Messages Scheduling for Parallel Data Redistribution between Clusters*

Johanne Cohen
LORIA-CNRS
Nancy, France

Emmanuel Jeannot
LORIA Université H. Poincaré
Nancy, France

Nicolas Padoy
T.U. München
Germany

Frédéric Wagner
LORIA-INRIA
Nancy, France

Abstract

We study the problem of redistributing in parallel 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 communications between two parallel machines interconnected by a backbone, we aim at minimizing the total time required for the completion of all communications 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-complete. We study a lower bound of the problem. We propose two 2-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. These algorithms have

*This work is partially funded by the INRIA ARC redGRID, the ACI GRID and the Région Lorraine

been implemented using MPI. Experimental results show that both algorithms outperform a brute-force TCP based solution, where no scheduling of the messages is performed.

Keywords: 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 ressources, be it computing power, software or data. However the classical problem of minimizing the communications / computations ratio remains and is even more difficult in most cases since communication times increase on slower networks. It is therefore important to try to minimize communication times. In this work we concentrate 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, 22, 25, 27] or for satellite-switched time division multiple access (SS/TDMA) [4, 15, 16]. The solution proposed in the article works for these two contexts.

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, 23], here no assumption is made on the redistribution pattern. Moreover, contrary to other works which assume that there is no bottleneck [1, 8], 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 applications framework. We provide here 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 [20] are composed of several codes that interact with each other. They are used for simulating complex system. Such a system is composed of several models, each model being simulated by one parallel code or component [29]. Moreover, in distributed code coupling, each code/component is run on different parallel machine or cluster [2, 24]. During the simulation the models interact with each-other and therefore the parallel codes need to exchange data. This exchange of data is therefore a redistribution between distant clusters.

2. Parallel task execution: Recent work in the field of mixed parallelism [30, 26, 5], have shown the 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 for metacomputing: Several metacomputing environments implement the client-agent-server model [6], such as Netsolve or Ninf [18]. In this model the agent has in charge to map a client request to a server, the request is then being processed in an RPC way. One of the drawback 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]. As the service can be parallel the data can be distributed among the node of the cluster and 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-Complete for any value of $k (k < n)$. We exhibit a lower bound for the number of steps of the redistribution as well as a lower bound for the sum of the duration of each step. Next, we propose two algorithms (called GGP and OGGP) that have a low complexity and a 2-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 in letting the transport layer doing the scheduling and managing alone the congestion.

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.

Let us consider a parallel application that must execute the first part of its computation 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 in 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, 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 (resp. 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) $k \times d_1 \leq D$, (b) $k \times d_2 \leq D$, (c) $k \leq n_1$ and (d) $k \leq n_2$.

We denote by d the speed of each communication. For instance let us assume that $n_1 = 200$, $n_2 = 100$, $d_1 = 10$Mbit/s, $d_2 = 100$Mbit/s and $D = 1$GBbit/s ($D = 1000$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 (which is supported by the backbone) and each network card of $\mathcal{C}_2$ can receive the data at $d =$10 Mbit/s.

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 complete it later. In practice, this involves a non-negligible 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 and β as follows: (1) The traffic matrix T, can be replaced by the matrix $Q = (q_{i,j}) = (\frac{t_{i,j}}{d})_{1\leq i\leq n_1, 1\leq j\leq n_2}$ that represents the communication times for each message. (2) The matrix Q can be replaced by the matrix $M = (m_{i,j}) = (\frac{q_{i,j}}{\beta})_{1\leq i\leq n_1, 1\leq j\leq n_2}$ that represents the fraction of startup delay required for sending each message.

In the following we will *always* consider the normalized problem ($\beta = 1$).

The matrix M can be represented by a bipartite graph $G = (V_1, V_2, E)$ (see Figure 1) and a positive edge-weight function $w : E \rightarrow \mathbb{Q}$. Each node of cluster $\mathcal{C}_1$ (resp. $\mathcal{C}_2$) is represented by a node of V_1 (resp. V_2). Hence, $|V_1| = n_1$ and $|V_2| = n_2$. The weight of an edge between node i and j is equal to $m_{i,j}$.

	1'	2'	3'	4'
1	8	3	0	0
2	2	3	0	0
3	0	4	0	0
4	0	0	2	5

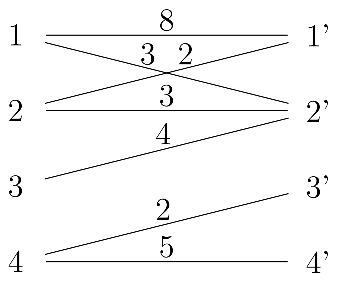

Figure 1: *A Communication Matrix Seen as a Bipartite Graph*

We use the 1-port model for the communication and at most k communications can occur during one step. Hence, a communication step is a weighted matching of G with at most k edges. The weights refer to the amount of exchanged data. We denote the matching corresponding to a communication step by a *valid weighted matching* (for the remaining, a valid weighted matching contains at most k edges).

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

Given a weighted bipartite graph $G = (V_1, V_2, E, w)$ where $w : E \to \mathbb{Q}$, an integer $k \geq 2$[1], find a collection $\{(M_1, W_1), (M_2, W_2), \ldots, (M_s, W_s)\}$ of valid weighted matchings such that:

1. Let w_i be the edge weight function of each matching M_i. It must respect the following inequalities: for any $e \in E$, $\sum_{i=1}^{s} w_i(e) \geq w(e)$. If $e \notin M_i$ then $w_i(e) = 0$.

2. For any $1 \leq i \leq s$, the matching M_i has at most k edges ($|M_i| \leq k$) and its cost is equal to the rational number $W_i = \max_{e \in M_i} w_i(e)$.

3. $(\sum_{i=1}^{s} W_i) + s$ is minimized. In the normalized form of the problem, each step has a cost equal to W_i plus 1 for the startup cost.

[1]the case $k = 1$ is not interesting because the backbone is saturated by one communication

In the remainder of this paper, we use the following notation: for any solution S of $KPBS$, if the cost of S is $\alpha + s$, the *number of steps* is s and the *useful transmission cost* equals α. See Figure 2 for an example.

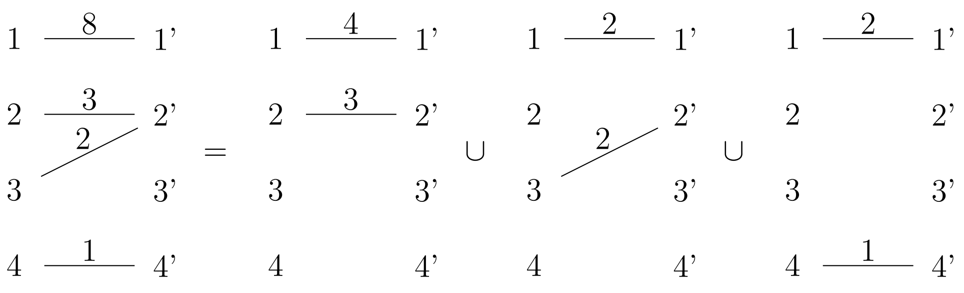

Figure 2: *An example for KPBS problem (k = 2). The cost of the solution is* $8 + 3 = 11$

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-Complete 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, 22, 25, 27]. The problem of minimizing the number of steps is studied in [13, 15], and the problem of minimizing the total cost is studied in [22]. In [7] and in [25], 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 [25] 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-complete 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, 23] 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-complete for the particular case where $k \geq \min(n_1, n_2)$ [12, 16]. We prove that it remains NP-complete for any fixed $k \geq 2$ (with a different reduction than in [12, 16]). The decision problem KPBS is defined as follow:

Instance: A weighted bipartite graph $G = (V_1, V_2, E, w)$ where $w : E \rightarrow \mathbb{Q}$, 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 $G = \sum_{i=1}^{s} M_i$, and $\sum_{i=1}^{s} w(M_i) + s \leq B$.

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

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

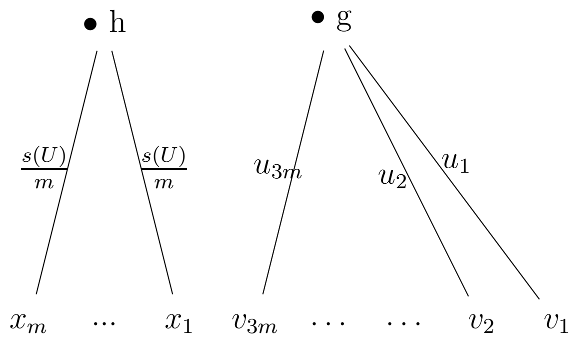

Figure 3: An example of graph G from an instance (U, s) of Partition Problem.

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

Question: Can U be partitioned into m disjoint sets $U_1, \ldots, U_m$ such that:

For $1 \leq i \leq m$, $\sum_{u \in U_i} s(u) = \frac{1}{m} \sum_{u \in U} s(u)$?

Let $s(U) = \sum_{i=1}^{m} s(u_i)$. We transform 3-partition to KPBS: Let $U = \{u_1, u_2, \ldots, u_m\}$ be a finite set and a size $s(u) \in Z^+$ for each $u \in U$ in an arbitrary instance of *3-Partition*. We must construct an instance of KPBS. We set $B = s(U) + 3m$ 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 $V_2 = \{g, h\}$
- $E = \{(x_i, h) : 1 \leq i \leq m\} \cup \{(v_i, g) : 1 \leq i \leq 3m\}$
- $w(x_i, h) = s(U)/m$ for $1 \leq i \leq m$
- $w(v_i, g) = s(u_i)$ for $1 \leq i \leq 3m$

Figure 3 gives an example of this transformation. Since *3-Partition* is NP-complete in the strong sense, G can clearly be constructed in polynomial time. We claim that the instance of KPBS admits a solution if and only if the instance of *3-Partition* has a solution.

($\Rightarrow$) Let $\{U_1, \ldots, U_m\}$ be a solution of the *3-Partition* instance. Then for $1 \leq j \leq m$, $1 \leq i \leq 3m$, valid matchings $\{(v_i, g)(x_j, h)\}$ having weights $s(u_i)$ such that $u_i \in U_j$ are a solution of KPBS. Indeed the sum of these matchings is $s(U)(\sum_{j=1}^{m}\sum_{u_i\in U_j} s(u_i) = s(U))$. The cost of this solution is exactly B.

($\Leftarrow$) Conversely, we suppose that the instance of KPBS admits a solution. Then the useful transmission cost is at least equal to $s(U)$ 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 KPBS is composed of m valid matchings. Thus the solution having the desired properties. can be written $(C_i)_{1\leq i\leq 3m}$. Since the size of a matching is at most $2(=k)$, for $1 \leq i \leq 3m$, C_i of the solution contains only one edge incident to g and to u_j (w.l.g. we assume that $j = i$) such that $w(C) \leq s(u_i)$. Necessarily C_i contains an edge incident to one vertex belonging to $\{x_1, \ldots, x_m\}$, having the weight $s(u_i)$ (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 C_i contains an edge adjacent to x_j. Then, since for $1 \leq j \leq m$,$w(x_j, h) = \frac{s(U)}{m}$, we have $\sum_{u\in U_j} s(u_i) = \frac{s(U)}{m}$. So sets $U_1, \ldots, U_m$ (such that for $1 \leq j \leq m$, $\sum_{u\in U_j} s(u) = \frac{1}{m}S(u)$) is a partition of U .

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

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 Figure 1, $W(G) = 8$, $P(G) = 27$ and $\Delta(G) = 3$.

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

$\eta_d(G) + \eta_s(G)$ *where*

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

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 $\eta_d(G) + \eta_s(G)$. ■

6 Algorithms

In this section we present two algorithms we propose to use for the KPBS problem. We start by presenting the *GGP* algorithm (Generic Graph Peeling) which is a polynomial time 2-approximation. This algorithm is relatively complex hence we describe it relying on a subalgorithm called *weight-regular extension algorithm* in order to simplify the presentation.

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: approximation ratio, worst case complexity and algorithm correctness. Study of the approximation ratio is difficult and we need to introduce another algorithm called *multigraph algorithm.* We prove that this algorithm is a pseudo polynomial 2-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.

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

  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 $I$ with $\frac{P(I)}{k} \geq W(I)$ and $\frac{P(I)}{k} \in \mathbb{N}$:
  If $\frac{P(H)}{k} < W(H)$
      build graph $I = (V_{1_I}, V_{2_I}, E_I, w_I)$ such that:
         - $E \subset E_I, V_1 \subset V_{1_I}, V_2 \subset V_{2_I}$
         - $\forall e \in E, w_I(e) = w_H(e)$
         - $P(I) = W(H) \times k$ with:
            - $\exists e = (s_1, s_2) \in E_I : s_1 \notin V_1, s_2 \notin V_2, w_I(e) \leq \max\left(W(H), \left\lceil \frac{P(H)}{k} \right\rceil\right) = \frac{P(I)}{k}$
            - $\forall d = (s_1, s_2) \in E_I : s_1 \notin V_1, s_2 \notin V_2, d \neq e$
             we have $w_I(d) = \max\left(W(H), \left\lceil \frac{P(H)}{k} \right\rceil\right) = \frac{P(I)}{k}$
            - $\forall v \in V_{1_I} \backslash V_1$ the degree of $v$ is 1, the edge $e$ incident to $v$ is incident
             to $v_2 \in V_{2_I} \backslash V_2$
  Else if $\frac{P(H)}{k} \notin \mathbb{N}$
      build graph $I = (V_{1_I}, V_{2_I}, E_I, w_I)$ such that:
         - $E \subset E_I, V_1 \subset V_{1_I}, V_2 \subset V_{2_I}$
         - $\forall e \in E, w_I(e) = w_H(e)$
         - $\frac{P(I)}{k} = \left\lceil \frac{P(H)}{k} \right\rceil$ with:
            - $\exists e = (s_1, s_2) \in E_I : s_1 \notin V_1, s_2 \notin V_2, w_I(e) \leq \max\left(W(H), \left\lceil \frac{P(H)}{k} \right\rceil\right) = \frac{P(I)}{k}$
            - $\forall d = (s_1, s_2) \in E_I : s_1 \notin V_1, s_2 \notin V_2, d \neq e$
             we have $w_I(d) = \max\left(W(H), \left\lceil \frac{P(H)}{k} \right\rceil\right) = \frac{P(I)}{k}$
            - $\forall v \in V_{1_I} \backslash V_1$ the degree of $v$ is 1, the edge $e$ incident to $v$ is incident
             to $v_2 \in V_{2_J} \backslash V_2$
      Else $I = H$
  3. Transform $I$ into a weight-regular graph:
  Build a $\frac{P(I)}{k}$-weight-regular graph $J = (V_{1_J}, V_{2_J}, E_J, w_J)$ using the algorithm
   described in Figure 7.
  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 $e \notin E$
```

Figure 4: **GGP algorithm**

6.1 GGP Algorithm

6.1.1 Simple Case

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.

An interesting propriety of weight-regular graph is that there always exists a perfect matching on any of them [8]. We are then able to pick a perfect matching which will be stored in the set of solutions. But as we want all communications in a given step to end 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 we can see that the graph left after removing the matching is still weight-regular because we removed the same amount of weight on each edge and since the matching was perfect, on each node. We then start again, removing another matching out of the graph. In the remaining of this paper, we call *peeling* the graph this procedure of step by step removing perfect matchings from it. The algorithm ends when the graph is empty. An illustration of this peeling procedure is shown on Figure 5.

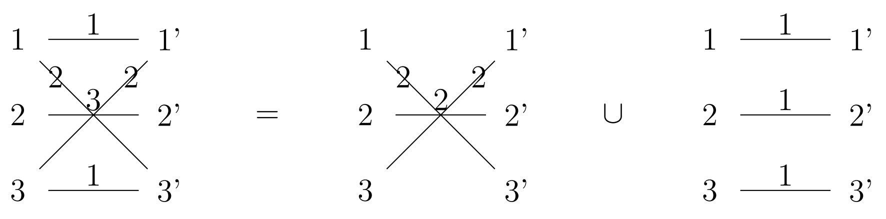

Figure 5: *Peeling a weight-regular graph*

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 while 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 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 Proposition 3.

Consider the example of Figure 6. The input of GGP is the first graph on the left, with $k = 1$. We can quickly see that peeling this graph directly would give as results one matching with all two edges. Therefore we first add the edges shown as dashed lines to the initial graph. After that we start peeling the graph and in any perfect matching you can take (for example the graph on the right) the number of real edges is always 1 and the number of virtual edges 2.

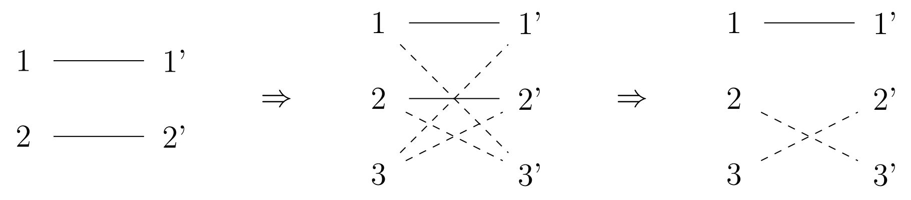

Figure 6: *Example of k constraint solving*

Hence if we put all that into order, GGP is divided into these 4 large steps:

1. Round all weights (step 1 in the formal description of Figure 4),

2. Prepare the graph for step 3 (step 2 in the formal description),

3. Build a weight-regular graph while also adding virtual edges taking care of the k constraint (step 3 in the formal description),

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 $mw : V_1 \cup V_2 \to \mathbb{N}$ 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 Figure 7.

```
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. Now 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 $V_{2_J}$.
```

Figure 7: **Weight-regular extension algorithm**

6.2 Properties

6.2.1 Approximation Ratio

We start by proving that the weight-regular extension algorithm of Figure 7 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 $w(s) = \frac{P(I)}{k}$.

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 $\forall s \in V_{1_J} \cup V_{2_J}, w(s) = \frac{P(I)}{k}$. ■

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 spliting 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 (Figure 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 pseudo-polynomial and therefore not useful in practice.

Theorem 2 proves that the multigraph algorithm is a 2-approximation one. Before that we need the following lemma. It proves that the lower bound of the useful transmission time remains unchanged when building graph H from graph I.

Lemma 1 $\eta_d(I) = \eta_d(H)$

Proof of Lemma 1 $\eta_d(H) = \max(\left\lceil \frac{P(H)}{k} \right\rceil, W(H))$. We have three possibilities:

```
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 $e \notin E$
```

Figure 8: **Multigraph algorithm**

1. $\frac{P(H)}{k} \geq W(H)$ and $\frac{P(H)}{k} \in \mathbb{N}$: $I = H$ and therefore $\eta_d(H) = \eta_d(I)$.

2. $\frac{P(H)}{k} \geq W(H)$ and $\frac{P(H)}{k} \notin \mathbb{N}$: we add no edge of weight greater than $W(H)$ and $\frac{P(I)}{k} = \left\lceil \frac{P(H)}{k} \right\rceil$ hence, $\eta_d(H) = \eta_d(I)$.

3. $\frac{P(H)}{k} < W(H)$: as we do not add any edge of weight greater than $W(H)$, $W(I) = W(H)$. Moreover we 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 2-approximation.*

Proof of Theorem 2 We name I' the graph obtained by $trans(I)$.

Take a weighted bipartite graph $G = (V_1, V_2, E, w_G)$ with k an integer as an input pattern. We apply the multigraph algorithm on G and obtain S' as the solution set of matchings. Since J is, by construction, a $\frac{P(I)}{k}$-weight-regular graph, and all weights are integers, we know that $J' = trans(J)$ is a $\frac{P(I)}{k}$ regular multigraph. As $P(I) = m(I')$ we have J' a $\frac{m(I')}{k}$ regular graph. Since at each step we remove a perfect matching and therefore an edge on each vertex of J' we know that $|S'| = \frac{m(I')}{k} = \eta_s(I')$. Therefore the cost of the solution S' is $c(S') = \eta_s(I') + \eta_s(I') \times 1 = 2\eta_s(I')$ since each step has a duration of 1 and an startup delay of 1. As $\frac{P(I)}{k} = \frac{m(I')}{k}$ and $W(I) = \Delta(I')$ we know that $\eta_s(I') = \max(\Delta(I'), \frac{m(I')}{k}) = \max(W(I), \frac{P(I)}{k}) = \eta_d(I)$ Therefore $c(S') = 2\eta_d(I)$. We can now use Lemma 1 to deduce that $c(S') = 2\eta_d(H)$.

Now consider $\eta_d(H)$. We have $\eta_d(H) = \max\left(W(H), \left\lceil\frac{P(H)}{k}\right\rceil\right) \leq \max\left(W(G) + \Delta(G), \left\lceil\frac{P(G)+m(G)}{k}\right\rceil\right)$ since no edge can see its weight increase by more than 1 when building H from G. Therefore $\eta_d(H) \leq \max\left(W(G), \left\lceil\frac{P(G)}{k}\right\rceil\right) + \max\left(\Delta(G), \left\lceil\frac{m(G)}{k}\right\rceil\right) = \eta_d(G) + \eta_s(G) = \eta(G)$. So we finally have $c(S') = 2\eta_d(H) \leq 2\eta(G)$. ■

Theorem 3 *GGP is a 2-approximation algorithm.*

Proof of Theorem 3 We prove that 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: $|S| \times 1 + \sum_{i=1}^{|S|} c(M_i)$. Similarly the cost of S' is: $|S'| \times 1 + \sum_{j=1}^{|S'|} c(M'_j)$. Therefore $c(S) \leq c(S')$. ■

6.2.2 Correctness

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 is proven by Proposition 3. It requires the following lemma.

Lemma 2 *Any perfect matching on J contains k edges belonging to I.*

Proof of Lemma 2 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 $|V_{1_I}| + |V_{2_I}| - k - (|V_{1_I}| - k) - (|V_{2_I}| - k) = k$. ∎

Proposition 3 *$\forall M \in S$, the solution given by GGP, $|M| \leq k$.*

Proof of Proposition 3 We know from Lemma 2 that a perfect matching M on J contains k edges belonging to I. Since I is builded by adding edges from H, M has at most k edges belonging to G. Since S is builded from perfect matchings on J from which all edges not belonging to G are removed we can directly conclude that $|M| \leq k$. ∎

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

Proof of Proposition 4 We can see that steps 1,2,3,4 and 6 of GGP are computed in linear time. However steps 5 requires finding a perfect matching which is done in $O(\sqrt{n(J)}m(J))$ using the hungarian method [21]. At each step we remove at least one edge (the edge of the matching with the lowest weight) hence we iterates at most $m(J)$ times finding matchings. Therefore the worst case complexity of step 5 is in $O(\sqrt{n(J)}m(J)^2)$.

Now to build H no edges or nodes 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 has no sense allowing to select in a matching more edges than node, 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 2 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

GGP is relatively fast, and gives good results. However, it is possible to find a family of graphs on which the 2-approximation ratio may be reached. 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 2-approximation ratio. It should be noted however, that we have no proof that OGGP could have a lower approximation ratio than 2.

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 doesn't 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.

The greedy algorithm depicted in Figure 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.

```
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 $G'$
```

Figure 9: **Algorithm for extracting a matching with maximal minimum weight**

Proposition 5 *Algorithm of Figure 9 returns a matching of maximal minimum weight.*

Proof of Proposition 5 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 $O(m(J) \times \sqrt{n(J)} m(J)) = O(m(J)^2 \sqrt{n(J)})$. 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 2-approximation algorithms as will show the simulation results. The heuristic on degrees

```
Input:  A bipartite graph $G$.
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 $G$.
```

Figure 10: **Heuristic on weights**

is the same as the heuristic on weights 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 $\mathcal{O}(m(G) \times n(G)^{1/2})$ 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 $\mathcal{O}(m(G)^2 \times n(G)^{1/2})$ which is better than the complexity of GGP.

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 Figures 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, even 1.8 for small weights, and the average is always below 2, and even 1.3 in case of large weights. Unfortunately, we did not succeed into giving an evaluation ratio for these two heuristics.

We explain the convex shape of the plots as follows:

- when $k = 1$ the two heuristics obtain the optimal solution which consists in one communication per steps;
- 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.

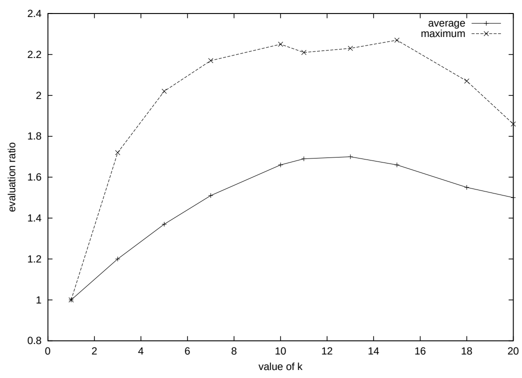

Figure 11: Heuristic on weights. Simulation with small weights.

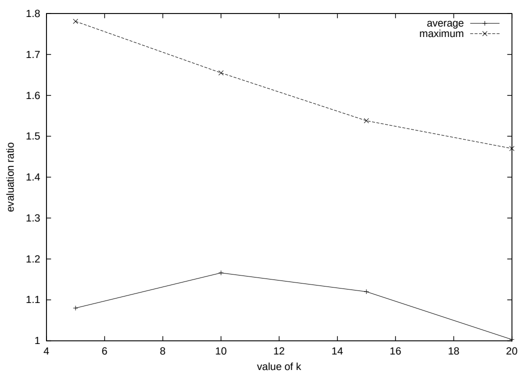

Figure 12: Heuristic on weights. Simulation with large weights.

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.

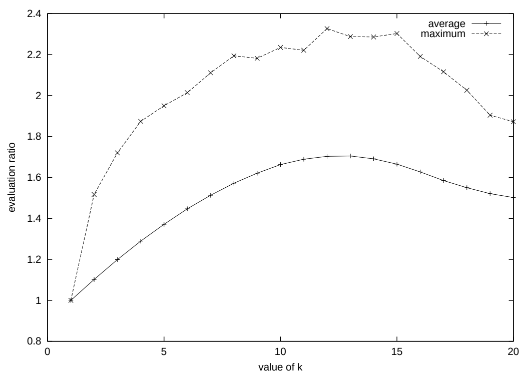

Figure 13: Heuristic on edges. Simulation with small weights.

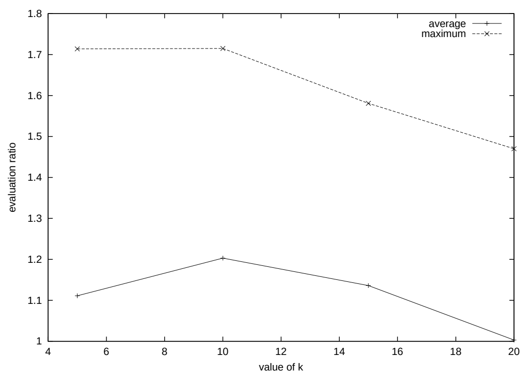

Figure 14: Heuristic on edges. Simulation with large weights.

8.2.1 Comparing GGP and OGGP

Tests on Small Weights. Figure 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. Figure 15 displays how the evaluation ratio varies when k grows. The weights are generated randomly between 1 and 100000.

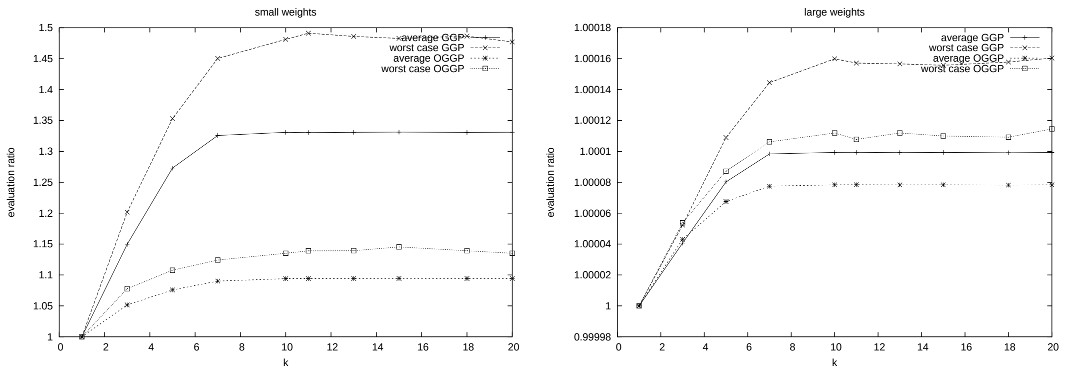

Figure 15: GGP and OGGP for Small Weights / Large Weights graphs

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.

8.2.2 Comparing GGP and Heuristics

We can clearly 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.

When comparing GGP and the heuristics on larger weights the gap between GGP and heuristics widens The huge difference between the algorithm and the heuristics is essentially given by the use of preemption. As the cost of splitting a communication (β) is small when compared to the cost of communications, we can achieve far better results, extremely close to the optimum.

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 outgoing bandwidth of each network card to $\frac{100}{k}$ Mbits per second. This was done using the *rshaper* [28] Linux kernel module. This module implements a software token bucket filter thus enabling a good control of the available bandwidth. We conduct experiments for $k = 3, k = 5, k = 7$.

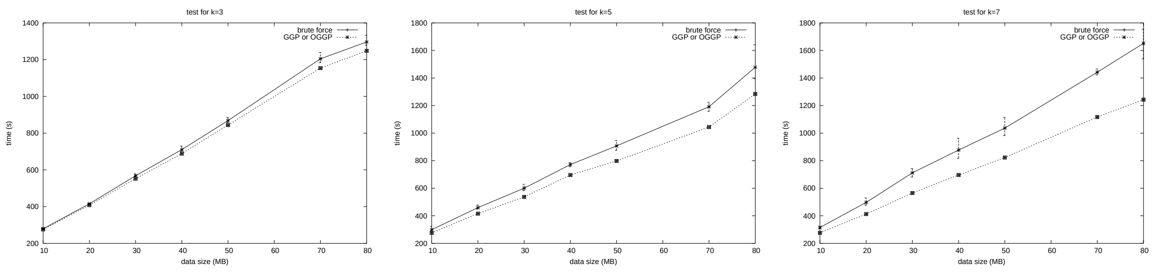

Figure 16: *Brute-Force vs. GGP or OGGP (*$k = 3, 5, 7$*)*

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. 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 Figure 16.

Several observations can be made:

- We achieve a 5% to 20% 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% 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 percents. 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-complete when k is constant. We have studied lower bounds related to KPBS. We have proposed a polynomial time approximation algorithm with ratio 2. We have then proposed an improvement of this algorithm. Two messages scheduling algorithms called GGP and OGGP for the redistribution problem with a lower complexity have been proposed. GGP and OGGP provide a solution at most twice longer than the optimal. We then studied two fast heuristics. 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 in letting the network manage the congestion alone (redistribution time can be reduced to up to 20%).

In our approach we set 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 happens 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 pre-redistribution in case a high-speed local network is available. This would enable 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 multi-step 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 [2]) a fully working redistribution library.

[2]`http://graal.ens-lyon.fr/~desprez/REDGRID`

References

[1] F. N. Afrati, T. Aslanidis, E. Bampis, and I. Milis. Scheduling in switching networks with set-up delays. *J. Comb. Optim.*, 9(1):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. In *IPDPS*, 2005.

[3] P. B. Bhat, V. .K. Prasanna, and C. .S. Raghavendra. Block Cyclic Redistribution over Heterogeneous Networks. In *11th International Conference on Parallel and Distrinuted Computing Systems (PDCS 1998)*, 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.*, 29(5):721–726, 1981.

[5] V. Boudet, F. Desprez, and F. Suter. One-step algorithm for mixed data and task parallel scheduling without data replication. In *IPDPS*, page 41, 2003.

[6] H. Casanova and J. Dongarra. NetSolve: A Network-Enabled Server for Solving Computational Science Problems. *International Journal of Supercomputer Applications and High Performance Computing*, 11(3):212 – 213, Fall 1997.

[7] H. Choi, H.-A. Choi, and M. Azizoglu. Efficient Scheduling of Transmissions in Optical Broadcast Networks. *IEEE/ACM Trans. Net.*, 4(6):913–920, 1996.

[8] P. Crescenzi, D. Xiaotie, and C. H. Papadimitriou. On Approximating a Scheduling Problem. *Journal of Combinatorial Optimization*, 5:287–297, 2001.

[9] B. Del-Fabbro, D. Laiymani, J.-M. Nicod, and L. Philippe. Data management in grid applications providers. In *DFMA*, pages 315–322, 2005.

[10] F. Desprez, J. Dongarra, A. Petitet, C. Randriamaro, and Y. Robert. Scheduling Block-Cyclic Array Redistribution. *IEEE TPDS*, 9(2):192–205, 1998.

[11] F. Desprez and E. Jeannot. Improving the GridRPC Model with Data Persistence and Redistribution. In *3rd International Symposium on Parallel and Distributed Computing (ISPDC)*, Cork, Ireland, July 2004.

[12] S. Even, A. Itai, and A. Shamir. On the complexity of timetable and multicommodity flow problem. *SIAM J. Comput.*, 5:691–703, 1976.

[13] A. Ganz and Y. Gao. A Time-Wavelength Assignment Algorithm for WDM Star Network. In *IEEE INFOCOM'92*, pages 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.*, COM-30(11):2475–2481, November 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 Journal of Supercomputing*, 20(3):243–265, 2001.

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

[19] C.-H. Hsu, Y.-C. Chung, D.-L. Yang, and C.-R. Dow. A generalized processor mapping technique for array redistribution. *IEEE TPDS*, 12(7):743–757, 2001.

[20] Oak Ridge National Labs. Mxn. `http://www.csm.ornl.gov/cca/mxn`.

[21] S. Micali and V. V. Vazirani. An $o(\sqrt(v)e)$ algorithm for finding a maximum matching in general graphs. In *Proc. 21st Ann IEEE Symp. Foundations of Computer Science*, pages 17–27, 1980.

[22] M. Mishra and K. Sivalingam. Scheduling in WDM Networks with Tunable Transmitter and Tunable Receiver Architecture. In *NetWorld+Interop Engineers Conference*, Las Vegas, NV, May 1999.

[23] N. Park, V. K. Prasanna, and C. S. Raghavendra. Efficient algorithms for block-cyclic array redistribution between processor sets. *IEEE TPDS*, 10(12):1217–1239, 1999.

[24] C. Pérez, T. Priol, and A. Ribes. A parallel corba component model for numerical code coupling. In *Proc. of the 3rd International Workshop on Grid Computing*, number 2536 in LNCS, pages 88–99, Baltimore, Maryland, USA, November 2002.

[25] G. R. Pieris and Sasaki G.H. Scheduling Transmission in WDM Broadcast-and-Select Networks. *IEEE/ACM Transaction on Networking*, 2(2), April 1994.

[26] A. Radulescu, C. Nicolescu, A. J. C. van Gemund, and P. Jonker. CPR: Mixed Task and Data Parallel Scheduling for Distributed Systems. In *IPDPS*, page 39, 2001.

[27] N. Rouskas and V. Sivaraman. On the Design of Optimal TDM Schedules for Broadcast WDM Networks with Arbitrary Transceiver Tuning Latencies. In *IEEE INFOCOM'96*, pages 1217–1224, 1996.

[28] A. Rubini. Linux module for network shaping `http://ar.linux.it/software/\#rshaper`.

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

[30] J. Turek, J. Wolf, and P. Yu. Approximate Algorithms Scheduling Parallelizable Tasks. In *Proceedings of the fourth annual ACM Symposium on Parallel Algorithms and Architectures (SPAA'92)*, pages 323–332. ACM Press, 1992.