Data-Locality Aware Dynamic Schedulers for Independent Tasks with Replicated Inputs

Olivier Beaumont, Thomas Lambert, Loris Marchal, Bastien Thomas

Data-Locality Aware Dynamic Schedulers for Independent Tasks with Replicated Inputs

Olivier Beaumont*, Thomas Lambert†, Loris Marchal‡ and Bastien Thomas§
*Inria and University of Bordeaux, Talence, France
†University of Manchester, Manchester, United Kingdom
‡ENS of Lyon, Lyon, France
§ENS of Rennes, Ker Lann, France
olivier.beaumont@inria.fr, thomas.lambert@inria.fr, loris.marchal@inria.fr, bastien.thomas@ens-rennes.fr

***Abstract*—In this paper we concentrate on a crucial parameter for efficiency in Big Data and HPC applications: data locality. We focus on the scheduling of a set of independant tasks, each depending on an input file. We assume that each of these input files has been replicated several times and placed in local storage of different nodes of a cluster, similarly of what we can find on HDFS system for example. We consider two optimization problems, related to the two natural metrics: makespan optimization (under the constraint that only local tasks are allowed) and communication optimization (under the constraint of never letting a processor idle in order to optimize makespan). For both problems we investigate the performance of dynamic schedulers, in particular the basic greedy algorithm we can for example find in the default MapReduce scheduler. First we theoretically study its performance, with probabilistic models, and provide a lower bound for communication metric and asymptotic behaviour for both metrics. Second we propose simulations based on traces from a Hadoop cluster to compare the different dynamic schedulers and assess the expected behaviour obtained with the theoretical study.**

I. Introduction

Data locality is often 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 may lead to large spendings in both processing time, network congestion and energy consumption. Many distributed systems, in particular the ones used in big data analysis, rely on a file system that comes with a built-in replication strategy. This is for instance the case of the Hadoop Distributed File System (HDFS), which splits files into large blocks 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. While the original motivation for replication is data resilience (the whole data may be recovered in case of a node failure, or even a rack failure), it also improves data locality. The MapReduce scheduler takes advantage of the replication when allocating map tasks: whenever a node has a slot available for a map task, it is allocated a local task if possible, that is, a task whose block is stored locally.

In this paper, we study how to schedule independent tasks whose input data are replicated among a distributed platform. We concentrate on two performance criteria: the overall processing time, or makespan, and the amount of communication among processors, i.e. of data movement. We target simple, dynamic schedulers, such as the one of MapReduce. 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. One objective of the present paper is thus to assess the performance of such dynamic schedulers to provide theoretical guarantees on their performance. Our contributions are the following:

- We propose close-form formulas to compute the amount of communications of a greedy scheduler for tasks with same duration, with or without initial data replication;
- We analyse the problem of allocating only local tasks, that is, preventing any communication at runtime and prove that it relates to the study of balls-into-bins processes with "power of k choices";
- We propose simple locality-aware heuristics for tasks with heterogeneous durations;
- We perform numerous simulations both to validate our theoretical findings and to show that our simple heuristics behave well compared to more complex solutions from the literature.

The rest of the paper is organized as follows. Section II states the problem and notations, Section III reviews the related work. We analyse dynamic schedulers for tasks with homogeneous duration in Section IV and propose scheduling heuristics for heterogeneous tasks in Section V. Simulations are reported in Section VI to experimentally verify the theoretical results obtained in the previous section and to assess the performance of scheduling heuristics. Finally, Section VII concludes the paper.

II. Problem modeling

We consider the problem of processing n independent tasks, named $T_1 \ldots, T_n$ on a distributed computing platform made of m nodes (or processors), denoted by $P_1, \ldots, P_m$. Each task depends on a single input data, or data chunk, which is not used by other tasks. These n chunks are originally distributed on the nodes, with replication factor r: there are r copies of each input data in the platform. We assume that no two

copies of the same data are present on the same node, and that replicas are uniformly distributed at random. We consider that all nodes of the system have the same processing capability, and that each task T_i has a duration t_i on any processor. In the following, we will consider both homogeneous task durations ($t_i = t$ for all tasks) and heterogeneous task durations. In the latter case, the scheduler is said to be *clairvoyant* if it knows each task duration before the execution, and *non-clairvoyant* otherwise. In both cases, we consider that all chunks have the same data size S.

Our objective is to process all tasks on the available processors and to take advantage of data locality to avoid moving to much data around, by carefully choosing the set of tasks processed by each processor (task allocation). The problem we address is thus bi-criteria, as there is a natural trade-off between time and data movement. We formally define the first objective as the makespan, i.e., the total completion time of all processors. The completion time of a processor P_i is the sum of the durations of the tasks assigned to P_i. The second objective is the amount of data movements or communications, i.e., the total size of data sent among nodes at runtime. Since we consider homogeneous data sizes, this amount of communications is directly proportional to the number of non local tasks, that is, the number of tasks that are not processed on one of the processors holding its input data:

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

In the case of homogeneous tasks, it is easy to achieve optimal makespan: never leave a processor idle by assigning non local tasks if needed, as done in traditional list scheduling. Conversely it is easy to perform only local tasks: keep a processor idle if it does not own unprocessed chunks anymore. Thus, both metrics will be considered, always under the assumption that the other one is set to its optimal value. More precisely, when optimizing the makespan metric, we assume that only local tasks are performed. When optimizing the communication metric, we would like the makespan to be optimal. In the case of homogeneous task duration, the optimal makespan is easily computed as $\lceil n/m \rceil$. In the case of heterogeneous durations, it is NP-complete to minimize the makespan (even on two processors by reduction to 2-Partition [1]). Therefore, we lower our constraint on the makespan, and we only forbid that a processor is kept idle when it could start an unprocessed task. This restricts our search of communication efficient algorithms to List Schedules, i.e., to schedules that never leave a resource idle while there are tasks to process.

Finally, we assume the there is a centralised coordinator, or scheduler, which takes all allocation decisions. Note that we are interested in algorithms with low runtime complexity, such as greedy schedulers, so that they can be incorporated in large scale schedulers. Thus, we do not usually look for optimal strategies (except to design a reference lower bound, as in Section V-B).

III. Related Work on Independent Tasks Scheduling

A. Data Locality in Map-Reduce

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 [2]. By default, HDFS replicates randomly data chunks several times onto the nodes (usually 3 times). 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. [3], Xie and Lu [4] and Isard et al. [5] strive to increase the data locality by proposing better schedulers than the default one.

Ibrahim et al. [6] 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 to assign local tasks. They experimentally show that this reduces the number of non-local map tasks. A more precise description of *Maestro* is given in Section V.

B. 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 [7]. More specifically, we prove in Section IV-B that it is possible to simulate the greedy algorithm of the makespan problem 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 have to be processed locally, then the maximum load of a bin is equal to the makespan achieved by 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. [8]. 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 [7] 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 choice algorithms, where r random candidate bins are pre-selected for each ball, and then the ball is assigned to the candidate bin whose load is minimum. It is well known that having more than one choice strongly improves load balancing. We refer the interested reader to Mitzenmacher [9] and Richa et al. [10] for surveys that illustrate the power of two choices. Typically, combining previous results with those of Berenbrink et al. [11], 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. Peres et al. [12] study cases with a non-integer r. In this case, with a given parameter β, for each ball, with probability β the assignation 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 $O(\log \log m)$ regular balls-into-bins gap. The combination of multiple choice games with weighted balls has also been considered by Peres et al. [12]. 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.

C. Scheduling Tasks with Replicated Inputs

Several papers have studied a related scheduling problem, where data locality is important: where to allocate tasks which dependents on several input data initially distributed on various repositories [13], [14], [15]. However, contrarily to our model, these studies assume that (i) tasks depend on several input data and (ii) some input data may be shared among tasks. This makes the problem much more combinatorial. These papers propose heuristics for this problem and test them through simulations.

IV. Analysis of Greedy Strategies on Homogeneous Tasks

A. Lower Bound for Communication metric

We start with the communication metric, which seems the most natural one: given a set of tasks and their replicated input chunks, allocate the tasks to the processors so as to minimize the data movement, while ensuring a perfect load-balancing: no processor should be kept idle when there are still some unprocessed tasks.

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

Our analysis below assume that the average number of tasks per processor is small in comparison to both the total number of tasks ($n/m = o(n)$) and the number of processors ($n/m = o(m)$). We start by estimating the amount of communication of such a scheduler without replication.

Theorem 1. *The amount of communication of the greedy scheduler without initial data replication is lower bounded by $\sqrt{nm/2\pi}$ (provided that m divides n).*

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 when it has finished his own local work.

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

We consider the number of chunks that are sent to each processor during the distribution. We denote by N_k the number of processors that received exactly k chunks. In the end, as all tasks have the same duration, each processor will process exactly n/m tasks. Thus, in our variant, a processor with k tasks will request $n/m - k$ tasks if $k < n/m$ and none otherwise. This allows to express the total amount of communications:

$$V = \sum_{0 \leq k < n/m} (n/m - k) N_k$$

We now compute the probability that a processor gets exactly k chunks during the data distribution. This is is similar to the probability that a bin receive k balls in a balls-into-bins distribution process [16] and is given by: $Pr(k) = \binom{n}{k}(1/m)^k(1 - 1/m)^{n-k}$. When $k \ll n, m$ and $k > 0$, we approximate this probability by $Pr(k) = e^{-n/m}(n/m)^k/k!$. Then, N_k is simply $m \times Pr(k)$. This allows to compute V:

$$\begin{aligned} V &= \frac{n}{m} N_0 + \sum_{1 \leq k < n/m} (n/m - k) m e^{-n/m} \frac{(n/m)^k}{k!} \\ &= \frac{n}{m} N_0 + m e^{-n/m} \sum_{1 \leq k < n/m} \frac{(n/m)^{k+1}}{k!} - \frac{(n/m)^k}{(k-1)!} \\ &= \frac{n}{m} m e^{-n/m} + m e^{-n/m} \left(\frac{(n/m)^{n/m}}{(n/m - 1)!} - n/m \right) \end{aligned}$$

$$\geq me^{-n/m}\frac{(n/m)^{n/m+1}}{\sqrt{2\pi n/m}}(me/n)^{n/m}$$
$$\geq \sqrt{nm/2\pi}.$$

□

Figure 1 illustrates this lower bound, by showing the comparison between this above formula and a simulation of GREEDY-COMM with $m = 50$ processors and different values of n. The results show that this lower bound accurately describes the behaviour of GREEDY-COMM strategy, probing the reliability of this probabilistic approach.

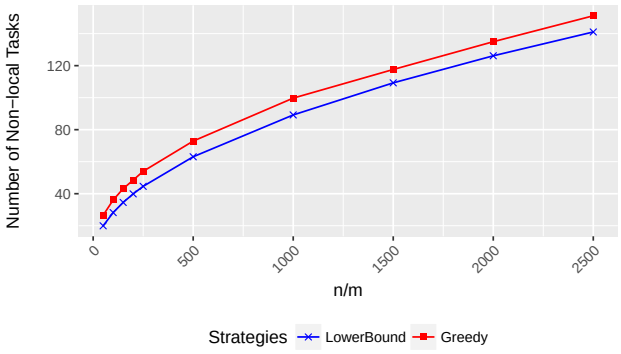

Figure 1. Number of Non-Local Tasks – homogeneous case, no replication

It is possible to apply the same method when considering replication. Unfortunately, this results in a more complex formula.

Theorem 2. *The amount of communication of the greedy scheduler with initial data replication is lower bounded (provided that m divides n) by:*

$$me^{-rn/m}\sum_{0\leq k<n/m}(n/m-k)\frac{(rn/m)^k}{k!}.$$

Proof. We apply the same reasoning as in the previous proof. We first consider the random distribution of $r\times n$ distinct data chunks without replication. This leads to a probability for a node to receive k chunks given by $Pr'(k)=\binom{rn}{k}(1/m)^k(1-1/m)^{rn-k}$. The distribution of n chunks with replication r leads to a probability which is not larger: $Pr(k)\leq Pr'(k)$. Then, it is possible to upper bound N_k, the number of processors with exactly k chunks:

$$N_k = m\times Pr(k)\leq m\times Pr'(k).$$

As previously, when $k\ll n,m$ and $k>0$, we approximate this probability by $Pr'(k)=e^{-rn/m}(rn/m)^k/k!$. We use the same way of computing the total volume of communication, which results in

$$V\leq me^{-rn/m}\sum_{0\leq k<n/m}(n/m-k)\frac{(rn/m)^k}{k!}.$$

□

B. Link between the Makespan Problem and Balls-into-Bins

We consider here the dual problem, that is, we aim at minimizing the makespan when only local tasks can be processed. We consider the following basic greedy strategy, called GREEDY-MAKESPAN: when a processor finishes some task, if it still holds the input chunk of some unprocessed task, it randomly chooses such a task and process it. Otherwise, it stops its execution.

For the analysis of this algorithm, we move to the case of heterogeneous task durations, when task T_i has a duration of t_i. We also consider that processors have slightly different initial availability times: ϵ_j is the availability time of processor j. However, the GREEDY-MAKESPAN scheduler has no knowledge of the task durations before their execution. We assume, as in [8] or [12], 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 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 scheduler behavior.

We now prove that in presence of replication, the expected makespan of the GREEDY-MAKESPAN scheduler is closely related to the balls-into-bins problem with r multiple choices, see Mitzenmacher [9]. The process of distributing n balls $B_1,\ldots,B_n$ of sizes $t_1,\ldots,t_n$ into m 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 shows the relation between the simple dynamic scheduler and the balls-into-bins process.

Theorem 3. *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,*

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

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

Lemma 4. *Let $j_b(i)$ denote the index of the bin where ball b_i is placed and let $j_p(i)$ denote 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 $k\in C_1$. Conversely, T_1 is replicated onto all the processors P_k, where $k\in C_1$. 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 indexes $1,\ldots,i-1$, and let us consider the set of bins B_k and the set of processors P_k such that $k\in C_i$. By construction, at this instant, processors $P_k,\ k\in C_i$ have only processed tasks whose index is smaller than i. Let us denote by $S_i=\{T_{i_1},\ldots,T_{i_{n_i}}\}$

this set of tasks, whose indexes i_k's are smaller than i. These tasks have been processed on the processors whose indexes 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 $P_k,\ k \in C_i$, 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, which achieves the proof of the theorem. □

Thanks to this result, we can apply known bounds on the maximum load for balls-into-bins processes derived in the literature, as related in the previous section. In particular, going back to the case of tasks with identical processing times, the expected makespan when $r \geq 2$ is known to be of order $n/m + O(\log \log m)$ (with high probability) [11].

In addition to this result for the makespan evaluation of GREEDY-MAKESPAN, Berenbrink et al. [8] introduce the number of "holes" in a balls-into-bins process, *i.e.* the number of balls that are lacking in the least loaded bins to reach a perfect balancing, and proves it can be bounded by a linear function of m. In our case, with the use of GREEDY-COMM, the holes can be interpreted as the number of time a processor will be idle before the end of the execution and will therefore steal a task. We investigate in Section VI this $O(m)$ asymptotic behaviour for communication cost of GREEDY-COMM.

V. HEURISTICS FOR HETEROGENEOUS TASK DURATIONS

We recall the general setting: there are n tasks to process $T_1, \ldots, T_n$ onto a set of m processors $P_1, \ldots, P_m$, as described in Section II. Each task T_i depends on its associated data chunk. We assume that each data chunk is replicated r times by the distributed file system, and for the sake of simplicity, we will denote by $R_i^{(1)}, \ldots R_i^{(r)}$ the indices of the processors that store the data chunk associated to task T_i.

In what follows, in order to assess the efficiency of the different strategies, we consider a more general setting that the one considered in Section IV, where all tasks are assumed to have the duration. More specifically, we assume that the duration of task T_i on any resource (we assume that resources are homogeneous at this point) is given by t_i. We will consider both clairvoyant heuristics (to which t_i is known) and non-clairvoyant heuristics (to which t_i is unknown).

A. Non Clairvoyant Heuristics for Makespan Minimization without Communications

- **Greedy**: When a resource P_j becomes idle, it chooses a task at random between the unprocessed tasks for which it owns the input file and then process it locally. If there is no such local unprocessed task, its stays idle . This corresponds, in the case of homogeneous tasks, to GREEDY-MAKESPAN for which we propose a theoretical study of Section IV (from now we denote both GREEDY-MAKESPAN and GREEDY-COMM as GREEDY, the use of one or another depending on the context).
- **Maestro** : This corresponds to the strategy proposed by Ibrahim et al. in [6]. In order to explain the functioning of MAESTRO, we introduce the parameters defined by the authors
  - For a processor P_i, HCN_i denotes its host chunk number, *i.e.* the number of data chunks from unprocessed tasks present on the processor.
  - For a task T_j, its chunk weight is $Cw_j = 1 - \left(\sum_{i=i_1}^{i=r} \frac{1}{HCN_i}\right)$ where $P_{i_1}, \ldots, P_{i_r}$ are the processors that host the duplicates of the data chunk of T_j. This value represents the probability for T_j to be non-locally executed. If a chunk is on a processor with many other chunks, then it is more likely to be executed by another processor (because the first processor may not finish its other tasks on time).
  - For two processors $P_i, P_{i'}$, $Sc_i^{i'}$ denotes the number of shared chunks between P_i and $P_{i'}$. Using the above notations, $Sc_i^{i'}$ is the number of tasks j for which there exists k and k' with $i = R_j^{(k)}$ and $i' = R_j^{(k')}$.
  - For a processor P_i, its node weight is $NodeW_i = \sum_{j=1}^{HCN_i} \left(1 - \sum_{k=1}^{r} \frac{1}{HCN_{i_{j,k}} + Sc_{i_{j,k}}^{i}}\right)$ where $(P_{j,1}, \ldots, P_{j,r})$ are the processors that host data chunk of T_j. Quickly, the higher is $NodeW_i$, the less P_i has a chance to process non-local task.

  During the first phase, MAESTRO assigns m tasks, one per processor. To do this, it first chooses the processor P_i with the lowest $NodeW_i$ (a processor with few hosted chunks or/and many shared chunks with other critical processors) and allocates to P_i the task whose chunk is present on P_i with the largest chunk weight (*i.e.* the task with the most chance to be non-locally executed). More precisely MAESTRO gives to the processor with highest chance to process non-local tasks the task it hosts with the highest chance to be processed non-locally.
  During its second phase, that is purely dynamic, each time a processor is idle, the local task with the highest chunk weight is assigned to this processor and processed. If there is no local task, an arbitrary unprocessed task is processed after the loading of the data chunk (if non-local tasks are allowed, otherwise the processor stays idle). For a precise pseudo-code of this two phases see [17].
- **Locality Aware Greedy (2 variants)**: These heuristics are based on the same idea as **Maestro** [6], at a lower computational cost. As for **Greedy**, decisions are taken when a resource P_j becomes idle. Among the unprocessed tasks for which P_j owns associated data chunks, a greedy strategy is used 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 completion time is expected to be highest, which corresponds to the task for which candidate processors still own a large number of unprocessed tasks. To estimate the relative expected the completion time of the tasks depending on the state of the resources owning its data chunk, we in turn rely on two possible heuristics, LocAwareGreedyMin and LocAwareGreedyAvg.

To implement these heuristics, each resource P_j maintains the number $n_j(t)$ of still unprocessed tasks at time t for which it owns the input data files. 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 $f(T_i) = \min\left(n_{i^{(1)}}, \ldots, n_{i^{(r)}}\right)$ LocAwareGreedyMin, $f(T_i) = \text{mean}\left(n_{i^{(1)}}, \ldots, n_{i^{(r)}}\right)$ LocAwareGreedyAvg,

B. Clairvoyant Heuristics for Makespan Minimization without Communications

- **Earliest Time First** We 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 i and the load of P_j is minimal with already allocated tasks T_1 to T_{i-1}.
- **Locality Aware, Clairvoyant Greedy (2 variants)**: Each resource P_j maintains the duration $\text{Rem}_j(t)$ of still unprocessed at time t tasks for which it owns the input data files and its ready time RT_j, given already allocated tasks.

  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

$$f(T_i) = \min_{1 \leq l \leq r} \left(RT_{i^{(l)}} + \text{Rem}_{i^{(l)}}\right)$$

LocAwareClairGreedyMin,

$$f(T_i) = \text{mean}_{1 \leq l \leq r} \left(RT_{i^{(l)}} + \text{Rem}_{i^{(l)}}\right)$$

LocAwareClairGreedyAvg.

C. List Based Heuristics for Communication Minimization

Previous heuristics can be adapted in the context of Communication Minimization (see Section II). We recall that in that case, we restrict our search of communication efficient algorithms to List Schedules, that is, to schedules that never deliberately leave a processor idle where there remain tasks to be processed.

In all the heuristics proposed in Section V-A and Section V-B, decisions are made when a resource becomes idle. Then, the idle resource choose a task among the unprocessed tasks for which a local chunk is known when such a local chunk exist, and remains idle otherwise. In the case of communication minimization, when no local unprocessed task exists, then the resource chooses a task among all unprocessed tasks, using the same heuristic.

We use this general scheme to extend all proposed heuristics (Greedy, Maestro, or Data Aware Greedy) to the case of communication minimization. Due to lack of space, we do not formally present all these heuristics, but we compare them using simulations in Section VI.

VI. Simulations

Because of the lack of space, we focus in this section on the communication metric. We propose two case: homogeneous and heterogeneous case.

A. Homogeneous tasks

Let us first focus on homogeneous tasks. The number m of processors is set to 50 and we study the influence of other parameters, *i.e.* n (number of tasks) and r (replication). For each pair (n, r) and each heuristic, 250 runs are performed, each time randomly drawing the initial data chunk distribution.

a) Impact of replication on non-local communications: In the case without replication, we proved on Figure 1 that Theorem 1 provides an accurate lower bound for the number of non local tasks and that **Greedy** strategy is close to optimal. Let us now consider the case when there are r initial replicas of each data chunk. The fraction of non-local tasks for the different heuristics presented in Section V is depicted in Figure 2.

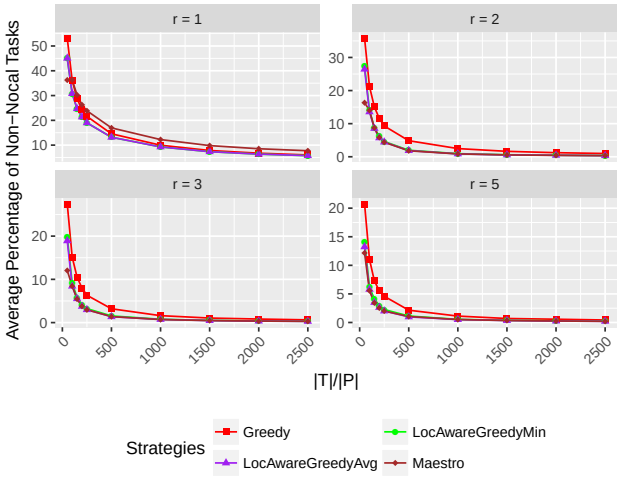

Figure 2. Fraction of Non-Local Tasks – homogeneous case with different levels of replication

The first notable information in Figure 2 is the gain from $r = 1$ to $r = 2$, that is coherent with the balls-into-bins model results where the overhead significantly decrease with the possibility to choose bins (if the overhead decreases, so does the makespan and thus the number of processor with many stolen tasks). Next, increasing the number of replicas from $r = 2$ to larger values has a modest impact on the efficiency of the different heuristics. Therefore, the choice of $r = 3$ in HDFS appears to be a good trade-off between the storage cost of replication and the availability of data.

Second, we can observe that **Greedy** is sub-optimal. In particular, all the other heuristics, that take into account the state of the resources on which local tasks are replicated achieve better performance. In the context of homogeneous tasks, we consider the clairvoyant versions with uniform weights, since all tasks durations are known in advance and are identical. In all cases (except for $r = 1$ for unknown reasons), **Maestro** achieves the best results, in particular for $n/m = 1$ where its sophisticated first wave improves the efficiency. Nevertheless, its computational time (during this same first wave) is much higher than those of our data aware greedy heuristics, and even when $r = 2$ and n/m is small (but larger than 1), the performance of **Maestro** and LocAwareGreedyAvg are undistinguishable, what justifies their interest.

b) Asymptotic stability of the number of non-local tasks per processor: On Figure 3 we propose a different representation of the same set of experiments where the average number of non-local tasks per processor is depicted. In all these experiments, m is constant and increasing the value of n/m corresponds to increasing the number of tasks n. From the literature [8]) (see Section IV-B), the number of "holes" in a balls-into-bins process, *i.e.* the number of lacking balls in the least loaded bins to reach a perfect balancing has been proved to be of order $O(m)$. Figure 3 shows that after a transitional phase for small n/m, all heuristics then have a stable number of non-local tasks per processors, with almost no influence from the number of tasks, but we can also observe that the constant also depends on r.

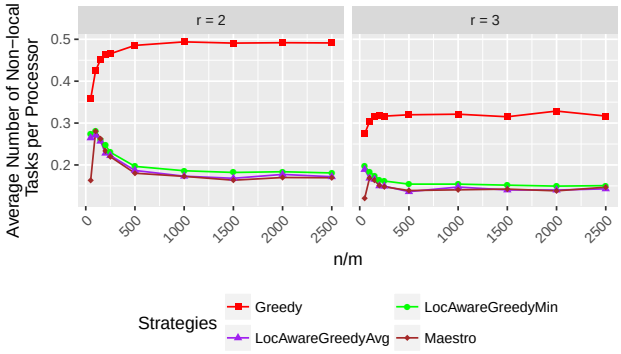

Figure 3. Average Number of Non-Local Tasks per Processor – homogeneous case with different levels of replication

B. Heterogeneous tasks

It is well known that in practice, even Map tasks of a given jobs do not have the exact same duration. In order to build realistic instances for the heterogeneous settings, we rely on actual traces from [18]. These traces come from the jobs running during 10 months on a Hadoop cluster. In order to emulate the on-line setting, the computational time of each task is randomly picked up at the beginning of its simulated execution. In the following, we focus on jobs with at most 5000 tasks (what corresponds nearly to 89% of the jobs). In addition, we classify the jobs depending on their Normalized Standard Deviation (NSD), *i.e.* the standard deviation of their computational times divided by the mean value of the computational times.

We depict the fraction of non local tasks in Figure 4 ($n = p$), Figure 5 ($n = 2p$) and Figure 6 ($n = 50p$), for different Normalized Standard Deviation. In all this figures, we use boxplots the represent the results of the different experiments. We recall that a boxplot should be read as follow. The box represent the values of the second and third quarter with the horizontal bar set at the median. The vertical bar above and below the box goes from the second to the ninth decile.

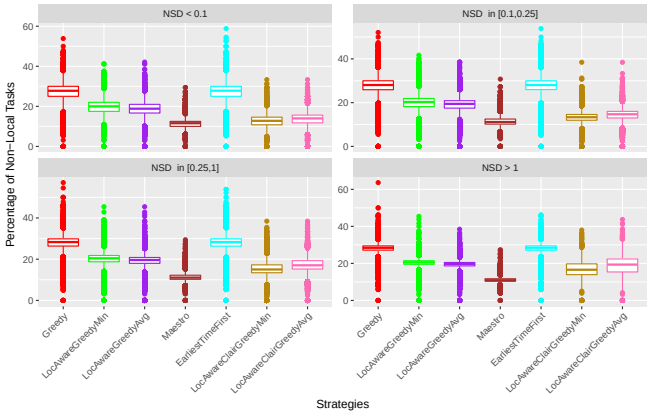

Figure 4. Fraction of Non-Local Tasks – heterogeneous case with $r = 3$ and $n/m = 1$

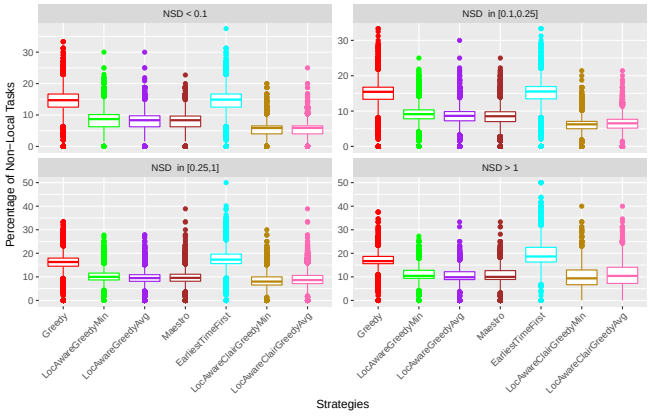

Figure 5. Fraction of Non-Local Tasks – heterogeneous case with $r = 3$ and $n/m = 2$

In all cases, we can observe that LocAwareGreedyMin and LocAwareGreedyAvg overperform the basic **Greedy** strategy, and that if tasks duration are known in advance (clairvoyant case), LocAwareClairGreedyMin and LocAwareClairGreedyAvg overperform the **Earliest Time First** strategy. Another interesting property is that the impact of heterogeneity on the number of non local tasks is not strong and only slightly depends on the value of the NSD. This is certainly related to the greedy behavior of the different strategies, when allocation decisions are not performed statically offfline

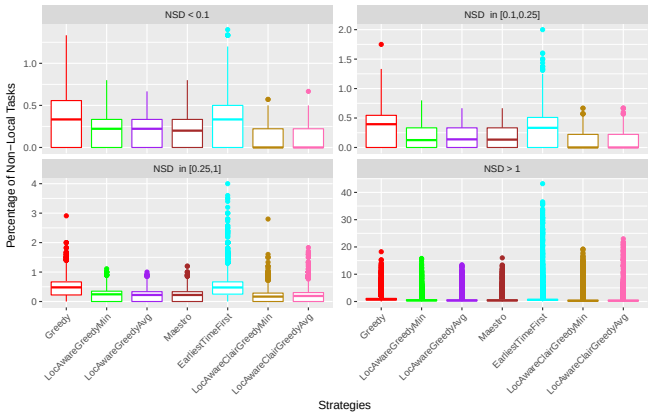

Figure 6. Fraction of Non-Local Tasks – heterogeneous case with $r = 3$ and $n/m = 50$

but at runtime, based on the state of the nodes. A node that receives longer tasks will very likely perform less tasks and local tasks only, whereas a node that receives shorter 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 IV-B for the makespan and [8] for the fraction of non-local tasks) is already difficult and becomes much harder in the heterogeneous setting.

When $n = p$, *i.e.* when there is a single round of tasks distribution, we can observe that the best strategy is **Maestro**, what is not surprising since it benefits from the sophisticated but high cost first wave (and only in this case). The performance of clairvoyant strategies are close to the one of **Maestro** whereas **Maestro** is a non clairvoyant strategy, which proves that when the number of tasks per processor is low, and with respect to the number of non local tasks, choosing an allocation that balances the number of local tasks is more important than an allocation that balances the workload. When $n = 2p$ and even more when $n = 50p$, we can observe that clairvoyant heuristics perform better, which is not surprising since a larger number of tasks provides more opportunities for load balancing. We can also observe that among non-clairvoyant heuristics, there is no difference between cheap heuristics LocAwareGreedyMin and LocAwareGreedyAvg and more expensive **Maestro** strategy as soon as $n \geq 2p$.

VII. Conclusion

In this paper, we provide formulas to estimate the makespan and the number of communications induced by a natural greedy demand-driven heuristic for allocating independent tasks over a distributed file system that uses replication. Then, we propose several more sophisticated but cheap heuristics, whose complexity for allocating a new task is logarithmic in the number of unprocessed tasks. Then, using simulations based on an actual MapReduce production trace, we both prove that the bounds proved in theoretical formulas are almost tight and that proposed heuristics behave very well in practice against state of the art allocation strategies. This paper opens several perspectives on the theoretical part, since the very good results achieved by greedy strategies in the heterogeneous case are still not well explained by theory.

References

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

[2] D. Borthakur, "HDFS Architecture Guide," *HADOOP http://hadoop. apache. org/common/docs/current/hdfs design. pdf*, p. 39, 2008.

[3] M. Zaharia, D. Borthakur, J. Sen Sarma, K. Elmeleegy, S. Shenker, and 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.

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

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

[6] S. Ibrahim, H. Jin, L. Lu, B. He, G. Antoniu, and 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.

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

[8] P. Berenbrink, T. Friedetzky, Z. Hu, and R. Martin, "On weighted Balls-into-Bins Games," *Theoretical Computer Science (TCS)*, vol. 409, no. 3, pp. 511–520, 2008.

[9] M. Mitzenmacher, "The Power of Two Choices in Randomized Load Balancing," *Transactions on Parallel and Distributed Systems (TPDS)*, vol. 12, no. 10, pp. 1094–1104, 2001.

[10] A. W. Richa, M. Mitzenmacher, and R. Sitaraman, "The Power of Two Random Choices: A Survey of Techniques and Results," *Combinatorial Optimization*, vol. 9, pp. 255–304, 2001.

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

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

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

[14] K. Kaya, B. Uçar, and C. Aykanat, "Heuristics for scheduling file-sharing tasks on heterogeneous systems with distributed repositories," *J. Parallel Distrib. Comput.*, vol. 67, no. 3, pp. 271–285, 2007. [Online]. Available: https://doi.org/10.1016/j.jpdc.2006.11.004

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

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

[17] T. Lambert, "On the Effect of Replication of Input Files on the Efficiency and the Robustness of a Set of Computations," Ph.D. dissertation, Université de Bordeaux, Talence, 2017.

[18] S. Kavulya, J. Tan, R. Gandhi, and 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.