Messages Scheduling for Parallel Data Redistribution between Clusters

Johanne Cohen, Emmanuel Jeannot, Nicolas Padoy, and Frédéric Wagner

Abstract—We study the problem of redistributing data between clusters interconnected by a backbone. We suppose that at most k communications can be performed at the same time (the value of k depending on the characteristics of the platform). Given a set of messages, we aim at minimizing the total communication time assuming that communications can be preempted and that preemption comes with an extra cost. Our problem, called k-$Preemptive$ *Bipartite Scheduling (KPBS)* is proven to be NP-hard. We study its lower bound. We propose two $\frac{8}{3}$-approximation algorithms with low complexity and fast heuristics. Simulation results show that both algorithms perform very well compared to the optimal solution and to the heuristics. Experimental results, based on an MPI implementation of these algorithms, show that both algorithms outperform a brute-force TCP-based solution, where no scheduling of the messages is performed.

Index Terms—Message scheduling, data redistribution, grid computing, approximation algorithm, code coupling.

1 Introduction

In recent years, the emergence of cluster computing, coupled with fast wide area networks has allowed the apparition of grid computing, enabling parallel algorithms to take advantage of various distant resources, be it computing power, software, or data. However, the classical problem of minimizing the communications/computations ratio remains and is even more difficult since communication times increase on slower networks. It is therefore important to try to minimize communication times. In this work, we focus on the scheduling of the messages when a parallel data redistribution has to be realized on a network, called a backbone. Two parallel machines are involved in the redistribution: the one that holds the data and the one that will receive the data. If the parallel redistribution pattern involves a lot of data transfers, the backbone can become a bottleneck. Thus, in order to minimize the parallel data redistribution time and to avoid the overloading of the backbone it is required to schedule each data transfer.

The message scheduling problem appears in the context of data redistribution but also in the context of packet switching for wavelength-division multiplexed (WDM) optical network [7], [13], [23], [26], [28] or for satellite-switched time division multiple access (SS/TDMA) [4], [15], [16]. The solution proposed in this paper works for these two cases.

Data redistribution has mainly been studied in the framework of high-performance parallel computing [1], [8], [10]. In this paper, we study a generalization of the parallel data redistribution. Indeed, contrary to some previous works that only deal with block-cyclic redistribution [3], [10], [24], here no assumption is made on the redistribution pattern. Moreover, some workk [1], [8] assume that there is no bottleneck. In this paper, it is not the case and we suppose that the ratio between the throughput of the backbone and the throughput of each of the n nodes of the parallel machines is k. Hence, at most k communications can occur at the same time. We study the problem for all values of k. We focus on the case $k < n$ (the backbone is a bottleneck) whereas the case $k \geq n$ has been tackled in [1], [8].

Redistributing data between clusters has recently received considerable attention as it occurs in many application frameworks. Here, we provide three examples of such frameworks taken from distributed code-coupling, parallel task execution and persistence, and redistribution for metacomputing:

1. Distributed code coupling. Code coupling applications [21] are composed of several codes that interact with each other. They are used for simulating complex systems. Such a system is composed of several models, each model being simulated by one parallel code or component [30]. Moreover, in distributed code coupling, each code/component is running on a different parallel machine or cluster [2], [25]. For instance, the hydrogrid project [20] aims at modeling and simulating fluid and solute transport in subsurface geological media. In this application, two parallel codes are coupled: one for flow simulation and one for transport simulation. For performance reasons, each parallel code requires a very large cluster to execute on: they cannot execute on the same cluster. During the simulation the models interact with each other and, therefore, the parallel codes need to exchange data. Hence, this exchange of data is a redistribution between distant clusters.
2. Parallel task execution. Recent works in the field of mixed parallelism [31], [27], [5] have shown the

- *J. Cohen and E. Jeannot are with Loria-INRIA-Lorraine, Campus Scientifique, BP 239, 54506 Vendoeuvre les Nancy, France. E-mail: {Johanne.Cohen, emmanuel.Jeannot}@loria.fr.*
- *N. Padoy is with Technische Universität München, Germany. E-mail: padoy@in.tum.de.*
- *F. Wagner is with LIA—Laboratoire d'informatique d'avignon, 339, chemin des Meinajaries, Agroparc BO 1228, 84911 AVIGNON Cedex 9, France. E-mail: wagnerf@loria.fr.*

Manuscript received 22 July 2004; revised 9 May 2005; accepted 31 Oct. 2005; published online 24 Aug. 2006.
Recommended for acceptance by R. Eigenmann.
For information on obtaining reprints of this article, please send e-mail to: tpds@computer.org, and reference IEEECS Log Number TPDS-0178-0704.

potential of executing data parallel tasks concurrently on different clusters. In some cases, these tasks need to communicate with each other and data has to be redistributed between each cluster that host the task.

3. Persistence and redistribution in GridRPC systems. Several metacomputing environments implement the client-agent-server model [6], such as Netsolve or Ninf [18]. In this model, the agent has to map a client request to a server; the request is then being processed in an RPC way. One of the drawbacks of this approach is that data are sent back to the client at the end of every computation. This implies unnecessary communications when computed data are needed by an other server in further computations. Some recent enhancements of this model deal with data management and allow data to be persistent on the server [9], [11]. Because the service can be parallel, the data can be distributed among the node of the cluster when this data is required for further computation on a distant server. In this case, a parallel data redistribution occurs between distant clusters.

The contribution of this paper is the following: We prove that the problem of scheduling any parallel data redistribution pattern is NP-hard for any value of $k(k < n)$. We exhibit a lower bound for the number of steps of the redistribution and a lower bound for the sum of the durations of each step. Next, we propose two algorithms (called GGP and OGGP) that have a $\frac{8}{3}$-approximation bound. On the other hand, we study simple and fast heuristics that achieve a good average performance. Simulation results show that both GGP and OGGP outperform the heuristics and are close to the optimal solution. Moreover, we have implemented these algorithms and tested them on real examples using MPI. Results show that we outperform a TCP-based brute-force solution that consists of letting the transport layer do the scheduling and managing the congestion alone.

2 Description of the Problem

2.1 Modelization of the Problem

We consider the following heterogeneous architecture made of two clusters of workstations $\mathcal{C}_1$ and $\mathcal{C}_2$ connected together by a backbone of throughput D. Let n_1 be the number of nodes of $\mathcal{C}_1$ and n_2 be the number of nodes of $\mathcal{C}_2$. All the nodes of the first cluster have a throughput d_1 and the nodes of the second have a throughput d_2.

We assume that any node of $\mathcal{C}_1$ can communicate to any node of $\mathcal{C}_2$. This requires that each node has its own address (as in GRID'5000[1] or for the icluster project[2]) or, if the nodes are behind a NAT, that the router/front-end implements a specific port forwarding mechanism to each node. The bottleneck that might come from such a mechanism can easily be captured by our model.

Let us consider a parallel application that must execute the first part of its computations on $\mathcal{C}_1$ and the second part on $\mathcal{C}_2$. This is the case where an application is made of several parallel components, data parallel tasks, or requests with dependencies. During the execution of the application, parallel data must be redistributed from the first cluster to the second one.

We assume that the communication pattern of the redistribution is computed by the application. We focus on efficiently transmitting the data, not on computing the pattern itself. For computing the pattern in the case of block-cyclic redistribution, see [17]. This pattern is modeled by a *traffic matrix* $T = (t_{i,j})_{1 \leq i \leq n_1, 1 \leq j \leq n_2}$, where $t_{i,j}$ represents the amount of information that must be exchanged between node i of cluster $\mathcal{C}_1$ and node j of cluster $\mathcal{C}_2$.

For a given traffic pattern and for a particular architecture, our goal is to minimize the total transmission time. In order to perform the redistribution, one naive solution consists of sending all the data from all the nodes of $\mathcal{C}_1$ to all the nodes of $\mathcal{C}_2$ at the same time and let the transport layer (for instance, TCP) schedule the segments. This solution, as we will show in the results section, is suboptimal for many reasons. If the traffic matrix is very large and dense with high coefficient, a lot of traffic is generated at the same time. This traffic cannot be handled either by the backbone (when the aggregated bandwidth of the emitting card is greater than the bandwidth of the backbone) or by the cards themselves (when the incoming traffic has a throughput greater than the throughput of a given card). In both cases, TCP segments will be dropped. TCP will detect the problem and start to control the congestion by reducing the window size and, therefore, reduce the amount of data sent at a given time. To avoid these problems, we use the knowledge we have (i.e., the traffic matrix) to perform optimizations at the application level and control by ourselves the congestion by defining a schedule for all the communications.

We consider two constraints relative to the communications:

1. **The 1-port constraint**. A transmitter (respectively, a receiver) cannot perform more than one communication at a given moment. However, more than one communication can occur at the same time as long as the receiver/transmitter pair is different. A parallel transmission of messages between different pairs is called a *step*.
2. **The k constraint**. The maximum number of communications that can occur during a step is denoted by k. This number depends mainly on the ratio D/d_1 and D/d_2. It comes from the fact that no congestion occurs when the aggregated bandwidth generated by cluster $\mathcal{C}_1$ or received by $\mathcal{C}_2$ is not larger than the bandwidth D of the backbone. Therefore, k must respect the following equations:

   a. $kd_1 \leq D$,
   b. $kd_2 \leq D$,
   c. $k \leq n_1$, and
   d. $k \leq n_2$.

We denote by d the speed of each communication $d = \min(d_1, d_2, D)$. For instance, let us assume that $n_1 = 200$, $n_2 = 100$, $d_1 = 10\text{Mbit/s}$, $d_2 = 100\text{Mbit/s}$, and $D = 1\text{GBbit/s}$ ($D = 1,000\text{Mbit/s}$). In that case, $k = 100$ because $\mathcal{C}_1$ can send 100 outgoing communications at 10 Mbit/s generating a total of 1 Gbit/s aggregated bandwidth and each network card of $\mathcal{C}_2$ can receive the data at $d = 10$ Mbit/s.

1. www.grid5000.org.
2. icluster.imag.fr.

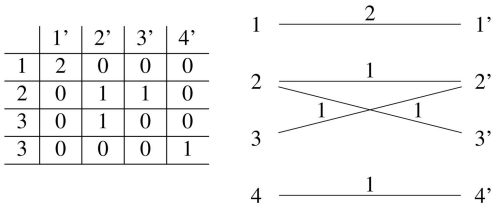

Fig. 1. Correspondance between matrix $\mathcal{M}$ and a bipartite graph.

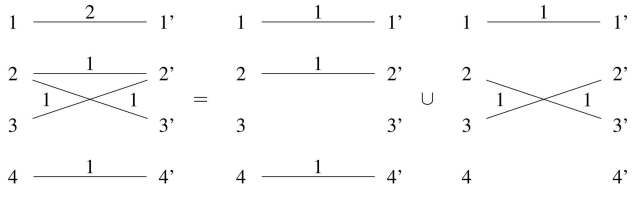

Fig. 2. An example for KPBS problem with the graph of Fig. 1 and $k = 3$. The solution contains two steps and the useful transmission cost is equal to 2 $(= 1 + 1)$. The cost of the solution is $2 + 2\beta$. Thanks to preemption, the edge of cost 2 is decomposed into two steps.

A common approach to minimize the overall transmission time is to enable preemption, i.e., the possibility to interrupt the transmission of a message and to complete it later. In practice, this involves a nonnegligible cost, called *startup delay* and denoted here by β, which is the time necessary to start a new *step*.

2.2 Formulation of the Problem

Let T be a traffic matrix, k be the maximum number of communications at each step, β be the startup delay, and d be the speed of each communication.

We can normalize the problem by d: The traffic matrix T, can be replaced by the matrix $\mathcal{M} = (m_{i,j}) = (\frac{t_{i,j}}{d})_{1 \leq i \leq n_1, 1 \leq j \leq n_2}$ that represents the communication time.

Before describing the problem formally, we need to introduce some terminology on graphs. Let $G = (V_1, V_2, E)$ be a bipartite graph with vertex set $V_1 \cup V_2$ and edges set $E \subseteq V_1 \times V_2$: If $(i, j) \in E$, then $i \in V_1$ and $j \in V_2$. A set $M \subseteq E$ is called a *matching* if no vertex $v \in V_1 \cup V_2$ is incident with more than one edge in M. A matching is called *perfect* if all vertices of $V_1 \cup V_2$ are incident to exactly one edge in M. A *weighted matching* (M, w) is a matching associated to a function that gives the weight of all its edges: $w : M \rightarrow \mathbf{Q}^+$.

The matrix $\mathcal{M}$ is equivalent to a *weighted bipartite graph* $G = (V_1, V_2, E, w)$ (see Fig. 1) where $w : E \rightarrow \mathbf{Q}^+, \forall e = (i, j) \in E$, and $w(e) = m_{i,j}$. Each node of cluster $\mathcal{C}_1$ (respectively, $\mathcal{C}_2$) is represented by a node of V_1 (respectively, V_2). Hence, $|V_1| = n_1$ and $|V_2| = n_2$.

Given a weighted bipartite graph $G = (V_1, V_2, E, w)$ that represents the communication to execute with their time, we tackle the problem of scheduling these communications. The solution must describe when a node of V_1 must communicate to a node of V_2 and for how long. The solution will be composed of steps. Each step needs to follow the constraints presented in the previous section:

- The 1-port model avoids communication contention: During one step, a given node cannot communicate with more than one node. Since an edge of graph G represents an communication between two nodes, a step will be modeled by a matching of G.
- Moreover, the backbone is a bottleneck: At most, k communications can occur during one step. Hence, a communication step will be represented by a matching with at most k edges.

We recall that the preemption is allowed: A communication between two given nodes might be stopped and resumed later. Therefore, a given edge of G might occur in several matchings, each representing a different communication step. In order to describe which part of the whole communication is done during a given step, we add weight function to each the matching. These weights refer to the amount of exchanged data. Hence, an edge of G can be decomposed and be present into several matchings provided that the sum of all the weights of this edge in these matchings is not smaller than the weight of the original edge in G.

We denote the matching and its weighted function corresponding to a communication step by a *valid weighted matching* (for the remaining, a valid weighted matching contains at most k edges).

In order to execute a schedule, we execute each step one after the other. Step i corresponds to a valid weighted matching (M_i, w_i) and for each edge $e = (u, v) \in M_i$ we send data between node u of cluster $\mathcal{C}_1$ to node v of cluster $\mathcal{C}_2$ for a time equal to $w_i(e)$ (or $w_i(e) \times d$ amount of data). We execute step $i + 1$ when all communications of step i are done. The duration of step i is therefore $\beta + W_i$ where β is the startup cost of a communication and $W_i = \max_{e \in M_i} w_i(e)$. Fig. 2 gives an example of a valid schedule.

We call this optimization problem *k-Preemptive Bipartite Scheduling (KPBS)*, formally defined as follows:

Given a weighted bipartite graph $G = (V_1, V_2, E, w)$, where $w : E \rightarrow \mathbf{Q}^+$, an integer[3] $k \geq 2$ and a rational β, find a collection $\{(M_1, w_1), (M_2, w_2), \ldots, (M_s, w_s)\}$ of valid weighted matchings such that:

1. The matchings are a decomposition of E: $\cup_{i=1}^{s} M_i = E$.
2. All the functions w_i, $1 \leq i \leq s$, must respect the following inequality: $\forall e \in E, \sum_{i \in \{j | e \in M_j\}} w_i(e) \geq w(e)$.
3. Any matching M_i contains at most k edges $(|M_i| \leq k, i \in [1, s])$ and its cost is equal to the rational number $\beta + W_i$, where $W_i = \max_{e \in M_i} w_i(e)$.
4. $\left(\sum_{i=1}^{s} (\beta + W_i)\right)$ is minimized.

2.2.1 Case $\beta = 1$

It is important to see that solving the case $\beta = 1$ is sufficient to consider. Indeed, if $\beta \neq 1$, one can divide (normalize) all the edges weights by β, then solve the problem assuming $\beta = 1$ and multiply the edges weight of the matchings of the solution by the original value of β. Therefore, in the remainder of this paper, we will consider that $\beta = 1$. β will not appears in the NP-completeness proof or as a parameters of the algorithms (Sections 4 and 6).

In the remainder of this paper, we use the following notation: For any solution S of $KPBS$, the cost of S is $\alpha + s$ (β is considered equals to 1), where s is the *number of steps* and α is the *useful transmission cost*.

3. The case $k = 1$ is not interesting because the backbone is saturated by one communication.

3 Related Work

This problem has been partially studied in the context of Satellite-Switched Time-Division Multiple access systems (SS/TDMA) [4], [15], [16]. In [4], the problem with $\beta = 0$ is studied and an optimal algorithm with $O(mn)$ steps is described. In [15], an optimal algorithm that finds the minimal number of steps is described. In [16], the problem without preemption is studied. It is shown NP-hard and a heuristic is given.

The KPBS problem partially falls in a field originated by packet switching in communication systems for optical network called wavelength-division multiplexed (WDM) broadcast network [7], [13], [23], [26], [28]. The problem of minimizing the number of steps is studied in [13], [15], and the problem of minimizing the total cost is studied in [23]. In [7] and [26], the authors consider a version of the KPBS problem where the number of receivers is equal to the number of messages that can be transmitted at the same time ($k = n_2$) and where the setup delay can be overlapped by the communication time (in [26], authors also assume that all messages have the same size). In that case, a list-scheduling algorithm is proven to be a 2-approximation algorithm [7]. The case where the backbone is not a constraint ($k \geq \min(n_1, n_2)$) has been studied in [1], [8] and it is known as the *preemptive bipartite scheduling (PBS)*. PBS was proven to be NP-hard in [12], [16]. Approximating the PBS problem within a ratio number smaller than $\frac{7}{6}$ has been proven impossible unless $P = \mathrm{NP}$ [8]. Several approximation algorithms for the PBS problem have been proposed in the literature. In [8], two different polynomial time 2-approximation algorithms for PBS have been proposed and in [1], an improvement of this result is given.

In [19], the problem of mapping the data to the processors for minimizing the communications is studied in the context of local block cyclic redistributions. It aims at minimizing the amount of data to transfer and not the communication time.

In the context of block cyclic redistribution, many works exist (see [3], [10], [24], for example). In this case, the backbone is not a constraint and the redistribution pattern is not arbitrary. Hence, all these problems are less general than KPBS.

4 Complexity Results

This problem has already been proven NP-hard for the particular case where $k \geq \min(n_1, n_2)$ [12], [16]. We prove that it remains NP-hard for any fixed $k \geq 2$ (with a different reduction than in [12], [16]). The *decision* problem of KPBS (D-KPBS) is defined as follows:

Instance: A weighted bipartite graph $G = (V_1, V_2, E, w)$ where $w : E \to \mathbf{Q}^+$, an integer k, a rational number B.

Question: Is there a collection $\{(M_1, w_1), (M_2, w_2), \ldots, (M_s, w_s)\}$ of valid weighted matchings such that $E = \cup_{i=1}^{s} M_i$ and $\sum_{i=1}^{s} W_i + s \leq B$ with for any $e \in E$, $\sum_{i \in \{j | e \in M_j\}} w_i(e) \geq w(e)$?

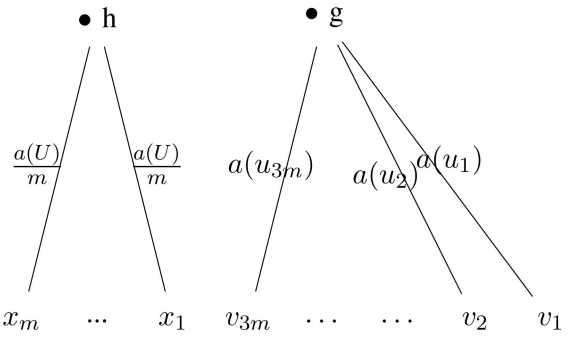

Fig. 3. An example of graph G from an instance (U, s) of Partition Problem.

Theorem 1. *Let $k \geq 2$ be a fixed integer. D-KPBS is NP-complete in the strong sense.*

Proof of Theorem 1. It is easy to see that D-KPBS belongs to NP. We show that it can be reduced to the *3-Partition* problem [14] defined as follows:

Instance: A finite set $U = \{u_1, u_2, , u_{3m}\}$ and a size $a(u) \in \mathbf{Z}^+$ for each $u \in U$.

Question: Can U be partitioned into m disjoint sets $U_1, , U_m$ such that: For $1 \leq i \leq m$, $\sum_{u \in U_i} a(u) = \frac{1}{m} \sum_{u \in U} a(u)$?

Let $a(U) = \sum_{i=1}^{3m} a(u_i)$. We transform 3-partition to D-KPBS: Let $U = \{u_1, u_2, \ldots, u_{3m}\}$ be a finite set and a size $a(u) \in \mathbf{Z}^+$ for each $u \in U$ in an arbitrary instance of *3-Partition* problem. Now, an instance of D-KPBS are constructed from set U and function a. We set $B = a(U) + 3m$ and $k = 2$. We consider the following weighted bipartite graph $G = (V_1, V_2, E, w)$:

- $V_1 = \{v_1, v_2, \ldots, v_{3m}, x_1, \ldots, x_m\}$ and $V_2 = \{g, h\}$.
- $E = \{(x_i, h) : 1 \leq i \leq m\} \cup \{(v_i, g) : 1 \leq i \leq 3m\}$.
- $w(x_i, h) = a(U)/m$ for $1 \leq i \leq m$.
- $w(v_i, g) = a(u_i)$ for $1 \leq i \leq 3m$.

Fig. 3 gives an example of this transformation. G can clearly be constructed in polynomial time. We claim that the instance of D-KPBS admits a solution if and only if the instance of *3-Partition* has a desired partition.

($\Rightarrow$) Let $\{U_1, \ldots, U_m\}$ be a solution of the *3-Partition* instance. Then, a collection $\{(M_i, w_i) : 1 \leq i \leq 3m\}$ of valid weighted matchings is defined as follows: for $1 \leq i \leq 3m$, $M_i = \{(v_i, g)(x_j, h)\}$, $w_i((v_i, g)) = a(u_i)$, and $w_i((x_j, g)) = a(u_i)$, where $u_i \in U_j$. This collection is a solution of D-KPBS. Indeed the sum of these matchings is $a(U)$ because $\sum_{j=1}^{m} \sum_{u_i \in U_j} a(u_i) = a(U)$. The cost of this solution is exactly B.

($\Leftarrow$) Conversely, we suppose that the instance of D-KPBS admits a solution. Then, the useful transmission cost is at least equal to $a(U)$ because of the edges incident to vertex h. There are at least $3m$ steps because vertex g has $3m$ neighbors. Since the cost is lower than or equal to B, both previous inequalities are equalities. Therefore, for $1 \leq i \leq 3m$, no edge incident to v_i can be split and the solution of the instance of D-KPBS is composed of $3m$ valid matchings. Thus, the solution having the desired properties can be written $(M_i)_{1 \leq i \leq 3m}$. Now, we

will determine weight functions $(w_i)_{1\leq i\leq 3m}$. Since the size of a matching is at most $2(=k)$, for $1 \leq i \leq 3m$, all valid matchings C_i of the solution contains only one edge incident to g and to u_j (w.l.g. we assume that $j = i$) and, thus, $w_i((v_j, g)) = a(u_i)$. Necessarily, M_i contains an edge incident to one vertex belonging to $\{x_1, \ldots, x_m\}$, having the weight $a(u_i)$ (the same weight as the other edge of the matching).

For $1 \leq j \leq m$, let U_j be the set of the u_i such that M_i contains an edge adjacent to x_j. Then, for $1 \leq j \leq m$, we have $\sum_{u_i \in U_j} a(u_i) = \sum_{u_i \in U_j} w_i((v_i, g)) = \frac{a(U)}{m}$. Hence, sets $U_1, \ldots, U_m$ (such that for $1 \leq j \leq m$, $\sum_{u \in U_j} a(u) = \frac{1}{m} a(u)$) is a partition of U.

Since for any k fixed, we have the same proof, Theorem 1 is proven. □

Since the problem decision problem D-KPBS is NP-complete, the optimization problem KPBS is NP-hard. The remainder of this paper is devoted to find approximation algorithms and to experiment them.

5 Lower Bounds

Before giving a lower bound for the optimal solution, we give some graph notations. We define the weight $w(v)$ of a node v of G to be the sum of weights of all edges incident to vertex v. We denote the maximum of $w(v)$ over all vertices by $W(G)$. Let $P(G)$ be the sum of the weights of all edges of graph G. We denote the maximum degree of the bipartite graph G by $\Delta(G)$, its number of edges by $m(G)$ and its number of vertices by $n(G)$. For example, in Fig. 1, $W(G) = 2$, $P(G) = 6$ and $\Delta(G) = 2$.

Proposition 1. *Let $G = (V_1, V_2, E, w)$ be a weighted bipartite graph. Let k be an integer, β a rational. The cost of the optimal solution for the instance $\langle G, k, \beta \rangle$ of KPBS is at least $\eta(G) = \eta_d(G) + \beta\eta_s(G)$ with*

$$\eta_d(G) = \max\left(W(G), \frac{P(G)}{k}\right),$$
$$\eta_s(G) = \max\left(\Delta(G), \left\lceil \frac{m(G)}{k} \right\rceil\right).$$

Proof of Proposition 1. $\eta_s(G)$ is a lower bound for the number of steps. The first term of the maximum accounts for the fact that two edges incident to the same node cannot appear in the same step and the second term for the fact that a step contains at most k edges. $\eta_d(G)$ is a lower bound for the useful transmission cost and is obtained similarly. The total cost is therefore minimized by $\eta_d(G) + \beta\eta_s(G)$. □

6 Algorithms

In this section, we present two algorithms we propose to use for the KPBS problem. As discussed before, we consider only the case $\beta = 1$ here. We start by presenting the *GGP* (Generic Graph Peeling) which is a polynomial time $\frac{8}{3}$-approximation algorithm. This algorithm is relatively complex, hence we describe it relying on a subalgorithm called *weight-regular extension algorithm* in order to simplify the presentation.

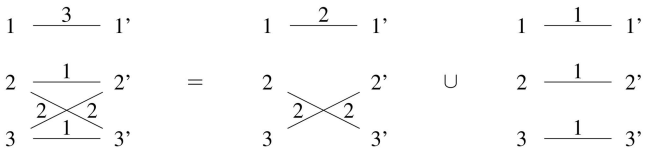

Fig. 4. Peeling a weight-regular graph leads to a weight-regular graph.

We first introduce the main ideas behind these algorithms together with a more formal description of the different steps.

We then continue this section by an analysis of the different properties of GGP. Three key properties are studied: algorithm correctness, approximation ratio, and worst-case complexity. In order to study the approximation ratio, we need to introduce another algorithm called *multigraph algorithm*. We prove that this algorithm is a pseudo polynomial $\frac{8}{3}$-approximation and that GGP can always give better results than it.

Finally, we introduce the *OGGP* algorithm (Optimized GGP) which is a direct enhancement of GGP and compute its worst-case complexity.

6.1 GGP Algorithm

6.1.1 Simple Case where G is Weight-Regular, with All Edges of Integer Weights

Solving the KPBS problem is easy in the case where there is no constraint on k (i.e., $k = n$) and the input graph G holds the following properties: G is weight-regular, with all edges of integer weights. A weight-regular graph is a graph such that for each of its nodes the sum of all weights of adjacent edges is the same (see Fig. 4).

The algorithm is based on an interesting propriety: Any weight-regular graph has a perfect matching [8]. We are therefore able to pick such a perfect matching which can be communication step. But, as all communications in a given step should to be ended simultaneously (to minimize waiting and therefore overall cost) we cut the duration of all communications (i.e., the weight of any edge in the matching) to the smallest one. By doing this, the graph left after removing the matching is still weight-regular because we removed the same amount of weight on each node (see Fig. 4). We then start again, removing another matching from the graph. In the remainder of this paper, we call *peeling* a graph this procedure of step by step removing perfect matchings from it. The algorithm ends when the graph is empty. In Fig. 4, we removed a perfect matching of weight 2 for all edges leading to a weight-regular graph (which is also a perfect matching).

6.1.2 General Case

For the general case, we start by modifying the input graph to obtain a weight-regular graph as in the simple case. We do so by taking into account latency and the k-constraint.

The difficulty of the KPBS problem comes from the startup delay cost associated to each step. This means that in order to be efficient we should avoid generating a too high number of steps and, therefore, avoid cutting any edge into too little pieces. In particular, we do not want to subdivise an edge with a cost already lower than the startup delay cost (which has a value of 1 due to normalization). To achieve that, the first step of the GGP algorithm is to round

Input: A bipartite graph $G = (V_1, V_2, E, w_G)$, an integer k.
Output: A set of valid weighted matchings S.

let $V_1 = \{v_1, \ldots, v_{n_1}\}$, $V_2 = \{u_1, \ldots, u_{n_2}\}$, where $n_1 = |V_1|$, $n_2 = |V_2|$
1. Build a graph $H = (V_1, V_2, E, w_H)$ such that $\forall e \in E, w_H(e) = \lceil w_G(e) \rceil$
2. Build a graph $I = (V_{1_I}, V_{2_I}, E_I, w_I)$ with $\frac{P(I)}{k} \geq W(I)$ and $\frac{P(I)}{k} \in \mathbb{N}$:
$\phi = \max\left(W(H), \left\lceil \frac{P(H)}{k} \right\rceil\right)$
δ: number of nodes added to V_1 and V_2: $\delta = \left\lceil \frac{\phi.k - P(H)}{W(H)} \right\rceil$
$V_{1_I} = \{v'_1, \ldots, v'_{n_1+\delta}\}$, $V_{2_I} = \{u'_1, \ldots, u'_{n_2+\delta}\}$
$E_I = E_1 \cup E_2$, where
$E_1 = \left\{(v'_i, u'_j) | (v_i, u_j) \in E, i \in [1, n_1], j \in [1, n_2]\right\}$, $\forall (v'_i, u'_j) \in E_1, w_I\left((v'_i, u'_j)\right) = w\left((v_i, u_j)\right)$
if $\delta = 0$ then
$E_2 = \emptyset$
else
$E_2 = \left\{(v'_{n_1+i}, u'_{n_2+i}) | i \in [1, \delta]\right\}$
if $\delta \neq 1$ then
$\forall i \in [1, \delta - 1], w_I\left((v'_{n_1+i}, u'_{n_2+i})\right) = W(H)$
if $(\phi.k - P(H)) \bmod W(H) \neq 0$ then
$w_I\left((v'_{n_1+\delta}, u'_{n_2+\delta})\right) = (\phi.k - P(H)) \bmod W(H)$
else
$w_I\left((v'_{n_1+\delta}, u'_{n_2+\delta})\right) = W(H)$
3. Transform I into a $\frac{P(I)}{k}$-weight-regular graph $J = (V_{1_J}, V_{2_J}, E_J, w_J)$
using the algorithm described in Fig 6.
4. $S = \emptyset$
5. While $E_J \neq \emptyset$ do:
5.1. Choose a perfect matching M in J
5.2. Change w_M such that $\forall e \in M, w_M(e) = s(M)$ the smallest weight of the edges of M
5.3. Add M to S, the set of solution matchings
5.4. $\forall e \in M$ change $w_J(e)$ to $w_J(e) - w_M(e)$
5.5. Remove from E_J all edges of weight 0
6. Remove all edges e in S such that $e \notin E$

Fig. 5. GGP algorithm.

all weights on all edges to their next upper integers and after that considering only matchings with integer costs in the algorithm.

The other main objective of the GGP algorithm is to avoid having more than k edges in a matching. In order to do that, we add some *virtual* edges in the input graph. By choosing them carefully, we can ensure that any perfect matching will contain at most k *real* edges (i.e., edges from the original graph): see Section 6.2.1.

Hence, if we put all that into order, GGP is divided into these four large steps:

1. Building graph H by rounding all weights (Step 1 in the formal description of Fig. 5).
2. Building graph I for preparing Step 3 (Step 2 in the formal description).
3. Build a weight-regular graph J while also adding virtual edges taking care of the k constraint (Step 3 in the formal description, and Fig. 6).
4. Peel the graph (Steps 4, 5, and 6 in the formal description).

To build the weight-regular graph of Step 3 we need to define the function $mw : V_1 \cup V_2 \rightarrow \mathbf{N}$ the function assigning to a node s the *missing weight* $mw(s)$ of this node for the graph to be $\frac{P(I)}{k}$ weight-regular. We have $mw(s) = \frac{P(I)}{k} - w(s)$. The algorithm building the weight-regular graph is shown in Fig. 6.

Example. Consider the example of Fig. 7. The input of GGP is the graph given on Fig. 1 with $k = 3$. After Step 3 of GGP, nodes 5 and 5′ as well as edges shown as dashed lines have been added to the original graph. Then, we start peeling the graph and in any perfect matching the number of real edges is always $k = 3$ and the number of virtual edges is 2. The final solution (real edges) is the same as the one given in Fig. 2.

6.2 Properties

We start by proving that the weight-regular extension algorithm of Fig. 6 is correct.

Proposition 2. *J is $\frac{P(I)}{k}$-weight-regular.*

Proof of Proposition 2. We need to prove $\forall s \in V_{1_J} \cup V_{2_J}$, $w(s) = \frac{P(I)}{k}$. We consider two cases: the case where s was already in I and the case where s is a new node.

If s was already in I, the algorithm implies $mw(s) = 0$ and, therefore, $w(s) = \frac{P(I)}{k}$.

Input: The weighted bipartite graph $I = (V_{1_I}, V_{2_I}, E_I, w_I)$ of step 2 of GGP, an integer k.
Output: A weighted bipartite graph J such that J is a $\frac{P(I)}{k}$ weight-regular bipartite graph.

1. Copy the graph I in J: $V_{1_I} = V_{1_J}, V_{2_I} = V_{2_J}$, $E_I = E_J$, and $\forall e \in E_J, w_J(e) = w_I(e)$.
2. We use a variable cn which is a node. cn starts being undefined.
3. foreach $s \in V_{1_I}$(s also in V_{1_J}) do:
   3.1 Compute $mw(s)$.
   If cn is undefined or $mw(cn) = 0$ then
      3.1.1 add a new node n to V_{2_J}
      3.1.2 $cn = n$
      3.1.3 add an edge (s, cn) to E_J with $w_J(s, cn) = mw(s)$
   else
      3.1.4 if $mw(cn) \geq mw(s)$ then
         3.1.4.1 add an edge (s, cn) to E_J with $w_J(s, cn) = mw(s)$
      else
         3.1.4.2 add an edge (s, cn) to E_J with $w_J(s, cn) = mw(cn)$
         3.1.4.3 add a new node n to V_{2_J}
         3.1.4.4 $cn = n$
         3.1.4.5 add an edge (s, cn) to E_J with $w_J(s, cn) = mw(s)$
4. Do the same for all nodes in V_{2_J}.

Fig. 6. Weight-regular extension algorithm.

If s was not in I, the weights on the added edges is the sum of the missing weights for all nodes of one side, that is $\sum_{t \in V_{1_I}} mw(t)$ and $\sum_{t \in V_{2_I}} mw(t)$. We know that $\sum_{t \in V_{1_I}} mw(t) = \sum_{t \in V1_I} \frac{P(I)}{k} - w(t)$. It is therefore equal to $|V_{1_I}| \times \frac{P(I)}{k} - P(I) = (|V_{1_I}| - k)\frac{P(I)}{k}$ since there is no edge between two nodes on the same side. This value divided by $\frac{P(I)}{k}$ gives $(|V_{1_I}| - k)$ which means it is possible to add edges to $(|V_{1_I}| - k)$ new nodes s with $w(s) = \frac{P(I)}{k}$. The same reasoning holds for V_{2_I}. As we build all nodes sequentially, never leaving a node s with $w(s) < \frac{P(I)}{k}$ we have $\forall s \in V_{1_J} \cup V_{2_J}, w(s) = \frac{P(I)}{k}$. □

6.2.1 Correctness of GGP

The correctness of GGP follows from the respect of the 1-port and the k-constraint. The 1-port constraint is ensured by choosing matchings. The respect of the k-constraint requires the following lemma:

Lemma 1. *Any perfect matching on J contains k edges belonging to I.*

Proof of Lemma 1. Let M be a perfect matching on J. We have from proof of Proposition 2 that V_{1_J} is V_{1_I} with $|V_{2_I}| - k$ new nodes. Similarly, V_{2_J} is V_{2_I} with $|V_{1_I}| - k$ new nodes. Therefore, we have $|V_{1_J}| = |V_{2_J}| = |V_{1_I}| + |V_{2_I}| - k$, which is the number of edges in M. We know that none of the nodes added are connected together and also that any edge connected to a new node is not in E_I. Therefore, we have one edge for each node added that is in M and not in E_I. Since we added $|V_{1_I}| - k$ and $|V_{2_I}| - k$ nodes the number of nodes in M and in E_I is $|V_{1_I}| + |V_{2_I}| - k - (|V_{1_I}| - k) - (|V_{2_I}| - k) = k$. □

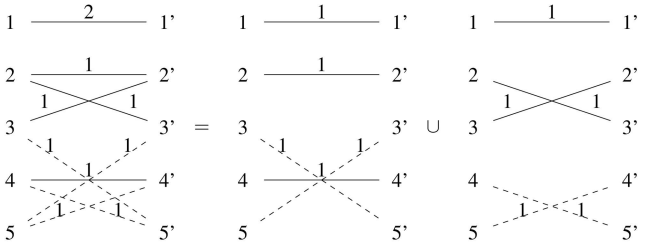

Fig. 7. Example of $k = 3$ constraint solving. The Graph J (after Step 3 of GGP) is built from the graph of Fig. 1 and peeled to obtain the solution (unbroken edges) of Fig. 2.

Since I is built by adding edges from H, M has at most k edges belonging to G. To build the solution, matchings on J from which all edges not belonging to G are removed. Therefore, any matching of the solution given by GGP respects the k-constraint.

6.2.2 Approximation Ratio

We define the $trans$ function which takes a weighted bipartite graph $G = (V_1, V_2, E, w_G)$ as argument and returns the corresponding multigraph $G' = (V_1', V_2', E')$ by splitting all edges such that an edge $e \in E$ of weight $w_G(e)$ is turned into $w_G(e)$ edges of weight 1.

With this function, we can now define the Multigraph algorithm (Fig. 8). Basically, this algorithm starts by constructing the same graph J as GGP but is different in the ways it peels the graph. We build $J' = trans(J)$ which is a regular graph and peel it into perfect matchings using the property that there always exist a perfect matching on a regular graph. Thus, we obtain a set of matchings whose costs are always 1, solution of KPBS. However, as the number of edges in J' depends on the weights of the edges of J the algorithm is only pseudopolynomial and therefore not useful in practice.

Theorem 2 proves that the multigraph algorithm is a $\frac{8}{3}$-approximation algorithm. Before that, we need the following lemma. It proves that the lower bound of the useful transmission time of graph I is equal to $\max\left(\lceil\frac{P(H)}{k}\rceil, W(H)\right)$.

Input: A bipartite graph $G = (V_1, V_2, E, w_G)$, an integer k.
Output: A set of matchings S.

1. Build the graph J as described in steps 1, 2 and 3 of GGP
2. Build $J' = trans(J)$
3. $S' = \emptyset$
4. While $E_{J'} \neq \emptyset$ do:
   5.1. Choose a perfect matching M' in J' with $\forall e \in M w_M(e) = 1$
   5.2. Add M' to S', the set of solution matchings
   5.3. Remove from $E_{J'}$ all edges of M'
6. Remove all edges e in S' such that $e \notin E$

Fig. 8. Multigraph algorithm.

Lemma 2. $\eta_d(I) = \max\left(\lceil\frac{P(H)}{k}\rceil, W(H)\right)$.

Proof of Lemma 2. Consider the three possible cases when building I:

1. $\frac{P(H)}{k} \geq W(H)$ and $\frac{P(H)}{k} \in \mathbf{N}$: $I = H$ and, therefore,
$$\eta_d(I) = \max\left(\frac{P(I)}{k}, W(I)\right) = \max\left(\frac{P(H)}{k}, W(H)\right) = \max\left(\left\lceil\frac{P(H)}{k}\right\rceil, W(H)\right).$$
2. $\frac{P(H)}{k} \geq W(H)$ and $\frac{P(H)}{k} \notin \mathbf{N}$: We add no edge of weight greater than $W(H)$ and $\frac{P(I)}{k} = \left\lceil\frac{P(H)}{k}\right\rceil$, hence $\eta_d(I) = \max\left(\left\lceil\frac{P(H)}{k}\right\rceil, W(H)\right)$.
3. $\frac{P(H)}{k} < W(H)$: As we add no edge of weight greater than $W(H)$, $W(I) = W(H)$. Moreover, we only add edges until $\frac{P(I)}{k} = W(H)$, hence $\eta_d(I) = \eta_d(H) = W(H)$. □

Theorem 2. *The multigraph algorithm is a* $\frac{8}{3}$-approximation *algorithm.*

Proof of Theorem 2. With as input parameters $G = (V_1, V_2, E, w_G)$ a bipartite graph and k an integer, we apply the multigraph algorithm on G to obtain S' the set of wanted matchings. J is, by construction, a $\frac{P(I)}{k}$-weight-regular graph and all of its weights are integers, therefore $J' = trans(J)$ is a $\frac{P(I)}{k}$-regular multigraph. Since $P(I) = m(I')$, J' is a $\frac{m(I')}{k}$-regular graph, where $I' = trans(I)$. As at each step of the main iteration, we remove a perfect matching from J', $|S'| = \frac{m(I')}{k} = \eta_s(I')$. The cost of the solution S' is therefore $c(S') = \eta_s(I') + \eta_s(I')$ because each step has a duration of 1 and the startup delay β is considered equals to 1. Therefore,

$$c(S') = 2\eta_s(I'). \tag{1}$$

If all edges of G have a weight less than 1, then all edges of H have a weight of 1 and $W(H) = 1$. Hence, all edges added to H to build I have a weight of 1. This means that $\eta_s(I') = \eta_s(I)$ because I and I' are then identical graphs. As the weight of any edge in I is 1, we have $P(I) = m(I)$ and $W(I) = \Delta(I)$ and, therefore, $\eta_s(I) = \max\left(\left\lceil\frac{P(I)}{k}\right\rceil, W(I)\right) = \max\left(\left\lceil\frac{P(H)}{k}\right\rceil, W(H)\right)$ by construction of I. As the weight of each edge in H is also 1, we can conclude that $\eta_s(I) = \eta_s(H)$. Finaly, as $m(H) = m(G)$ and $\Delta(H) = \Delta(G)$, we have $\eta_s(I') = \eta_s(I) = \eta_s(H) = \eta_s(G)$ and, therefore, (1) becomes $c(S') = 2\eta_s(G) \leq 2\eta(G)$. Therefore, the algorithm is a 2-approximation algorithm when all edges of G have a weight less than 1.

As $\frac{P(I)}{k} = \frac{m(I')}{k}$ and $W(I) = \Delta(I')$, we know that $\eta_s(I') = \max\left(\Delta(I'), \frac{m(I')}{k}\right) = \max\left(W(I), \frac{P(I)}{k}\right) = \eta_d(I)$.

Consequently, $c(S') = 2\eta_d(I)$. We now use Lemma 2 to deduce that

$$c(S') = 2\max\left(\left\lceil\frac{P(H)}{k}\right\rceil, W(H)\right). \tag{2}$$

Let us first suppose that $\left\lceil\frac{P(H)}{k}\right\rceil > W(H)$.

Equation (2) becomes $c(S') = 2\left(\left\lceil\frac{P(H)}{k}\right\rceil\right)$.

In the algorithm building H from G, no edge sees its weight increasing by more than one unit. Therefore, we have $P(H) \leq P(G) + m(G)$. This leads to

$$\begin{aligned} c(S') &\leq 2\left(\left\lceil\frac{P(G) + m(G)}{k}\right\rceil\right) && (3)\\ &\leq 2\left(\left\lceil\frac{P(G)}{k}\right\rceil + \left\lceil\frac{m(G)}{k}\right\rceil\right)\\ &\leq 2\left(\left\lceil\frac{P(G)}{k}\right\rceil + \max\left(\left\lceil\frac{m(G)}{k}\right\rceil, \Delta(G)\right)\right) && (4)\\ &\leq 2\left(\frac{P(G)}{k} + 1 + \eta_s(G)\right)\\ &\leq 2(\eta_d(G) + 1 + \eta_s(G)) = 2\eta(G) + 2. && (5) \end{aligned}$$

Since $W(H)$ is an integer and we supposed that $\left\lceil\frac{P(H)}{k}\right\rceil > W(H)$, we can deduce that $\frac{P(H)}{k} > W(H)$. This means that $m(H) > k$ (otherwise, the number of edges of H would be greater than k and $P(H)$ would be less than $k \times W(H)$—by definition of $W(H)$—and, consequently, $\frac{P(H)}{k}$ would be less than $W(H)$).

By construction $m(H) = m(G)$, therefore $\left\lceil\frac{m(G)}{k}\right\rceil \geq 2$. We deduce that $\eta_s(G) \geq 2$.

We can assume that the weight of at least one edge of G is greater than 1 or equal to 1 (the case when all the edges have a weight smaller than 1 has been treated above and leads to an approximation ratio of 2). We have $W(G) \geq 1$ and, therefore, $\eta_d(G) = \max\left(\frac{P(G)}{k}, W(G)\right) \geq 1$. Consequently, $\eta(G) \geq 2 + 1 = 3$.

Inequation (3) becomes

$$\frac{c(S')}{\eta(G)} \leq 2 + \frac{2}{\eta(G)} \leq 2 + \frac{2}{3} = \frac{8}{3}.$$

This means that in this case, the algorithm is a $\frac{8}{3}$-approximation algorithm.

Now, we still have to study the case where $\left\lceil\frac{P(H)}{k}\right\rceil \leq W(H)$. In this case, (2) becomes $c(S') = 2W(H)$.

However, $W(H) \leq W(G) + \Delta(G)$ because we have added at most one to each edge weight of G to build H. Hence,

$$\begin{aligned} c(S') &\leq 2(W(G) + \Delta(G))\\ &\leq 2\left(\max\left(\frac{P(G)}{k}, W(G)\right) + \max\left(\left\lceil\frac{m(G)}{k}\right\rceil, \Delta(G)\right)\right)\\ &\leq 2(\eta_d(G) + \eta_s(G)) = 2\eta(G). \end{aligned}$$

Therefore, for this case, the approximation ratio is 2.

Consequently, the approximation ratio for the multigraph algorithm is $\frac{8}{3}$ (in the worst case). □

From the above theorem, it follows that GGP is a $\frac{8}{3}$-approximation algorithm. Indeed, for any schedule S obtained by GGP of cost $c(S)$ it exists a schedule S' obtained by the multigraph algorithm of cost $c(S')$ such that $c(S) \leq c(S')$. By construction, a solution S obtained by GGP can be decomposed into a solution S' where $\forall M_i \in S$ of cost $c(M_i)$ there exists $c(M_i)$ identical matchings M'_j of cost 1 in S'. We therefore have $\sum_{i=1}^{|S|} c(M_i) = \sum_{j=1}^{|S'|} c(M'_j)$ and $|S| \leq |S'|$.

The cost of S is: $|S| + \sum_{i=1}^{|S|} c(M_i)$. Similarly, the cost of S' is: $|S'| + \sum_{j=1}^{|S'|} c(M'_j)$. Therefore, $c(S) \leq c(S')$.

6.2.3 Complexity

We have shown that the Multigraph algorithm is pseudo-polynomial. Here, we show that GGP is polynomial and compute its worst-case complexity.

Proposition 3. *GGP has a worst-case complexity in* $O(\sqrt{n(G)}(m(G)+n(G))^2)$.

Proof of Proposition 3. Steps 1, 2, 3, 4, and 6 of GGP are computed in linear time. However, Step 5 requires finding a perfect matching which is done in $O(\sqrt{n(J)}m(J))$ using the hungarian method [22]. At each step, we remove at least one edge (the edge of the matching with the lowest weight); hence, we iterate at most $m(J)$ times. Therefore, the worst-case complexity of Step 5 is in $O(\sqrt{n(J)}m(J)^2)$.

Now, to build H, no edge or node are added, hence $n(H) = n(G)$ and $m(H) = m(G)$. To build I, we add at most k edges and, therefore, $2k$ nodes, hence $n(I) \leq n(G) + 2k$ and $m(I) \leq m(G) + k$. Since it makes no sense allowing the selection in a matching of more edges than nodes, we limit here k to $n(G)$. We can then deduce that $n(I) \leq 3n(G)$ and $m(I) \leq m(G) + n(G)$.

In the weight-regular extension algorithm, for each node in I (each iteration), we add at most one new node in J. Therefore, $n(J) \leq 2n(I)$. Similarly, for each node in I, we add at most two new edges in J and, therefore, $m(J) \leq m(I) + 2n(I)$. Hence, $n(J) \leq 6n(G)$ and $m(J) \leq m(G) + 7n(G)$.

This leads to a worst-case complexity of Step 5 of $O(\sqrt{n(G)}(m(G)+n(G))^2)$. □

6.3 OGGP

6.3.1 Algorithm

It is possible to find a family of graphs on which GGP reaches a 2-approximation ratio. We developed a modified version of GGP called OGGP which gives good results on this family of graphs, and better results than GGP in the general case. Being a direct extension of GGP, OGGP inherits the $\frac{8}{3}$-approximation ratio. It should be noted however, that we have no proof that OGGP could have a lower approximation ratio than $\frac{8}{3}$.

The principle is the following: In GGP, when choosing a perfect matching, the weight-regular graph guarantees that there exists a perfect matching. However, there is often more than one perfect matching and GGP does not specify which one to choose, but uses a random one. In OGGP, we simply try to choose the matching that might give the best results among all matchings. Intuitively, we would like to issue as much communications as possible. By taking the longest possible communication steps, we might reduce the total number of steps and, therefore, the communication time. A communication step time is given by the smallest weight of all edges in the matching. To have the largest communication step, we need to find the perfect matching whose smallest weight is maximal.

Input: A bipartite graph G.
Output: M: the perfect weighted matching with maximal minimum weight.

1. $G' = \emptyset, M = \emptyset, G'' = G$
2. while M is not perfect in G do:
   2.1. choose $e \in E(G'') | \forall e' \in E(G''), w(e) \geq w(e')$
   2.2. $E(G') = E(G') \cup e$
   2.3. $E(G'') = E(G'') \backslash e$
   2.4. M = a maximal matching in G'

Fig. 9. Algorithm for extracting a matching with maximal minimum weight.

The greedy algorithm depicted in Fig. 9 finds a perfect matching which smallest edge's weight is maximal. It is based on the algorithm described in [4] that maximizes the minimum weight of a matching.

Proposition 4. *The algorithm of Fig. 9 returns a matching of maximal minimum weight.*

Proof of Proposition 4. Let l be the last edge added at the previous step in G', we have $l \in M$ because without l it was not possible to find a perfect matching in G. We also have $\forall e \in G, w(e) > w(l) \Rightarrow e \in G'$. Suppose that M' is a maximal matching better than M (it's minimum weight is larger than the one of M). M' is such that: $\forall e \in M', f(e) > f(l)$. Therefore, we have $M' \subset G'$. This is a contradiction, hence M is a perfect matching maximizing the minimum weight. □

6.3.2 Complexity

The new matching algorithm complexity is $O(m(J)\sqrt{n(J)}\, m(J)) = O(m(J)^2\sqrt{n(J)})$. Therefore, the complexity of OGGP is $O(\sqrt{n(G)}(m(G)+n(G))^3)$.

7 Heuristics

Here are two heuristics that appear to work well in practice (a heuristic on weights and a heuristic on degrees). They are faster than GGP and OGGP, but are not approximation algorithms as will show the simulation results. The heuristic on degrees is the same as the heuristic on weights (Fig. 10) except that line 2 is changed into "*2. Keep only the k (or less if there are less than k edges) edges with highest degrees.*"

Complexity: We use the Hungarian method of complexity $\mathcal{O}(m(G)n(G)^{1/2})$ for finding a maximum cardinality matching in a bipartite graph. For both heuristics, at each step, at least one edge is removed from G. Therefore, the complexity of both heuristics is $\mathcal{O}(m(G)^2\sqrt{n(G)})$ which is better than the complexity of GGP.

Input: A weighted bipartite graph G, an integer k.
Output: A set of valid weighted matchings.

1. Find a maximal matching.
2. Keep only the k (or less if there are less than k edges) edges whose weights are the largest.
3. Set all the weights of the matching equal to the lowest one.
4. Subtract the matching from G.
5. Loop until there is no more edge left in G.

Fig. 10. Heuristic on weights.

8 Experiments

8.1 Simulation of the Heuristics

We have tested each heuristic (with k fixed) on a sample of 100,000 random graphs (the number of edges, the edges, and, finally, the weights were chosen randomly with a uniform distribution) with 20 nodes on each side. We made a difference between lightly and heavily weighted graphs. Small weights were taken between 1 and 20, whereas large weights were taken between 1 and 100,000. The result of a heuristic is calculated as the solution cost divided by the lower bound η. We call this ratio the *evaluation ratio*.

In Figs. 11, 12, 13, and 14, the plots show the average and the maximum calculated over the samples.

For these tests, the maximum is always below 2.4 with an average under 1.8 for small weights, and below 2, with an average under 1.3 in case of large weights.

We explain the convex shape of the plots as follows:

- When $k = 1$, the two heuristics obtain the optimal solution which consists of one communication per step.
- When k is greater than 2 and lower than a certain value (close to $n/2$), the quality of the solution degrades (compared to the lower bound). We believe that this is due to the fact that, at each step, the number of valid matchings increases.
- When k is greater than $n/2$, the quality of the solution tends to improve. At each stage of the two heuristics, the choice of valid matchings decreases, therefore the heuristics are less likely to select bad valid matchings.

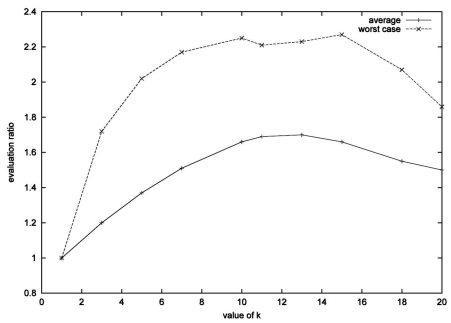

Fig. 11. Heuristic on weights. Simulation with small weights.

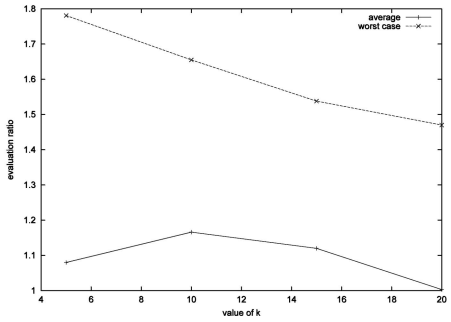

Fig. 12. Heuristic on weights. Simulation with large weights.

8.2 Simulation of GGP and OGGP

The simulation of GGP and OGGP has been conducted under the same conditions as the simulation of the heuristics. OGGP and GGP have been implemented into a C++ library, and executed on random graphs as described in the previous section.

8.2.1 Comparing GGP and OGGP

Tests on Small Weights. Fig. 15 displays how the evaluation ratio varies when k grows. The weights are generated randomly between 1 and 20.

We can see that as k grows, the evaluation ratio grows and stabilizes. OGGP gives better results than GGP even when comparing the worst case obtained with OGGP and the average obtained with GGP.

Tests on Large Weights. Fig. 15 displays how the evaluation ratio varies when k grows. The weights are generated randomly between 1 and 100,000.

Results are similar, but with an evaluation ratio far closer to 1 on large weights. On these cases, the difference between GGP and OGGP is smaller.

We can see that GGP is giving better results than the heuristics. Although the difference is not extremely high on average, when comparing worst cases, heuristics are taking 1.5 more time than GGP.

8.3 Real-World Experiments with MPI

In order to validate theoretical results and simulations, we have conducted several real-world experiments. We used two clusters of 10 1.5 GHz Pentium computers running Linux. Network cards were 100Mbits Ethernet adapters and the two clusters were interconnected within a local network by two 100Mbits switches. In order to test interesting cases, that is where $k \neq 1$, we limited the available incoming and

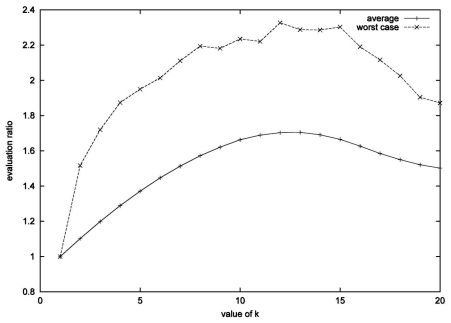

Fig. 13. Heuristic on edges. Simulation with small weights.

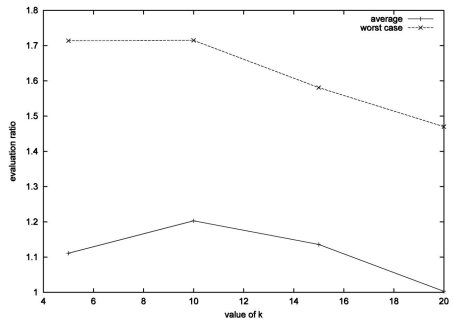

Fig. 14. Heuristic on edges. Simulation with large weights.

outgoing bandwidth of each network card to $\frac{100}{k}$ Mbits per second. This was done using the *rshaper* [29] Linux kernel module. This module implements a software token bucket filter thus enabling a control of the available bandwidth. We conduct experiments for $k = 3, k = 5, k = 7$.

Two different types of redistribution have been implemented. First, a brute force TCP-only approach: We start all communications simultaneously and wait until all transfers are finished. In this case, the network transport layer (TCP) is responsible for the congestion control. The second approach allows us to test our algorithms: We divide all communications into different steps, synchronized by a barrier, and only one synchronous communication can take place in each step for each sender. Both algorithms have been implemented using MPICH version 1.2.4. We have not implemented an exponential algorithm finding the optimal solution (which could seem possible as the number of nodes and edges is not very high) because designing such an algorithm is difficult, and anyway our algorithms are already close enough to the optimum. All communication times have been measured using the *ntp_gettime* function call from the GNU libc.

In our tests, the 10 nodes of the first cluster have to communicate to each 10 nodes of the second cluster. The size of the data to transfer between two given cluster nodes is uniformly generated between 10 and n MB. We plot the total communication time obtained when n increases as shown in Fig. 16.

Several observations can be made:

- We achieve a 5 to 20 percent reduction of communications costs. Although we are alone on a local network, where TCP is efficient, we are able to achieve better results.
- The barriers cost extremely little time. Although OGGP algorithm has 50 percent less steps of communication, it gives the same result as GGP. However, we believe the cost of synchronizations may increase if we introduce some random perturbations on the network.
- The brute-force approach does not behave deterministically. When conducting several time the same experiments we see a time variation of up to 10 percent. It is interesting to see that our approach on the opposite behaves deterministically.
- As the available bandwidth decreases (i.e., k increases) we increase the benefits of using GGP or $OGGP$ over the brute-force approach.

9 Conclusions

In this paper, we have formalized and studied the problem (called KPBS) of redistributing parallel data over a backbone. Our contribution is the following: We have shown that KPBS remains NP-hard when k is constant. We have studied lower bounds related to KPBS. We have proposed two messages scheduling algorithms called GGP and OGGP for the redistribution problem. These algorithms have an approximation ratio of $\frac{8}{3}$. We then studied two fast heuristics.

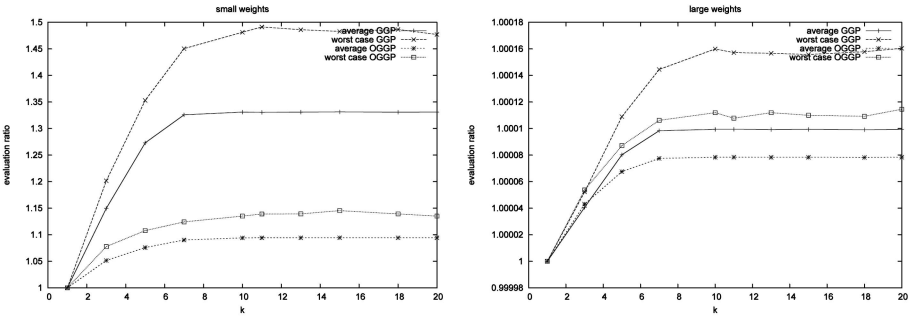

Fig. 15. GGP and OGGP for small weights/large weights graphs.

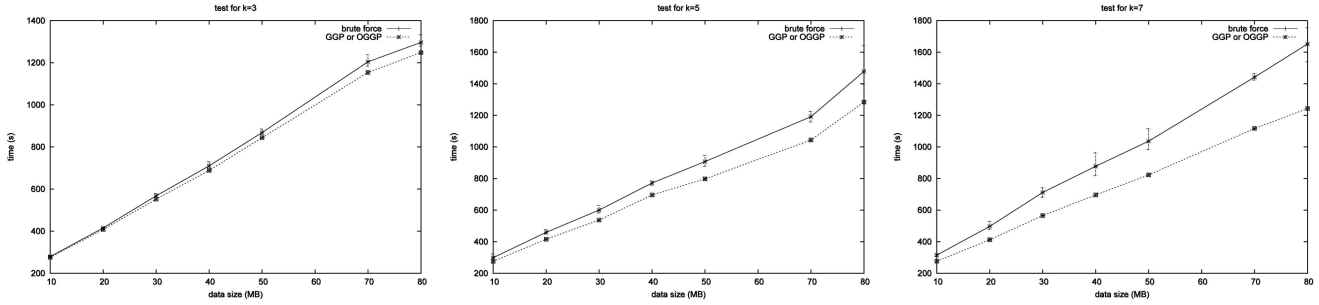

Fig. 16. Brute-Force versus GGP or OGGP ($k = 3, 5, 7$).

Simulations show that OGGP outperforms GGP that outperforms the heuristics. We have performed real experiments on two clusters. Results show that our scheduling algorithms outperform the brute-force approach that consists of letting the network manage the congestion alone (redistribution time can be reduced to up to 20 percent).

In our approach, we limit the maximum number of messages during one step. This is especially useful when the redistribution is performed between two clusters interconnected by a backbone and when this backbone is a bottleneck. However, the algorithms we have proposed can also be used when a redistribution occurs on the same parallel machines or in the context of SS/TDMA systems or WDM network.

In our future work, we want to extend the model to handle more complex redistributions. First, we would like to consider achieving a local preredistribution in case a high-speed local network is available. This would enable us to aggregate small communications together, or on the opposite to dispatch communications to all nodes in the cluster. Second, we would like to study the problem when the throughput of the backbone varies dynamically or when the redistribution pattern is not fully known in advance. We think that our multistep approach could be useful for these dynamic cases. The final goal of this work is to produce (together with the people involved in the INRIA ARC redGRID[4]) a fully working redistribution library.

ACKNOWLEDGMENTS

This work was partially funded by the INRIA ARC redGRID, the ACI GRID, and the Région Lorraine.

REFERENCES

[1] F.N. Afrati, T. Aslanidis, E. Bampis, and I. Milis, "Scheduling in Switching Networks with Set-Up Delays," *J. Combinatorial Optimization,* vol. 9, no. 1, pp. 49-57, 2005.

[2] F. Bertrand, R. Bramley, D. Bernholdt, J.A. Kohl, A. Sussman, J.W. Larson, and K. Damevski, "Data Redistribution and Remote Method Invocation in Parallel Component Architectures," *Proc. Int'l Parallel and Distributed Processing Symp.,* 2005.

[3] P.B. Bhat, V.K. Prasanna, and C.S. Raghavendra, "Block Cyclic Redistribution over Heterogeneous Networks," *Proc. 11th Int'l Conf. Parallel and Distrinuted Computing Systems (PDCS '98),* 1998.

[4] G. Bongiovanni, D. Coppersmith, and C.K. Wong, "An Optimum Time Slot Assignment Algorithm for an SS/TDMA System with Variable Number of Transponders," *IEEE Trans. Comm.,* vol. 29, no. 5, pp. 721-726, 1981.

[5] V. Boudet, F. Desprez, and F. Suter, "One-Step Algorithm for Mixed Data and Task Parallel Scheduling without Data Replication," *Proc. Int'l Parallel and Distributed Processing Symp.,* p. 41, 2003.

[6] H. Casanova and J. Dongarra, "NetSolve: A Network-Enabled Server for Solving Computational Science Problems," *Int'l J. Supercomputer Applications and High Performance Computing,* vol. 11, no. 3, pp. 212-213, Fall 1997.

[7] H. Choi, H.-A. Choi, and M. Azizoglu, "Efficient Scheduling of Transmissions in Optical Broadcast Networks," *IEEE/ACM Trans. Networking,* vol. 4, no. 6, pp. 913-920, 1996.

[8] P. Crescenzi, D. Xiaotie, and C.H. Papadimitriou, "On Approximating a Scheduling Problem," *J. Combinatorial Optimization,* vol. 5, pp. 287-297, 2001.

[9] B. Del-Fabbro, D. Laiymani, J.-M. Nicod, and L. Philippe, "Data Management in Grid Applications Providers," *Proc. Int'l Conf. Distributed Frameworks for Multimedia Applications (DFMA),* pp. 315-322, 2005.

[10] F. Desprez, J. Dongarra, A. Petitet, C. Randriamaro, and Y. Robert, "Scheduling Block-Cyclic Array Redistribution," *IEEE Trans. Parallel and Distributed Systems,* vol. 9, no. 2, pp. 192-205, Feb. 1998.

[11] F. Desprez and E. Jeannot, "Improving the GridRPC Model with Data Persistence and Redistribution," *Proc. Third Int'l Symp. Parallel and Distributed Computing (ISPDC),* July 2004.

[12] S. Even, A. Itai, and A. Shamir, "On the Complexity of Timetable and Multicommodity Flow Problem," *SIAM J. Computers,* vol. 5, pp. 691-703, 1976.

[13] A. Ganz and Y. Gao, "A Time-Wavelength Assignment Algorithm for WDM Star Network," *Proc. IEEE INFOCOM Conf.,* pp. 2144-2150, 1992.

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

[15] I.S. Gopal, G. Bongiovanni, M.A. Bonuccelli, D.T. Tang, and C.K. Wong, "An Optimal Switching Algorithm for Multibean Satellite Systems with Variable Bandwidth Beams," *IEEE Trans. Comm.,* vol. 30, no. 11, pp. 2475-2481, Nov. 1982.

[16] I.S. Gopal and C.K. Wong, "Minimizing the Number of Switching in an SS/TDMA System," *IEEE Trans. Comm.,* 1985.

[17] M. Guo and I. Nakata, "A Framework for Efficient Data Redistribution on Distributed Memory Multicomputers," *The J. Supercomputing,* vol. 20, no. 3, pp. 243-265, 2001.

[18] S. Sekiguchi, H. Nakada, and M. Sato, "Design and Implementations of Ninf: Towards a Global Computing Infrastructure," *Future Generation Computing Systems,* vol. 15, pp. 649-658, 1999.

[19] C.-H. Hsu, Y.-C. Chung, D.-L. Yang, and C.-R. Dow, "A Generalized Processor Mapping Technique for Array Redistribution," *IEEE Trans. Parallel and Distributed Systems,* vol. 12, no. 7, pp. 743-757, 2001.

[20] The Hydrogrid Project, http://www-rocq.inria.fr/kern/HydroGrid/HydroGrid-en.html, 2006.

[21] Oak Ridge National Labs, Mxn, http://www.csm.ornl.gov/cca/mxn, 2006.

[22] S. Micali and V.V. Vazirani, "An $o(\sqrt{(v)}e)$ Algorithm for Finding a Maximum Matching in General Graphs," *Proc. 21st Ann IEEE Symp. Foundations of Computer Science,* pp. 17-27, 1980.

[23] M. Mishra and K. Sivalingam, "Scheduling in WDM Networks with Tunable Transmitter and Tunable Receiver Architecture," *Proc. NetWorld + Interop Eng. Conf.,* May 1999.

[24] N. Park, V.K. Prasanna, and C.S. Raghavendra, "Efficient Algorithms for Block-Cyclic Array Redistribution between Processor Sets," *IEEE Trans. Parallel and Distributed Systems,* vol. 10, no. 12, pp. 1217-1239, Dec. 1999.

[25] C. Pérez, T. Priol, and A. Ribes, "A Parallel Corba Component Model for Numerical Code Coupling," *Proc. Third Int'l Workshop Grid Computing,* Nov. 2002.

[26] G.R. Pieris and G.H. Sasaki, "Scheduling Transmission in WDM Broadcast-and-Select Networks," *IEEE/ACM Trans. Networking,* vol. 2, no. 2, Apr. 1994.

[27] A. Radulescu, C. Nicolescu, A.J.C. van Gemund, and P. Jonker, "CPR: Mixed Task and Data Parallel Scheduling for Distributed Systems," *Proc. Int'l Parallel and Distributed Processing Symp.,* p. 39, 2001.

[28] N. Rouskas and V. Sivaraman, "On the Design of Optimal TDM Schedules for Broadcast WDM Networks with Arbitrary Transceiver Tuning Latencies," *Proc. IEEE INFOCOM Conf.,* pp. 1217-1224, 1996.

[29] A. Rubini, "Linux Module for Network Shaping," http://ar.linux.it/software/\#rshaper, 2005.

[30] C. Szyperski, *Component Software: Beyond Object-Oriented Programming.* ACM Press, 1999.

[31] J. Turek, J. Wolf, and P. Yu, "Approximate Algorithms Scheduling Parallelizable Tasks," *Proc. Fourth Ann. ACM Symp. Parallel Algorithms and Architectures (SPAA '92),* pp. 323-332, 1992.

4. http://graal.ens-lyon.fr/~desprez/REDGRID.

Johanne Cohen received the PhD degree in computer science from the University of Paris XI in 1999. She is currently at the French Centre National de la Recherche Scientifique (CRNS) and located at LORIA Laboratory (Nancy, France). Her main research interests include the study of telecommunication networks and approximation methods for hard combinatorial problems.

Emmanuel Jeannot received the master's and PhD master's degrees in computer science (respectively, in 1996 and 1999) both from Ecole Normale Supérieure de Lyon. He is currently a full-time researcher at INRIA (Institut National de Recherche en Informatique et en Automatique) and is doing his research at the LORIA Laboratory. From September 1999 to September 2005, he was an associate professor at the Université Henry Poincaré, Nancy 1. His main research interests are scheduling for heterogeneous environments and grids, data redistribution, grid computing software, adaptive online compression, and programming models.

Nicolas Padoy studied computer science at the Ecole Normale Supérieure de Lyon, France, and at the Technische Universität München, Germany. After some research in distributed computing, he is now starting a joint PhD in medical computer vision at the Université Henry-Poincaré, France, and at the Technische Universität München, Germany.

Frédéric Wagner has been studying computer science at the Université Louis Pasteur in Strasbourg. After receiving the master's degree on automatic compiler optimizations, he received the PhD degree in Nancy at the Université Henri Poincaré on communications scheduling for parallel computing. He currently has a temporary position at the University of Avignon.