On the Importance of Bandwidth Control Mechanisms for Scheduling on Large Scale Heterogeneous Platforms

Olivier Beaumont, Hejer Rejeb

On the Importance of Bandwidth Control Mechanisms for Scheduling on Large Scale Heterogeneous Platforms

Olivier Beaumont, Hejer Rejeb
INRIA Bordeaux – Sud-Ouest
University of Bordeaux, LaBRI
Bordeaux, France

***Abstract*—We study three scheduling problems (file redistribution, independent tasks scheduling and broadcasting) on large scale heterogeneous platforms under the Bounded Multi-port Model. In this model, each node is associated to an incoming and outgoing bandwidth and it can be involved in an arbitrary number of communications, provided that neither its incoming nor its outgoing bandwidths are exceeded. This model well corresponds to modern networking technologies, it can be used when programming at TCP level and is also implemented in modern message passing libraries such as MPICH2. We prove, using the three above mentioned scheduling problems, that this model is tractable and that even very simple distributed algorithms can achieve optimal performance, provided that we can enforce bandwidth sharing policies. Our goal is to assert the necessity of such QoS mechanisms, that are now available in the kernels of modern operating systems, to achieve optimal performance. We prove that implementations of optimal algorithms that do not enforce prescribed bandwidth sharing can fail by a large amount if TCP contention mechanisms only are used. More precisely, for each considered scheduling problem, we establish upper bounds on the performance loss than can be induced by TCP bandwidth sharing mechanisms, we prove that these upper bounds are tight by exhibiting instances achieving them and we provide a set of simulations using SimGRID to analyze the practical impact of bandwidth control mechanisms.**

I. Introduction

We consider three different problems to assess the impact of bandwidth sharing mechanisms on the performance of scheduling algorithms in large scale distributed platforms.

The first scheduling problem we consider (see Section II-A) arises in the context of large scale distributed storage systems such as Vespa [4], developped by Yahoo!, when systems reconfiguration take place. In this paper, following the work proposed in [6], we consider the case where one disk is added to the system. In the case of Vespa, the storage system replicates the data in order to tolerate component failures and the placement of data replicas on resources is enforced by external mechanisms based on CRUSH [28].

The second scheduling problem (see Section II-B) is related to independent tasks scheduling. We assume that initially, a single node (the master) holds or generate a large amount of independent equal-sized tasks, such as in volunteer computing applications run on platforms like BOINC [1] or Folding@home [18]. These tasks will be processed by slave nodes, whose both communication (in terms on latencies and bandwidths) and computation capabilities are strongly heterogeneous. In the context of volunteer computing applications, the number of tasks to be processed is huge so that makespan minimization does not make sense. Therefore, we rather consider throughput maximization, where the aim is to maximize the number of tasks that can be processed within one time unit once steady state has been reached, as advocated in [5].

The third scheduling problem we consider (see Section II-C) is related to broadcasting a large size message. Broadcasting in computer networks is the focus of a vast literature [17], [27], [26]. The one-to-all broadcast, or single-node broadcast, is the most primary collective communication pattern: initially, only the source processor holds (or generate) the data that needs to be broadcast; at the end, there is a copy of the original data residing at each processor. Parallel algorithms often require to send identical data to all other processors, in order to disseminate global information (typically, input data such as the problem size or application parameters). The same framework applies for broadcasting a live stream of data, such as a movie. In this paper, we concentrate on a simple scenario, where the nodes are organized as a star platform (the source node being at the center), and where all the communications take place directly between the source node and the clients.

Since we target large scale distributed platforms, we do not assume that the topology of the platform is known in advance, since automatic discovery mechanisms such as ENV [25] or AlNEM [12] are too slow to be used in large scale dynamic settings. Therefore, we rather associate to each node local properties (namely its

incoming and outgoing bandwidths and its processing capability), whose values can easily be determined at runtime. Thus, the network topologies we consider are rather logical overlay networks rather than physical networks.

To model contentions, we rely on the bounded multi-port model, that has already been advocated by Hong et al. [15] for independent task distribution on heterogeneous platforms. In this model, node P_i can serve any number of clients P_j simultaneously, each using a bandwidth $b_{i,j}$ provided that its outgoing bandwidth is not exceeded, *i.e.*, $\sum_j b_{i,j} \leq B_i^{\text{out}}$. Similarly, P_j can simultaneously receive messages from any set of clients P_i, each using a bandwidth $b_{i,j}$ provided that its incoming bandwidth is not exceeded, *i.e.*, $\sum_i b_{i,j} \leq B_j^{\text{in}}$. This corresponds well to modern network infrastructure, where each communication is associated to a TCP connection.

This model strongly differs from the traditional one-port model used in the scheduling literature, where connections are made in exclusive mode: the server can communicate with a single client at any time-step. In the context of large scale platforms, the networking heterogeneity ratio may be high, and it is unacceptable to assume that a 100MB/s server may be kept busy for 10 seconds while communicating a 1MB data file to a 100kB/s DSL node. In the context of large scale distributed platforms, we will assume that all connections are directly handled at TCP level. It is worth noting that at TCP level, several QoS mechanisms such as qdisc, available in modern operating systems, enable a prescribed sharing of the bandwidth [7], [16]. In particular, it is possible to handle simultaneously several connections and to fix the bandwidth allocated to each connection. In our context, these mechanisms are particularly useful since in optimal schedules, the bandwidth allocated to a connection between P_i and P_j may be lower than both B_i^{out} and B_j^{in}. Therefore, the model we propose encompasses the benefits of both bounded multi-port model and one-port model. It enables several communications to take place simultaneously, what is compulsory in the context of large scale distributed platforms, and practical implementation is achieved using TCP QoS mechanisms.

We prove, using the three above mentioned scheduling problems, that this model is tractable and that simple distributed algorithms can achieve optimal performance, provided that we enforce bandwidth sharing policies. Our goal is to assert the necessity of such QoS mechanisms to obtain a prescribed share of bandwidths, that are now available in the kernels of modern operating systems. More precisely, we prove that implementations of optimal algorithms that do not enforce prescribed bandwidth sharing can fail by a large amount if TCP contention mechanisms are used. This result is asserted both by providing theoretical worst cases analysis and through simulations using SimGRID.

For the sake of simplicity, all the applications we consider are based on the master-worker paradigm. Therefore, their implementations are not fully distributed since it is assumed that the master node knows about the characteristics of all slaves. Nevertheless, in a more realistic context, the basic knowledge each node must have in order to implement the algorithms proposed in this paper is the state and characteristics of its neighbours and in this sense, the algorithms we propose are distributed algorithms. Distributed implementations for the independent tasks scheduling problem can be derived from multi-commodity flow algorithms proposed in [2], [3]. Similarly, in the case of the broadcast application, a randomized distributed implementation has recently been proposed in [21]. On the other hand, we are not aware of any distributed implementation of the data redistribution scheduling problem.

To assert the importance of bandwidth sharing mechanisms, we propose for each of the scheduling problems mentionned above two different implementations. For each problem, the first implementation does not make use of sophisticated QoS mechanisms for bandwidth sharing mechanisms whereas the second does. Nevertheless, it is worth noting that both implementations rely on the same knowledge and therefore that the comparison between both implementations is fair.

The rest of the paper is organized as follows. In Section II, we formalize the scheduling problems we consider and we describe how to model the kind of fairness TCP implements in presence of contentions. In Sections III, IV and V, we study the maximal performance loss that can be induced by TCP bandwidth sharing mechanisms in presence of contentions. More precisely, for each scheduling problem we consider, *i.e.* File Redistribution (Section III), Independent Tasks Scheduling (Section IV) and Broadcasting (Section III), we establish upper bounds on the performance loss induced by TCP bandwidth sharing mechanisms, we prove that these upper bounds are tight by exhibiting instances achieving these bounds and we provide a set of simulations to analyze the practical importance of bandwidth control mechanisms. At last, we provide in Section VI some future works and concluding remarks.

II. Problems and Communication Modeling

A. Data Redistribution

In the context of large scale distributed storage systems such as Vespa [4], developed by Yahoo!, we consider the case where a disk is added to the system. In Vespa, the set of files that should be transfered to the added disks is known in advance and the scheduling problem consists in finding for each file, among the existing replicas of it, the one that should be used for the transfer so as to minimize to overall completion time. Let us denote by $S = \{F_1, F_2, \ldots, F_k\}$ denote the set of files that should be transfered to destination node D and let us denote by x_k^i the indicator function so that $x_k^i = 1$ if source node S_i holds a replica of file F_k and 0 otherwise. The size of file F_k is denoted by s_k. In order to make the problem tractable, we assume that a given file can be sent partially from several source nodes (otherwise, the problem becomes NP-Complete and is analyzed in [6]). Let us also denote by B^{in} the incoming bandwidth at destination node D and by B_i^{out} the outgoing bandwidth at source node D_i. The following linear program provides a lower bound for the time necessary to complete all transfers

$$\text{Minimize } T \text{ subject to}$$
$$\begin{cases} \forall i,k & z_k^i \leq x_k^i B_i^{\text{out}} \text{ and } z_k^i \geq 0 \\ \forall k, & \sum_i z_k^i = s_k \\ \forall i & \sum_k z_k^i \leq B_i^{\text{out}} T \\ & \sum_i \sum_k z_k^i \leq B^{\text{in}} T \end{cases},$$

where z_k^i denotes the size of the part of file F_k transfered from S_i to D.

Clearly, any solution (if we average bandwidth usages over time) must satisfy above conditions, so that the optimal value T_{opt} of the linear program is a lower bound on the achievable makespan. On the other hand, let us consider an implementation such that each source disk S_i sends a part of file F_k to D at constant rate $\frac{z_k^i}{T_{\text{opt}}}$. Such an implementation would achieve all file transfers by time T_{opt}.

We will show in Section III how to achieve optimality using bandwidth control mechanisms and prove that without such a mechanism, *i.e.* relying only on TCP contention mechanisms, the performance of such an implementation may be as bad as $2T_{\text{opt}}$.

B. Independent Tasks Scheduling

We consider an elementary master-slave platform to process a huge number of independent equal-sized tasks. Initially, the master node M holds (or generate at a given rate) a large number of tasks that will be processed by a set of slave nodes P_i. The master node is characterized by its outgoing bandwidth B^{out} whereas a slave node P_i, $1 \leq i \leq N$ is characterized by both its incoming bandwidth B_i^{in} and its processing capability w_i. Since all tasks are equal-sized, we normalize all B^{out}, B_i^{in} and w_i in terms of tasks (transmitted or processed) per time unit. Let us consider the following linear program

$$\text{Maximize } \rho_{\text{opt}} = \textstyle\sum_i \rho_i \text{ subject to}$$
$$\begin{cases} \forall i & \rho_i \leq \min(B_i^{\text{in}}, w_i) \text{ and } \rho_i \geq 0 \\ & \sum_i \rho_i \leq B^{\text{out}} \end{cases},$$

where ρ_i denotes the number of tasks that the master node delegates to P_i per time unit. We formulate the optimization problem as a linear program for the sake of generality, since this approach can be extended to more complicated platforms than star-shaped platforms. Nevertheless, in the case of a star platform, the optimal throughput and the fraction of tasks allocated to each slave processor can be determined in linear program by setting

$$\begin{cases} \rho_i' = \min(B_i^{\text{in}}, w_i) \\ \alpha = \min\left(1, \frac{B^{\text{out}}}{\sum_i \rho_i'}\right) \\ \rho_i = \alpha \rho_i' \end{cases}$$

If we consider any valid solution of the independent tasks scheduling problem over a long time period T and if we denote by x_i the average number of tasks processed P_i per time unit, *i.e.* $x_i = N_i(T)/T$, then the x_is satisfy the conditions of the linear program, so that $\sum_i x_i \leq \rho_{\text{opt}}$ and $\sum_i N_i(T) \leq \rho_{\text{opt}} T$, what proves that ρ_{opt} is an upper bound on the achievable throughput. On the other hand, let us consider a solution where the master node continuously sends tasks to P_i at rate ρ_i and tasks are immediately processed by P_i. Since the conditions of the linear program are satisfied, after an initialization phase whose duration is a constant and that corresponds to the necessary time for all the slaves to receive their first task, this solution is valid and processes $\sum_i \rho_i$ tasks per time unit. Therefore, if we consider an arbitrarily large execution time, then the duration of the initialization phase can be neglected and the achieved throughput tends to ρ_{opt}. We will show in Section IV how to achieve optimality using bandwidth control mechanisms and prove that without such a mechanism, *i.e.* relying only on TCP contention mechanisms, the performance of such an implementation may be as bad as $3/4\rho_{\text{opt}}$.

C. Broadcasting

In the broadcast setting, a source node S holds (or generate at a given rate) a large file that must be sent to all client nodes. Numerous broadcast algorithms have been designed for parallel machines such as meshes, hypercubes, and variants (see among others [17], [27]). In the context, of content distribution systems, it is at the core of live streaming distribution systems such as CoolStreaming [29] or SplitStream [10]. In both cases, we are interested in the distribution of a large message to all the nodes of a large scale platform. Thus, we are not interested in minimizing the makespan for a given message size but rather to maximize the throughput (*i.e.* the maximum broadcast rate, once steady state has been reached).

In this context, the source node S is characterized by its outgoing bandwidth B^{out} whereas a client node P_i is characterized by both its incoming bandwidth B_i^{in} and its outgoing bandwidth B_i^{out} since it may be used as an intermediate source once it has received some part of the message. In the most general case, the goal is to design an overlay network $G = (P, E, c)$ such that P_i sends messages to P_j at rate $c(P_i, P_j)$.

The optimal broadcast rate on G can be characterized using flows. Indeed, theorems [11], [13] relate the optimal broadcast rate with the minimum source-cut of a weighted graph. $\forall j$, we can denote as $\text{cut}(j)$ the minimum value of a cut of G into two set of clients C_1 and C_2 such that $C_1 \bigcup C_2 = P$, $S \in C_1$ and $P_j \in C_2$. $\forall j,\ \text{cut}(j)$ denotes the maximal value of a flow between the source node S and P_j and therefore represents an upper bound of the broadcast rate. Moreover, it is proven in [11] that this bound is actually tight, *i.e.* that the optimal broadcast rate for graph G is equal to $\text{mincut}(G) = \min_j \text{cut}(j)$. Efficient algorithms [13] have been designed to compute the set of weighted trees that achieve this optimal broadcast rate from $c(P_i, P_j)$ values.

Therefore, we can use the linear programming approach proposed in proposed in [19] to compute the optimal broadcast rate $\rho*$ and $\forall i, j,\quad c(P_j, P_i)$, the overall bandwidth used between nodes P_j and P_i. Once all $c(P_i, P_j)$ values have been determined, Massoulié et al. [21] recently proposed a decentralized randomized algorithm to implement broadcast that achieves a throughput arbitrarily close to $\rho*$, in the case where all incoming bandwidths have infinite capacity. In this context, a single communication between P_i and P_j can reach the maximum outgoing bandwidth of P_i, so that we can fully make use of available bandwidth without dealing with contentions. In this paper, we will consider a simpler setting, where client nodes are organized as a star network with the source node at the center and client nodes have no outgoing bandwidth. On the other hand, we do not make any assumption on the incoming bandwidth of the client nodes. In particular, incoming bandwidths may be smaller than B^{out}, what requires to do several communications simultaneously to aggregate bandwidth up to B^{out}, and therefore requires to deal with contentions.

Similarly to the case of independent tasks scheduling, it is worth noting that in the case of the star graph, the optimal broadcast rate $\rho*$ can be determined in linear time by setting

$$\rho* = \min\left(\min_i B_i^{\text{in}}, \frac{B^{\text{out}}}{N}\right).$$

We will show in Section V how to achieve optimality using bandwidth control mechanisms and prove that without such a mechanism, *i.e.* relying only on TCP contention mechanisms, the performance of such an implementation may be arbitrarily smaller than $\rho*$.

D. TCP Contention Modeling

Our goal is to study the influence in presence of contentions of TCP bandwidth sharing mechanisms on the performance of several scheduling algorithm implementations. More precisely, our goal is to prove that TCP mechanisms to deal with congestion must be bypassed by associating to each communication a prescribed bandwidth so that contentions are automatically removed. In order to understand what kind of fairness TCP implements in presence of contentions, several sophisticated models have been proposed [22], [24], [20]. In this paper, we will model contentions using the RTT-aware Max-Min Flow-level method that has been proposed in [8] and validated using NS-2 Network Simulator [23] in [9].

Let us consider the basic platform depicted in Figure 1, that will be used throughout this paper. Let us denote by B^{out} the outgoing bandwidth of node S, by b_i^{in} the incoming bandwidth of node P_i and by λ_i the latency between S and P_i. Let us consider the case where S simultaneously sends messages to all P_is (the case where all P_is simultaneously send a message to S gives the same results). Then, the bandwidth c_i allocated to the communication between S and P_i using RTT-aware Max-Min Flow-level method is returned by the following algorithm (where B^{rem} denotes the

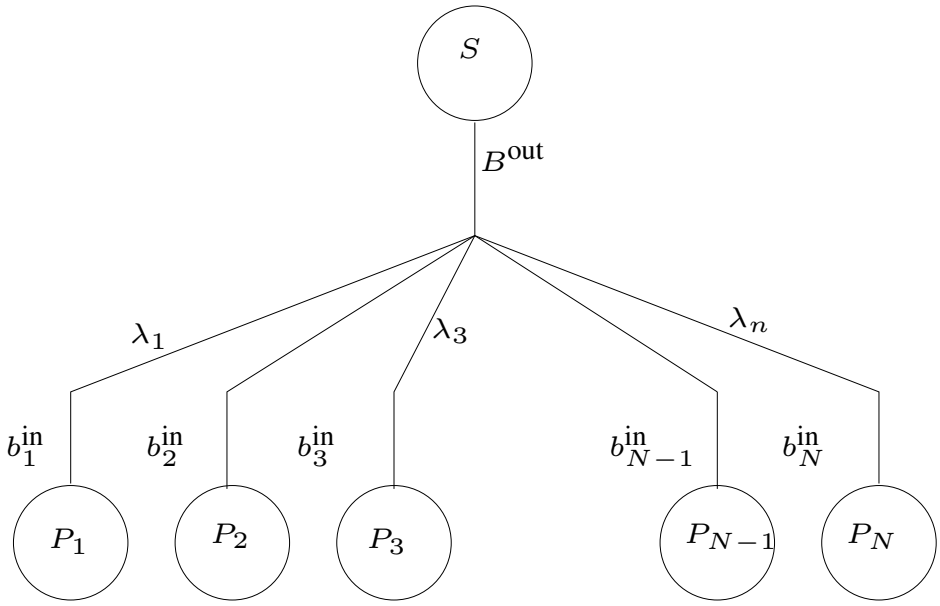

Figure 1. Bandwidth sharing in presence of contentions

remaining bandwidth).

Set	$\text{marked}_i = 0 \quad \forall i; \quad B^{\text{rem}} = B^{\text{out}}$
While $\exists i$,	$\text{marked}_i = 0$ and
	$b_i^{\text{in}} \leq \frac{\frac{1}{\lambda_i}}{\sum_{j,\ \text{marked}_j=0} \frac{1}{\lambda_j}} B^{\text{rem}}$
	$c_i = b_i^{\text{in}}; \text{marked}_i = 1;$
	$B^{\text{rem}} = B^{\text{rem}} - b_i^{\text{in}};$
EndWhile	
Forall i,	If $\text{marked}_i = 0$
	then Set $c_i = \frac{\frac{1}{\lambda_i}}{\sum_{j,\ \text{marked}_j=0} \frac{1}{\lambda_j}} B^{\text{rem}}$
EndForAll	

Using this model, in the case where all b_i^{in} values are large (for instance larger than B^{out}), the bandwidth allocated to the communication between S and P_i only depends on the latency of the link and is inversely proportional to the latency of the link. On the other hand, if all b_i^{in} values are very small, then the bandwidth allocated to the communication between S and P_i is b_i^{in}. Let us now prove two basic lemmas related to this model.

Lemma 2.1: If $\sum_i b_i^{\text{in}} \leq B^{\text{out}}$, then $\forall i,\ c_i = b_i^{\text{in}}$.

Proof: Let us first prove that initially $\exists i, \quad \text{marked}_i = 0$ and $b_i^{\text{in}} \leq \frac{\frac{1}{\lambda_i}}{\sum_{j,\ \text{marked}_j=0} \frac{1}{\lambda_j}} B^{\text{out}}$. All nodes are unmarked. Let us suppose that $\forall i, \quad b_i^{\text{in}} > \frac{\frac{1}{\lambda_i}}{\sum_j \frac{1}{\lambda_j}} B^{\text{out}}$. Then, $\sum_i b_i^{\text{in}} > \frac{\sum_i \frac{1}{\lambda_i}}{\sum_j \frac{1}{\lambda_j}} B^{\text{out}} = B^{\text{out}}$, what is absurd. Therefore, there is at least one node P_{i_1} such that $b_{i_1}^{\text{in}} \leq \frac{\frac{1}{\lambda_{i_1}}}{\sum_{j,\ \text{marked}_j=0} \frac{1}{\lambda_j}} B^{\text{out}}$. The algorithm marks this node and allocates a bandwidth $b_{i_1}^{\text{in}}$ to P_{i_1} and $B^{\text{rem}} = B^{\text{out}} - b_{i_1}^{\text{in}}$. Since initially $\sum_i b_i^{\text{in}} \leq B^{\text{out}}$, then $\sum_{i \neq i_1} b_i^{\text{in}} \leq B^{\text{rem}}$ and we can prove claimed result by induction. ■

Lemma 2.2: If $\sum_i b_i^{\text{in}} \geq B^{\text{out}}$, then $\sum_i c_i \geq B^{\text{out}}$.

Proof: At the end of the While loop, let us denote by $\mathcal{S}$ the set of nodes that have been marked. Then, $\forall P_i \in \mathcal{S}, \quad c_i = b_i^{\text{in}}$ and $B^{\text{rem}} = B^{\text{out}} - \sum_{P_i \in \mathcal{S}} c_i$. At the end of the For loop, $\forall P_i \notin \mathcal{S}, \quad c_i = \frac{\frac{1}{\lambda_i}}{\sum_{P_j \notin \mathcal{S}} \frac{1}{\lambda_j}} B^{\text{rem}}$, so that $\sum_{P_i \notin \mathcal{S}} c_i = B^{\text{rem}}$ and $\sum_i c_i = \sum_{P_i \in \mathcal{S}} c_i + \sum_{P_i \notin \mathcal{S}} = B^{\text{out}} - B^{\text{rem}} + B^{\text{rem}} = B^{\text{out}}$. ■

III. Data Redistribution

A. Implementation

In this section, we consider the practical implementation of file redistribution scheduling algorithms described in Section II-A. In the case where files can be split and sent from several sources to the destination disk, we have seen that a simple linear program provides the set of file transfers that minimizes the makespan. More precisely, the solution of the linear program

provides for each file F^k the size z_k^i of the part of F_k that should be transfered from S_i to D.

In order to assess the impact of bandwidth sharing mechanisms on the overall performance of scheduling algorithms, we will consider two different implementations of the data redistribution algorithm.

1) **Implementation 1:** each source disk S_i has the list of the part of the files F_k that it has to send to destination disk D and it sends them synchronously to the destination node.
2) **Implementation 2: Implementation 2** is exactly the same as **Implementation 1** except that we bound the outgoing bandwidth of S_i to $\frac{\sum_k z_k^i}{T}$.

B. Simulation Results using SimGRID

We present the simulation results obtained on random but realistic instances with SimGRID. It is worth noting that in the case of file redistribution, as in the case of steady state scheduling (Section IV) and broadcasting (Section V), we consider simple star platforms. In this context, the simulation of the bounded Multi-port model in SimGRID has been validated in [9] using NS-2 Network Simulator [23]. Since we are interested in the impact of TCP bandwidth sharing mechanism in presence of contention, we consider the cases where $\sum_i B_i^{\text{out}} = 1.2B^{\text{in}}$ and $\sum_i B_i^{\text{out}} = 2B^{\text{in}}$, that correspond respectively to low and high level of contentions. Since the latency has a major impact on the bandwidth sharing when using TCP, we also consider the case when the latency are almost homogeneous (random values between 10^{-5} and 3×10^{-5}) or strongly heterogeneous (10^{-x}, where x is a random value between 3 and 7). In order to evaluate the impact of the number of nodes, we consider the case where $N = 10$ and $N = 20$).

The following table represents the ratio between the makespan obtained with **Implementation 2** and the makespan obtained using **Implementation 1**. All values correspond to 20 different simulations, and in all case, we depict the minimum, maximum and mean ratio over the 20 simulations.

	$\sum_i B_i^{\text{out}} = 1.2B^{\text{in}}$			
Homogeneous	ratio	min.	max.	mean
	$N = 10$	1.18	1.45	1.26
	$N = 20$	1.25	1.40	1.30
Heterogeneous	ratio	min.	max.	mean
	$N = 10$	1.30	1.57	1.45
	$N = 20$	1.22	1.64	1.51
	$\sum_i B_i^{\text{out}} = 2B^{\text{in}}$			
Homogeneous	ratio	min.	max.	mean
	$N = 10$	1.02	1.17	1.11
	$N = 20$	1.05	1.15	1.11
Heterogeneous	ratio	min.	max.	mean
	$N = 10$	1.09	1.26	1.16
	$N = 20$	1.06	1.29	1.13

The simulation results prove the impact if the bandwidth control on the performance of file redistribution scheduling algorithms. We can notice that, as expected, the impact is more important when the heterogeneity is high (in this case, the bandwidth allocated to some nodes in presence of contentions may be very small, thus delaying their transfers) and when the level of contention is relatively low. Indeed, in the case where $\sum_i B_i^{\text{out}} = 2B^{\text{in}}$, even if some transfers are almost completely delayed first, $\sum_i B_i^{\text{out}}$ once the first set of transfers has ended is still large so that B^{in} bandwidth is not wasted (what happens in the case $\sum_i B_i^{\text{out}} = 1.2B^{\text{in}}$.

C. Worst case analysis

In previous section, we have seen that bounding the available bandwidth out of source nodes can improve the overall makespan. We now prove that the ratio between the optimal makespan using bandwidth control and the makespan when TCP contention mechanisms are used in presence of contention is upper bounded by 2. We also prove that this bound is tight by exhibiting a platform where this ratio can be arbitrarily close to 2.

1) Upper bound for the makespan performance loss: Let us consider a platform with several source disks S_i and a destination disks D and let us consider the makespan M_2 to complete all file transfers using **Implementation 2**. To model contentions, we will rely on the RTT-aware Max-Min Flow-level method that has been introduced in Section II-D. We will prove that the makespan M_2 using **Implementation 2** cannot be larger than twice the makespan M_1 obtained using **Implementation 1**. The proof is based on the same ideas as the classical Graham's bound [14].

Let us distinguish two sets of instants during the execution of **Implementation 2**. The first phase consists in the instants such that the incoming bandwidth of D is fully used and the second phase consists in all other instants. Let us denote by T_1 the duration of the first phase and by T_2 the duration of the second phase, so that $T_1 + T_2 = M_2$.

Theorem 3.1: $T_1 \leq M_1$ and $T_2 \leq M_1$, so that $T_1 + T_2 \leq 2M_1$

Proof: Let us first consider the first phase. During this phase, the incoming bandwidth of D is fully used so that the overall size of data S_1 transmitted during phase 1 is exactly $S_1 = B^{\text{in}} \times T_1$. By construction, $S_1 \leq S$, where S denotes the overall size of data that must be transmitted to D, so that $T_1 \leq \frac{S}{B^{\text{in}}}$. Moreover, $\frac{S}{B^{\text{in}}}$ is a lower bound for the completion time of all file transfers, so that $T_1 \leq T_{\text{opt}} \leq M_1$.

Let us now consider the second phase and more specifically a source disk S_{last} that is involved in a file transfer at the end of Phase 2. During an instant t of Phase 2, let us denote by $\mathcal{U}(t)$ the set of nodes that actually send data to D. Using the notations of Lemma 2.2, $\sum_{S_i \in \mathcal{U}(\sqcup)} c_i(t) < B^{\text{in}}$ so that $\sum_i B_i^{\text{out}} < B^{\text{in}}$ and, because of Lemma 2.1, $\forall S_i \in \mathcal{U}(t), \quad c_i(t) = B_i^{\text{out}}$. Therefore, since S_{last} is still sending data at the end of Phase 2, it has been sending data to D at all the instants of Phase 2 with rate $B_{\text{last}}^{\text{out}}$. Therefore, the overall amount of data sent by S_{last} during Phase 2 is at least $B_{\text{last}}^{\text{out}} \times T_2$. Clearly, the overall amount of data sent by S_{last} is at most $B_{\text{last}}^{\text{out}} \times M_1$, so that $T_2 \leq M_1$. This achieves the proof of the theorem. ■

2) Worst Case Example:

Theorem 3.2: $\frac{M_2}{M_1}$ can be arbitrarily close to 2.

Proof: Let us now prove that the bound of 2 in Theorem 3.1 is tight. To obtain this result, let us consider the following platform, made of two source disks S_1 and S_2 and a destination disk D with the following characteristics

$$S_1: \quad \lambda_1 = \epsilon^3, \quad B_1^{\text{out}} = 1; \qquad S_2: \quad \lambda_2 = \epsilon, \quad B_2^{\text{out}} = \epsilon$$

$$D: \quad B^{\text{in}} = 1,$$

where ϵ stands for an arbitrarily small quantity and λ_1 and λ_2 denote the latencies between S_1 and D and S_2 and D respectively. Since latencies are arbitrarily small, we will not consider the delays introduced by these latencies but rather concentrate on their impact on bandwidth sharing using the RTT-aware Max-Min Flow-level algorithm presented in Section II-D.

Let us assume that S_1 has to send to D a file of size 1 and that S_2 has to send a file of size ϵ. In the optimal solution, S_1 continuously sends data during time $1 + \epsilon$ to D using bandwidth $\frac{1}{1+\epsilon}$ and S_2 continuously sends data during time $1 + \epsilon$ to D using bandwidth $\frac{\epsilon}{1+\epsilon}$ so that at time $1 + \epsilon$, D has received both files.

Let us now consider what happens if we rely on TCP bandwidth sharing mechanisms to deal with contentions.

$$\begin{cases} B_1^{\text{out}} = 1 > \frac{\frac{1}{\epsilon^3}}{\frac{1}{\epsilon}+\frac{1}{\epsilon^3}} = 1 - \epsilon^2 + o(\epsilon^2) \\ B_2^{\text{out}} = \epsilon > \frac{\frac{1}{\epsilon}}{\frac{1}{\epsilon}+\frac{1}{\epsilon^3}} = \epsilon^2 + o(\epsilon^2) \end{cases},$$

so that (c_i values are attributed in the Forall loop) $c_1 = 1 - \epsilon^2 + o(\epsilon^2)$ and $c_2 = \epsilon^2 + o(\epsilon^2)$. Therefore, S_1 ends up its transfer at time $1 + \epsilon^2$. At this time, S_2 has transfered $\epsilon^2 + o(\epsilon^2)$ data so that it needs extra $1 - \epsilon$ time to ends up its transfer using its maximal bandwidth ϵ. Therefore, the overall necessary time to transfer both files using TCP bandwidth sharing mechanism is $(2 - \epsilon)$, *i.e.* $(2 - 3\epsilon)$ times the time necessary to do the file transfers optimally, what achieves the proof of the theorem. ■

IV. Steady State Scheduling

A. Implementation

In this section, we consider the implementation of a scheduling algorithm to process independent equal-sized tasks on a master-slave heterogeneous platform. Initially, the master node holds (or generate at a given rate) a large number of tasks that will be processed by a set of slave nodes P_i. The master node is characterized by its outgoing bandwidth B^{out} whereas a slave node P_i is characterized by both its incoming bandwidth B_i^{in} and its processing capability w_i. Since all tasks are equal-sized, we normalize all B^{out}, B_i^{in} and w_i in terms of tasks per time unit. We have seen in Section II-B that a simple linear program provides for each slave node P_i the rate ρ_i at which the master should send tasks to P_i in order to maximize the overall throughput, *i.e.* the overall (rational) number of tasks that can be processed using this platform within one time unit. As in the case of file redistribution, in order to assess the impact of bandwidth sharing mechanisms on the overall performance of scheduling algorithm, we consider two different implementations of the scheduling algorithm.

1) **Implementation 1:** In order to avoid starvation, each slave node starts with two tasks in its local buffer. Each time P_i starts processing a new task, it asks for another task and the master node initiates the communication immediately.
2) **Implementation 2: Implementation 2** is exactly the same as **Implementation 1** except that we

bound the bandwidth used by M to send tasks to P_i to ρ_i.

In what follows, we will denote by T_1 the achieved throughput when using **Implementation 1** and by T_2 the achieved throughput when using **Implementation 2**.

B. Simulation Results using SimGRID

We present the simulation results obtained on random but realistic instances with SimGRID. We use exactly the same settings as in Section III-B for the communications. The following table represents the ratio between the throughput obtained with **Implementation 2** and the throughput obtained using **Implementation 1**. All values correspond to 20 different simulations, and in all case, we depict the minimum, maximum and mean ratio over the 20 simulations. For each simulation, to estimate the throughput, we run both implementations on 200 tasks. In order to estimate the impact of communications rather than processing, the processing rate of the processors are set so that in the optimal solution, processors are limited by their communication capabilities.

	$\sum_i B_i^{\text{out}} = 1.2B^{\text{in}}$			
Homogeneous	ratio	min.	max.	mean
	$N = 10$	1.01	1.03	1.02
	$N = 20$	1.00	1.02	1.01
Heterogeneous	ratio	min.	max.	mean
	$N = 10$	1.01	1.04	1.03
	$N = 20$	1.01	1.03	1.02
	$\sum_i B_i^{\text{out}} = 2B^{\text{in}}$			
Homogeneous	ratio	min.	max.	mean
	$N = 10$	1.01	1.04	1.02
	$N = 20$	1.00	1.03	1.01
Heterogeneous	ratio	min.	max.	mean
	$N = 10$	1.01	1.04	1.03
	$N = 20$	1.00	1.03	1.02

The difference between both implementations is much smaller than for file redistribution (and broadcasting). This is due to fact that contrarily to other situations, compensation between processors can take place. The processors with small latencies process more tasks with **Implementation 1** than with **Implementation 2**. In order to obtain more significant difference, we can make the processors saturated in computations in the optimal solution, and form two groups of equivalent aggregated processing power, one with small latencies and one with high latencies. In this case, the ratio is closer to the $\frac{4}{3}$ bound proved below.

C. Worst case analysis

In previous section, we have seen that bounding the bandwidth used by a communication between M and P_i improves the achieved throughput. In this section, we prove that the ratio between the optimal throughput using bandwidth control and the throughput when TCP contention mechanisms are used in presence of contention is smaller than $\frac{4}{3}$. We also prove that this bound is tight by exhibiting a platform where this ratio can be arbitrarily close to $\frac{4}{3}$.

1) Upper bound for the throughput performance loss:

Theorem 4.1: $T_2 \leq \frac{4}{3}T_1$

Proof: Let us consider the result obtained using **Implementation 1** over a long period of time and let us denote by x_i the average number of tasks processed by P_i during one time unit. If $\sum x_i = B^{\text{out}}$, then **Implementation 1** achieves asymptotically optimal throughput and the theorem is true. Otherwise, let us denote by t_1 the average fraction of time when the bandwidth of the master is fully used. On the other hand, let us denote by $B^{\text{out}}_{\text{ave}}$ the average used bandwidth when the bandwidth of the master is not fully used, *i.e.* during fraction of time $(1 - t_1)$ (see Figure 2). Using these notations, we can find a first upper bound of the throughput W wasted using implementation 1, $W \leq (1 - t_1)(B^{\text{out}} - B^{\text{out}}_{\text{ave}})$.

Let us now consider the set $\mathcal{S}_1$ of slave processors that are not used at their best rate, *i.e.* such that $x_i < \min(w_i, B_i^{\text{in}})$ and by $\mathcal{S}_2$ the set of processors such that $x_i = \min(w_i, B_i^{\text{in}})$. Moreover, let us denote by $\rho^{(k)}_{\text{opt}}$, $k = 1, 2$ the overall throughput achieved by the slaves of set $\mathcal{S}_k$ in the optimal solution. We can notice that $\sum_{P_i \in \mathcal{S}_2} x_i \geq \rho^{(2)}_{\text{opt}}$.

Since the processors of $\mathcal{S}_1$ are not used at their maximal processing rate, they are continuously requesting tasks using **Implementation 1**. Therefore, at each instant when the bandwidth of M is not fully used, slave $P_i \in \mathcal{S}_1$ is receiving tasks at rate B_i^{in}. Therefore, $B^{\text{out}}_{\text{ave}} \geq \sum_{P_i \in \mathcal{S}_1} B_i^{\text{in}}$ and $\sum_{P_i \in \mathcal{S}_1} x_i \geq (1 - t_1)B^{\text{out}}_{\text{ave}}$. Moreover, by definition, $\rho^{(1)}_{\text{opt}} \leq \sum_{P_i \in \mathcal{S}_1} B_i^{\text{in}}$. Therefore,

$$\begin{array}{rcl} \rho^{(1)}_{\text{opt}} - \sum_{P_i \in \mathcal{S}_1} x_i & \leq & \sum_{P_i \in \mathcal{S}_1} B_i^{\text{in}} - (1 - t_1)B^{\text{out}}_{\text{ave}} \\ & \leq & B^{\text{out}}_{\text{ave}} - (1 - t_1)B^{\text{out}}_{\text{ave}} \\ & \leq & t_1 B^{\text{out}}_{\text{ave}}. \end{array}$$

and therefore,

$$W = \rho^{(2)}_{\text{opt}} - \sum_{P_i \in \mathcal{S}_2} x_i + \rho^{(1)}_{\text{opt}} - \sum_{P_i \in \mathcal{S}_1} x_i \leq t_1 B^{\text{out}}_{\text{ave}}.$$

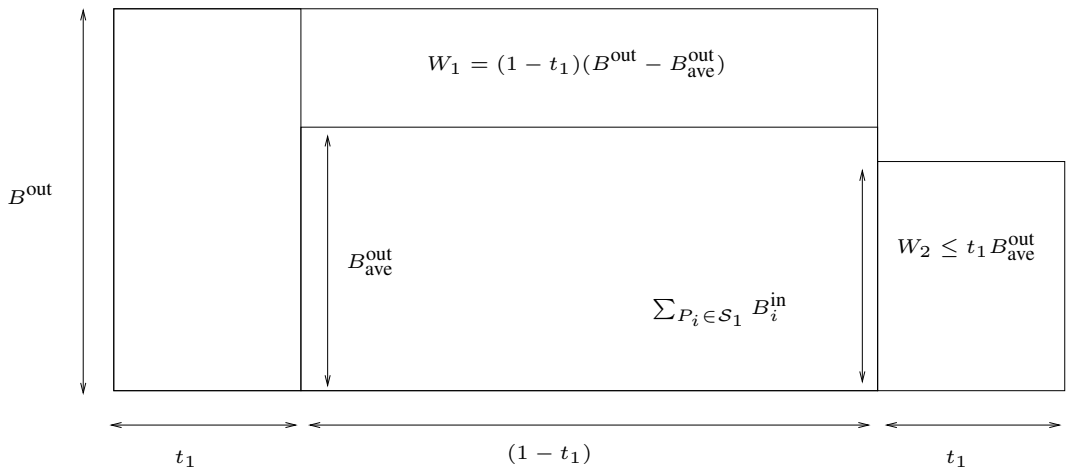

Figure 2. Bandwidth sharing using TCP and **Implementation 1**

Using both upper bounds of W, we obtain $W \leq f(t_1, B^{\text{out}}_{\text{ave}}) = \min((1-t_1)(B^{\text{out}} - B^{\text{out}}_{\text{ave}}), t_1 B^{\text{out}}_{\text{ave}})$. If we first consider $f(t_1, B^{\text{out}}_{\text{ave}})$ as a function of $B^{\text{out}}_{\text{ave}} \in [0, B^{\text{out}}]$, we observe that $f(t_1, B^{\text{out}}_{\text{ave}})$ is minimal when $B^{\text{out}}_{\text{ave}} = (1-t_1)B^{\text{out}}$ and $f(t_1, (1-t_1)B^{\text{out}}) = t_1(1-t_1)B^{\text{out}}$ so that $\forall t_1, B^{\text{out}}_{\text{ave}}, \;\; W \leq \frac{B^{\text{out}}}{4}$, what achieves the proof of the theorem. ■

2) Worst Case Example:

Theorem 4.2: $\frac{T_2}{T_1}$ can be arbitrarily close to $\frac{4}{3}$.

Proof: Let us now prove that the bound of $\frac{4}{3}$ is tight. To obtain this result, let us consider the following platform, made of two slave nodes P_1 and P_2 and a master node M with the following characteristics

$$P_1: \; \lambda_1 = \epsilon^3, \;\; w_1 = 1, \;\; B^{\text{in}}_1 = 2;$$

$$P_2: \; \lambda_2 = \epsilon, \;\; w_1 = 1, \;\; B^{\text{in}}_2 = 1;$$

$$D: \; B^{\text{out}} = 2,$$

where ϵ stands for an arbitrarily small quantity and λ_1 and λ_2 denote the latencies between M and P_1 and M and P_2 respectively. As previously, since latencies are arbitrarily small, we will not consider the delays introduced by these latencies but rather concentrate on their impact on bandwidth sharing using the RTT-aware Max-Min Flow-level algorithm presented in Section II-D.

Using **Implementation 2**, P_1 starts computing its first task at time 0 and ends up at time 1. The master starts sending a new task at time 0 using bandwidth 1 and the communication ends up at time 1. The same process applies to P_2, so that exactly 2 tasks are processed every time unit, hence $T_2 = 2$.

Let us now consider what happens if we rely on TCP bandwidth sharing mechanisms to deal with contentions.

$$\begin{cases} B^{\text{out}}_1 = 2 > \frac{\frac{1}{\epsilon^3}}{\frac{1}{\epsilon}+\frac{1}{\epsilon^3}} \times 2 = 2 - 2\epsilon^2 + o(\epsilon^2) \\ B^{\text{out}}_2 = \epsilon > \frac{\frac{1}{\epsilon}}{\frac{1}{\epsilon}+\frac{1}{\epsilon^3}} \times 2 = 2\epsilon^2 + o(\epsilon^2) \end{cases},$$

so that (c_i values are attributed in the Forall loop) $c_1 = 2 - \epsilon^2 + o(\epsilon^2)$ and $c_2 = 2\epsilon^2 + o(\epsilon^2)$. Therefore, P_1 receives its first task at time $\frac{1}{2} + 2\epsilon^2 + o(\epsilon^2)$ and P_2 receives only $\epsilon^2 + o(\epsilon^2)$ tasks at time $\frac{1}{2} + 2\epsilon^2 + o(\epsilon^2)$. Between time $\frac{1}{2} + 2\epsilon^2 + o(\epsilon^2)$ and time 1, P_2 receives tasks at rate 1 since it is the only one requiring tasks. Thus, at time 1, P_2 has received $\frac{1}{2} + O(\epsilon^2)$ tasks. At time 1, the same scheme applies since P_1 requires a new task and will receive it by time $\frac{3}{2} + O(\epsilon^2)$ while P_2 receives extra $O(\epsilon^2)$ tasks. Thus, P_2 will end up receiving its first task at time $2 - \epsilon^2$. Then, the same scheme applies during each time period of size 2.
Therefore, **Implementation 1** processes 2 tasks every time unit while **Implementation 2** processes 3 tasks every 2 time units, what achieves of the proof of the theorem. ■

V. Broadcast under Bounded Multiport Model

A. Implementation

In the broadcast problem under the bounded multiport model, we are given a source node S whose outgoing bandwidth is B^{out} and a set of clients P_i. We denote by B^{in}_i the incoming bandwidth of the P_i. Moreover, we assume that S holds (or generate) a large size message and that all client nodes should receive the whole message. In this context, our goal is to maximize the

throughput, *i.e.* the average size of the message received by any client during one time unit. In the case where $\forall i, \quad B_i^{\text{in}} > B_j^{\text{out}}$, a simple randomized and distributed algorithm has been recently proposed by Massoulié et al. [21]: during the execution, a node compares the set of packets that it has received with the set of packets received by its neighbor nodes. Then, it sends a packet to the node that has received the less packets yet. Remarkably enough, it has been proved in [21] that this algorithm achieves quasi-optimal performance in the case of a complete graph. Unfortunately, the proof strongly relies on the assumption that $\forall i, \quad B_i^{\text{in}} > B_j^{\text{out}}$ since it requires that any single communication between P_i and P_j consumes bandwidth B_i^{in}. Therefore, there is no need to use several communications to aggregate bandwidth and thus to use the whole capacity of S. Then, the whole execution takes place without contentions.

Therefore, we concentrate in this section on a simpler setting, where the platform is a star-shaped platform with the master at the center. On the other hand, we do not make any assumption on the values of B_i^{in} and B^{out} (since the platform is a star, the client nodes do not have any outgoing bandwidth). In this simple case, if N denotes the number of clients, the achievable throughput $\rho*$ is given by $\rho* = \min(\frac{B^{\text{out}}}{N}, \min_i B_i^{\text{in}})$. We consider two different implementations of the broadcast operation.

1) **Implementation 1:** Every time unit, the source S initiates simultaneously a communication with each client node, and sends a message of size $\rho*$ containing last generated packets to each client.
2) **Implementation 2: Implementation 2** is exactly the same as **Implementation 1** except that we bound the bandwidth used by S to send the message to P_i to $\rho*$.

In order to compare both implementations, we will execute both programs for a long time period T. Let $x_i^k(T)$ denote the size of the message received at time T by P_i using Implementation k. The performance of Implementation k is given by $\rho_k = \lim_{T\to+\infty} \frac{\min_i x_i^k(T)}{T}$. In what follows, we prove that ρ_1 can be arbitrarily smaller than $\rho*$.

B. Simulation Results

We present the simulation results obtained on random but realistic instances with SimGRID. We use exactly the same settings as in Section III-B for the communications. The following table represents the ratio between ρ_2 and ρ_1, the throughput obtained with **Implementation 2** and the throughput obtained using **Implementation 1**. All values correspond to 20 different simulations, and in all case, we depict the minimum, maximum and mean ratio over the 20 simulations. For each simulation, to estimate the throughput, we run both implementations for time 500.

	$\sum_i B_i^{\text{out}} = 1.2B^{\text{in}}$			
Homogeneous	ratio	min.	max.	mean
	$N = 10$	1.01	1.03	1.02
	$N = 20$	1.00	1.02	1.01
Heterogeneous	ratio	min.	max.	mean
	$N = 10$	1.01	1.09	1.04
	$N = 20$	1.00	1.04	1.03
	$\sum_i B_i^{\text{out}} = 2B^{\text{in}}$			
Homogeneous	ratio	min.	max.	mean
	$N = 10$	1.01	1.22	1.07
	$N = 20$	1.00	1.09	1.03
Heterogeneous	ratio	min.	max.	mean
	$N = 10$	1.01	1.79	1.47
	$N = 20$	1.00	1.33	1.19

The simulation results prove that the throughput achived by **Implementation 1** may be much smaller than the troughput achived by **Implementation 2**, especially when the latencies are strongly heterogeneous. Indeed, in this case, when several communications take place simultaneously, the processors with high latencies get a very small part of the bandwidth. Since new communications are launched every time step, the size of data received by these processors is significantly lower, especially in the case of high contentions $\sum_i B_i^{\text{out}} = 2B^{\text{in}}$.

C. Worst Case Analysis

Theorem 5.1: ρ_1 can be arbitrarily smaller than ρ_2 and $\rho*$.

Proof: Let us consider the following platform consisting of N clients. The source node S has outgoing bandwidth $B^{\text{out}} = N$. The first $N-1$ clients $P_i, \ i = 1 \dots N-1$ have incoming bandwidth $B_i^{\text{in}} = \frac{N}{N-1}$ and the latency between S and $P_i, \ i = 1 \dots N-1$ is given by $\lambda_i = \epsilon^2$. At last, client P_N has incoming bandwidth 1 and the latency between S and P_N is ϵ. At last, we assume that ϵ is arbitrarily small and in particular $\epsilon \times N << 1$. Using this platform, **Implementation 2** achieves optimal throughput $\rho_2 = \rho* = 1$. Indeed, all clients are simultaneously served every time step with bandwidth $\rho*$ and all transfers finish within one time unit.

Using **Implementation 1**, the sum of the bandwidths of the client nodes involved in communications with S

at time 0 is given by $N+1$, so that contentions take place at the source node. Using the algorithm presented in Section II-D to model TCP bandwidth sharing in presence of contentions, we obtain $\forall i = 1, \ldots, N-1,\ c_i = \frac{N}{N-1}(1+O(\epsilon))$ and $c_N = \frac{N\epsilon}{N-1} + o(\epsilon)$. Therefore, all P_is, $i \leq N-1$ receive the first message at time $1 - 1/N + O(\epsilon)$ whereas at that time, P_N has only received a message of size $O(\epsilon)$. During the interval between $1-1/N$ and 1 (instant when a new message is broadcast to all clients), P_N is the only node communicating with S and $C_N = 1$. Thus, at time 1, P_N has received a message of size $1/N + O(\epsilon)$. The same scheme applies between time 1 and 2 and it will take a time N to P_N to completely receive the very first message. Hence, the overall performance is $\rho_1 = 1/N$, what achieves the proof of the theorem. ■

VI. Conclusions

In this paper, we have studied the influence of bandwidth control mechanisms on the performance of several scheduling algorithms on large scale distributed platforms. In this context, the topology is not known since Internet is the underlying network, and the volatility of resources and the changes in their performance make automatic discovery tools inefficient. We have therefore proposed to model communication costs and contentions using a very limited set of parameters, that can be determined at runtime (incoming and outgoing bandwidths and latencies). Rather than relying on traditional one-port model, that is not well suited to very heterogeneous resources since it may induce important waste in performance, we modeled communications using the Bounded Multi-port Model, where several incoming and outgoing communications can be done simultaneously provided that bandwidth capacities are not exceeded. More specifically, we have compared on three classical scheduling problems (namely file redistribution schemes, independent tasks and collective communication scheduling) the performance obtained with implementations using bandwidth and implementations relying on TCP bandwidth sharing in presence of contention. For each problem, we have a proved an upper bound on the maximal performance loss that can be induced by TCP bandwidth sharing, we have proved that this bound is tight by exhibiting instances achieving it and we have compared the performance of implementations using bandwidth sharing control or relying on TCP bandwidth sharing mechanisms in presence of contentions on random realistic instances. This work shows that in the context of large scale distributed platforms, where latencies are strongly heterogeneous, the use of bandwidth control mechanisms, that are available in modern operating systems, is compulsory to achieve good performance.

References

[1] D.P. Anderson. BOINC: A System for Public-Resource Computing and Storage. In *5th IEEE/ACM International Workshop on Grid Computing*, pages 365–372, 2004.

[2] B. Awerbuch and T. Leighton. A simple local-control approximation algorithm for multicommodity flow. In *Annual Symposium on Foundations Of Computer Science (FOCS 1993)*, volume 34, pages 459–459. IEEE COMPUTER SOCIETY PRESS, 1993.

[3] B. Awerbuch and T. Leighton. Improved approximation algorithms for the multi-commodity flow problem and local competitive routing in dynamic networks. In *Proceedings of the twenty-sixth annual ACM symposium on Theory of computing*, page 496. ACM, 1994.

[4] R. Baeza-Yates and R. Ramakrishnan. Data challenges at Yahoo! In *Proceedings of the 11th international conference on Extending database technology: Advances in database technology*, pages 652–655. ACM New York, NY, USA, 2008.

[5] C. Banino, O. Beaumont, L. Carter, J. Ferrante, A. Legrand, and Y. Robert. Scheduling Strategies for Master-Slave Tasking on Heterogeneous Processor Platforms. *IEEE TPDS*, 2004.

[6] C. Banino-Rokkones, O. Beaumont, and H. Rejeb. Scheduling Techniques for Effective System Reconfiguration in Distributed Storage Systems. In *IEEE IC-PADS'08*, pages 80–87, 2008.

[7] Martin A. Brown. Traffic Control HOWTO. Chapter 6. Classless Queuing Disciplines. *http://tldp.org/HOWTO/Traffic-Control-HOWTO/classless-qdiscs.html*, 2006.

[8] H. Casanova. Network modeling issues for grid application scheduling. *International Journal of Foundations of Computer Science*, 16(2):145–162, 2005.

[9] H. Casanova and L. Marchal. A network model for simulation of grid application. *RR-40-2002*, 40, 2002.

[10] M. Castro, P. Druschel, A.M. Kermarrec, A. Nandi, A. Rowstron, and A. Singh. SplitStream: high-bandwidth multicast in cooperative environments. *ACM SIGOPS Operating Systems Review*, 37(5):298–313, 2003.

[11] J. Edmonds. Edge disjoint branchings. In *Combinatorial Algorithms: Courant Computer Science Symposium 9: January 24-25, 1972*, page 91. Algorithmics Press, 1972.

[12] L. Eyraud-Dubois, A. Legrand, M. Quinson, and F. Vivien. A First Step Towards Automatically Building Network Representations. *Lecture Notes in Computer Science*, 4641:160, 2007.

[13] H.N. Gabow and KS Manu. Packing algorithms for arborescences (and spanning trees) in capacitated graphs. *Mathematical Programming*, 82(1):83–109, 1998.

[14] R.L. Graham, E.L. Lawler, J.K. Lenstra, and A.H.G.R. Kan. Optimization and approximation in deterministic sequencing and scheduling: A survey. *Annals of Discrete Mathematics*, 5(2):287–326, 1979.

[15] B. Hong and V.K. Prasanna. Distributed adaptive task allocation in heterogeneous computing environments to maximize throughput. *IEEE IPDPS*, 2004.

[16] B. Hubert et al. Linux Advanced Routing & Traffic Control. Chapter 9. Queueing Disciplines for Bandwidth Management. *http://lartc.org/lartc.pdf*, 2002.

[17] S.L. Johnsson and C.T. Ho. Optimum broadcasting and personalized communication in hypercubes. *IEEE Transactions on Computers*, 38(9):1249–1268, 1989.

[18] S.M. Larson, C.D. Snow, M. Shirts, and V.S. Pande. Folding@ Home and Genome@ Home: Using distributed computing to tackle previously intractable problems in computational biology. *Computational Genomics*, 2002.

[19] Z. Li, B. Li, D. Jiang, and L.C. Lau. On achieving optimal throughput with network coding. In *Proceedings IEEE INFOCOM*, 2005.

[20] L. Massoulie and J. Roberts. Bandwidth sharing: objectives and algorithms. In *IEEE INFOCOM*, 1999.

[21] L. Massoulie, A. Twigg, C. Gkantsidis, and P. Rodriguez. Randomized decentralized broadcasting algorithms. In *IEEE INFOCOM*, pages 1073–1081, 2007.

[22] M. Mathis, J. Semke, J. Mahdavi, and T. Ott. The macroscopic behavior of the TCP congestion avoidance algorithm. *ACM SIGCOMM Computer Communication Review*, 27(3):67–82, 1997.

[23] S. McCanne, S. Floyd, K. Fall, K. Varadhan, et al. Network simulator ns-2, 2000.

[24] J. Padhye, V. Firoiu, D. Towsley, and J. Kurose. Modeling TCP throughput: A simple model and its empirical validation. *ACM SIGCOMM Computer Communication Review*, 28(4):303–314, 1998.

[25] G. Shao, F. Berman, and R. Wolski. Using effective network views to promote distributed application performance. In *International Conference on Parallel and Distributed Processing Techniques and Applications*. CSREA Press, June 1999.

[26] Y.C. Tseng, S.Y. Wang, and C.W. Ho. Efficient broadcasting in wormhole-routed multicomputers: anetwork-partitioning approach. *IEEE TPDS*, 10(1):44–61, 1999.

[27] J. Watts and R.A. Geijn. A pipelined broadcast for multidimensional meshes. *Parallel Processing Letters*, 5(2):281–292, 1995.

[28] S.A. Weil, S.A. Brandt, E.L. Miller, and C. Maltzahn. CRUSH: Controlled, scalable, decentralized placement of replicated data. In *Proceedings of the ACM/IEEE conference on Supercomputing*. ACM, 2006.

[29] X. Zhang, J. Liu, B. Li, and Y.S.P. Yum. CoolStreaming/DONet: A data-driven overlay network for peer-to-peer live media streaming. In *Proceedings IEEE INFOCOM*, 2005.