

Matching-Based Allocation Strategies for Improving Data Locality of Map Tasks in MapReduce

Olivier Beaumont, Thomas Lambert, Loris Marchal, Bastien Thomas

▶ **To cite this version:**

Olivier Beaumont, Thomas Lambert, Loris Marchal, Bastien Thomas. Matching-Based Allocation Strategies for Improving Data Locality of Map Tasks in MapReduce. [Research Report] RR-8968, Inria - Research Centre Grenoble – Rhône-Alpes; Inria Bordeaux Sud-Ouest. 2016. hal-01386539v2

HAL Id: hal-01386539

https://inria.hal.science/hal-01386539v2

Submitted on 2 Nov 2016 (v2), last revised 24 Oct 2017 (v5)

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

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



Matching-Based Allocation Strategies for Improving Data Locality of Map Tasks in MapReduce

Olivier Beaumont, Thomas Lambert, Loris Marchal, Bastien Thomas

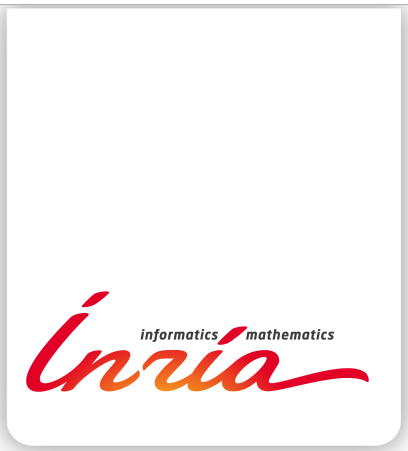
RESEARCH REPORT

N° 8968

November 2016

Project-Teams ROMA and RealOpt

ISSN 0249-6399 ISRN INRIA/RR--8968--FR+ENG



Matching-Based Allocation Strategies for Improving Data Locality of Map Tasks in MapReduce

Olivier Beaumont*, Thomas Lambert*, Loris Marchal†, Bastien Thomas‡

Project-Teams ROMA and RealOpt

Research Report n° 8968 — November 2016 — 34 pages

Abstract: MapReduce is a well-know framework for distributing data-processing computations on parallel clusters. In MapReduce, a large computation is broken into small tasks that run in parallel on multiple machines, and scales easily to very large clusters of inexpensive commodity computers. Before the Map phase, the original dataset is first split into chunks, that are replicated (a constant number of times, usually 3) and distributed onto the computing nodes. During the Map phase, nodes request tasks and are allocated first tasks associated to local chunks (if any). Communications take place when requesting nodes do not hold any local chunk anymore. In this paper, we provide the first complete theoretical data locality analysis of the Map phase of MapReduce, and more generally, for bag-of-tasks applications that behaves like MapReduce. We show that if tasks are homogeneous (in term of processing time), once the chunks have been replicated randomly on resources with a replication factor larger than 2, it is possible to find a priority mechanism for tasks that achieves a quasi-perfect number of communications using a sophisticated matching algorithm. In the more realistic case of heterogeneous processing times, we prove using an actual trace of a MapReduce server that this priority mechanism enables to complete the Map phase with significantly fewer communications, even on realistic distributions of task durations.

Key-words: MapReduce, Analysis of Randomized Algorithms, Matchings, Resource Allocation and Scheduling, Balls-into-bins.

* Inria & University of Bordeaux, France
† CNRS, LIP, École Normale Supérieure de Lyon, INRIA, France
‡ ENS of Rennes, France

RESEARCH CENTRE
GRENOBLE – RHÔNE-ALPES

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

Stratégies d'allocations à base de couplages pour améliorer la localité des tâches Map dans MapReduce

Résumé : MapReduce est un modèle de programmation très connu pour les applications distribuées de traitement de données sur grappes de calcul. Dans ce modèle, le calcul est découpé en petites tâches qui sont lancées en parallèles sur de nombreux processeurs. Il permet d'utiliser facilement de très grandes grappes de calcul faites de processeurs standards. Avant la première phase de Map, les données sont d'abord découpées en gros fragments, qui sont répliqués et distribués sur la plate-forme. Pendant la phase de Map, les processeurs demandent du travail et se voient alloués en priorité des tâches associées aux fragments locaux (s'il y en a). Lorsque ce n'est pas (ou plus) le cas, des communications ont lieu pour transmettre des fragments de données. Dans ce rapport, nous proposont une étude théorique de la localité des données dans la phase de Map d'une application MapReduce, et plus généralement pour toutes applications de type "sac de tâches" qui se comporte de façon similaire. Nous montrons que si les tâches ont des temps de traitement homogènes, une fois que les fragments ont été répliqués et distribués aléatoirement aux processeurs avec un facteur de réplication plus grand que 2, il est possible de trouver un méchanisme de priorité pour les tâches qui réalise un nombre quasi-parfait de communications en utilisant un algorithme de couplage sophistiqué. Dans le cas plus réalistes de temps de traitement hétérogènes, nous montrons qu'avec des données issues de traces d'exécution de MapReduce, ce méchanisme permet de réaliser la phase de Map avec très peu de communications.

Mots-clés : MapReduce, Analyse d'algorithmes randomisés, couplages, allocation de ressource et ordonnancement

1 Introduction

MapReduce is a well-known framework that has been introduced by Google and has been popularized by implementations like Hadoop [1] for distributing data-processing computations on parallel clusters. In MapReduce, a large computation is broken into small tasks that run in parallel on multiple machines, and scales easily to very large clusters of inexpensive commodity computers. MapReduce is a very successful example of a dynamic scheduler, as one of its crucial feature is its inherent capability of handling hardware failures and processing capability heterogeneity, thus hiding this complexity to the programmer, by relying on on-demand allocations and the online detection of nodes that perform poorly (in order to re-assign tasks that slow down the process).

In a classical MapReduce application, the original dataset is first split into chunks and distributed on the computing nodes. Then, computation is decomposed into two phases: a Map phase followed by a Reduce phase, each of them being composed of several tasks. Map tasks emit <key,value> pairs which are sorted by key and all pairs with the same keys are aggregated by the Reduce tasks. In the textbook example of the WORDCOUNT application, whose purpose is to count the number of occurrences of each word in a text, the Map tasks consist in emitting a key-value pair $<w,1>$ for each word w in a text chunk. Then, the Reduce tasks are given as input all pairs corresponding to the same word and compute its total number of occurrences in the text. However, not all MapReduce applications use this classical framework. In particular, MapReduce is also widely used to distribute independent task applications, which are then only composed of Map tasks. For these applications, data locality is the main source of communications. There have been relatively few theoretical studies of data locality in MapReduce and its impact on communications [2].

In MapReduce, minimizing the runtime amount of communications is a crucial issue. The initial distribution of the chunks onto the platform is performed by a distributed filesystem such as HDFS [3]. By default, HDFS replicates randomly data chunks several (usually 3) times onto the nodes. This replication has two main advantages. First, it improves the reliability of the process, limiting the risk of losing input data. Second, replication tend to minimize the number of communications. 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 the job priority and history. Then, it checks whether the job has a local unprocessed data chunk on the processor. If yes, such a local task is allocated, and otherwise, a non-local task is allocated and the associated data chunk is sent from a distant node. Therefore, intuitively, having more replicas gives more opportunities for a given chunk of being processed locally (without inducing a communication). In what follows, we will assess precisely the impact of replication on the number of non local tasks. For instance, Section 4.1 shows that replication is closely related to the power of 2 choices in balls into bins games, that is known to improve the performance of classical balls into bins.

This paper is devoted to a precise analysis of the influence of replication on

data locality for map tasks, and to the rigorous analysis of another dynamic scheduling algorithm introduced in Section 4 that computes, after the initial randomized allocation performed by the distributed file system, an optimal allocation under the assumption that the processing times of all tasks have the same duration. Indeed, depending on the size of the job [4] shows that the number of non-local tasks can be around 12-17% , and their processing takes is 1.2 to 2 times longer because of data fetching time. The quality of the data locality of a scheduling policy, given a replication mechanism, is therefore a crucial issue and can be evaluated through several metrics. In this paper, we will either consider the number of communications, *i.e.* the number of map tasks that are not processed locally on a node that holds the corresponding input data chunk or the makespan without communications, *i.e.* the necessary time to process all tasks if communications were forbidden and all tasks had to be processed locally. These two metrics enable to well estimate the quality of a scheduling algorithm, although they are not strictly equivalent.

The rest of the paper is organized as follows. We discuss related work on matching algorithms, randomized balls into bins algorithms and MapReduce scheduling algorithms in Section 2 and the modeling of MapReduce in Section 3. The new scheduling algorithm based on matchings that we propose is presented in Section 4 and its performance analysis is provided in Section 5. Finally, simulation results assessing the actual efficiency of our algorithm on actual data traces are presented in Section 6 and concluding remarks in Section 7.

2 Related Work

In this section, we review the literature in three research areas that are close to this study: data locality for MapReduce (in particular during the Map phase), matchings (in particular for bipartite graphs), and balls-into-bins games.

2.1 Locality in MapReduce

A number of paper have studied the problem of data locality in MapReduce, and in particular during the Map phase.

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

Ibrahim et al. [4] also outlines 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 allocation scheme that intends to maximize the number of local tasks. To this end, *Maestro* estimates which processors will be allocated the smallest number of local tasks, given the distribution of the repli-

cas. These nodes are selected first to allocate local tasks. They experimentally show that this reduces the number of non-local map tasks.

Xie et al. [6] propose a simple allocation 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 allocated a local task. They propose a "peeling" algorithm which first serves the processor with a single unprocessed replica (and thus allocates to them their single local task), and then switches to a classical random allocation for other processors. Using evolution of random graphs, they show that the peeling algorithm leads to better data locality, and achieves close to optimal allocation for small to medium load.

Wang et al. [7] consider the problem of map task locality from a stochastic network perspective and propose a new queuing algorithm to simultaneously maximize throughput and minimize delay in heavy loaded conditions.

Guo et al. [8] consider the locality of map tasks. They propose an algorithm based on the Linear Sum Assignement Problem to compute an allocation with minimal communications. Unfortunately, in the case where there are more tasks than processors, there 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 reallocation.

Finally, some papers [9, 10] have focused on optimizing data locality in the Reduce phase, by placing reduce tasks close to their input data computed by the map tasks.

2.2 Matchings in Bipartite Graphs

Many studies and algorithms have been proposed for matching problems, in particular for bipartite graph, a particular case with many applications, for example in assignment problems [11].

A first research direction deals with the existence of matchings, in particular perfect matchings, *i.e.* matchings whose size is the number of vertices. For instance, the work of Erdös and Rényi on random bipartite graph [12] proves that there exists a perfect matching of a bipartite graph of $2n$ vertices with asymptotic probability $e^{-2e^{-c}}$ when the number of edges is $n \ln n + cn + o(n)$. Walkup [13], instead of an assumption on the number of edges, uses a condition on the minimum degree of the vertices. Asymptotically, a regular bipartite graph (*i.e.* whose both sets of vertices are of equal size) has a perfect matching if its minimum degree is at least 2.

A second research direction deals with the efficient computation of matchings. Many algorithms rely on augmenting paths introduced in [14]. An augmenting path is a path that that induces an improvement of the current existing flow (or matching) by permuting some edges of the path with some of the actual solution. For bipartite graphs, very efficient algorithms exist to find an optimal matching, such as the Hopcroft-Karp Algorithm [15] with a complexity of $O(\sqrt{n}m)$, where n denotes the number of vertices and m the number of

edges, or the one proposed by Goel et al. [16] for regular bipartite graph, whose expected complexity is $O(n \log n)$. There also exists approximation algorithms with better computations time that no longer guarantee the computation of a perfect matching [17, 18].

2.3 Variants of Balls-into-Bins

An extreme solution for improving data locality is to forbid all communications, and allow processors to compute only the chunks they own locally. This might create some load imbalance, since some of the processors may stop their computations earlier.

Such a process is closely related to balls into bins problems [19]. More specifically, we will prove in Section 4.1 that it is possible to simulate the greedy algorithm for allocating Map tasks with a variant of a Balls into Bins game. In these randomized processes, n balls are placed randomly into p bins and the expected load of the most loaded bin is considered. If we consider a process where data chunks are not replicated and correspond to balls, processing resources correspond to bins, and tasks have to be processed locally, then the maximum load of a bin is equal to the makespan achieved by greedy allocation. The case of weighted balls, that corresponds to tasks whose lengths are not homogeneous, has been considered in [20]. It is shown in [20] that when allocating a large number of small balls with total weight W, one ends up with a smaller expected maximum load than the allocation of a smaller number of uniform balls with the same total weight. In the case of identical tasks [19], when $n/\text{polylog}(n) \leq p \leq n \log n$, *i.e.* typically when the average number of tasks allocated to each resource is a few units, then the expected maximum load is of order $\frac{\log n}{\log\left(\frac{n \log n}{p}\right)}$, it is therefore arbitrarily large and grows as $\frac{\log n}{\log \log n}$ when the number of tasks increases.

These techniques have been extended to multiple choice algorithms, where r random candidate bins are pre-selected for each ball, and the ball is allocated to the candidate bin whose load is minimal. It is well known that having more than one choice strongly improves load balancing. We refer the interested reader to [21, 22] for surveys that illustrate the power of two choices. Typically, combining previous results with those of [23], it can be proved that whatever the expected number of balls per bin, the expected maximum load is of order $n/p + \log \log n$ even in the case where $r = 2$, what represents a very strong improvement over the single choice case. The combination of multiple choice games with weighted balls has been considered in [24]. In this case, each ball comes with its weight w_i and is allocated, 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 allocated to it.

3 Modeling of Map-Reduce

We model the allocation mechanism of MapReduce as follows. We assume that there are p identical *processors*, *i.e.* processors with the same processing capabilities and equipped with their own data storage. There are n of *tasks* to process, and each task corresponds to the processing of a *chunk*. In the following theoretical analysis in Section 4 and 5, we assume that all tasks have the same processing needs. In the simulation Section 6, we remove this assumption to analyze the effect of task heterogeneity on the performance of schedulers. Chunks are initially replicated on the processors: r replicas of each chunk are distributed uniformly at random on the p processors. Note that we ignore inter/intra rack replication in this study.

We consider the following basic MapReduce-like scheduler: whenever a processor is available, if it holds local unprocessed chunks, it randomly selects one of these chunks for processing. Otherwise, a random chunk is allocated to this processor, and associated data is sent throughout the network. As soon as the processing of a chunk starts on a processor, all its replicas are discarded.

The problem of finding an allocation that minimizes the communication volume will be denoted by MinComm throughout this paper. As explained in the introduction, we will also consider a variant of this problem, which focuses on minimizing the makespan when communication of chunks is forbidden, that will be denoted by MinMakespan. The simple greedy scheduler easily translates to a solution of the MinMakespan problem. Whenever a processor is available, one of its local chunk (if any) is allocated for processing. Otherwise, the processor simply stops.

4 Finding a Better Allocation

In this section, we focus on the MinMakespan problem presented in the previous section. To analyze the performance of the greedy allocation mechanism, we prove in Section 4.1 its equivalence to the balls-into-bins with multiple choice problem. Then, we propose an optimal allocation mechanism based on matchings in Sections 4.2, 4.3 and 4.4. In Section 5, we concentrate on the critical case $n = p$ and we prove that in this case, load imbalance is at most 1 with high probability.

4.1 Analysis of the Greedy Allocation

In this section, we prove that in presence of replication, the expected makespan without communications is closely related to multiple choices balls-into-bins algorithms. This result holds in a more general context that the one introduced in the previous section and we consider here heterogeneous task processing times. Moreover, contrarily to what happens in the case of weighted balls and bins, we do not have to know ball weights in advance. More specifically, we propose an algorithm that allocates the replicas of the tasks to processors without

any knowledge of their durations. The greedy Allocation used at runtime then automatically balances the load between the processors. In what follows, for the sake of realism, we assume that processors start their execution at slightly different instants modeled by ϵ_i, that are expected to be small. We also assume that since tasks have heterogeneous durations, no ties have to be broken, as in [20, 24].

Algorithm 1: ReplicatedAllocation($\mathcal{C}_1, \ldots, \mathcal{C}_n, \epsilon_1, \ldots \epsilon_p$)

Allocation:
for $i = 0 \ldots n$ **do**
  place a copy of task T_i on each $P_{i^{(k)}}$, $1 \leq k \leq r$
  where $\mathcal{C}_i = \{i^{(1)}, \ldots, i^{(r)}\}$;
Execution:
When P_j ends a task, execute on P_j the available local task with the smallest index, if any ;

Algorithm 1 depicts the behavior of the greedy scheduler in presence of replicated chunks and the resulting allocation is entirely defined by the set of random choices $\mathcal{C}_i$'s of the r locations where task T_i is replicated (tasks processing ordering is given by their indexes). Similarly, let us consider Algorithm 2, similar to the one proposed in [20] and ruled by the same set of random choices $\mathcal{C}_i$'s and the initial load of the p bins $\epsilon_1, \ldots, \epsilon_p$.

Algorithm 2: Greedy($\mathcal{C}_1, \ldots, \mathcal{C}_n, \epsilon_1, \ldots, \epsilon_p$)

Allocation:
for $i = 0 \ldots n$ **do**
  Place ball b_i of weight w_i into the less loaded bin among bins $\{B_{i^{(1)}}, \ldots, B_{i^{(r)}}\}$ where $\mathcal{C}_i = \{i^{(1)}, \ldots, i^{(r)}\}$;

Theorem 1. *Let us denote by* $\textsc{MaxLoad}(\mathcal{C}_1, \ldots, \mathcal{C}_n, \epsilon_1, \ldots, \epsilon_p)$ *the maximal load of a bin using Greedy*$(\mathcal{C}_1, \ldots, \mathcal{C}_n, \epsilon_1, \ldots, \epsilon_p)$ *and let us denote by* $C_{\max}(\mathcal{C}_1, \ldots, \mathcal{C}_n, \epsilon_1, \ldots, \epsilon_p)$ *the makespan achieved using Replicated Allocation*$(\mathcal{C}_1, \ldots, \mathcal{C}_n, \epsilon_1, \ldots, \epsilon_p)$. *Then,*

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

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

Lemma 2. *Let $j_b(i)$ denote the index of the bin where ball b_i is allocated and let $j_p(i)$ denote the index of the processor where ball b_i is processed, then $j_b(i) = j_p(i)$.*

Proof. Let us consider ball b_1 and task ϵ_1. b_1 is allocated to 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 that issues a request, *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 allocated to, by induction hypothesis. Therefore, for each $P_k,\ k \in C_i$, the time at which P_k ends processing the tasks allocated to it and whose index is smaller than i is exactly the weight of the balls with index smaller than i allocated to B_k. Therefore, the processor P_k that issues the request for T_i first is the one such that t_k plus the weight of the balls with index smaller than i allocated to B_k is minimal, so that $j_b(i) = j_p(i)$, what achieves the proof of the lemma. □

Therefore, the makespan achieved by Replicated Allocation$(\mathcal{C}_1, \ldots, \mathcal{C}_n, t_1, \ldots, t_p)$ is equal to the load of most loaded bin in Greedy$(\mathcal{C}_1, \ldots, \mathcal{C}_n, t_1, \ldots, t_p)$, what 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 (see Section 2.3).

4.2 Matching-Based Allocations

We now focus on the design of better allocations than the one produced by above-mentioned greedy scheduler by the use of matching techniques.

Let P denote the set of processors and T the set of tasks. Let $E \subseteq P \times T$ denote a set of edges such that $(p_i, t_j) \in E$ if and only if the data chunk associated to t_j is replicated on processor p_i. The initial distribution of data chunks is entirely defined by bipartite graph $G = (P, T, E)$.

Definition 3 (Assignment)**.** *Let $G = (P, T, E)$ be a bipartite graph. An* assignment *of G is a subset A of E such that:*

$$\forall t_j \in T,\ \exists \text{ a unique } p_i \in P \text{ such that } (p_i, t_j) \in A.$$

An assignment is an allocation of the tasks between the processors with respect of the initial repartition of the input files. The following metrics will be used to evaluate the efficiency of an assignment.

Definition 4 (Degree in an assignment, maximum degree and total load imbalance)**.** *Let A be an assignment of $G = (P, T, E)$.*

- *The* degree in A *of a vertex p_i of P, denoted $d_A(p_i)$, is the degree of p_i in $G' = (P, T, A)$, the sub-graph of G induced by A.*

- *the* maximum degree $d(A)$ *of* A *is defined as* $D(A) = \max_{p_i \in P} d_A(p_i)$.
- *The* total load imbalance $Imb(A)$ *of* A *is given by* $Imb(A) = \sum_{\substack{p_i \in P \\ d_A(p_i) > \left\lceil \frac{|T|}{|P|} \right\rceil}} d_A(p_i) - \left\lceil \frac{|T|}{|P|} \right\rceil$.

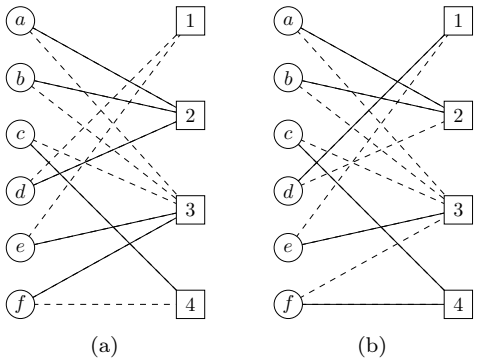


Figure 1: Two examples of assignments for the same problem.

These notions are illustrated on Figure 1, where tasks are on the left part and processors on the right. Solid edges represent an assignment and dashed ones are the initial edges that are not used in the assignment. Each task can be uniquely associated to one solid edge, but processors can be assigned to more than one task. On Figure 1(a), the maximum degree is 3 (reached on 2) and the total load imbalance is 1. On Figure 1(b), the maximum degree is 2 (reached on 2 and 4) and the total load imbalance is 0.

Intuitively, the degree of a processor in an assignment A is the number of tasks it has to process and the maximum degree of an assignment is the expected makespan for the MinMakespan problem. The total load imbalance of an assignment is the number of data chunks that have to be transferred in order to achieve perfect load balancing and therefore represents an upper bound on the amount of communication in the MinComm problem. Note that $D(A) = \left\lceil \frac{|T|}{|P|} \right\rceil \iff \mathrm{Imb}(A) = 0$.

4.3 Reduction to a Maximal Matching

In this section, we establish the relationship between assignment and matchings.

Definition 5 (Matching)**.** *Let* $G = (V, E)$ *be a graph. A* matching *of* G *is a subset* M *of* E *such that*

$$\forall e, e' \in E, e \cap e' = \emptyset.$$

More precisely, for a bipartite graph $G = (P, T, E)$, a matching M is a subset of E such that for all $(p_1, t_1), (p_2, t_2)$ in M:

$$(p_1, t_1) \neq (p_2, t_2) \implies p_1 \neq p_2 \wedge t_1 \neq t_2.$$

There are some notable differences between an assignment and a matching in a bipartite graph $G = (P, T, E)$. First, unlike in an assignment, there could exist no edge from a vertex t_j of T in a matching. Second, in a matching, there is at most one edge from every vertex p_i of P. We propose to build an auxiliary bipartite graph G^m by replicating m the set of processors, that is directly related through the next lemma to the existence of an assignment of maximum degree m.

Definition 6 (G^m)**.** *Let $G = (P, T, E)$ be a bipartite graph and m be an integer. We define $G^m = (P^m, T, E^m)$ with:*

- $P^m = \bigcup_{p_i \in P} \{p_i^1, \ldots, p_i^m\}$.
- $(p_i^k, t_j) \in E^m$ *if and only if* $(p_i, t_j) \in E$.

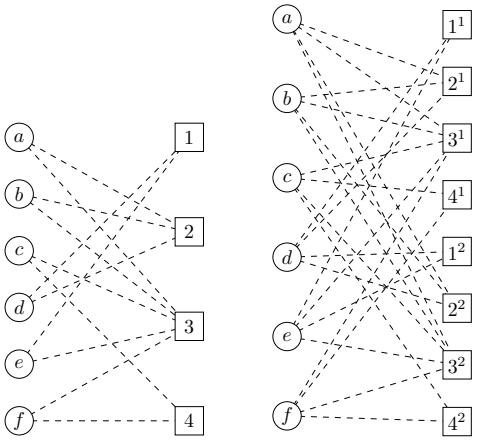


Figure 2: An example of replication of a bipartite graph. On the left G, on the right G^2.

An illustration of this notion is provided in Figure 2.

Lemma 7. *Let $G = (P, T, E)$ be a bipartite graph with $|T| = n$ and m be an integer. There exists an assignment of maximum degree m if and only if there exists a matching of size n in G^m.*

Proof. Let us first suppose that there exists an assignment A of maximum degree m of G. For $p_i \in P$ let $v_A(p_i)$ denotes its neighborhood in the sub-graph induced

by A, *i.e.* $v_A(p_i) = \{t_j \in T,\ (p_i, t_j) \in A\}$ (and $|v_A(p_i)| = d_A(p_i)$). Let M be a subset of E^m with $M = \bigcup_{p_i \in P} M_{p_i}$ with

$$M_{p_i} = \big\{(p_i^1, t_1), \ldots, (p_i^{d_A(p_i)}, t_{d_A(p_i)}), \\ \{t_1, \ldots, t_{d_A(p_i)}\} = v_A(p_i)\big\}.$$

Since $\forall p_i \in P$, $d_A(p_i) \leq m$, then $\forall p_i$, $\{p_i^1, \ldots, p_i^{d_A(p_i)}\}$ is a valid subset of P^m. Let (p_i^k, t_j) and $(p_{i'}^{k'}, t_{j'})$ denote any two edges of M.

- If $i \neq i'$, then $p_i^k \neq p_{i'}^{k'}$. In addition, by definition of an assignment, $v_A(p_i) \cap v_A(p_{i'}) = \emptyset$ (otherwise there is a contradiction with $\exists! p_i \in P, (p_i, t_j) \in A$) and then $t_j \neq t_{j'}$.
- If $i = i'$, then $t_j = t_{j'}$ if and only if $k = k'$.

Therefore, M is a valid matching. Moreover, since $v_A(p_i) \cap v_A(p_{i'}) = \emptyset$, then $M_{p_i} \cap M_{p_{i'}} = \emptyset$. Therefore, $|M| = \sum |M_{p_i}| = \sum d_A(p_i) = n$ by definition of an assignment. Thus, there exists a matching of cardinal n in G^m.

Let us now assume that there exists a matching M of G^m with $|M| = n$. Let us build a subset A of E such that

$$(p_i, t_j) \in A \Leftrightarrow \exists k,\ (p_i^k, t_j) \in M.$$

Let (p_i, t_j) and $(p_{i'}, t_{j'})$ be any two edges of A. There exists (k, k') such that $(p_i^k, t_j), (p_{i'}^{k'}, t_{j'}) \in M$. By definition of a matching, $t_j = t_{j'}$ if and only $p_i^k = p_{i'}^{k'}$. Hence $t_j = t_{j'}$ if and only if $p_i = p_{i'}$. Moreover, $|A| = |M| = n$ and therefore, $\forall t_j$, there exists p_i such that $(p_i, t_j) \in A$. Thus A is an assignment of G and $D(A) \leq m$ as $d_A(p_i) \leq m$ for all $p_i \in P$. □

4.4 Algorithm to Find an Optimal Assignment

Lemma 7 says that finding an assignment for a bipartite graph G is equivalent to finding a maximal matching (in G^m). More precisely, we propose in this section an algorithm that (i) builds G^m, (ii) finds a maximal matching, and (iii) converts it into an assignment for G. This algorithm, summarized in Algorithm 3, returns an assignment of minimum maximal degree.

The complexity of Algorithm 3 is nevertheless high in practice. Indeed, if there exist very efficient algorithms to compute maximum matchings on bipartite graph [15], Algorithm 3 requires to build G^m graph until it finds an assignment. In what follows, we propose, following classical ideas from the literature on the matching and flows problems [14], a more direct algorithm to find an assignment. Let us first adapt the notion of alternating path.

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

Algorithm 3: NaiveAssignment(G)

Input: A bipartite graph $G = (P, T, E)$
Output: An assignment A of G of minimum maximal degree
$m = \left\lceil \frac{|T|}{|P|} \right\rceil$;
$A = \emptyset$;
while *A is not an assignment* **do**
    Find a maximal matching M of G^m ;
    if $|M| = |T|$ **then**
        $A \leftarrow$ the conversion of M into an assignment of size m ;
    $m \leftarrow m + 1$;
return A

- *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 A is a path $(x_1, \ldots, x_k)$ of G such that:*
  - *If $x_i \in T$, then $(x_i, x_{i+1}) \in A$.*
  - *If $x_i \in P$, then $(x_i, x_{i+1}) \notin A$.*

An example of alternating path is described in Figure 3. On the left hand side, solid edges represent an assignment, and dashed ones the unused edges. On the right hand side, the proposed path (to improve the clarity of the scheme, edges that are not in the path has been removed) is an alternating one according to the previous assignment.

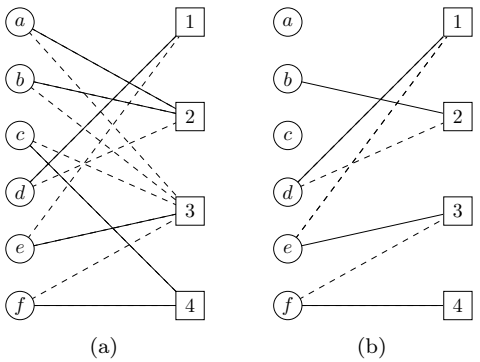


Figure 3: Example of Alternating path.

Alternating paths can be used to improve an existing assignment. Indeed, Lemma 9 states that if the starting and the ending vertices of an alternating

path are in P, then it is possible to build an assignment that improves the degree of the last vertex and increases by one the degree of the first one.

Lemma 9. *Let $G = (P, T, E)$ be a bipartite graph, let A be an assignment of G and let $x = (p_{i_1}, t_{j_1}, \ldots, p_{i_k})$ be an alternating path of G according to A. Let $\otimes$ be the xor operation ($A \otimes B = (A \cup B) \setminus (A \cap B)$) and let x be assimilated to its edges, then,*

- *$A \otimes x$ is an assignment.*
- $d_{A\otimes x}(p_{i_1}) = d_A(p_{i_1}) + 1$
- $d_{A\otimes x}(p_{i_k}) = d_A(p_{i_k}) - 1$
- $\forall p_i \in P \setminus \{p_{i_1}, p_{i_k}\},\ d_{A\otimes x}(p_i) = d_A(p_i)$.

Proof. Let $G = (P, T, E)$ be a bipartite graph, let A be an assignment of G and let $x = (p_{i_1}, t_{j_1}, \ldots, t_{j_{k-1}}, p_{i_k})$ be an alternating path of G according to A. Let $t_j \in T$:

- If $t_j = t_{j_l} \in \{t_{j_1}, \ldots, t_{j_{k-1}}\}$, then, by definition of an alternating path, $(t_{j_l}, p_{i_{l+1}})$ is in A (and, by definition of an assignment, it is the only edge from t_j in A) and (p_{i_l}, t_{j_l}) is not. Therefore (p_{i_l}, t_{j_l}) is in $A \otimes x$ (and it is the only edge from t_j in $A \otimes x$) and $(t_{j_l}, p_{i_{l+1}})$ is not.
- Otherwise, the unique edge in A from t_j is not in x (and x has no edges from t_j) and therefore is also present in $A \otimes x$.

Therefore, $\forall t_j \in T$, there is an unique edge from t_j in $A \otimes x$ that is thus an assignment.

Furthermore, let p_i be in P, then

- if $p_i = p_{i_1}$, then (p_{i_1}, t_{j_1}) is added to its neighborhood in $A \otimes x$,
- if $p_i = p_{i_k}$, then $(t_{j_{k-1}}, p_{i_k})$ is removed from its neighborhood in $A \otimes x$,
- if $p_i = p_{i_l} \in \{p_{i_2}, \ldots, p_{i_{k-1}}\}$, then (p_{i_l}, t_{j_l}) is added to its neighborhood and the edge $(t_{j_{l-1}}, p_{i_l})$ is removed from its neighborhood in $A \otimes x$,
- else there is no modification of its neighborhood from A to $A \otimes x$.

Therefore,

- $d_{A\otimes x}(p_{i_1}) = d_A(p_{i_1}) + 1$
- $d_{A\otimes x}(p_{i_k}) = d_A(p_{i_k}) - 1$
- $\forall p_i \in P \setminus \{p_{i_1}, p_{i_k}\},\ d_{A\otimes x}(p_i) = d_A(p_i)$.

□

Definition 10. *Let $G = (P, T, E)$ be a bipartite graph and A be an assignment of G. An* augmenting path *according to A is an alternating path $x = (p_{i_1}, t_{j_1}, \ldots, p_{i_k})$ such that $d_A(p_{i_1}) + 1 < d_A(p_{i_k})$.*

Lemma 9 says that an augmenting path can be used to build an assignment that can even decrease the maximum degree if p_{i_k} is the only vertex such that $d_A(p_{i_k}) = D(A)$. In fact, a stronger property holds true: if there is no augmenting path, then the assignment has a minimum maximum degree. This result is formalized in Theorem 11.

Theorem 11. *Let $G = (P, T, E)$ be a bipartite graph, let A be an assignment of G. If there is no augmenting path according to A, then A has a minimum maximum degree.*

We will rely on the following lemmas to prove Theorem 11.

Lemma 12. *(Berge's Lemma [25]) Let $G = (P, T, E)$ be a bipartite graph and M be a matching of G. M is maximal if and only if there no alternating path $x = (p_{i_1}, t_{j_1}, \ldots, t_{j_{k-1}})$ such that there is no edge from p_{i_1} and from $t_{j_{k-1}}$ in M.*

Lemma 13. *Let A and A' be two assignments of a same bipartite graph $G = (P, T, E)$. If there exists p_i such that $d_A(p_i) > d_{A'}(p_i)$, then there exists an alternating path $x = (p_d, \ldots, p_f)$ according to A such that $d_A(p_d) < d_{A'}(p_d)$ and p_f is the vertex verifying $d_A(p_f) > d_{A'}(p_f)$ with the largest degree.*

Proof. To prove above lemma, we need a more general definition of a replicated graph. Instead to replicate each vertex of P m times, we replicate p_i m_{p_i} times (the number or replication can be different from a vertex to another). In this case, Lemma 7 can be adapted: let $G = (P, T, E)$ be a bipartite graph with $|T| = n$ and $(m_i)_{1 \leq i \leq p}$ be a set of integers. Then, there exists an assignment A with, for all $p_i \in P$, $d_A(p_i) \leq m_i$ if and only if there exists a matching of size n in the replicated graph of G where each p_i is replicated m_i times.

Let now A and A' be two assignments of a bipartite graph G such that $\exists p_i$ that fulfills $d_A(p_i) > d_{A'}(p_i)$. Let p_f be the vertex satisfying $d_A(p_f) > d_{A'}(p_f)$ with the largest degree. Let us consider the graph G' that is a replicated graph of G where p_f is replicated $d_A(p_f) - 1$ times and where $p_i \in P \setminus \{p_f\}$ is replicated $max(d_A(p_i), d_{A'}(p_i))$. Let $t \in T$ be such that $(p_f, t) \in A$. Thanks to the modified version of Lemma 7, we can define a matching M' of G' of size $|T|$ and a matching M of size T associated to the partial assignment $A \setminus (p_f, t)$.

M is not a matching of maximum size since M' is larger. Therefore, according to Lemma 12, there exists an alternating path $x = (p_{i_1}, t_{j_1}, \ldots, t_{j_{k-1}})$ such that there is no edge from p_{i_1} and from $t_{j_{k-1}}$ in M. The only possible free vertex from T is t, thus this alternating path must fulfill $x = (p_d^{k_d}, \ldots, t)$. In addition, to have a free replicate in G' according to M, p_d must fulfill $d_A(p_d) < max(d_A(p_d), d_{A'}(p_d))$ and therefore $d_A(p_d) < d_{A'}(p_d)$.

We then easily check that the path $(p_d, \ldots, t, p_f)$ is an alternating path and, as $d_A(p_d) < d_{A'}(p_d)$, thus claimed result. □

Lemma 14. *Let $G = (P, T, E)$ be a bipartite graph and let A, A' be two assignments of G. Therefore there exist finite sequences of assignments $(A_k)_{k \leq l}$ and paths $(x_k)_{k \leq l-1}$ (l is the length of the sequence) such that*

- $A_0 = A$

- $\forall p_i \in P,\ d_{A_l}(p_i) = d_{A'}(p_i)$
- $\forall k \leq l$, x_k *is alternating according to* A_k *and* $A_{k+1} = A_k \otimes x_k$
- *If* $x_k = (p_{d_k}, \ldots, p_{f_k})$, *then* $d_{A_k}(p_{d_k}) < d_{A'}(p_{d_k})$ *and* $d_{A_k}(p_{f_k}) > d_{A'}(p_{f_k})$ *and* p_f *is the greatest such* p_i.
- $\forall p_i \in P$, *the sequence* $(|d_{A_k}(p_i) - d_{A'}(p_i)|)_{k \leq l}$ *is decreasing.*

Proof. Let us suppose that these sequences have been built until rank k. If $\forall p_i \in P,\ d_{A_k}(p_i) = d_{A'}(p_i)$, then sequences are built. Else, according to Lemma 13, there exists an alternating path $x_k = (p_{d_k}, \ldots, p_{d_f})$ such that $d_{A_k}(p_{d_k}) < d_{A'}(p_{d_k})$ and p_f is the vertex with the larger degree verifying $d_{A_k}(p_{f_k}) > d_{A'}(p_{f_k})$. Let us define $A_{k+1} = A_k \otimes x_k$.

Let us prove that sequences are finite. Let us consider the sequence $(u_k)_{k \in \mathbb{N}}$ defined by $u_k = \sum_{p_i \in P} |d_{A_k}(p_i) - d_{A'}(p_i)|$.

$$
\begin{aligned}
u_{k+1} &= \sum_{p_i \in P} |d_{A_{k+1}}(p_i) - d_{A'}(p_i)| \\
&= \sum_{p_i \in P \setminus \{p_{d_k}, p_{f_k}\}} |d_{A_{k+1}}(p_i) - d_{A'}(p_i)| + |d_{A_{k+1}}(p_{d_k}) - d_{A'}(p_{d_k})| + |d_{A_{k+1}}(p_{f_k}) - d_{A'}(p_{f_k})| \\
&= \sum_{p_i \in P \setminus \{p_{d_k}, p_{f_k}\}} |d_{A_k}(p_i) - d_{A'}(p_i)| + |d_{A_k}(p_{d_k}) + 1 - d_{A'}(p_{d_k})| + |d_{A_k}(p_{f_k}) - 1 - d_{A'}(p_{f_k})| \\
&= \sum_{p_i \in P \setminus \{p_{d_k}, p_{f_k}\}} |d_{A_k}(p_i) - d_{A'}(p_i)| + |d_{A_k}(p_{d_k}) - d_{A'}(p_{d_k})| - 1 + |d_{A_k}(p_{f_k}) - d_{A'}(p_{f_k})| - 1 \\
&= u_k - 2
\end{aligned}
$$

Therefore there exists an index l such that $u_l = 0$ and hence $\forall p_i \in P,\ d_{A_l}(p_i) = d_{A'}(p_i)$ and the sequences are finite. Note that similar calculations prove that $(|d_{A_k}(p_i) - d_{A'}(p_i)|)_{k \leq l}$ is decreasing for all p_i.

□

With Lemma 14, we can now prove Theorem 11. Let $G = (P, T, E)$ be a bipartite graph and let A be an assignment of G. Let us suppose that A has not a minimum maximum degree. Therefore, there exists A' such that $D(A') < D(A)$.

Thank to Lemma 14, we know that there are finite sequences of assignments $(A_k)_{k \leq l}$ and paths $(x_k)_{k < l}$ such that

- $A_0 = A$,
- $A_l = A'$,
- $\forall p_i \in P,\ d_{A_l}(p_i) = d_{A'}(p_i)$,
- $\forall k < m$, x_k is alternating and $A_{k+1} = A_k \otimes x_k$.

In particular $D(A_0) > D(A_l)$. Therefore, there exists k_0 such that $D(A_{k_0}) > D(A_{k_0+1}) = D(A_{k_0} \otimes x_{k_0})$. Let $x_{k_0} = (p_{d_{k_0}}, \ldots, p_{f_{k_0}})$. Since x_{k_0} is alternating

- $d_{A_{k_0} \otimes x_{k_0}}(p_{d_{k_0}}) = d_{A_{k_0}}(p_{d_{k_0}}) + 1$,
- $d_{A_{k_0} \otimes x_{k_0}}(p_{f_{k_0}}) = d_{A_{k_0}}(p_{f_{k_0}}) - 1$,
- $\forall p_i \in P \setminus \{p_{d_{k_0}}, p_{f_{k_0}}\},\ d_{A_{k_0} \otimes x_{k_0}}(p_i) = d_{A_{k_0}}(p_i)$.

Yet, $D(A_{k_0}) > D(A_{k_0} \otimes x_{k_0})$ and, since this is the only one with decreasing degree, $D(A_{k_0}) = d_{A_{k_0}}(p_{f_{k_0}})$. Moreover,

$$\begin{aligned} &d_{A_{k_0}}(p_{d_{k_0}}) + 1 = d_{A_{k_0} \otimes x_{k_0}}(p_{d_{k_0}}), \\ &d_{A_{k_0}}(p_{d_{k_0}}) + 1 \leq D(A_{k_0} \otimes x_{k_0}), \\ &d_{A_{k_0}}(p_{d_{k_0}}) + 1 < D(A_{k_0}), \\ &d_{A_{k_0}}(p_{d_{k_0}}) + 1 < d_{A_{k_0}}(p_{f_{k_0}}). \end{aligned}$$

Therefore, x_{k_0} is augmenting, $D(A_{k_0}) = d_{A_{k_0}}(p_{f_{k_0}})$ and $d_{A_{k_0}}(p_{d_{k_0}}) < d_{A'}(p_{d_{k_0}})$.

We have proved if there exists an index k_0 such that $D(A_{k_0}) > D(A')$, there exists an augmenting path according to A_{k_0} with some additional properties. Let us define k_0 as the smallest k such that $D(A_k) > D(A')$. There exists an augmenting path $x = (p_d, \ldots, p_f)$ according to A_k (and that augmenting path could differ from x_k) and such that $d_{A_k}(p_f) = D(A_k)$ and $d_{A_{k_0}}(p_d) < d_{A'}(p_d)$. Let us try to prove that $k_0 = 0$ in order to reach a contradiction.

Let $x = (p_d, \ldots, p_f)$ be such an augmenting path in A_{k_0} $D(A_{k_0}) > D(A')$.

First, let us note that $D(A_{k_0-1}) \geq D(A_{k_0}) > D(A')$. Indeed, for all $p_i \neq p_{d_{k_0-1}}$, $d_{A_{k_0-1}}(p_i) \geq d_{A_{k_0}}(p_i)$.

Second, let us assume that x and x_{k_0-1} are disjoint. In this case, $d_{A_{k_0-1}}(p_d) = d_{A_{k_0}}(p_d) < d_{A'}(p_d)$ and $d_{A_{k_0-1}}(p_f) = d_{A_{k_0}}(p_f) = D(A_{k_0}) \leq D(A_{k_0-1})$. If $D(A_{k_0}) = D(A_{k_0-1})$. Then, x is a valid augmenting path and $d_{A_{k_0-1}}(p_f) = D(A_{k_0-1})$. Otherwise, $D(A_{k_0-1}) = d_{A_{k_0-1}}(p_{f_{k_0-1}})$ $(D(A_{k_0-1}) > D(A_{k_0}) > D(A')$ and thus, there exists at least one p_i such that $d_{A_{k_0-1}}(p_i) > d_{A'}(p_i)$ and by construction this is also the case for $p_{f_{k_0-1}}$), so that

$$d_{A_{k_0-1}}(p_{d_{k_0-1}}) + 1 = d_{A_{k_0}}(p_{d_{k_0-1}}) \leq D(A_{k_0}) < d_{A_{k_0-1}}(p_{f_{k_0-1}}).$$

Hence, x_{k_0-1} is an augmenting path with $D(A_{k_0-1}) = d_{A_{k_0-1}}(p_{f_{k_0-1}})$ and $d_{A_{k_0-1}}(p_{d_{k_0-1}}) < d_{A'}(p_{d_{k_0-1}})$ by construction. Thus x and x_{k_0-1} are not disjoint since otherwise, we would reach a contradiction with the definition of k_0.

Therefore, we know that x and x_{k_0-1} are not disjoint. Let v_i and v_j be two vertices of G such that

- v_i is the first vertex in x to be in x_{k_0-1},
- v_j is the last vertex in x to be in x_{k_0-1}.

Let us suppose that v_i is before v_j in x_{k_0-1}. Then, $(p_d, \dots, v_i, \dots, v_j, \dots, p_f)$ is a valid alternating path in A_{k_0-1}. Because $(|d_{A_k}(p_i) - d_{A'}(p_i)|)_{k \leq l}$ is decreasing, $d_{A_{k_0-1}}(p_d) \leq d_{A_{k_0}}(p_d) < d_{A'}(p_d)$. Similarly $d_{A_{k_0-1}}(p_f) \geq d_{A_{k_0}}(p_f)$ and thus $d_{A_{k_0-1}}(p_d) + 1 < d_{A_{k_0-1}}(p_f)$ and $(p_d, \dots, v_i, \dots, v_j, \dots, p_f)$ is augmenting. If $D(A_{k_0-1}) = D(A_{k_0})$, then we reach a contradiction with the definition of k_0. Otherwise we know that $D(A_{k_0-1}) = d_{A_{k_0-1}}(p_{f_{k_0-1}})$ and can prove that $(p_d, \dots, v_i, \dots, v_j, \dots, p_{f_{k_0-1}})$ is an augmenting path with all the properties we want, thus another contradiction.

Finally, we prove that v_i is after v_j in x_{k_0-1}. Therefore, $y = (p_{d_{k_0}}, \dots, v_j, \dots, p_f)$ is an alternating path according to A_{k_0-1}. As previously, $d_{A_{k_0-1}}(p_{d_{k_0}}) < d_{A'}(p_{d_{k_0}}) \leq D(A')$. Thus $d_{A_{k_0-1}}(p_{d_{k_0}}) + 1 \leq D(A')$. Similarly, $D(A') < D(A_{k_0}) = d_{A_{k_0}}(p_f) \leq d_{A_{k_0-1}}(p_f)$. Thus, y is augmenting and if $D(A_{k_0-1}) = D(A_{k_0})$, then we reach a contradiction with the definition of k_0. Otherwise, we consider $y' = (p_d, \dots, v_i, \dots, p_{f_{k_0}})$ and reach a similar contradiction.

Hence, we prove that, in any case, $k_0 > 0$ implies the existence of an augmenting path according to A_{k_0-1} with all desired properties. Therefore, $k_0 = 0$ and there is an augmenting path according to $A_0 = A$, what achieves the proof of the theorem.

□

Based on Theorem 11, we can build an algorithm to find an assignment with minimum maximum degree as follows.

Algorithm 4: FindAssignment(G)

Input: A bipartite graph $G = (P, T, E)$
Output: An assignment A of G of minimum maximal degree
$A = \emptyset$;
foreach $t_j \in T$ **do**
  Choose a random p_i such that $(p_i, t_j) \in E$ and add this edge to A;
while *there exists an augmenting path according to* A **do**
  Compute an augmenting path x according to A;
  $A \leftarrow A \otimes x$;
return A

To prove the termination of Algorithm 4, let us consider $\sum d_A(p_i)^2$. If $x = (p_d, \dots, p_f)$ is an augmenting path, then

$$\begin{aligned}\sum d_{A\otimes x}(p_i)^2 - \sum d_A(p_i)^2 &= (d_A(p_d)+1)^2 + (d_A(p_f)-1)^2 \\ &\quad - d_A(p_d)^2 - d_A(p_f)^2 \\ &= 2(d_A(p_d) + 1 - d_A(p_f)) < 0\end{aligned}$$

Therefore, $\sum d_A(p_i)^2$ is decreasing during the execution of Algorithm 4. Since this value is bounded (trivially by 0), there is an instant where there

no longer exists an augmenting path and Algorithm 4 terminates, returning an assignment with minimum maximum degree, what solves MinMakespan problem.

Note that we can adapt all these results to prove that if there is no augmenting path, the assignment also achieves an optimal total load imbalance, thus providing an optimal solution for MinComm as stated in the following theorem.

Theorem 15. *Let $G = (P, T, E)$ be a bipartite graph, let A be an assignment of G. If there is no augmenting path according to A then A has a minimum total load imbalance.*

Lemma 16. *Let $G = (P, T, E)$ be a bipartite graph and A be an assignment of G. Let $x = (p_d, \ldots, p_f)$ be an alternating path according to A. Then,*

- *if $d_A(p_d) < \left\lceil \frac{|T|}{|P|} \right\rceil < d_A(p_f)$ hence $Imb(A \otimes x) = Imb(A) - 1$,*
- *if $d_A(p_d) > \left\lceil \frac{|T|}{|P|} \right\rceil > d_A(p_f)$ hence $Imb(A \otimes x) = Imb(A) + 1$,*
- *otherwise $Imb(A \otimes x) = Imb(A)$.*

Lemma 16 can be proved using Lemma 9. Note that an alternating path $x = (p_d, \ldots, p_f)$ such that $d_A(p_d) < \left\lceil \frac{|T|}{|P|} \right\rceil < d_A(p_f)$ is an augmenting path.

Let us now prove Theorem 15. Let $G = (P, T, E)$ be a bipartite graph and let A be an assignment of G. Let us suppose that A has not an optimal total load imbalance. In this case, there exists an assignment A' such that $\mathrm{Imb}(A) > \mathrm{Imb}(A')$. Thanks to Lemma 14, we can build two finite sequences $(A_k)_{k \leq l}$ and $(x_k)_{k \leq l-1}$ such that

- $A_0 = A$,
- $A_l = A'$,
- $\forall p_i \in P,\ d_{A_l}(p_i) = d_{A'}(p_i)$,
- $\forall k < m$, x_k is alternating and $A_{k+1} = A_k \otimes x_k$.

In particular $\mathrm{Imb}(A_0) > \mathrm{Imb}(A_l)$. Therefore, there exists k_0 such that $\mathrm{Imb}(A_{k_0}) > \mathrm{Imb}(A_{k_0+1}) = \mathrm{Imb}(A_{k_0} \otimes x_{k_0})$. Let $x_{k_0} = (p_{d_{k_0}}, \ldots, p_{f_{k_0}})$, then, thanks to Lemma 16, $d_{A_{k_0}}(p_{d_{k_0}}) < \left\lceil \frac{|T|}{|P|} \right\rceil < d_{A_{k_0}}(p_{f_{k_0}})$. Moreover, by construction of the sequences, $d_{A_{k_0}}(p_{f_{k_0}}) > d_{A'}(p_{f_{k_0}})$.

Thus, we have proven that there exists k_0 such that $x = (p_d, \ldots, p_f)$ is an alternating path according to A_{k_0} and such that $d_{A_{k_0}}(p_d) < \left\lceil \frac{|T|}{|P|} \right\rceil < d_{A_{k_0}}(p_f)$ and $d_{A_{k_0}}(p_f) > d_{A'}(p_f)$. Let k_0 be the smallest k such that there exists such a path according to A_k (not necessarily x_k). In order to prove that $k_0 = 0$, let us assume by contradiction that $k_0 > 0$.

Let $x = (p_d, \ldots, p_f)$ be an alternating path according to A_{k_0} such that $d_A(p_d) < \left\lceil \frac{|T|}{|P|} \right\rceil < d_A(p_f)$ and $d_{A_{k_0}}(p_f) > d_{A'}(p_f)$.

Let us suppose that x and x_{k_0-1} are disjoint. In this case, x is a valid alternating path according to A_{k_0-1}, $d_{A_{k_0-1}}(p_d) = d_{A_{k_0}}(p_d) < \left\lceil \frac{|T|}{|P|} \right\rceil$ and $d_{A_{k_0-1}}(p_f) = d_{A_{k_0}}(p_f) > \left\lceil \frac{|T|}{|P|} \right\rceil$. Thus, we reach a contradiction with the definition of k_0.

Let us assume that x and x_{k_0-1} are not disjoint. In this case, let us define v_i and v_j such that

- v_i is the first vertex in x to be in x_{k_0-1},
- v_j is the last vertex in x to be in x_{k_0-1}.

If v_i is before v_j in x_{k_0-1}, then $(p_d, \ldots, v_i, \ldots, v_j, \ldots p_f)$ is a valid alternating path according to A_{k_0-1}. Furthermore, if $v_i \neq p_d$, p_d is not in x_{k_0-1} and $d_{A_{k_0-1}}(p_d) = d_{A_{k_0}}(p_d) < \left\lceil \frac{|T|}{|P|} \right\rceil$. Otherwise, $d_{A_{k_0-1}}(p_d) = d_{A_{k_0}}(p_d) - 1 < \left\lceil \frac{|T|}{|P|} \right\rceil$. Thus, in all cases $d_{A_{k_0-1}}(p_d) < \left\lceil \frac{|T|}{|P|} \right\rceil$ and similarly, we prove that $\left\lceil \frac{|T|}{|P|} \right\rceil < d_{A_{k_0-1}}(p_f)$ and reach a contradiction with the definition of k_0.

Therefore, v_j is before v_i in x_{k_0-1}. Let us consider the path $y = (p_d, \ldots, v_i, \ldots, p_{f_{k_0-1}})$, that is a valid alternating path according to A_{k_0-1}. As previously, $d_{A_{k_0-1}}(p_d) < \left\lceil \frac{|T|}{|P|} \right\rceil$. In addition, $d_{A'}(p_f) < d_{A_{k_0}}(p_f) \leq d_{A_{k_0-1}}(p_f)$ by assumption and because $(|d_{A_k}(p_i) - d_{A'}(p_i)|)_{k \leq l}$ is decreasing. Thus, by construction of the p_{f_k}'s, $d_{A_{k_0-1}}(p_f) \leq d_{A_{k_0-1}}(p_{f_{k_0-1}})$. Therefore, $\left\lceil \frac{|T|}{|P|} \right\rceil < d_{A_{k_0-1}}(p_{f_{k_0-1}})$ and y is an alternating path with all desired properties. Therefore, we reach a contradiction with the definition of k_0.

Hence, we prove by contradiction that $k_0 = 0$ and thus that there exists an alternating path $x = (p_d, \ldots, p_f)$ according to A such that $d_A(p_d) < \left\lceil \frac{|T|}{|P|} \right\rceil < d_A(p_f)$. Therefore, there exists an augmenting path according to A, what achieves the proof of Theorem 15 by contraposition.

□

5 Performance Analysis of Optimal Allocations

In Section 3, we have shown how the performance of the greedy algorithm could be analyzed using Balls into Bins results and we have proposed in Section 4 a new algorithm FindAssignment that computes efficiently the optimal assignment, given an initial randomized distribution of data chunks similar to what is achieved by HDFS. In this section, we aim at analyzing the performance of FindAssignment. More specifically, we aim at proving that the maximum degree of FindAssignment is with high probability only a few units above the lower bound $\lceil n/p \rceil$. We call *perfect* an assignment that reaches the $\lceil n/p \rceil$ bound, and *quasi-perfect* an assignment whose maximum degree is $\lceil n/p \rceil + 1$. Note that

FindAssignment is a deterministic algorithm that computes the assignment of minimal maximum degree and that (i) randomness comes only from the initial chunk distribution and (ii) $\lceil n/p \rceil$ is an absolute bound that does not depend of the quality of the distribution. Therefore, the results in this section show that a small replication (2 is enough in general) is enough to enable the existence of matchings with very small makespan.

In what follows, we mostly rely on [13] for the analysis techniques and we use some results on bipartite graph's matching from the literature. In fact, thanks to Lemma 7, we can use a well-known result on matching on bipartite graph.

Theorem 17 (Hall's Theorem [26])**.** *Let $G = (P, T, E)$ be a bipartite graph. There exists a perfect matching (of cardinal $min(|P|, |T|)$) of G if and only if for all subset T' of T, its neighborhood P' verifies $|P'| \geq |T'|$.*

Based on Theorem 17 we can prove the following lemma.

Lemma 18. *Let $G = (P, T, E)$ be a bipartite graph. There exists an assignment of G of maximum degree m if and only if for all subset T' of T, its neighborhood P' satisfies $m|P'| \geq |T'|$.*

Proof. Let T' be a subset of T and let $v_G(T')$ be its neighborhood in G. By construction of $G^m = (P^m, T, E^m)$, $|v_{G^m}(T')| = m|v_G(T')|$. In addition, thank to Lemma 7, there exists an assignment of G of maximum degree m if and only if there is a perfect matching of G^m, that is equivalent (Theorem 17) to $\forall T' \subseteq T$, $|T'| \leq |v_{G^m}(T')| = m|v_G(T')|$. □

5.1 Probability of Existence of a Quasi-Perfect Assignment

In what follows, we use Lemma 18 to analyze the probability that there exists such an assignment of maximum degree m with a random initial distribution of the data chunks. Let t_j be a task of T and let P' be a subset of P of cardinal q. The probability that the entire neighborhood of t_j is included in P' is $\frac{\binom{q}{r}}{\binom{p}{r}}$, this is simply a draw without replacement.

Let $X_{P'}$ be the random variable that gives the number of tasks whose neighborhood is included in P'. $X_{P'}$ conforms to a binomial law of parameter n (the number of tasks) and $\frac{\binom{q}{r}}{\binom{p}{r}}$ (the probability that a task has its neighborhood included in P') with q the cardinal of P'. Therefore,

$$Pr(X_{P'} = k) = \binom{n}{k} \left(\frac{\binom{q}{r}}{\binom{p}{r}} \right)^k \left(1 - \frac{\binom{q}{r}}{\binom{p}{r}} \right)^{n-k}.$$

Let $Y_{q,k}$ be the number of subset of P of size q such that exactly k element of T have their neighborhood included in this subset. Formally,

$$Y_{q,k} = \sum_{\substack{P' \subseteq P \\ |P'|=q}} \mathbb{1}_{X_{P'}=k}$$

where $\mathbb{1}_{X_{P'}=k}$ is the random variable which is equal to 1 when $X_{P'} = k$ and to 0 otherwise. Therefore $E(\mathbb{1}_{X_{P'}=k}) = Pr(X_{P'} = k)$ and thus, by linearity of expectation,

$$\begin{aligned}
E(Y_{q,k}) &= E\left(\sum_{\substack{P'\subseteq P\\|P'|=q}} \mathbb{1}_{X_{P'}=k}\right)\\
&= \sum_{\substack{P'\subseteq P\\|P'|=q}} E(\mathbb{1}_{X_{P'}=k})\\
&= \sum_{\substack{P'\subseteq P\\|P'|=q}} Pr(X_{P'} = k)\\
&= \sum_{\substack{P'\subseteq P\\|P'|=q}} \binom{n}{k}\left(\frac{\binom{q}{r}}{\binom{p}{r}}\right)^k\left(1-\frac{\binom{q}{r}}{\binom{p}{r}}\right)^{n-k}\\
&= \binom{p}{q}\binom{n}{k}\left(\frac{\binom{q}{r}}{\binom{p}{r}}\right)^k\left(1-\frac{\binom{q}{r}}{\binom{p}{r}}\right)^{n-k}.
\end{aligned}$$

Lemma 18 states that there exists an assignment of maximal degree m if and only if, for all subset T' of T, $m|P'| \geq |T'|$, where P' is the neighborhood of T'. Therefore, there is no such assignment if and only if there exists T' and its neighborhood P' such that $m|P'| < |T'|$ and therefore if and only if there exists (q, i), with $q \in [0, p], i \in \mathbb{N}^*$, such that $Y_{q,mq+i} \geq 1$.

Note that event $(Y_{q,mq+m+i} \geq 1)$ is included in event $(Y_{q+1,m(q+1)+i} \geq 1)$. Indeed, if there is a set of q processors that contains the entire neighborhood of $mq + m + i$ tasks, then there is also a set of $q + 1$ processors that contains this neighborhood (obtained by adding any processor that is not already in the subset) and thus $Y_{q,mq+m+i} \geq 1$ implies $Y_{q+1,m(q+1)+i} \geq 1$. And, at contrary, $Y_{q+1,m(q+1)+i} = 0$ implies $Y_{q,mq+m+i} = 0$. Therefore, we can consider the events $(Y_{q,mq+i} \geq 1)$ with $i \in [1, m]$ only. Let us define the random variable $Z = \sum_{q=0}^{p}\sum_{i=1}^{m} Y_{q,mq+i}$. If $Z = 0$, then $Y_{q,mq+i} = 0$ for all (q, i) and there exists an assignment of maximum degree m. Otherwise, if $Z \geq 1$ there exists a (q, i) such that $Y_{q,mq+i} \geq 1$ and then there is no assignment of maximum degree m.

Using to Markov inequality, we have

5.2 Existence of quasi-perfect assignment in the case $n = p$

For the particular case of a regular bipartite graph where $n = p$, a quasi-perfect assignment has size 2, and we prove in Theorem 19 that the probability of non-existence of assignment of size 2 is asymptotically 0.

$$\begin{aligned} Pr(Z \geq 1) &\leq \tfrac{E(Z)}{1} \\ &\leq \sum_{q=0}^{p}\sum_{i=1}^{m} Y_{q,mq+i} \\ &\leq \sum_{q=0}^{p}\sum_{i=1}^{m} \tbinom{p}{q}\tbinom{n}{mq+i} \\ &\quad \times \left(\frac{\tbinom{q}{r}}{\tbinom{p}{r}}\right)^{mq+i}\left(1-\frac{\tbinom{q}{r}}{\tbinom{p}{r}}\right)^{n-mq-i}. \end{aligned}$$

Theorem 19. *With the previous notations and with $n = p$, $m = 2$ and $r \geq 2$:*

$$Pr(Z \geq 1) = O\left(\frac{1}{p}\right).$$

We first recall the bound on $Pr(Z \geq 1)$:

$$Pr(Z \geq 1) \leq \sum_{q=0}^{p}\sum_{i=1}^{m} Y_{q,mq+i}$$

with

$$Y_{q,mq+i} = \binom{p}{q}\binom{n}{mq+i}\left(\frac{\tbinom{q}{r}}{\tbinom{p}{r}}\right)^{mq+i}\left(1-\frac{\tbinom{q}{r}}{\tbinom{p}{r}}\right)^{n-mq-i}.$$

Proof. First, note that

$$\left(\frac{\tbinom{q}{r}}{\tbinom{p}{r}}\right)^{mq+i}\left(1-\frac{\tbinom{q}{r}}{\tbinom{p}{r}}\right)^{n-mq-i} \leq \left(\frac{q}{p}\right)^{r(mq+i)}\left(1-\left(\frac{q}{p}\right)^{r}\right)^{n-mq-i}.$$

The intuition is that a drawing with replacement, the right case, produces a graph with inferior degree than a drawing without replacement, the left case, and so a higher probability of non-existence of an assignment of maximum degree m. Therefore:

$$E(Y_{q,mq+i}) \leq \binom{p}{q}\binom{n}{mq+i}\left(\frac{q}{p}\right)^{r(mq+i)}\left(1-\left(\frac{q}{p}\right)^{r}\right)^{n-mq-i}$$

We now suppose that $n = p$, $m = 2$ and $r = 2$ (Z decreased with the increasing of r). Therefore,

$$Pr(Z \geq 1) \leq \sum_{q=0}^{p} Y_{q,2q+1} + Y_{q,2q+2}$$

with

$$E(Y_{q,2q+i}) = \binom{p}{q}\binom{p}{2q+i}\left(\frac{q}{p}\right)^{4q+2i}\left(1-\left(\frac{q}{p}\right)^{2}\right)^{p-2q-i}.$$

A first important remark is that $\binom{p}{2q+1} = \binom{p}{2q+2} = 0$ for $q > \frac{p}{2}$. Therefore, in this case $Y_{q,2q+1} = 0$ and $Y_{q,2q+2} = 0$. Furthermore $Y_{0,k} = 0$. Hence:

$$Pr(Z \geq 1) \leq \sum_{q=1}^{\frac{p}{2}} E(Y_{q,2q+1}) + E(Y_{q,2q+2}).$$

In addition,

$$\begin{aligned}
E(Y_{q,2q+2}) &= \binom{p}{q}\binom{p}{2q+2}\left(\frac{q}{p}\right)^{4q+4}\left(1-\left(\frac{q}{p}\right)^2\right)^{p-2q-2} \\
&= \binom{p}{q}\binom{p}{2q+1}\frac{p-2q-1}{2q+2}\left(\frac{q}{p}\right)^{4q+2}\left(\frac{q}{p}\right)^2\left(1-\left(\frac{q}{p}\right)^2\right)^{p-2q-1}\left(\frac{p^2-q^2}{p^2}\right)^{-1} \\
&= E(Y_{q,2q+1})\frac{p-2q-1}{2q+2}\left(\frac{q}{p}\right)^2\frac{p^2}{p^2-q^2} \\
&= E(Yq,2q+1)\frac{p-2q-1}{p-q}\frac{q}{2q+2}\frac{q}{p+q}.
\end{aligned}$$

Note that $\frac{q}{2q+2} < \frac{1}{2}$ and, as $p \geq q$, $\frac{q}{p+q} \leq \frac{1}{2}$. In addition, $x \mapsto \frac{p-2x-1}{p-x}$ is a decreasing function on $\left[0, \frac{p}{2}\right]$. Therefore $\frac{p-2q-1}{p-q} \leq \frac{p-1}{p} < 1$. Therefore,

$$E(Y_{q,2q+2}) < \frac{E(Y_{q,2q+1})}{4}.$$

From this bound on $E(Y_{q,2q+2})$ we can obtain

$$\begin{aligned}
Pr(Z \geq 1) &= \sum_{q=1}^{\frac{p}{2}} E(Y_{q,2q+1}) + E(Y_{q,2q+2}) \\
&< \frac{5}{4}\sum_{q=1}^{\frac{p}{2}} E(Y_{q,2q+1}) \\
&< \frac{5}{4}\sum_{q=1}^{\frac{p}{2}} \binom{p}{q}\binom{p}{2q+1}\left(\frac{q}{p}\right)^{4q+2}\left(1-\left(\frac{q}{p}\right)^2\right)^{p-2q-1} \\
&< \frac{5}{4}\sum_{q=1}^{\frac{p}{2}} \binom{p}{q}\binom{p}{2q+1}\left(\frac{q}{p}\right)^{4q+2} \\
&< \frac{5}{4}\sum_{q=1}^{\frac{p}{2}} u_q
\end{aligned}$$

with $u_q = \binom{p}{q}\binom{p}{2q+1}\left(\frac{q}{p}\right)^{4q+2}$.

Let us now study the sequence $(u_q)_{q\in\mathbb{N}^*}$. First,

$$\begin{aligned}
u_{q+1} &= \binom{p}{q+1}\binom{p}{2q+3}\left(\frac{q+1}{p}\right)^{4q+6} \\
&= \binom{p}{q}\frac{p-q}{q+1}\binom{p}{2q+1}\frac{(p-2q-1)(p-2q-2)}{(2q+2)(2q+3)}\left(\frac{q}{p}\right)^{4q+2}\left(\frac{q+1}{q}\right)^{4q+2}\left(\frac{q+1}{p}\right)^{4} \\
&= u_q\frac{p-q}{q+1}\frac{(p-2q-1)(p-2q-2)}{(2q+2)(2q+3)}\left(\frac{q+1}{q}\right)^{4q+2}\left(\frac{q+1}{p}\right)^{4} \\
&< u_q\frac{p-q}{q+1}\left(\frac{p-2q}{2q+2}\right)^{2}\left(\frac{q+1}{q}\right)^{4q+2}\left(\frac{q+1}{p}\right)^{4} \\
&< u_q\frac{p-q}{p}\frac{q+1}{q+1}\left(\frac{p-2q}{p}\right)^{2}\left(\frac{q+1}{2q+2}\right)^{2}\left(\frac{q+1}{q}\right)^{4q+2}\left(\frac{q+1}{p}\right) \\
&< u_q\left(1-\frac{q}{p}\right)\left(1-2\frac{q}{p}\right)^{2}\frac{1}{4}\frac{q}{p}\left(\frac{q+1}{q}\right)^{4q+3}
\end{aligned}$$

The function $x \mapsto x(1-x)(1-2x)^2$ is $\leq \frac{1}{16}$ on $[0,1]$. Therefore:

$$\frac{u_{q+1}}{u_q} < \frac{1}{64}\left(\frac{q+1}{q}\right)^{4q+3}.$$

Yet, $\lim\limits_{q\to+\infty}\left(\frac{q+1}{q}\right)^{4q+3} = e^4 < 64$. Then, there exist q_0, independent of p, such that u_q is decreasing from q_0. Furthermore,

$$\frac{u_{q+1}}{u_q} < \frac{1}{4}\frac{q}{p}\max_{q\in\mathbb{N}^*}\left(\left(\frac{q+1}{q}\right)^{4q+3}\right) = \frac{1}{4}\frac{q}{p}2^7 = 32\frac{q}{p}.$$

Thus, if $\frac{p}{q} \geq 32$, u_q is decreasing. Therefore, by choosing p large enough to fulfill $p \geq 32q_0$, then $\forall q \leq q_0$, $(u_q)_{q\in\mathbb{N}^*}$ is decreasing. Thus, for $p \geq 32q_0$ u_q is a

decreasing sequence. We can conclude

$$\begin{aligned}
\Pr(Z \geq 1) &< \frac{5}{4} \sum_{q=1}^{p/2} u_q \\
&< \frac{5}{4} \sum_{q=1}^{p/2} u_1 \\
&< \frac{5}{4} \sum_{q=1}^{p/2} \binom{p}{1}\binom{p}{3}\left(\frac{1}{p}\right)^6 \\
&< \frac{5}{4}\frac{p}{2} p \frac{p(p-1)(p-2)}{6}\left(\frac{1}{p}\right)^6 \\
&< \frac{5}{4}\frac{p}{2} p \frac{p^3}{6}\left(\frac{1}{p}\right)^6 \\
&< \frac{5}{48p} = O\left(\frac{1}{p}\right)
\end{aligned}$$

□

In general case, we rely on a numerical evaluation of this probability. For $r \geq 2$ and $m = \left\lceil \frac{n}{p} \right\rceil + 1$ (the generalization of the result of Theorem 19), this probability is very small, around 10^{-4} for $r = 2$, and decreases with n (see Figure 4). Note that the probability is even lower with r and for $r = 3$ the probability is smaller than 10^{-8}. Therefore, there exists with high probability a quasi-perfect assignment, whose maximum degree is equal to $\frac{n}{p} + 1$. This is a significant improvement from the previous $\frac{n}{p} + \Theta(\log \log n) + O(1)$ derived using balls-into-bins analysis for the greedy scheduler. Note that a maximum degree equal to $\frac{n}{p} + 1$ allows us to bound the total load imbalance by $\frac{p}{2}$.

6 Simulations Results

In this section, we perform simulations to evaluate the practical improvement of the proposed strategy based on matchings over the classical greedy scheduler. Indeed, the theoretical analysis provided in Section 5 is limited to homogeneous tasks and our aim is to assess the efficiency of proposed algorithm in practice, where tasks exhibit some heterogeneity. In this section we set the number of replication of input files to $r = 3$, the default value commonly used in MapReduce.

6.1 Makespan

In this section, we concentrate on the MinMakespan problem. We compare the four following strategies,

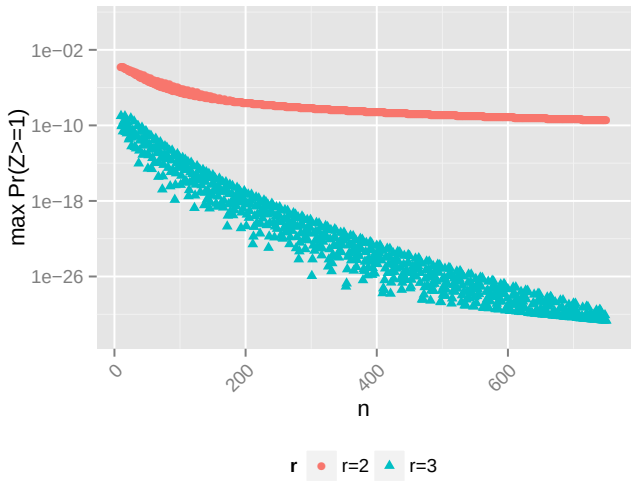


Figure 4: $\max_{p\in[1,n]} Pr(Z \geq 1)$ in function of n (with logarithm scale on y-axis).

- Greedy: the purely dynamic scheduler inspired from MapReduce: each requesting processor is allocated a local task, if any,
- Assignment: a purely static strategy where an optimal assignment dictates on which processor tasks must be processed,
- Hybrid: an hybrid strategy that mixed the previous two: processors first follows the static assignment, and then switch to the Greedy strategy if there remains local tasks to process,
- DynamicAssignment: an adapting strategy which recomputes an optimal assignment at the termination of each task (using augmenting paths). This new assignment is used to choose the next task to perform on this processor

on tasks whose processing times follow three different distributions

- Homogeneous: all tasks have the same homogeneous duration,
- Uniform-x: all tasks have a duration $y \times Mean$ where y is chosen uniformly at random in $[1-x, 1+x]$,
- Bi-Modal-x: all tasks have the same duration except 1/20 of them whose duration is x times the average duration ($x > 1$).

Note that the second distribution models a small variance in task processing time, while the third one models homogeneous tasks with stragglers, i.e., a small fraction of tasks which perform poorly.

We simulate a homogeneous platforms with $p = 25, 50, 75, 100$ processors and $n = 1, 5, 10, 20 \times p$ tasks. For each scenario, 250 simulations were performed, and results are depicted using classical box-plots. The metric of choice is the normalized makespan obtained by the different strategies, *i.e.* the sequential time divided by the number of processors.

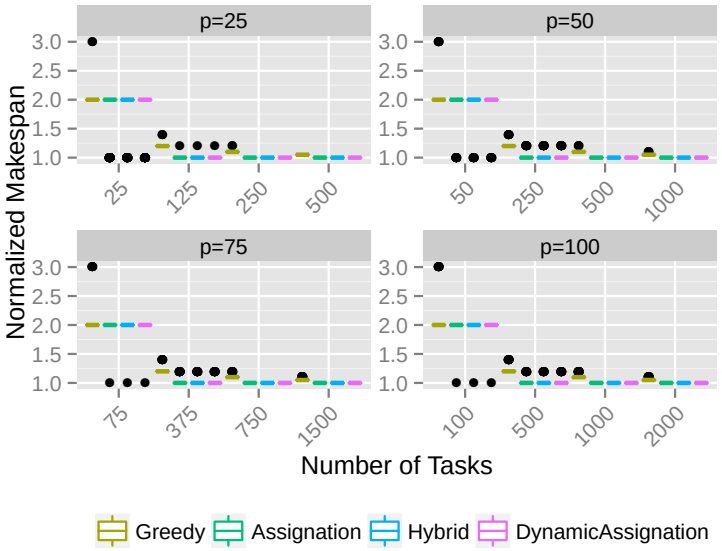
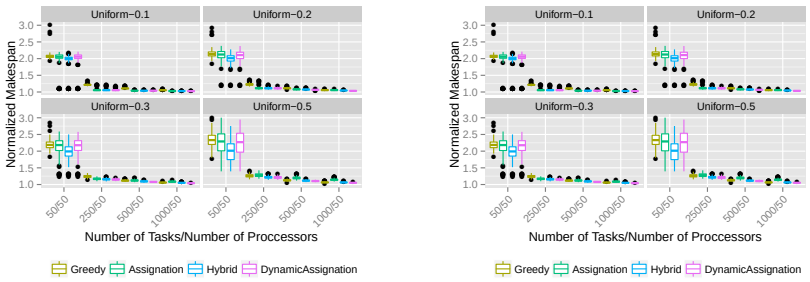


Figure 5: Normalized Makespan for tasks of type Homogeneous.

Let us first concentrate on the results for tasks from Homogeneous, depicted in Figure 5. First, variance is always very small. The results mostly depend on the value $\frac{n}{p}$, rather than of intrinsic values of p and n. In Section 5, we have shown that the makespan of the optimal matching is, with high probability, $\leq \frac{n}{p} + 1$. Therefore, when $\frac{n}{p}$ increases, the relative the quality of the normalized makespan of matching-based strategies increase and is near optimal. We can also observe that Greedy has a significant overhead when $n > p$.

For Uniform and Bi-Modal distributions, results are depicted on Figures 6(a) and 6(b). For Uniform distributions, Hybrid and DynamicAssignment (especially when the number of tasks increase) are the most efficient strategies. For Bi-Modal distribution, Greedy, Hybrid and DynamicAssignment achieve similar results for increasing value of $\frac{n}{p}$, but for small values, Hybrid is slightly better. In this case, Assignment suffers from a purely static approach, as the optimal assignment does not lead to good load-balancing anymore. Thus, as soon as there is some noticeable variance in the durations of the tasks, we cannot rely only on the static allocation given by Assignment anymore, and the use of dynamic strategies as Hybrid is necessary.



(a) Normalized Makespan for tasks of type Uniform.

(b) Normalized Makespan for tasks of type Bi-Modal.

Figure 6: Normalized Makespan for tasks of type Uniform and Bi-Modal.

6.2 Communications

Let us now focus on the MinComm problem, and thus allow processor to process non-local tasks so as to reach perfect load balancing, at the cost of some communications. Above Greedy, Hybrid and DynamicAssignment strategies are adapted as follows: the makespan-minimizing version is first used, and if a processor is idle and has no more local tasks to process, it then randomly steals a non-local tasks from the most loaded processor. In this section, we compare the fractions of non-local tasks of the different strategies at the end of the computation.

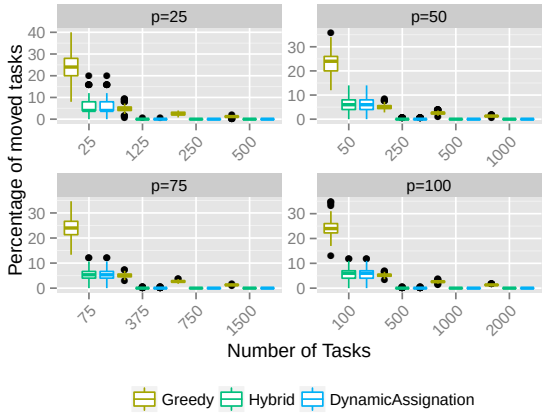
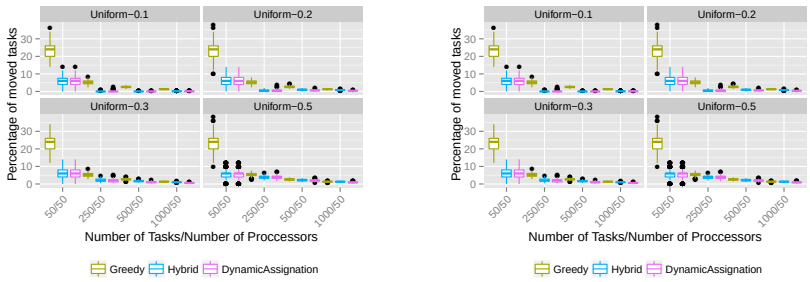


Figure 7: Percentage of non local tasks for tasks of type Homogeneous.

Results for the Homogeneous (respectively Uniform and Bi-Modal) distributions are presented in Figure 7 (respectively 8(a) and 8(b)). Note that if



(a) Percentage of moved tasks for tasks of type Uniform.

(b) Percentage of moved tasks for tasks of type Bi-Modal.

Figure 8: Percentage of moved tasks for tasks of type Uniform and Bi-Modal.

there are small differences, the ranking of the strategies is the same as with the MinMakespan metric. This corroborates the link between maximum degree and total load imbalance presented in Section 4. Note also that, like for the makespan results, the value $\frac{n}{p}$ seems more important than the intrinsic value of n and p (see Figure 7, similar observations have been made for the others distributions).

Therefore, according to these simulations with random task durations, it appears that matching-based strategies are more efficient than purely dynamic greedy strategies, even with non-homogeneous distributions. However, if Hybrid has only a (reasonable) pre-computation overhead compared to Greedy, the situation is different for DynamicAssignment, whose execution overhead is proportional to the number of edges (the cost of searching an augmenting path) and therefore to the number of tasks. Therefore DynamicAssignment cannot to be used for large number of tasks and for small number of tasks, it does not outperform Hybrid.

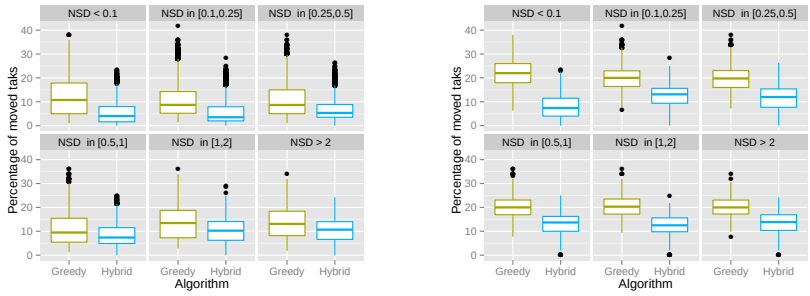
6.3 Simulations on Traces

In this section, we use traces recording jobs during 10 months on a Hadoop production cluster [27]. We selected in these traces all jobs that contain only Map tasks (and more than 10 of them), and computed the duration of all their tasks. These task distributions were used as an input for our simulation, using 50 different random task orders. Different task orders were used to minimize the impact of significantly longer tasks and have a fair comparison between the different strategies.

In order to minimize the computation time of the simulations we only run Greedy and Hybrid (Assignment gives strictly worse result than Hybrid and the computational time induced by DynamicAssignment is too large). We consider a platform of 50 processors and consider the results when $n \geq p$. We also limit the number of tasks to 10^4. Note that on the considered platform, the difference

in results between a job of 10000 tasks and a job of 50000 tasks is negligible. We then classify the different jobs depending on their number of tasks and their normalized standard deviation (the standard deviation divided by the mean) of the duration of the tasks.

We mainly focus on communication metric. We can make two main observations. First, for relatively small numbers of tasks (less than 500, *i.e.* 10 times the number of processors), see Figure 9(a), the results of Hybrid are better than those of Greedy. The difference between the strategies decreases when normalized standard deviation increases but can be up to 5%. Hybrid performs much better when the number of tasks is small and the difference is about 15% for jobs with less than 100 tasks, see Figure 9(b).



(a) Percentage of non local tasks for jobs with less than 500 tasks

(b) Percentage of non local tasks for jobs with less than 100 tasks

Figure 9: Percentage of non local tasks according to several values of normalized standard deviation (NSD)

Figure 10 reports the results for large number of tasks. Hybrid performs better than Greedy for small standard deviations (more specifically, the results of Hybrid are more stable than the ones of Greedy). When the variance increases, Greedy becomes slightly more efficient. Note that, except for huge standard deviations, both algorithms induce less than 5% of non-local tasks.

To conclude this section, we can observe that Hybrid is the most efficient for small number of tasks, even when the standard deviation between the duration of tasks is important. However, the difference decreases when n increases (and p is kept constant, what correspond to an easier case) an Hybrid can even be slightly outperformed by Greedy if, in addition, the variance becomes important. For for the makespan metric, our simulations show that when the standard deviation is large (larger than 0.5 times the mean) Hybrid and Greedy exhibit the same performance. For smaller standard deviations, Hybrid is slightly better, in particular when the number of tasks is close to the number of processors (as in the case of communications, the difference between them decreases when the number of tasks increases since the problem becomes easier), see Figure 11.

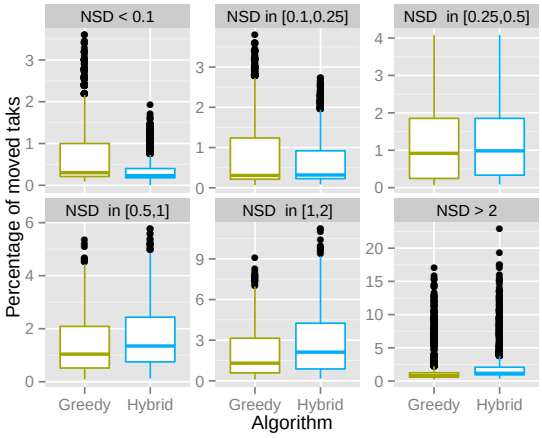


Figure 10: Percentage of moved tasks for jobs with more than 500 tasks according to several value of normalized standard deviation (NSD)

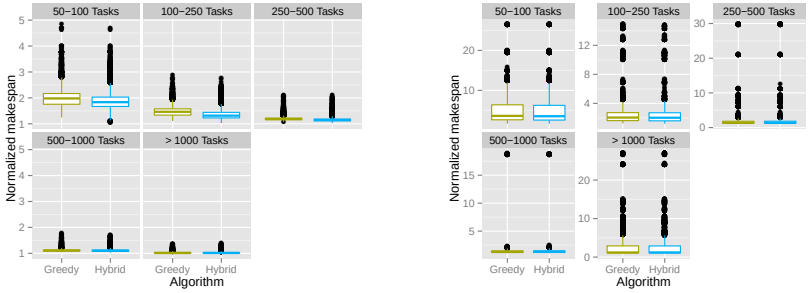
7 Conclusion

In this paper we consider the problem of data locality for map tasks in the framework MapReduce. To do this, we do not modify the replication mechanism provided by the distributed filesystem, but concentrate on the runtime strategies that aim to minimize the number of transferred data chunks. To model this, we introduce two metrics, MinComm and MinMakespan. We analyze the performance of the greedy scheduler used in Hadoop with respect to optimal strategies, based on the computation of matchings. We provide a detailed theoretical analysis of the expected performance of both algorithm, where randomness comes from the initial randomized algorithm used to distribute the chunks. We assert the efficiency of the matching-based approach by showing that it achieves quasi optimal performance ($\frac{n}{p} + 1$ where the lower bound is $\frac{n}{p}$) with high probability, with a mathematical proof in the case $n = p$ and with numerical analysis in the other cases. In addition, we provide simulation results both of synthetic distributions and on actual Hadoop traces and we advocate the use of an hybrid version. Perspectives of this work are two-fold. On the theoretical side, the extension of the proof to more general settings and the analysis for given distributions of task durations are of interest. On the practical side, the design of more robust allocation strategies, in particular in presence of stragglers, is needed.

References

[1] T. White, *Hadoop: The definitive guide.* " O'Reilly Media, Inc.", 2012.

[2] Z. Guo, G. Fox, and M. Zhou, "Investigation of data locality in mapreduce," in *12th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing, CCGrid*



(a) Normalized Makespan for jobs with normalized standard deviation inferior to 0.5

(b) Normalized Makespan for jobs with normalized standard deviation superior to 0.5

Figure 11: Normalized Makespan according to several categories of number of tasks

2012, Ottawa, Canada, May 13-16, 2012, 2012, pp. 419–426.

[3] D. Borthakur, "Hdfs architecture guide," *HADOOP http://hadoop. apache. org/common/docs/current/hdfs design. pdf*, p. 39, 2008.

[4] S. Ibrahim, H. Jin, L. Lu, B. He, G. Antoniu, and S. Wu, "Maestro: Replica-aware map scheduling for mapreduce," in *12th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid)*, 2012, pp. 435–442.

[5] 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 *Proceedings of the 5th European Conference on Computer Systems (EuroSys)*. ACM, 2010, pp. 265–278.

[6] Q. Xie and Y. Lu, "Degree-guided map-reduce task assignment with data locality constraint," in *Proceedings of the 2012 IEEE International Symposium on Information Theory (ISIT)*, 2012, pp. 985–989.

[7] W. Wang, K. Zhu, L. Ying, J. Tan, and L. Zhang, "Map task scheduling in mapreduce with data locality: Throughput and heavy-traffic optimality," in *Proceedings of the IEEE INFOCOM*, 2013, pp. 1609–1617.

[8] Z. Guo, G. C. Fox, and M. Zhou, "Investigation of data locality in mapreduce," in *12th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid)*, 2012, pp. 419–426.

[9] M. Hammoud and M. F. Sakr, "Locality-aware reduce task scheduling for mapreduce," in *IEEE 3rd International Conference on Cloud Computing Technology and Science (CloudCom 2011)*, 2011, pp. 570–576.

[10] J. Tan, S. Meng, X. Meng, and L. Zhang, "Improving reducetask data locality for sequential mapreduce jobs," in *Proceedings of the IEEE INFOCOM*, 2013, pp. 1627–1635.

[11] R. E. Burkard, M. Dell'Amico, and S. Martello, *Assignment Problems, Revised Reprint*. Siam, 2009.

[12] P. Erdös and A. Rényi, "On random matrices," *Studia Sci. Math. Hungar.*, vol. 8, pp. 455–461, 1964.

[13] D. W. Walkup, "Matchings in random regular bipartite digraphs," *Discrete Mathematics*, vol. 31, no. 1, pp. 59–64, 1980.

[14] L. R. Ford Jr and D. R. Fulkerson, *Flows in networks*. Princeton university press, 2015.

[15] J. E. Hopcroft and R. M. Karp, "An $n^{5/2}$ algorithm for maximum matchings in bipartite graphs," *SIAM Journal on Computing*, vol. 2, no. 4, pp. 225–231, 1973.

[16] A. Goel, M. Kapralov, and S. Khanna, "Perfect matchings in $O(n \log n)$ time in regular bipartite graphs," *SIAM Journal on Computing*, vol. 42, no. 3, pp. 1392–1404, 2013.

[17] J. Langguth, F. Manne, and P. Sanders, "Heuristic initialization for bipartite matching problems," *J. Exp. Algorithmics*, vol. 15, pp. 1.3:1.1–1.3:1.22, Mar. 2010.

[18] F. Dufossé, K. Kaya, and B. Uçar, "Two approximation algorithms for bipartite matching on multicore architectures," *Journal of Parallel and Distributed Computing*, vol. 85, pp. 62 – 78, 2015.

[19] M. Raab and A. Steger, "Balls into bins: A simple and tight analysis," in *Randomization and Approximation Techniques in Computer Science*. Springer, 1998, pp. 159–170.

[20] P. Berenbrink, T. Friedetzky, Z. Hu, and R. Martin, "On weighted balls-into-bins games," *Theoretical Computer Science*, vol. 409, no. 3, pp. 511–520, 2008.

[21] M. Mitzenmacher, "The power of two choices in randomized load balancing," *Parallel and Distributed Systems, IEEE Transactions on*, vol. 12, no. 10, pp. 1094–1104, 2001.

[22] 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.

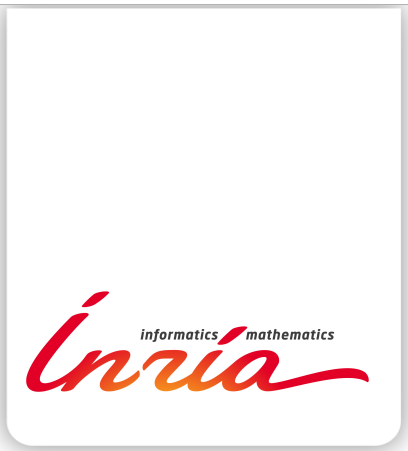
[23] P. Berenbrink, A. Czumaj, A. Steger, and B. Vöcking, "Balanced allocations: the heavily loaded case," in *Proceedings of the 22nd annual ACM symposium on Theory of computing*. ACM, 2000, pp. 745–754.

[24] Y. Peres, K. Talwar, and U. Wieder, "The $(1+\beta)$-choice process and weighted balls-into-bins," in *Proceedings of the twenty-first annual ACM-SIAM symposium on Discrete Algorithms*. Society for Industrial and Applied Mathematics, 2010, pp. 1613–1619.

[25] C. Berge, "Two theorems in graph theory," *Proceedings of the National Academy of Sciences*, vol. 43, no. 9, pp. 842–844, 1957.

[26] P. Hall, "On representatives of subsets," *Journal of the London Mathematical Society*, vol. s1-10, no. 1, pp. 26–30, 1935.

[27] S. Kavulya, J. Tan, R. Gandhi, and P. Narasimhan, "An analysis of traces from a production mapreduce cluster," in *10th IEEE/ACM International Conference on Cluster, Cloud and Grid Computing*, 2010, pp. 94–103.



RESEARCH CENTRE
GRENOBLE – RHÔNE-ALPES

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

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

ISSN 0249-6399