Two Fast and Efficient Message Scheduling Algorithms for Data Redistribution through a Backbone

Emmanuel Jeannot
LORIA, Université H. Poincaré
Nancy, France
Emmanuel.Jeannot@loria.fr

Frédéric Wagner*
LORIA, INRIA-Lorraine
Nancy, France
Frederic.Wagner@loria.fr

Abstract

In this paper we study the problem of redistributing in parallel data between clusters interconnected by a backbone. This problem is a generalization of the well-known redistribution problem that appears in parallelism [9]. 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). We use the knowledge of the application in order to schedule the messages and perform a control of the congestion by ourselves. Previous results [7, 6] show that this problem is NP-Complete. We propose and study two fast and efficient algorithms for this problem. We prove that these algorithms are 2-approximation algorithms. Simulation results show that both algorithms perform very well compared to the optimal solution. These algorithms have 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.

1 Introduction

In this work we tackle the problem of efficiently scheduling messages when performing data redistribution between two clusters of machines.

This problem appears frequently in the context of grid computing and code coupling. Code coupling applications are composed of at least two codes running on two different parallel machines (or clusters) that are interconnected by a high-speed network (a backbone) [13, 12, 25]. The ability to perform efficiently (as fast as possible) the redistribution is critical for the performance of the whole application.

During the computation, data from one cluster have to be transmitted to another cluster. For instance, some data of the first node of the first cluster might need to be sent to the first node of the second cluster while some other data might need to be sent to another node of the second cluster. Here, no assumption is made on the redistribution pattern and therefore any node of the first cluster can send data of any size to any node of the second cluster.

In the literature a lot of work has been done in order to optimize communications when the redistribution phase is local (on the same cluster/parallel machine) [1, 3, 8, 9, 16].

Some work, like MxN [21], tackle the problem of designing a standard interface for data redistribution between two parallel machines. Cumulvs [15] is a computational steering environment that uses MxN and is able to perform remote visualization across a network. In PAWS (Parallel Application Work Space) [20], code coupling with component model that also uses MxN is addressed. In metachaos [26] the interoperability of parallel libraries is studied as well as the coupling of these libraries. However, these works do not tackle the redistribution problem from the algorithmic point of view as we do here.

In [7, 6], the problem of redistributing data between two clusters interconnected by a backbone is studied. The problem, called K-PBS (K-Preemptive Bipartite Scheduling), is shown to be NP-Complete and a 2-approximation algorithm is provided. However, this algorithm has a very high complexity that makes it useless in practice. Moreover, its design is based on theoretical graph materials that makes it very hard to implement.

The K-PBS problem takes into consideration the throughput of the backbone, as this backbone can be a bottleneck for many reasons. The main problem appears when it is required to send data from a cluster where the sum of the bandwidth of its network cards is greater than the bandwidth of the backbone. The maximum number of communications that can take place at the same time without generating the congestion depends on the characteristics of the platform and is called k.

The contribution of this paper is that we propose two fast and efficient algorithms for scheduling any arbitrary com-

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

munication pattern between two clusters when the backbone is a bottleneck. These algorithms approximate the solution within a factor of two from the optimal solution. They have a low complexity that makes them useful in practice. These algorithms work for any values of k and hence can also be used when the backbone is not a bottleneck. We have compared these two algorithms using random distribution patterns and we show that they perform very efficiently compared 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.

The paper is organized as follow: in Section 2 the problem of data redistribution is presented and all the notations are introduced. The related work appears in Section 3. The algorithms are described in Section 4. Results and experiments are given in Section 5. In Section 6 we state our concluding remarks.

2 Problem of Data Redistribution

2.1 Modelization

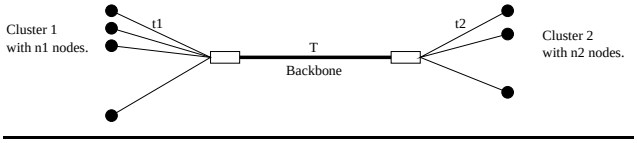

Figure 1. *Architecture for the Redistribution Problem*

We consider the following architecture (see Figure 1). Let there be two clusters $\mathcal{C}_1$ and $\mathcal{C}_2$ with respectively n_1 nodes and n_2 nodes. All the nodes of $\mathcal{C}_1$ have a network card of effective throughput t_1 Mbit/s and all the nodes of $\mathcal{C}_2$ have a network card of effective throughput t_2 Mbit/s [1]. A network, called a backbone, interconnects the two clusters and T denotes its throughput.

Let us suppose that an application is running and using both clusters (for example, a code coupling application). One part of the computation is performed on cluster $\mathcal{C}_1$ and the other part on cluster $\mathcal{C}_2$. During the application, data must be transmitted from $\mathcal{C}_1$ to $\mathcal{C}_2$, following a redistribution pattern. The question is: how to transmit all the data from $\mathcal{C}_1$ to $\mathcal{C}_2$ as fast as possible?

[1] the effective throughput of a network card is the throughput effectively achievable by this card on the computer.

In this paper we suppose that the redistribution pattern is given and computed by the application. We focus on efficiently transmitting the data, not on computing the pattern itself. Hence, the redistribution pattern is modeled by a traffic matrix $M = (m_{i,j})_{i \in [1,n_1], j \in [1,n_2]}$, where $m_{i,j}$ represent the amount of data to be sent from node i of cluster $\mathcal{C}_1$ to node j of cluster $\mathcal{C}_2$.

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 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. **1-port**. 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 message between different pair is called a *step*.

2. **k**. The maximum number of communications that can occur during a step is denoted by k. This number depends mainly on the ratio T/t_1 and T/t_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 T of the backbone. Therefore, k must respect the following equations: (a) $k \times t_1 \leq T$, (b) $k \times t_2 \leq T$ (c) $k \leq n_1$ and (d) $k \leq n_2$.

We denote by t the speed of each communication. For instance, let us assume that $n_1 = 200$, $n_2 = 100$, $t_1 = 10$Mbit/s, $t_2 = 100$Mbit/s and $T = 1$Gbit/s ($T = 1000$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 10 Mbit/s. Hence, $t = 10$ Mbit/s

In order to minimize the overall communication time, we allow preemption. This means that a communication between two given nodes can take place in several steps. In order to do this we need to assume that any part of the data to be transmitted can be sent in any arbitrarily small pieces (larger than 1 byte). In practice the preemption has a cost. We denote by β (called the *setup delay*) the time to setup a step (opening sockets, accessing data, etc.) and to preempt the communication.

We assume that communications steps are synchronous. An other approach using asynchronous communications is possible. In particular, in the developped algorithms the barriers between each communication step can be weakened with some post-processing. However, this is beyond the scope of this paper and will not be discussed here.

2.2 Formulation of the Problem

The problem of the data redistribution is known in the literature as K-PBS (K-Preemptive Bipartite Scheduling) [7, 6]. Here follows a summary of the formulation.

Let us consider the traffic matrix M and t the speed of the communication. One can transform M into a communication matrix $C = (c_{i,j})_{i\in[1,n_1],j\in[1,n_2]}$, such that $c_{i,j} = m_{i,j}/t$ is the *time* to transfer the data from node i to node j when no preemption occurs.

The communication matrix C can be seen as a bipartite graph $G = (V_1, V_2, E, f)$ with $n_1 = |V_1|$ nodes on the left side and $n_2 = |V_2|$ nodes on the right side. Each node of V_1 (resp. V_2) represents a node of cluster $\mathcal{C}_1$ (resp. $\mathcal{C}_2$). There is an edge between two nodes of the graph if there is a non-zero communication in the communication matrix. Finally, for any edge $e = (i, j) \in E$, the weight of this edge is the duration of the communication: $f(e) = c_{i,j}$.

A communication step must respect the two constraints explained above (1-port and at most k communications at a time). Therefore a communication step is modeled by a matching of at most k edges. A matching of a bipartite graph is a bipartite graph with at most one edge adjacent to each given node. Therefore, modeling a communication step by a matching ensures that a node will not send or receive more than one communication during this step. We limit the number of edges in the matching by k such that at most k communications will take place during this step.

Given a matching M_i representing communication step number i, the duration of this step is the duration of the longest communication, which is the largest weight of M_i. We denote the largest weight of matching M_i by $W(M_i)$.

We can formulate the problem K-PBS as follows:

- Let $G = (V_1, V_2, E, f)$ be a weighted bipartite graph, k a positive integer and β a positive rational number. G represents all the communications to perform, k the maximum number of communications during a step and β the startup delay.
- Find a set of s matchings $\{M_1, M_2, \ldots, M_s\}$ such that the maximum number of edges of each matching is bounded by k and $\cup_{i=1}^{s} M_i = G$. Each matching represents a communication step (see Figure 2 for an example).
- Minimize the redistribution time. Since each step has a startup cost β, the objective function to minimize is:

$$\sum_{i=1}^{s}(\beta + W(M_i))$$

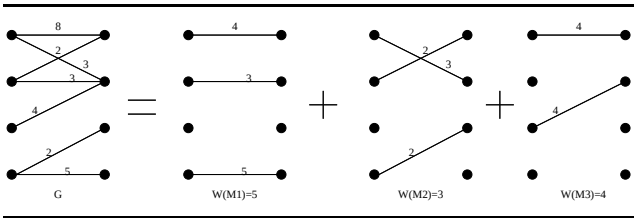

Figure 2. *Example of solution for Graph G, with $k = 3$ and 3 steps. When $\beta = 1$ the total cost of the redistribution is $(1+5)+(1+3)+(1+4) = 15$. Thanks to the preemption the edge of cost 8 is decomposed into 2 edges of cost 4*

2.3 Other Notations

We give here the rest of the notations we use in this paper.

- $m = |E|$ and $n = |V_1| + |V_2|$,
- $\Delta(s)$ is the degree of the node s,
- $P(G) = \sum_{e\in E} f_G(e)$ the sum of all edges weights,
- for any node s we call $w(s)$ the sum of the weights of all edges adjacent to s,
- $W(G) = \max_{s\in V_1\cup V_2}(w(s))$.

2.4 Discussion on the Value of k

When $k = 1$ all the communication have to be done in sequential and therefore the problem is easily solved. On the over hand, one might want to use the algorithms proposed in this paper to schedule the communications for a local redistribution (on the same parallel machine). For

instance, redistribute block-cyclic data from a virtual processor grid to an other virtual processor grid. In that case the maximum number of communications during one step is $\min(n_1, n_2)$, where n_1 (resp. n_2) is the number of nodes of the first (resp. second) virtual processor grid. Therefore, when $k = \min(n_1, n_2)$, the backbone is not a bottleneck and the problem consists in finding a solution for a local redistribution of any kind (block-cyclic to block-cyclic but not only).

3 Related Work

In our previous work [7, 6], we have introduced and studied the K-PBS problem. We have shown that K-PBS is an NP-complete problem and that approximating the K-PBS within a factor better than $4/3$ is impossible unless P=NP. We have proposed a 2-approximation algorithm. However, the time complexity of this algorithm is $\mathcal{O}(kn^{15/2} \times m^3)$. This large complexity makes it useless in practice. Moreover, this algorithm is based on very theoretical graph materials that make it very hard to implement. In [7, 6], a lower bound on the optimal solution of K-PBS is provided. This lower bound is based on the structure of the input graph, the value of k and β. We will use this lower bound in our simulation experiments to tell how far solutions of our algorithms are from the optimal solution.

Up to our knowledge, there is no other work on the K-PBS problem in its generality ($n_1 \neq n_2$ and k can have any value, etc.).

This problem has been partially studied in the context of Satellite-Switched Time-Division Multiple access systems (SS/TDMA) [4, 17, 18]. In [4], the problem with $\beta = 0$ is studied and an optimal algorithm with O(mn) steps is described. In [17] an optimal algorithm that finds the minimal number of steps is described. In [18] the problem without preemption is studied. It is shown NP-Complete and a heuristic is given.

The K-PBS problem partially falls in a field originated by packet switching in communication systems for optical network called wavelength-division multiplexed (WDM) broadcast network [5, 14, 23, 24, 27]. The problem of minimizing the number of steps is studied in [14, 17], and the problem of minimizing the total cost is studied in [23]. In [5] and in [24], the author consider a version of the K-PBS 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 [24] 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 [5].

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 [11, 18]. Approximating the PBS problem within a ratio number smaller than $\frac{7}{6}$ has been proven impossible unless $P =$ 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 the context of block cyclic redistribution many works exist (see [3, 9] for example). In this case the backbone is not a constraint and the redistribution pattern is not arbitrary. Hence, this problem is less general than K-PBS.

4 Algorithms

In this section, we present two 2-approximation algorithms with increasing complexity and efficiency.

4.1 WRGP Algorithm

In order to present our first algorithm called Generic Graph Peeling algorithm (GGP), we present a simple version of it, working only on weight-regular graphs and called Weight-Regular Graph Peeling algorithm (WRGP).

We suppose that k is not bounded and that the bipartite graph G, given as input, is weight-regular: the sum of the weights of all edges adjacent to a node is the same for any node in the graph. This means that each cluster node has the same amount of data to send or receive.

We are trying to waste the least amount of bandwidth at each communication step. In order to do that, for each step, we need two things:

- Issue a maximal number of communications,
- ensure all communications have the same size.

A perfect matching of a bipartite graph is a subgraph that pairs every vertex with exactly one other vertex. Therefore, if we have a communication step given by a perfect matching where all edges have the same weight, we know that this step is optimal. Of course, the whole scheduling may not be optimal, but at least during this step, the full bandwidth is being used.

Since a perfect matching can be found using the Hungarian Method [22], the problem is ensuring that all edges have the same weight. To achieve that, we use preemption. We cut each edges weights to the weight of the smallest edge in the matching.

Finally, the algorithm is based on the following property: in any weight-regular bipartite graph, there exists a perfect matching [8].

1. find a perfect matching M
2. take w the smallest weight of the edges of M
3. build a perfect matching M' such that
   - all edges of M' are also in M
   - all edges of M' are of weight $w = W(M)$

   and add it to the sets of communication
4. substract M' from the bipartite graph. If an edge reaches a weight of 0 remove it from the graph
5. start again at step 1, until the graph becomes empty

Figure 3. *WRGP algorithm*

We use this property in the WRGP algorithm depicted Figure 3. This algorithm is working because each time we modify the bipartite graph, it stays weight-regular as the matching M' is a perfect matching and all its edges have the same weight w. A small example of this is shown in Figure 4.

With this algorithm, we ensure that each communication step is using the full bandwidth.

Complexity of WRGP At each iteration, at least one edge is removed. This means we have at most m iterations. One of the best algorithms to find a perfect matching [22] has a complexity of $O(m\sqrt{n})$ hence the complexity of WRGP is $O(m^2\sqrt{n})$. However WRGP is independent from the matching algorithm chosen. On some cases it could be better to use the $O(n^{\frac{3}{2}}\sqrt{\frac{m}{\log(n)}})$ matching algorithm developed in [2].

4.2 GGP Algorithm

The WRGP algorithm works only for weight-regular graphs and does not take into account our β parameter. We explain here how to extend the WRGP algorithm into a general case algorithm called GGP.

4.2.1 Communication Cost

In WRGP, we make active use of preemption to cut large communications into smaller chunks. However, preemption comes with a cost. As each step of communication costs a constant time β to establish the connection we need to make sure that the costs of preemption are never higher than the possible benefits.

It is difficult to ensure that preemption is never used too much or too little. We have chosen the following: we will not split communications of duration less than β. By doing that, we may find some cases where we are suboptimal, but we limit the number of communication steps in the general case. This fact has been taken into account in the 2-approximation proof. A set of suboptimal examples reaching the approximation ratio of 2 may be found in [19].

Implementing this constraint in the previous algorithm is straightforward. We just have to normalize all weights by β and round them to the upper integer at the beginning.

As the size of each communication step is given by the size of the smallest communication in the corresponding matching, it will never be smaller than 1, since all weights have been rounded.

4.2.2 k Constraint, Building Weight-regular Graphs

In this section we show how to transform any bipartite graph into a weight-regular graph such that any perfect matching of the weight-regular graph has at most k edges belonging to the original graph.

Two different cases have to be distinguished:

1. $W(G) \leq \frac{P(G)}{k}$ and $k|P(G)$.

   In this case we build $G' = (V_1', V_2', E', f_G)$ a $\frac{P(G)}{k}$-weight-regular graph by adding $|V_2| - k$ nodes to V_1, $|V_1| - k$ nodes to V_2 and some edges.

   Remark that $\Sigma_{s \in V_1} w(s) = P(G)$. In order to have a $\frac{P(G)}{k}$-weight-regular graph we need $\Sigma_{s \in V_1} w(s) = \frac{P(G)}{k}|V_1|$. This means, we need to add edges of total weight $\frac{P(G)}{k}|V_1| - P(G) = P(G)(\frac{|V_1|}{k} - 1) = \frac{P(G)}{k}(|V_1| - k)$. Therefore we add $|V_1| - k$ nodes to V_2, each of weight $\frac{P(G)}{k}$. We do the same for all nodes in V_2. It is important to note that the new edges are never connecting two new nodes. A detailed description of this part can be found in [19].

Proposition 1 *Any perfect matching of G' has k edges belonging to G*

Proof of proposition 1 Each perfect matching of G' has $|V_1'| = |V_2'| = |V_1| + |V_2| - k$ edges. On this matching we have $|V_1| - k$ (nodes added to V_2) $+|V_2| - k$ (nodes added to V_1) edges not belonging to G. This means we have: $|V_1| + |V_2| - k - (|V_1| - k + |V_2| - k) = k$ edges belonging to the original G graph. ∎

2. $W(G) > \frac{P(G)}{k}$ or $W(G) \leq \frac{P(G)}{k}$ and $k \not| P(G)$

   In that case we add new edges (each one connecting two new nodes) to build G' such that: $\frac{P(G')}{k} = W(G)$

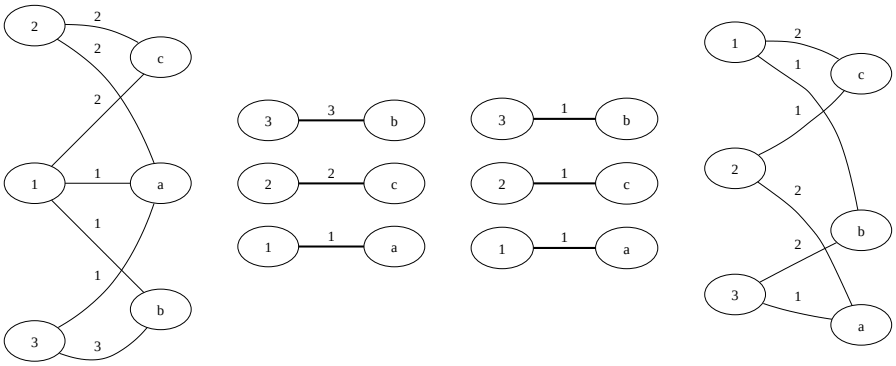

Figure 4. *WGRP. From left to right: the weigth-regular graph at iteration i, the matching M, the matching M' the weigth-regular graph at iteration $i+1$*

if $W(G) > \frac{P(G)}{k}$ or $k|P(G')$ if $W(G) \leq \frac{P(G)}{k}$ and $k \not| P(G)$. All edges added except the last one have the largest possible weight: $W(G)$. We call the resulting graph G'. After that, we refer to the first case and apply the above technique to build a weight-regular graph G''. The proposition 1 shows that any perfect matching of G'' has k edges belonging to G' and therefore at most k edges belonging to G.

In this case there may be less than k edges from G in a perfect matching on G''. In particular when k (the number of allowed simultaneous communications) is greater than the total number of communications, it is clear that we will issue less than k communications. However in such cases, there will be less than k communications even in the optimal scheduling.

4.2.3 Algorithm

We can now extend WRGP for general bipartite graphs, and respect our constraints. We present in Figure 5 the Generic Graph Peeling algorithm (GGP) that gives a solution to the K-PBS problem.

Input: $G = (V_1, V_2, E, f_G)$ the weighted bipartite graph, k an integer, β a rational.

Output: R a set of matchings, a solution of K-PBS.

1. Normalize weights by β, round them to the upper integer,
2. add nodes and edges to G to build a weight-regular graph J,
3. apply algorithm WRGP to J,
4. extract R from the solution given by WRGP.

Figure 5. *GGP Algorithm*

4.2.4 Complexity

Step 2 has added two kind of edges to G. (1) Edges of weight $W(G)$. These edges are all removed at the last iteration. (2) Edges added to turn G into a weight-regular graph. There is at most $2n$ edges of this type. Therefore we have at most $m + 2n + 1$ iterations to peel J with WRGP. Hence the complexity of GGP is $O((m+n)^2\sqrt{n})$.

4.2.5 Approximation Factor

Theorem 1 *GGP is a two approximation algorithm.*

Sketch of proof In [7, 6], Cohen, Jeannot and Padoy introduced a 2–approximation algorithm for K-PBS. Any solution build by GGP can also be found by this algorithm. Therefore, GGP is a 2–approximation algorithm. A complete proof is given in [19]. ∎

4.3 OGGP Algorithm

With the GGP algorithm we just introduced, it is possible to find cases where the approximation ratio of 2 is neared. We have enhanced the algorithm to obtain better results on these cases. The second algorithm, called Optimized Generic Graph Peeling algorithm (OGGP), we are presenting in this section is an extension of the previous one. It is a 2-approximation algorithm as any solution found

by OGGP can also be found by GGP. However, we have not been able to find a set of cases reaching the approximation ratio.

One part of the GGP algorithm has not been optimized: the choice of the matching. Indeed, any matching-algorithm can be used. It is possible to use a more intelligent approach in order to bind the matching-algorithm with our specific problem. Intuitively we want to have the least number of communication steps. This means, we should try to find the largest possible communication steps. As the size of a step is given by the smallest communication in the matching, a good strategy would be to find a matching with the smallest weight on the edges being as large as possible. The algorithm depicted in Figure 6 finds a maximal matching which smallest edge's weight is maximum. It is based on the algorithm described in [4] that maximizes the minimum weight of a matching.

1. $G' = \emptyset, M = \emptyset, G'' = G$
2. while M is not maximal in G do:
   (a) choose $e \in E(G'')|\forall e' \in E(G''), f(e) \geq f(e')$
   (b) $E(G') = E(G') \cup e$
   (c) $E(G'') = E(G'') \backslash e$
   (d) M = a maximal matching in G'

Figure 6. *Algorithm for extracting a matching that maximizes the minimal weight*

Proof the algorithm of Fig 6 returns the desired matching. If we call l 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 maximal matching in G. We also have $\forall e \in G, f(e) > f(l) \Rightarrow e \in G'$. 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, so M is a maximal matching maximizing the minimum weight. ∎

Complexity This algorithm complexity is $O(m \times \sqrt{n}m) = O(m^2\sqrt{n})$. Therefore, the complexity of OGGP is $O((m+n)^3\sqrt{n})$

5 Experiments

5.1 Simulations

The GGP and OGGP algorithms have been implemented in a bipartite graphs library we developed. Moreover, for any given graph, it is possible to compute the minimum time required to redistribute the data. This is based on the lower bound of K-PBS described in [7, 6].

In each following simulation we generate some random bipartite graphs, execute GGP and OGGP, and we plot the ratio between the duration of the found solution and the duration given by the lower bound. We call this ratio the evaluation ratio. Hence, the closest to 1 is the evaluation ratio, the best is the found solution. The computation times of the different algorithms are not taken into account here, as they are all under one second. The graph are generated with a random number of nodes (up to 40) and a random number of edges (up to 400).

The graph of Figure 7 is showing how the ratio is changing as k increases. Each point of the graph is obtained by running 100000 simulations and taking the average or the maximal ratio. The weights of each edge is uniformly generated between 1 and 20 and β is set to 1.

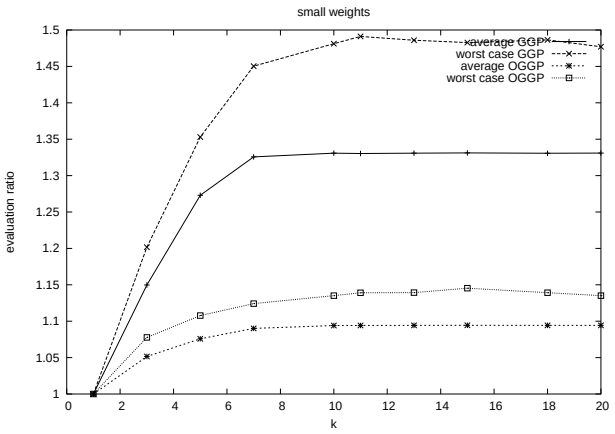

Figure 7. *Evaluation Ratios for Small Weights*

We see OGGP is giving far better results than GGP with its worst-case graph below GGP's average case. We should also note that the worst evaluation ratio reached is 1.15, well below 2.

On the graph of Figure 8, we conduct the same experiment, but with this time some more realistic values for the weights. We consider an amount of data to transfer far greater than β and generate weights of up to 10000.

This graph seems similar to the previous one, but the scale is not the same. The worst ratio reached is 1.00016, very close to the best achievable ratio. This means that for

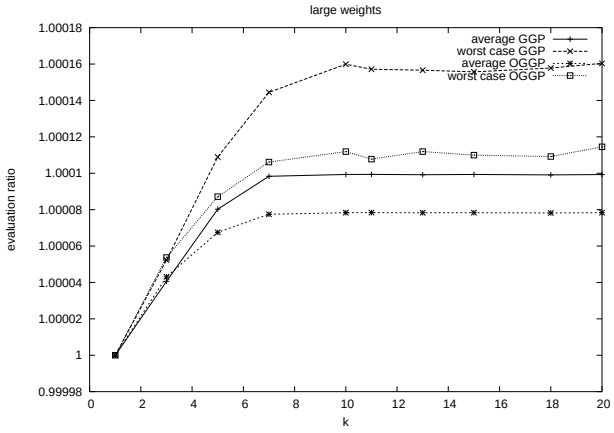

Figure 8. *Evaluation Ratios for Large Weights*

this kind of input, GGP and OGGP will behave in an identical manner, and that it will be difficult to find a better algorithm.

Finally we conducted some experiments on the cases where the weights are small (between 1 and 20) and $beta$ increases. For each input, k is randomly generated. The results are shown in Figure 9.

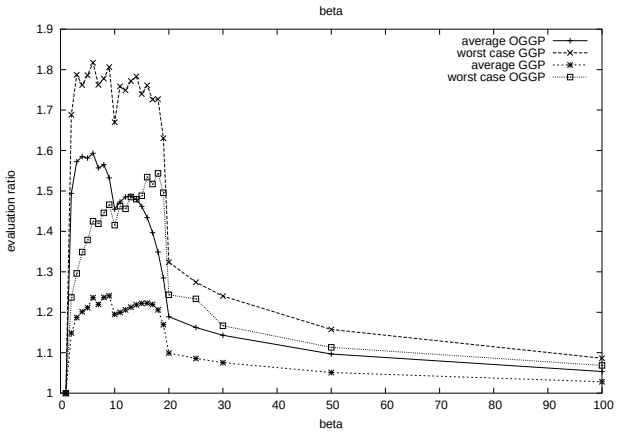

Figure 9. *Evaluation Ratios when β Increases*

We see that, with β smaller than the weights, evaluation ratios as high as 1.8 for GGP and 1.6 for OGGP are reached. After that, ratios drop quickly because the optimal cost rises with β. Note that OGGP has an average ratio of 1.2.

5.2 Real-World Experiments

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 conducted experiments for $k = 3, k = 5, k = 7$. Only $k = 3$ and $k = 7$ results are shown here.

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 [10]. 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 10 and 11.

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.

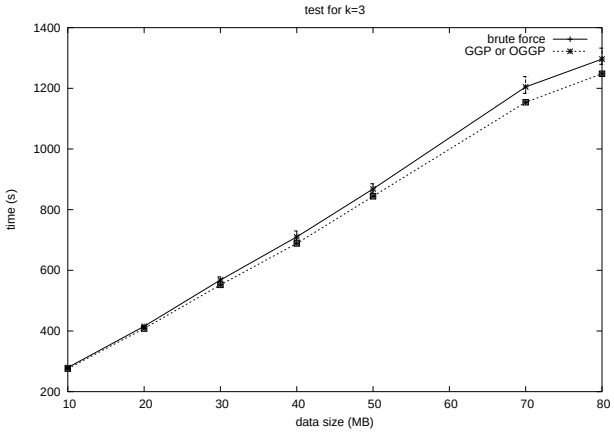

Figure 10. *Brute-Force vs. GGP or OGGP ($k = 3$)*

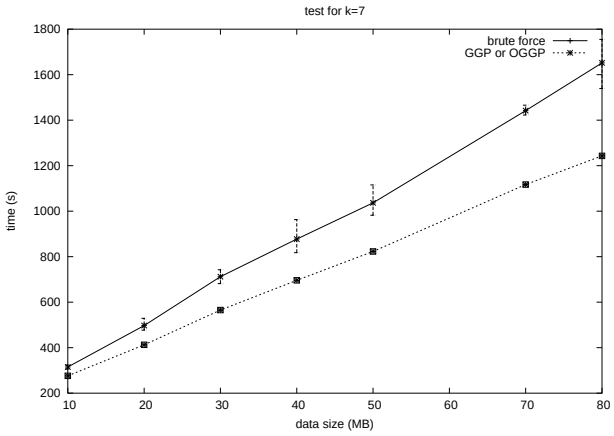

Figure 11. *Brute-Force vs. GGP or OGGP ($k = 7$)*

- As the available bandwidth decreases (i.e. k increases) we increase the benefits of using GGP or $OGGP$ over the brute-force approach.

6 Conclusion

In this paper we have studied and proposed two scheduling messages algorithms called GGP and OGGP for the redistribution problem. The novelty of our approach is that 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 redistribution happen on the same parallel machines or in the context of SS/TDMA systems or WDM network.

Our contribution is the following:

- The proposed algorithms have a low complexity and provide solution at most twice longer than the optimal one,
- simulation experiments show that OGGP outperform GGP due to an internal optimization when building each communication step. Moreover, these two algorithms find solutions very close to the optimal one when communications are long,
- 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 managing the congestion alone (redistribution time can be reduced to up to 20%).

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 allow 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) a fully working redistribution library.

References

[1] F. Afrati, T. Aslanidis, E. Bampis, and I. Milis. Scheduling in switching networks with set-up delays. In *AlgoTel 2002*, Mèze, France, May 2002.

[2] H. Alt, N. Blum, K. Melhorn, and M. Paul. Computing of maximum cardinality matching in a bipartite graph in time $O(n^{\frac{3}{2}}\sqrt{\frac{m}{\log(n)}})$. *Inf. Proc. Letters*, 37:237–240, 1991.

[3] P.B. Bhat, V.K. Prasanna, and C.S. Raghavendra. Block Cyclic Redistribution over Heterogeneous Networks. In *11th Int. Conf. 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 Transactions on Communications*, 29(5):721–726, 1981.

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

[6] J. Cohen, E. Jeannot, and N. Padoy. Messages Scheduling for Data Redistribution between Clusters. In *HeteroPar'03 workshop of SIAM PPAM 2003*, Czestochowa, Poland, September 2003. To appear in LNCS series.

[7] J. Cohen, E. Jeannot, and N. Padoy. Parallel data redistribution over a backbone. Technical Report RR-4725, INRIA-Lorraine, February 2003.

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

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

[10] N. Doss, W. Gropp, E. Lusk, and A. Skjellum. A High-Performance, Portable Implementation of the MPI Message Passing Interface Standard. Technical report, Argonne National Laboratory, 1996.

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

[12] I. Foster, J. Geisler, W. Gropp, N. Karonis, E. Lusk, G. Thiruvathukal, and S. Tuecke. Wide-area implementation of the Message Passing Interface. *Parallel Computing*, 24(12–13):1735–1749, 1998.

[13] Fraunhofer Institute Algorithms and Scientific Computing (SCAI). Mpcci - mesh-based parallel code coupling interface. `http://www.mpcci.org`.

[14] A. Ganz and Y. Gao. A Time-Wavelength Assignment Algorithm for WDM Star Network. In *IEEE INFOCOM'92*, pages 2144–2150, 1992.

[15] G.A. Geist, J.A. Kohl, and P.M. Papadopoulos. CUMULVS: Providing Fault-Tolerance, Visualization and Steering of Parallel Applications. *International Journal of High Performance Computing Applications*, 11(3):224–236, August 1997.

[16] A. Goldman, D. Trystram, and J. Peters. Exchange of Messages of Different Sizes. In *Solving Irregularly Structured Parblems in Parallel*, number 1457 in LNCS, pages 194 – 205. Springer Verlag, August 1998.

[17] 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 Transactions on Communications*, COM-30(11):2475–2481, November 1982.

[18] I.S. Gopal and C.K. Wong. Minimizing the number of switching in an ss/tdma system. *IEEE Trans. on Communications*, 33:497–501, 1885.

[19] E. Jeannot and F. Wagner. Message scheduling for data redistribution through high performance networks. Technical report, INRIA, 2004.

[20] K. Keahey, P. Fasel, and S. Mniszewski. PAWS: Collective Interactions and Data Transfers. In *10th IEEE Intl. Symp. on High Perf. Dist. Comp. (HPDC-10'01)*, pages 47–54, August 2001.

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

[22] S. Micali and 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.

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

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

[25] C. Pérez, T. Priol, and A. Ribes. A parallel corba component model for numerical code coupling. In Manish Parashar, editor, *Proc. of the 3rd International Workshop on Grid Computing*, number 2536 in LNCS, pages 88–99, Baltimore, Maryland, USA, November 2002. Springer-Verlag. In conjunction with *SuperComputing 2002* (SC'02).

[26] M. Ranganathan, A. Acharya, G. Edjlali, A. Sussman, and J. Saltz. Runtime coupling of data-parallel programs. In *Proceedings of the 1996 International Conference on Supercomputing*. ACM Press, May 1996.

[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`.