Figure 12: Results for the makespan metric and heterogeneous settings

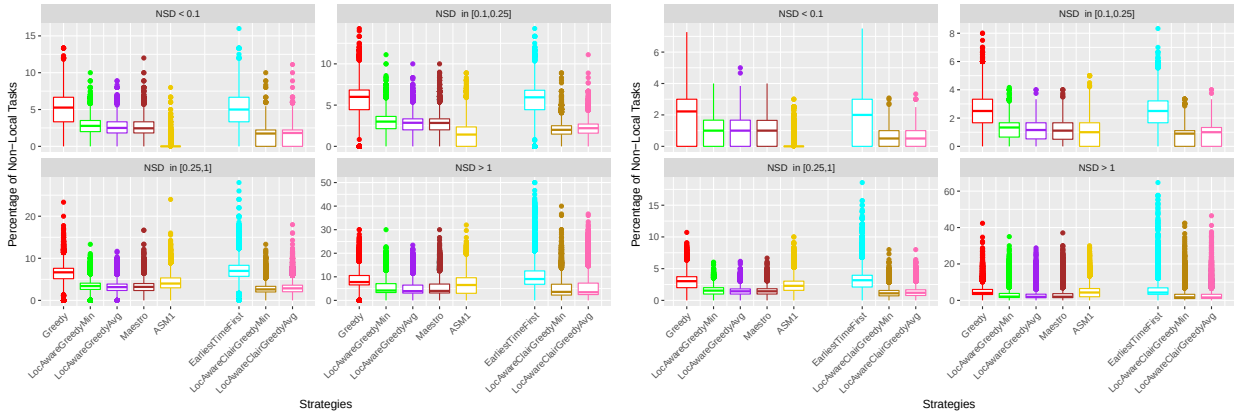
When $|T|/|P|$ is large enough, we observe some differences between $\mathcal{ASM}_1$ and greedy strategies. These differences mainly occur for small standard deviation, *i.e.* when the task durations are almost homogeneous. In such case, $\mathcal{ASM}_1$ performs even better than clairvoyant greedy strategies. Note that Static-$\mathcal{ASM}_1$ performs reasonably well for small standard deviation, but with no gain in comparison to $\mathcal{ASM}_1$, while it suffers with large standard deviation. Thus, the dynamic part in $\mathcal{ASM}_1$ is necessary.

7.3.2 Communication metric

As in the homogeneous case, we use the percentage of non-local tasks as a metric. Results are depicted in Figure 13.



(a) $\frac{|T|}{|P|} = 1$ (b) $\frac{|T|}{|P|} = 2$



(c) $\frac{|T|}{|P|} = 5$ (d) $\frac{|T|}{|P|} = 10$

Figure 13: Results for the communication metric and heterogeneous settings.

Let us consider the case $|T| = |P|$ (Figure 13(a)). For all strategies, no processor can be idle as long as there exists an unprocessed task. Therefore, each processor processes exactly one task (if the initial allocation gives no tasks to a processor, it will automatically steal one to another processor).

Therefore, the heterogeneity of computation times has no effect here on the communication cost. Thus, for all standard deviations, $\mathcal{ASM}_1$ performs well with about 7.5% of non-local tasks, to be compared with 11.4% for Maestro, 19.9% for LocAwareGreedyMin, 19.1% for LocAwareGreedyAvg and 25.8% for Greedy. The difference in this case between Maestro and the two other locality-aware strategies (LocAwareGreedyMin and LocAwareGreedyAvg) is explained by the first phase of Maestro (as

for homogeneous settings). The reason why LocAwareClairGreedyMin (15.7% of non-local tasks) and LocAwareClairGreedyAvg (14.2% of non-local tasks) perform better than LocAwareGreedyMin and LocAwareGreedyAvg is not obvious. However, all clairvoyant strategies are far less effective than $\mathcal{ASM}_1$.

Note that the small differences that appear between the results of two classes with different standard deviations likely come from the distribution of $|T|$ values, which differs from a class to another. We show in Section 7.2.3 that the number of processors (that is the number of tasks when $|T| = |P|$) slightly impact the results, in particular for small values.

When $|T|/|P| > 1$, the results are not as one-sided. First, as expected, a lower variance favors $\mathcal{ASM}_1$. The results of dynamic strategies are less impacted by heterogeneity, so that the gap between static and dynamic strategies decreases when heterogeneity increases. For instance, in the case $\frac{|T|}{|P|} = 5$ and $NSD < 0.1$ (Figure 13(c)), $\mathcal{ASM}_1$ results are close to those achieved in the homogeneous setting, with on average 0.13% of non-local-tasks whereas the fraction of non-local tasks reaches 2.61% with Maestro and 4.75% with Greedy. However the gap decreases for $NSD \in [0.1, 0.25[$ (1.51% against 2.67%) and finally, for greater NSD, Maestro performs better (5.13% for $\mathcal{ASM}_1$ against 3.96% for Maestro). Note that jobs with NSD value below 0.25 represent slightly more than half of the jobs.

Another important observation is that other locality-aware strategies performs similarly than Maestro, proving that it first wave has very few impact as soon as $|T|/|P| > 1$, as observed in the homogeneous case. We note that in general, LocAwareGreedyAvg slightly outperforms LocAwareGreedyMin.

In addition to NSD, the other important factor that influences the efficiency of the different strategies is the ratio $|T|/|P|$. All strategies produce less communications when $|T|/|P|$ increases, as already noticed in the homogeneous case. Therefore, as expected, when the number of tasks per processor is large and when the heterogeneity is important, the interest of relying on a mainly static efficient strategy like $\mathcal{ASM}_1$ decreases. Indeed, during the first phase, $\mathcal{ASM}_1$ tries to assign to each processor exactly $|T|/|P|$ tasks, *i.e.* the expected number of tasks to complete, whereas with Maestro or Greedy, tasks are assigned at runtime based on the state of the platform. Therefore, a high number of tasks and a higher heterogeneity favor dynamic strategies.

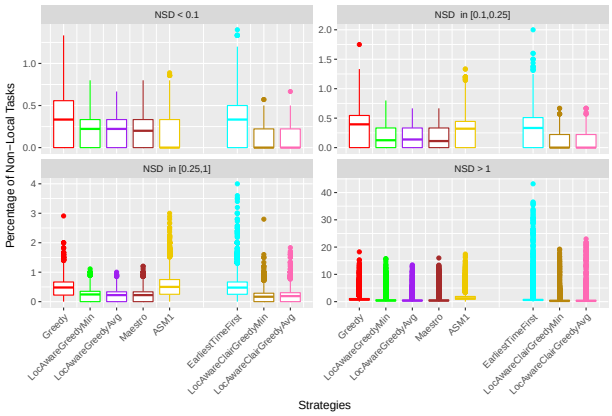


Figure 14: Results for the communication metric for $\frac{|T|}{|P|} = 50$.

For example, when $|T|/|P| = 1$ and $|T|/|P| = 2$, $\mathcal{ASM}_1$ is the most effective strategy, even for large standard deviation. However, as stated above, for $|T|/|P| = 5$ there are cases where $\mathcal{ASM}_1$ is less efficient than Maestro, LocAwareGreedyMin or LocAwareGreedyAvg and for $|T|/|P| = 10$ $\mathcal{ASM}_1$ is even challenged by Greedy in the case $NSD > 1$. In order to see if this trend is confirmed for larger ratios, we propose in Figure 14 an illustration of the case $|T|/|P| = 50$. In this setting, the only case where $\mathcal{ASM}_1$ is the indisputably better strategy is when $NSD < 0.05$, that represents few cases. However, even in the case where $\mathcal{ASM}_1$ is not the best strategy, its performance is good. For example, in the case $|T|/|P| = 50$, $\mathcal{ASM}_1$ allocates on average less than 2% of non local tasks. More generally, a percentage of non-local tasks

below 10% is almost always achieved (the only case where the average value is above 10% is for $|T|/|P| = 2$ and $NSD > 0.5$, but in this case Maestro is even worse).

For clairvoyant strategies, LocAwareClairGreedyMin and LocAwareClairGreedyAvg tend to slightly outperform LocAwareGreedyMin and LocAwareGreedyAvg. Thus, for large NSD and/or large $|T|/|P|$ values, they are dominant strategies. A possible explanation is that processors receiving longer tasks will very likely perform less tasks and local tasks only, whereas a processor receiving short tasks will choose the data chunks earlier and will get mostly local tasks too. This very interesting phenomenon would certainly deserve a theoretical analysis, but the analysis in the homogeneous case (see Section 4.2 for the makespan and [15] for the fraction of non-local tasks) is already difficult and becomes much harder in the heterogeneous setting. However, greedy clairvoyant strategies outperform non-clairvoyant ones only when the efficiency of $\mathcal{ASM}_1$ decreases. In such cases, we also observe a diminution of the gap between LocAwareGreedyMin/LocAwareGreedyAvg and LocAwareClairGreedyMin/LocAwareClairGreedyAvg. Thus, the benefit of greedy clairvoyant strategies is small.

To summarize this section, we prove that the hybrid strategy $\mathcal{ASM}_1$ outperforms all the other ones for small or medium values of the NSD or of $|T|/|P|$ ratio. In other cases, locality-aware strategy are dominant, with no clear winner between Maestro and LocAwareGreedyAvg, while the second is less expensive to compute due to its purely dynamic behavior.

8 Conclusion

We study how to allocate tasks whose input data is replicated on distributed processors. We provide closed form formulas to estimate the makespan and the number of communications induced by a natural greedy demand-driven heuristic, similar to the one use for Map tasks in MapReduce. We then investigate the use of sophisticated strategies based on semi-matching theory. We prove that if all tasks have the same duration, then the $\mathcal{ASM}_1$ algorithm returns optimal allocation for both makespan and communication metrics. When task durations are heterogeneous, we propose several fast heuristics taking advantage of data locality, depending if the information on task durations is available to the scheduler or not. We also perform a thorough experimental comparison, using simulations based on actual traces of MapReduce jobs in a production cluster. We prove that, except when heterogeneity and the number of tasks per processor are extremely large, $\mathcal{ASM}_1$ outperforms the best heuristic from the literature, Maestro, and also clairvoyant strategies, what shows the interest of designing efficient static scheduling strategies. In the other cases, clairvoyant strategies and Maestro achieve performance similar LocAwareGreedyAvg, another heuristic that we propose, while LocAwareGreedyAvg induces less computation cost (compared to Maestro) or requires less information (compared to clairvoyant strategies).

This work opens many perspectives in this direction. Indeed, predicting the performance of individual tasks and the duration of communications becomes more and more difficult, both for HPC and BigData applications. This motivates the design and massive use of dynamic schedulers, that take their decision at runtime, based on a correct state of the platform but a poor knowledge of the rest of the work to be processed, which sometimes leads to poor performance. The work presented here shows that in the context of Map tasks, injecting little static knowledge on the initial data distribution can help to make better decisions at runtime.

References

References

[1] O. Beaumont, T. Lambert, L. Marchal, B. Thomas, Data-Locality Aware Dynamic Schedulers for Independent Tasks with Replicated Inputs, in: IPDPSW 2018 IEEE International Parallel and Distributed Processing Symposium Workshops, IEEE, Vancouver, Canada, 2018, pp. 1–8.

[2] S. Kavulya, J. Tan, R. Gandhi, P. Narasimhan, An Analysis of Traces from a Production MapReduce Cluster, in: International Conference on Cluster, Cloud and Grid Computing (CCGrid), IEEE/ACM, 2010, pp. 94–103.

[3] M. R. Garey, D. S. Johnson, Computers and Intractability, a Guide to the Theory of NP-Completeness, W. H. Freeman and Co, 1979.

[4] D. Borthakur, HDFS Architecture Guide, HADOOP http://hadoop. apache. org/common/docs/current/hdfs design. pdf (2008) 39.

[5] M. Zaharia, D. Borthakur, J. Sen Sarma, K. Elmeleegy, S. Shenker, I. Stoica, Delay Scheduling: A Simple Technique for Achieving Locality and Fairness in Cluster Scheduling, in: European Conference on Computer Systems (EuroSys), ACM, 2010, pp. 265–278.

[6] Q. Xie, Y. Lu, Degree-Guided Map-Reduce Task Assignment with Data Locality Constraint, in: International Symposium on Information Theory (ISIT), IEEE, 2012, pp. 985–989.

[7] M. Isard, V. Prabhakaran, J. Currey, U. Wieder, K. Talwar, A. Goldberg, Quincy: Fair Scheduling for Distributed Computing Clusters, in: Symposium on Operating Systems Principles (SOSP), ACM, 2009, pp. 261–276.

[8] S. Ibrahim, H. Jin, L. Lu, B. He, G. Antoniu, S. Wu, Maestro: Replica-Aware Map Scheduling for MapReduce, in: International Symposium on Cluster, Cloud and Grid Computing (CCGrid), IEEE/ACM, 2012, pp. 435–442.

[9] W. Wang, K. Zhu, L. Ying, J. Tan, L. Zhang, Map Task Scheduling in MapReduce with Data Locality: Throughput and Heavy-Traffic Optimality, in: International Conference on Computer Communications (INFOCOM), IEEE, 2013, pp. 1609–1617.

[10] Z. Guo, G. C. Fox, M. Zhou, Investigation of Data Locality in MapReduce, in: International Symposium on Cluster, Cloud and Grid Computing (CCGrid), IEEE/ACM, 2012, pp. 419–426.

[11] L. R. Ford, D. R. Fulkerson, Flows in Networks, Princeton University Press, 2015.

[12] I. Gog, M. Schwarzkopf, A. Gleave, R. N. Watson, S. Hand, Firmament: Fast, Centralized Cluster Scheduling at Scale, in: Symposium on Operating Systems Design and Implementation (OSDI), USENIX, 2016, pp. 99–115.

[13] O. Selvitopi, G. V. Demirci, A. Turk, C. Aykanat, Locality-aware and load-balanced static task scheduling for mapreduce, Future Generation Computer Systems 90 (2019) 49 – 61.

[14] M. Raab, A. Steger, Balls into Bins: A Simple and Tight Analysis, in: Randomization and Approximation Techniques in Computer Science (RANDOM), Springer, 1998, pp. 159–170.

[15] P. Berenbrink, T. Friedetzky, Z. Hu, R. Martin, On weighted Balls-into-Bins Games, Theoretical Computer Science (TCS) 409 (3) (2008) 511–520.

[16] M. Mitzenmacher, The Power of Two Choices in Randomized Load Balancing, Transactions on Parallel and Distributed Systems (TPDS) 12 (10) (2001) 1094–1104.

[17] A. W. Richa, M. Mitzenmacher, R. Sitaraman, The Power of Two Random Choices: A Survey of Techniques and Results, Combinatorial Optimization 9 (2001) 255–304.

[18] P. Berenbrink, A. Czumaj, A. Steger, B. Vöcking, Balanced Allocations: The Heavily Loaded Case, in: Symposium on Theory of Computing (STOC), ACM, 2000, pp. 745–754.

[19] Y. Peres, K. Talwar, U. Wieder, The (1+ β)-Choice Process and Weighted Balls-into-Bins, in: Symposium on Discrete Algorithms (SODA), SIAM, 2010, pp. 1613–1619.

[20] A. Giersch, Y. Robert, F. Vivien, Scheduling tasks sharing files from distributed repositories, in: Proceedings of the 10th International Euro-Par Conference, 2004, pp. 246–253.

[21] K. Kaya, B. Uçar, C. Aykanat, Heuristics for scheduling file-sharing tasks on heterogeneous systems with distributed repositories, J. Parallel Distrib. Comput. 67 (3) (2007) 271–285. doi:10.1016/j.jpdc.2006.11.004.
URL https://doi.org/10.1016/j.jpdc.2006.11.004

[22] H. Casanova, A. Legrand, D. Zagorodnov, F. Berman, Heuristics for scheduling parameter sweep applications in grid environments, in: 9th Heterogeneous Computing Workshop, (HCW), 2000, pp. 349–363.

[23] M. Mitzenmacher, E. Upfal, Probability and Computing - Randomized Algorithms and Probabilistic Analysis, Cambridge University Press, 2005.

[24] N. J. Harvey, R. E. Ladner, L. Lovász, T. Tamir, Semi-matchings for bipartite graphs and load balancing, Journal of Algorithms 59 (1) (2006) 53–78.

[25] W. Horn, Minimizing average flow time with parallel machines, Operations Research 21 (3) (1973) 846–847.

[26] J. Bruno, E. G. Coffman Jr, R. Sethi, Scheduling independent tasks to reduce mean finishing time, Communications of the ACM 17 (7) (1974) 382–387.

[27] D. J. Abraham, Algorithmics of two-sided matching problems, Ph.D. thesis, Citeseer (2003).

[28] J. Fakcharoenphol, B. Laekhanukit, D. Nanongkai, Faster algorithms for semi-matching problems, ACM Transactions on Algorithms (TALG) 10 (3) (2014) 14.

[29] F. Galčík, J. Katrenič, G. Semanišin, On computing an optimal semi-matching, in: International Workshop on Graph-Theoretic Concepts in Computer Science, Springer, 2011, pp. 250–261.

[30] P. Sanders, S. Egner, J. Korst, Fast Concurrent Access to Parallel Disks, Algorithmica 35 (1) (2003) 21–55.

[31] A. Czumaj, C. Riley, C. Scheideler, Perfectly Balanced Allocation, Approximation, Randomization, and Combinatorial Optimization (APPROX) (2003) 240–251.

[32] T. Lambert, On the Effect of Replication of Input Files on the Efficiency and the Robustness of a Set of Computations, Ph.D. thesis, Université de Bordeaux, Talence (2017).