

# *Allocation of Clients to Multiple Servers on Large Scale Heterogeneous Platforms*

Olivier Beaumont — Lionel Eyraud-Dubois — Hejer Rejeb — Christopher Thraves

N° 6766

December 2008

Thème COM

 *Rapport  
de recherche*



## Allocation of Clients to Multiple Servers on Large Scale Heterogeneous Platforms

Olivier Beaumont\* , Lionel Eyraud-Dubois\* , Hejer Rejeb\* ,  
Christopher Thraves†

Thème COM — Systèmes communicants  
Équipe-Projet Cepage

Rapport de recherche n° 6766 — December 2008 — 17 pages

**Abstract:** In this paper, we consider the problem of allocating a large number of independent, equal-sized tasks to a heterogeneous large scale computing platforms, such as BOINC [1] or Folding@home [15]. We model the platform using a set of servers (masters) that initially hold (or generate) the tasks to be processed by a set of clients (slaves). All resources have different speeds of communication and computation and we model contentions using the bounded multi-port model. Under this model, a processor can be involved simultaneously in several communications, provided that its incoming and outgoing bandwidths are not exceeded. This model corresponds well to modern networking technologies, but for the sake of realism, another parameter needs to be introduced in order to bound the number of simultaneous connexions that can be opened at a server node. We prove that unfortunately, this additional parameter makes the problem of maximizing the overall throughput (*i.e.* the fractional number of tasks that can be processed within one time-unit) NP-Complete. This result is closely related to results on bin packing with splittable items and cardinality constraints. On the other hand, we also propose a polynomial time algorithm, based on a slight resource augmentation, to solve this problem. More specifically, we prove that, if  $d_j$  denotes the maximal number of connexions that can be opened at node  $\mathcal{S}_j$ , then the throughput achieved using this algorithm and  $d_j + 1$  is at least the same as the optimal one with  $d_j$ . We also provide a dual algorithm for minimizing the maximal number of connexions that need to be opened in order to achieve a given throughput. Finally, we also propose extensive simulations to assess the performance of the proposed algorithm.

**Key-words:** Independent Tasks Scheduling, Bin Packing, Resource Augmentation, Approximation Algorithms, Heterogeneous Computing

\* LaBRI

† This work was partially supported by French ANR project Alpage

## Allocation de Clients à Plusieurs Serveurs sur des Plates-Formes Hétérogènes à Grande Échelle

**Résumé :** Cet article s'intéresse à l'allocation d'un grand nombre de tâches indépendantes et identiques à une plate-forme de calcul hétérogène et à grande échelle, comme BOINC ou Folding@Home. La plate-forme est modélisée par un ensemble de serveurs (ou maîtres) qui détiennent ou génèrent les tâches qui doivent être effectuées par un ensemble de clients (ou esclaves). Les ressources ont des capacités de calcul et de communication différentes, et la contention est modélisée par le modèle multi-port borné. Ce modèle autorise un processeur à participer à plusieurs communications simultanément, tant que ses bandes passantes entrantes et sortantes ne sont pas dépassées. Cela correspond bien aux technologies réseau actuelles, mais il faut y ajouter un paramètre supplémentaire pour borner le nombre de connections qui peuvent être ouvertes sur chaque serveur. Nous prouvons que ce paramètre supplémentaire rend le problème de maximisation du débit total (c'est-à-dire le nombre de tâches fractionnaire qui peuvent être exécutées par unité de temps) NP-Complet. Ce résultat est proche d'autres résultats précédents à propos de *bin packing* avec des objets divisibles et des contraintes de cardinalité. De plus, nous proposons également un algorithme polynomial qui résout le problème avec une petite augmentation de ressources. Plus précisément, si  $d_j$  est le nombre maximal de connections que le serveur  $S_j$  peut ouvrir, alors notre algorithme atteint le débit d'une solution optimale en autorisant l'ouverture de  $d_j + 1$  connections. Cela fournit également une très bonne approximation au problème dual de minimisation du nombre de connections nécessaires pour obtenir un débit donné. Enfin, nous fournissons également des simulations étendues pour montrer la performance de l'algorithme proposé.

**Mots-clés :** ordonnancement de tâches indépendantes, bin packing, augmentation de ressources, algorithmes d'approximation, calcul hétérogène

## 1 Introduction

Scheduling computational tasks on a given set of processors is a key issue for high-performance computing, especially in the context of the emergence of large scale computing platforms such as BOINC [1] or Folding@home [15]. These platforms are characterized by their large scale, their heterogeneity and the variations in the performances of their resources. These characteristics strongly influence the set of applications that can be executed using these platforms. First, the running time of the application has to be large enough to benefit from the platform scale, and to minimize the influence of start-up times due to sophisticated middleware. Second, an application executed on such a platform typically consists of many small independent tasks. This allows to minimize the influence of variations in resource performances and to limit the impact of resource failures. From a scheduling point of view, the set of applications that can be efficiently executed is therefore restricted, and we can concentrate on "embarrassingly parallel" applications consisting in many independent tasks.

In this context, makespan minimization, i.e, minimizing the minimal time to process a given number of tasks, is usually intractable. An idea to circumvent the difficulty of makespan minimization is to lower the ambition of the scheduling objective. Instead of aiming at the absolute minimization of the execution time, why not consider asymptotic optimality and optimize the throughput (i.e. the fractional number of tasks that can be processed in one time-unit once steady-state has been reached)? After all, the number of tasks to be executed in applications such as Seti@home [2] or Folding@home [15] is huge. This approach has been pioneered by Bertsimas and Gamarnik [5] and has been extended to task scheduling in [3] and collective communications in [4]. Steady-state scheduling allows to relax the scheduling problem in many ways. Initialization and clean-up phases are neglected. The precise ordering and allocation of tasks and messages are not required, at least in the first step. The main idea is to characterize the activity of each resource during each time-unit: which rational fraction of time is spent sending and processing tasks and to which client tasks are delegated?

In this paper, we restrict our attention to steady-state scheduling of independent equal-sized tasks. In order to consider a more general model than current settings where a single server is used, we consider that a set of servers initially hold (or generate) the tasks to be processed. Each server  $\mathcal{S}_j$  is characterized by its outgoing bandwidth  $b_j$  (i.e. the number of tasks it can send during one time-unit) and its maximal degree  $d_j$  (i.e. the number of open connexions that it can handle simultaneously). On the other hand, each client  $\mathcal{C}_i$  is characterized by its capacity  $w_i$  (i.e. the number of tasks it can handle during one time-unit).  $w_i$  encompasses both its processing and communication capacities. More specifically, if  $\text{comp}_i$  denotes the number of tasks  $\mathcal{C}_i$  can process during one time-unit, and  $\text{comm}_i$  denotes the number of tasks it can receive during one time-unit, then we set  $w_i = \min(\text{comp}_i, \text{comm}_i)$ .

Our goal is to build a bipartite graph between servers and clients. We do not assume that the underlying network topology is known. Such an assumption would be completely unrealistic for large scale computing platforms such as BOINC, where Internet is the underlying network. Even for smaller scale platforms, such as Grids, automatic topology discovery tools, such as [10, 20] are much too slow for quickly evolving resources. Moreover, the underlying core

network is usually over-sized, so that contentions mostly take place at node networking interfaces.

To model contentions, we rely on the bounded multi-port model, that has already been advocated by Hong et al. [13] for independent task distribution on heterogeneous platforms. In this model, server  $\mathcal{S}_j$  can serve any number of clients simultaneously, each using a bandwidth  $w'_i \leq w_i$  provided that its outgoing bandwidth is not exceeded, i.e.  $\sum_i w'_i \leq b_j$ . This corresponds to modern network infrastructure, where each communication is associated to a TCP connexion.

This model strongly differs from the traditional one-port model used in scheduling literature, where connexions are made in exclusive mode: the server can communicate with a single client at any time-step. Previous results obtained in steady-state scheduling of independent tasks [3] have been obtained under this model, which is easier to implement. For instance, Saif and Parashar [16] report experimental evidence that achieving the performances of bounded multi-port model may be difficult, since asynchronous sends become serialized as soon as message sizes exceed a few megabytes. Their results hold for two popular implementations of the MPI message-passing standard, MPICH on Linux clusters and IBM MPI on the SP2. Nevertheless, in the context of large scale platforms, the networking heterogeneity ratio may be high, and it is unrealistic 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. Therefore, in our context, all connexions must directly be handled at TCP level, without using high level communication libraries.

It is worth noting that at TCP level, several QoS mechanisms enable a prescribed sharing of the bandwidth [6, 14]. In particular, it is possible to handle simultaneously several connexions and to fix the bandwidth allocated to each connexion. In our context, these mechanisms are particularly useful since  $w_i$  encompasses both processing and communication capabilities of  $\mathcal{C}_i$  and therefore, the bandwidth allocated to the connexion between  $\mathcal{S}_j$  and  $\mathcal{C}_i$  may be lower than both  $b_j$  and  $w_i$ . Nevertheless, handling a large number of connexions at server  $\mathcal{S}_j$  with prescribed bandwidths consumes a lot of kernel resources, and it may therefore be difficult to reach  $b_j$  by aggregating a large number of connexions. In order to avoid this problem, we introduce another parameter  $d_j$  in the bounded multi-port model, that represents the maximal number of connexions that can simultaneously be opened at server  $\mathcal{S}_j$ .

The rest of the paper is organized as follows. In Section 2, we present the communication model we use and formalize the scheduling problem we consider. We prove that if we introduce a bound on the maximal number of connexions that can be opened simultaneously at a server, then the problem of maximizing the overall throughput becomes NP-Complete. We also discuss related works dealing with the packing of splittable items with cardinality constraints. In Section 3, we also propose a sophisticated polynomial time algorithm, based on a slight resource augmentation to solve this problem. More specifically, we prove that, if  $d_j$  denotes the maximal number of connexions that can be opened at node  $\mathcal{S}_j$ , then the throughput achieved using this algorithm and degree  $d_j + 1$  is at least the same as the optimal one with degree  $d_j$ . We also provide a dual algorithm for minimizing the maximal number of connexions that need to be opened in order to achieve a given throughput. Section 4 presents extensive

simulation results comparing greedy based heuristics for our problem. At last, we provide in Section 5 some future works and concluding remarks.

## 2 Model and Related Works

### 2.1 Problem Modeling

Let us denote by  $b_j$  the capacity of server  $\mathcal{S}_j$  and by  $d_j$  the maximal number of connexions that it can handle simultaneously. The capacity of client  $\mathcal{C}_i$  is denoted by  $w_i$ . All capacities are normalized and expressed in terms of (fractional) number of tasks per time-unit. Moreover, let us denote by  $w_i^j$  the number of tasks per time-unit sent from server  $\mathcal{S}_j$  to client  $\mathcal{C}_i$ .

A valid solution can be depicted as a weighted bipartite graph between servers and clients (see Figure 1) where the following conditions are satisfied

$$\forall j, \quad \sum_i w_i^j \leq b_j \quad \text{capacity constraint at } \mathcal{S}_j \quad (1)$$

$$\forall j, \quad \text{Card}\{i, w_i^j > 0\} \leq d_j \quad \text{degree constraint at } \mathcal{S}_j \quad (2)$$

$$\forall i, \quad \sum_j w_i^j \leq w_i \quad \text{capacity constraint at } \mathcal{C}_i \quad (3)$$

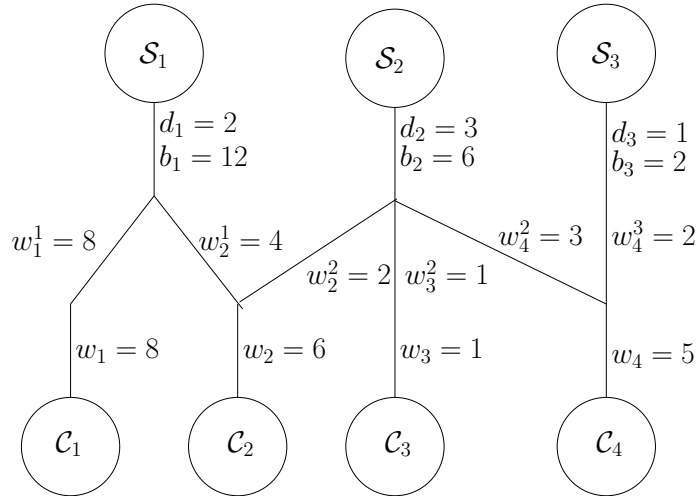


Figure 1: Optimal solution with 3 servers and 4 clients. Throughput=20

Our goal is to maximize the number of tasks that can be processed during one time-unit by the platform, what corresponds to problem MTBD.

**Maximize-Throughput-Bounded-Degree (MTBD):**

$$\text{Maximize } \sum_j \sum_i w_i^j \text{ under constraints (1),(2) and (3).}$$

The corresponding decision problem TBD-DEC can be formalized as follows.

**Throughput-Bounded-Degree-Dec (TBD-Dec):**

**Instance:** A set of  $m$  servers  $\mathcal{S}_1, \dots, \mathcal{S}_m$  with capacity  $b_j$  and degree  $d_j$ , a set of  $n$  clients  $\mathcal{C}_1, \dots, \mathcal{C}_n$  with capacity  $w_i$  and a bound  $K$

**Solution:** A weighted bipartite graph between servers and clients with weights  $w_i^j$  satisfying constraints (1),(2) and (3) and such that  $\sum_j \sum_i w_i^j \geq K$ .

TBD-DEC is clearly NP-Complete in the strong sense. For instance, we can use a reduction to **3-Partition** problem [11]. Indeed, let us consider an instance of **3-Partition** consisting of  $3m$  items  $a_i$  such that  $\sum a_i = mB$  and  $\forall i, \frac{B}{4} < a_i < \frac{B}{2}$  and let us set  $\forall j, d_j = 3, b_j = B, n = 3m, \forall i, w_i = a_i$  and  $K = mB$ . Since the overall out degree of the servers is at most  $3m$  and that all clients should be used in order to reach throughput  $mB$ , then each server should be connected to exactly 3 clients and no client should be connected to more than one server. Since the overall capacity of the server is  $mB$ , then each server should be connected to 3 clients whose aggregated capacity is exactly  $B$ , what achieves the NP-Completeness proof.

We will discuss related works dealing with bin packing of splittable items that provide more detailed complexity results in Section 2.2. It is worth noting that the complexity comes from the additional constraint related to the maximal number of connexions that a server can handle simultaneously. Indeed, without this constraint, the corresponding optimization problem becomes Maximize  $\sum_j \sum_i w_i^j$  under constraints (1) and (3) and can therefore be solved in

polynomial time using a linear program solver in rational numbers [17]. This situation is particularly annoying since the bound on the number of simultaneous connexions is a weak constraint. Indeed, even if it may be impossible for server  $\mathcal{S}_j$  to reach  $b_j$  by aggregating the bandwidths of a large number of connexions, the influence of one extra connexion on the aggregated bandwidth is very small. Therefore, the parameter  $d_j$  is mostly introduced to avoid pathological situations where thousands of nodes would connect to the same server.

In order to deal with this weak constraint, we propose in Section 3 a polynomial time algorithm that finds a solution where the maximal degree of a server is  $d_j + 1$  and whose throughput is at least as much as the optimal one with degree  $d_j$ . Since the degree constraint is weak, we can consider that, in practice, our algorithm is optimal. Moreover, the introduction of this extra parameter enables to avoid those pathological situations where too many clients would connect to the same server. We believe that this kind of techniques (resource augmentation on a weak parameter) may be used in many scheduling problems.

Based on the same ideas, the algorithm we propose is also an approximation algorithm for the following dual problem:

**Minimize-Degree-Given-Throughput (MDGT):** Minimize  $\alpha$  such that

$$\begin{array}{lll} \forall j, & \sum_i w_i^j \leq b_j & \text{capacity constraint at } \mathcal{S}_j \\ \forall j, & \text{Card}\{i, w_i^j > 0\} \leq d_j + \alpha & \text{degree constraint at } \mathcal{S}_j \\ \forall i, & \sum_j w_i^j \leq w_i & \text{capacity constraint at } \mathcal{C}_i \\ & \sum_i \sum_j w_i^j \geq T & \text{throughput larger than } T \end{array}$$

In particular, it is worth noting that if we set  $\forall j, d_j = 0$ , the optimal solution  $\alpha^*$  of the above optimization provides a solution where the maximal degree of a server is minimized. The corresponding decision problem is also trivially NP-Complete (on the previous instance with  $T = mB$ , deciding if  $\alpha^* = 3$  is equivalent to solve 3-Partition), but we provide in Section 3 an algorithm that outputs a valid solution with  $\alpha \leq \alpha^* + 1$ .

## 2.2 Related Works

A closely related problem is Bin Packing with Splittable Items and Cardinality Constraints. The goal in this problem is to pack a given set of items in as few bins as possible. The items may be split, but each bin may contain at most  $k$  items or pieces of items. This is very close to the problem we consider, with two main differences: in our case the number of servers (corresponding to bins) is fixed in advance, and the goal is to maximize the total bandwidth throughput (corresponding to the total packed size), whereas the goal in Bin Packing is to minimize the number of bins used to pack all the items. Furthermore, we consider heterogeneous servers.

As far as we know, Bin Packing with splittable items and cardinality constraints was introduced in the context of memory allocation in parallel processors by Chung et al. [7], who considered the special case when  $k = 2$ . They showed that even in that case this problem is NP-Complete, and proposed a  $3/2$ -approximation algorithm. Epstein and van Stee [9] showed that Bin Packing with splittable items and cardinality constraints is NP-Hard for any fixed value of  $k$ , and that the simple NEXT-FIT algorithm has an approximation ratio of  $2 - 1/k$ . They also designed a PTAS and a dual PTAS [8] for the general case with constant  $k$ .

Other related problems were introduced by Shachnai et al. [19], in which the size of an item increases when it is split, or there is a global bound on the number of fragmentations. The authors prove that these two problems do not admit a PTAS, and provide a dual PTAS and an asymptotic PTAS. In a multiprocessor scheduling context, another related problem is scheduling with allotment and parallelism constraints [18], where the goal is to schedule a certain number of tasks, where each task has a bound on the number of machines that can process it simultaneously, and another bound on the overall number of machines that can participate in its execution. This problem can also be seen as a splittable packing problem, but this time with a bound  $k_i$  on the number of times an item can be split. In [18], an approximation algorithm of ratio  $\max_i(1 + 1/k_i)$  is presented.

## 3 A Resource Augmentation Greedy Algorithm

In this section we present the algorithm SEQ for the problem MTBD, and we prove a result of optimality under resource augmentation.

### 3.1 The SEQ Algorithm

We present here a resource augmentation algorithm, in the sense that it provides a solution that slightly breaks one of the constraints of the problem, namely

constraint (2) that deals with the maximal degree of the servers. In the output solution of our SEQ algorithm, the number of clients that connect to a server  $\mathcal{S}_j$  is at most  $d_j + 1$  instead of  $d_j$  in constraint (2).

In the following, we will consider lists of clients sorted by increasing capacities, and if  $\mathcal{C} = \{\mathcal{C}_i\}$  denotes such a list, we will denote by  $\mathcal{C}(l, k) = \sum_{i=l}^k w_i$  the sum of the capacities of the clients between  $\mathcal{C}_l$  and  $\mathcal{C}_k$ , both of them included.

The SEQ algorithm maintains an ordered list of remaining clients, and at each step, picks up a server  $\mathcal{S}_j$  arbitrarily and goes through the list to find a suitable set of clients for this server. It only considers *consecutive* clients in the list, *i.e.* whose indexes form an interval of the form  $[l, l + d_j]$ . This property both decreases the complexity of the algorithm and ensures its correctness.

In order to avoid to waste connexions out of the server, it tries to allocate as many complete clients as possible. Thus, the only client that may be partially allocated to server  $\mathcal{S}_j$  is the last one in the interval, *i.e.* client  $\mathcal{C}_{l+d_j}$ . At last, we want to use all of the bandwidth of  $\mathcal{S}_j$ , so the sum of the capacities of the clients have to be at least the capacity  $b_j$  of the server. If such an interval exists (there may be several, but any of them does the trick), the SEQ algorithm allocates this set of clients to server  $\mathcal{S}_j$ . Client  $\mathcal{C}_{l+d_j}$  is then replaced by a new client whose capacity is equal to  $\mathcal{C}(l, l + d_j) - b_j$ . In that case the client  $\mathcal{C}_{l+d_j}$  will be linked to more than one server in the final solution. The list is then updated and reordered, and the algorithm goes on with the next server.

It may happen that there exists no interval with the desired properties, for two reasons. The first one is that  $d_j + 1$  clients are not enough to use all the bandwidth  $b_j$  (*i.e.* the overall capacity of the  $d_j + 1$  largest clients is not big enough). In this case, SEQ allocates to server  $\mathcal{S}_j$  the  $d_j + 1$  largest clients (the last ones in the list). On the other hand, if any set of  $d_j + 1$  clients has overall capacity larger than  $b_j$  (*i.e.* the overall capacity of the  $d_j + 1$  smallest clients is already too big), then the algorithm simply allocates the  $d$  smallest clients, where  $d$  is the smallest index such that  $\mathcal{C}(1, d) \geq b_j$ . In this case also, the last client may be split by creating a new client with capacity  $\mathcal{C}(1, d) - b_j$ . Algorithm 1 gives a more formal description of SEQ.

### 3.2 Approximation Results

In this section, we prove that for all instances  $I$  of MTBD, the throughput of the solution  $\text{SEQ}(I)$  computed by Algorithm SEQ is at least as good as any valid solution  $\mathcal{A}$  of instance  $I^1$ . To this end, we first claim that after the  $j$ -th step of SEQ, the remaining list of clients is *easier* to allocate with SEQ than the corresponding remaining clients of solution  $\mathcal{A}$  obtained by removing the clients allocated to servers  $\mathcal{S}_1, \dots, \mathcal{S}_j$ . In order to state formally what *easier* means, we define a relation  $\preceq$  on the (ordered) lists of clients.

**Definition 3.1** *Let  $\mathcal{C}$  and  $\mathcal{R}$  be two lists of clients of same length  $n$ , ordered by increasing capacities. We say that  $\mathcal{C}$  is easier than  $\mathcal{R}$  (denoted by  $\mathcal{C} \preceq \mathcal{R}$ ), if*

$$\forall k \leq n, \quad \mathcal{C}(1, k) \leq \mathcal{R}(1, k)$$

For the sake of simplicity, we consider that the length of the lists of clients remain  $n$  through all steps of the algorithm. Removed clients will thus be

<sup>1</sup>Remember that if the maximal degree of server  $\mathcal{S}_j$  is  $d_j$ , SEQ may use  $d_j + 1$  connexions out of  $\mathcal{S}_j$

---

**Algorithm 1** Algorithm SEQ
 

---

```

Set  $\mathcal{S} = \{\mathcal{S}_j\}_{j=1}^m$  and  $\mathcal{C} = \text{sort}(\{\mathcal{C}_i\}_{i=1}^n)$ ;
Set  $\mathcal{A} = \{\mathcal{A}_j = \{\emptyset\}\}_{j=1}^m$  and  $j = 1$ ;
for  $j = 1$  to  $m$  do
  if  $\exists l$  such that  $\mathcal{C}(l, l + d_j - 1) < b_j$  and  $\mathcal{C}(l, l + d_j) \geq b_j$  then
    Split  $\mathcal{C}_{l+d_j}$  in  $\mathcal{C}'_{l+d_j}$  and  $\mathcal{C}''_{l+d_j}$  with  $w_{l+d_j} = w'_{l+d_j} + w''_{l+d_j}$  and  $w''_{l+d_j} = b_j - \mathcal{C}(l, l + d_j - 1)$ 
    Set  $\mathcal{A}_j = \{\mathcal{C}_l, \mathcal{C}_{l+1}, \dots, \mathcal{C}_{l+d_j-1}, \mathcal{C}''_{l+d_j}\}$ 
    Remove  $\mathcal{C}_l, \mathcal{C}_{l+1}, \dots, \mathcal{C}_{l+d_j}$  from  $\mathcal{C}$ 
    Insert  $\mathcal{C}'_{l+d_j}$  in  $\mathcal{C}$ 
  end if
  if  $\mathcal{C}(1, d_j) \geq b_j$  then
    Search for the smallest  $l$  such that  $\mathcal{C}(1, l) \geq b_j$ 
    Split  $\mathcal{C}_l$  in  $\mathcal{C}'_l$  and  $\mathcal{C}''_l$  with  $w_l = w'_l + w''_l$  and  $w''_l = b_j - \mathcal{C}(1, l - 1)$ 
    Set  $\mathcal{A}_j = \{\mathcal{C}_1, \mathcal{C}_2, \dots, \mathcal{C}_{l-1}, \mathcal{C}''_l\}$ 
    Remove  $\mathcal{C}_1, \mathcal{C}_2, \dots, \mathcal{C}_l$  from  $\mathcal{C}$ 
    Insert  $\mathcal{C}'_l$  in  $\mathcal{C}$ 
  end if
  if  $\mathcal{C}(n - d_j, n) < b_j$  then
    Set  $\mathcal{A}_j = \{\mathcal{C}_{n-d_j}, \mathcal{C}_{n-d_j+1}, \dots, \mathcal{C}_n\}$ 
    Remove  $\mathcal{C}_{n-d_j}, \mathcal{C}_{n-d_j+1}, \dots, \mathcal{C}_n$  from  $\mathcal{C}$ 
  end if
end for
RETURN  $\mathcal{A} = \{\mathcal{A}_j\}_{j=1}^m$ 
    
```

---

considered as 0-capacity clients (and inserted at the beginning of the lists). Note that is does not change the behavior of the algorithm.

The following definition formalizes what the constraints of the problem induce on the lists of remaining clients of an arbitrary solution.

**Definition 3.2** Let  $\mathcal{R}$  be a list of clients of length  $n$ , in which client  $i$  has capacity  $r_i$ . We say that a list  $\mathcal{R}'$  is obtained from  $\mathcal{R}$  with degree  $d$  and capacity  $b$  (denoted by  $\mathcal{R} \xrightarrow{(d,b)} \mathcal{R}'$ ) if

- there exists  $C \subseteq \{1, \dots, n\}$  with  $\text{Card}(C) \leq d$ ,
- there exists  $(v_i)_{i \in C}$  such that  $\forall i \in C, v_i \leq r_i$  and  $\sum_{i \in C} v_i \leq b$ ,
- $\mathcal{R}' = \sigma(\mathcal{R}/r'_i \leftarrow r_i - v_i, i \in C)$  for some sorting permutation  $\sigma$ .

Let us now consider a given step of the algorithm SEQ, in which the considered server has capacity  $b$  and degree  $d$ . We denote by  $\mathcal{C}$  and  $\mathcal{C}'$  the lists of remaining clients respectively before and after this step. The central lemma of our proof is the following.

**Lemma 3.3** If  $\mathcal{C} \preceq \mathcal{R}$  and  $\mathcal{R} \xrightarrow{(d,b)} \mathcal{R}'$ , then  $\mathcal{C}' \preceq \mathcal{R}'$ .

**Proof:** We begin by proving two lower bounds for  $\mathcal{R}'(1, k)$ . Let  $\sigma$  be the permutation of  $\{1, \dots, n\}$  that is used to sort  $(\mathcal{R}/r'_i \leftarrow r_i - v_i, i \in C)$  in order to obtain  $\mathcal{R}'$ , which means  $r'_{\sigma(i)} = r_i$  if  $i \notin C$ , and  $r'_{\sigma(i)} = r_i - v_i$  if  $i \in C$ . Then,

$$\mathcal{R}'(1, k) = \sum_{i: i \notin c \wedge \sigma(i) \leq k} r_i + \sum_{i: i \in c \wedge \sigma(i) \leq k} r_i - v_i = \sum_{i: \sigma(i) \leq k} r_i - \sum_{i: i \in c \wedge \sigma(i) \leq k} v_i$$

For  $k \geq d$ , since there are at least  $k - d$  indexes  $i$  that satisfy  $i \notin c \wedge \sigma(i) \leq k$ , then  $\sum_{i: i \notin c \wedge \sigma(i) \leq k} r_i \geq \mathcal{R}(1, k - d)$ . Together with the fact that  $r_i - v_i \geq 0$ , we obtain the first upper bound

$$\mathcal{R}'(1, k) \geq \mathcal{R}(1, k - d) \quad \forall k > d. \quad (4)$$

Similarly, since there are  $k$  indexes  $i$  such that  $\sigma(i) \leq k$ , then  $\sum_{i: \sigma(i) \leq k} r_i \geq \mathcal{R}(1, k)$ . Together with the fact that  $\sum_{i \in C} v_i \leq b$ , we obtain the second upper bound

$$\mathcal{R}'(1, k) \geq \mathcal{R}(1, k) - b \quad (5)$$

To complete the proof, we need to evaluate  $\mathcal{C}'(1, k)$ . Since we identified three main situations when adding a server, we evaluate  $\mathcal{C}'(1, k)$  for each possible situation.

**Case 1:**  $\exists l$  such that  $A(l, l + d - 1) < b$  and  $A(l, l + d) \geq b$ . In this case the algorithm allocates completely clients  $\mathcal{C}_l, \mathcal{C}_{l+1}, \dots, \mathcal{C}_{l+d-1}$  to  $\mathcal{S}_j$  and splits  $\mathcal{C}_{l+d}$  into  $\mathcal{C}'_{l+d}$  and  $\mathcal{C}''_{l+d}$ . The first  $d$  clients of the list  $\mathcal{C}'$  will thus have zero capacity, and  $\mathcal{C}'_{l+d}$  will be reinserted in a position before  $\mathcal{C}_{l+d+1}$ , say between  $\mathcal{C}_p$  and  $\mathcal{C}_{p+1}$ .

Then  $\mathcal{C}' = \{\mathcal{C}'_l, \dots, \mathcal{C}'_{l+d-1}, \mathcal{C}_1, \dots, \mathcal{C}_p, \mathcal{C}'_{l+d}, \mathcal{C}_{p+1}, \dots, \mathcal{C}_{l-1}, \mathcal{C}_{l+d+1}, \dots, \mathcal{C}_n\}$ , and therefore

$$\mathcal{C}'(1, k) = \begin{cases} 0 & \text{for } k \leq d \\ \mathcal{C}(1, k - d) & \text{for } d < k \leq p + d \\ \mathcal{C}(1, k - d - 1) + w'_{l+d} & \text{for } p + d < k \leq l + d \\ \mathcal{C}(1, k) - b & \text{for } l + d < k \end{cases} \quad (6)$$

Indeed, for  $k \leq d$ ,  $\mathcal{C}'(1, k)$  is a sum over the completely allocated reinserted clients, and thus is a sum up to zero. For the second interval  $d < k \leq p + d$ ,  $\mathcal{C}'(1, k)$  is a sum of the first  $k - d$  capacities in  $\mathcal{C}$ , since they were shifted by  $d$  positions (due to the insertion of  $d$  clients at the beginning of the list). In the third interval  $p + d < k \leq l + d$ , the sum is the same than in the previous interval, but the last element in the sum is replaced by the size of the split client that has been inserted. Finally when  $l + d < k$ , the sum is equal to the sum in the original list, decreased by the total capacity allocated to  $\mathcal{S}$ .

Now, using equations (5) and (4), the fact that  $\mathcal{C} \leq \mathcal{R}$  and (6), we have

$$\begin{aligned} \mathcal{C}'(1, k) &= 0 \leq \mathcal{R}'(1, k) && \text{for } k \leq d \\ \mathcal{C}'(1, k) &= \mathcal{C}(1, k - d) \leq \mathcal{R}(1, k - d) \leq \mathcal{R}'(1, k) && \text{for } d < k \leq p + d \\ \mathcal{C}'(1, k) &= \mathcal{C}(1, k - d - 1) + w'_{l+d} \\ &\leq \mathcal{C}(1, k - d) \leq \mathcal{R}(1, k - d) \leq \mathcal{R}'(1, k) && \text{for } p + d < k \leq l + d \\ \mathcal{C}'(1, k) &= \mathcal{C}(1, k) - b \leq \mathcal{R}(1, k) - b \leq \mathcal{R}'(1, k) && \text{for } l + d < k. \end{aligned}$$

**Case 2:**  $A(1, d) \geq b$ . In this case, since SEQ uses the first  $l \leq d$  clients, there is no reordering of the list. The new list  $\mathcal{C}'$  can therefore be written as  $\{\mathcal{C}'_1, \dots, \mathcal{C}'_{l-1}, \mathcal{C}'_l, \mathcal{C}_{l+1}, \dots, \mathcal{C}_n\}$ , where  $\mathcal{C}'_i$  has zero capacity for  $i < l$ . Moreover, since the overall allocated capacity is equal to  $b$ , we have

$$\mathcal{C}'(1, k) = \begin{cases} 0 & \text{for } k \leq l - 1 \\ \mathcal{C}(1, k) - b & \text{for } k > l - 1 \end{cases}$$

Hence, by equation (5) together with the fact that  $\mathcal{C} \preceq \mathcal{R}$ , we have  $\mathcal{C}'(1, k) \leq \mathcal{R}'(1, k)$ .

**Case 3:**  $A(n - d, n) < b$ . In this case, SEQ allocates completely the  $d + 1$  last clients to  $\mathcal{S}$ , and therefore all reinserted clients  $\mathcal{C}'_i$  will have zero capacity and will be reinserted at the beginning of the list. The new list  $\mathcal{C}'$  can therefore be written as  $\{\mathcal{C}'_{n-d}, \dots, \mathcal{C}'_n, \mathcal{C}_1, \dots, \mathcal{C}_{n-d-1}\}$ . Therefore,

$$\mathcal{C}'(1, k) = \begin{cases} 0 & \text{for } k \leq d + 1 \\ \mathcal{C}(1, k - (d + 1)) & \text{for } k > d + 1 \end{cases}$$

Once again, by equation (4) together with  $\mathcal{C} \preceq \mathcal{R}$ , we have  $\mathcal{C}'(1, k) \leq \mathcal{R}'(1, k)$ . ■

Now we can state and prove our main result.

**Theorem 3.4** *Let  $\mathcal{A}$  be any valid solution of an instance  $I$ , and  $\text{SEQ}(I)$  be the solution given by algorithm SEQ. Then, the throughput of  $\text{SEQ}(I)$  is at least as much as the throughput of  $\mathcal{A}$ .*

**Proof:** Let us denote by  $\mathcal{LC}$  and  $\mathcal{LS}$  the lists of clients and servers of instance  $I$ , respectively. Let  $w_i^j$  be the bandwidth allocated to client  $\mathcal{C}_i$  on server  $\mathcal{S}_j$  in the solution  $\mathcal{A}$ . For all  $0 \leq j \leq m$ , we will denote by  $\mathcal{LC}_j$  the list of remaining clients using SEQ after step  $j$  (hence  $\mathcal{LC}_0 = \mathcal{LC}$ ), and by  $\mathcal{LR}_j$  the list of remaining clients if we remove servers  $\mathcal{S}_1, \dots, \mathcal{S}_j$  from solution  $\mathcal{A}$ , i.e.  $\mathcal{LR}_j = (w_i - \sum_{k=1}^j w_i^k)_{i \leq n}$ .

Since  $\mathcal{A}$  satisfies the constraints (1), (2) and (3), then  $\mathcal{LR}_{j-1} \xrightarrow{(d_j, b_j)} \mathcal{LR}_j$ . Lemma 3.3 provides the following implication:  $\mathcal{LC}_j \preceq \mathcal{LR}_j \Rightarrow \mathcal{LC}_{j+1} \preceq \mathcal{LR}_{j+1}$  for all  $j < m$ . Since  $\mathcal{LC}_0 = \mathcal{LR}_0 = \mathcal{LC}$ , a simple induction proves that  $\mathcal{LC}_j \preceq \mathcal{LR}_j$  for all  $j$ . In particular,

$$\mathcal{LC}_m \preceq \mathcal{LR}_m, \text{ and } \mathcal{LC}_m(1, n) \leq \mathcal{LR}_m(1, n). \quad (7)$$

Remark now that  $\mathcal{LC}_m(1, n)$  is the sum of the capacities of the clients that have not been allocated (or have been allocated only partially because of splittings) using SEQ, and so the throughput of  $\text{SEQ}(I)$  is given by  $\sum_i w_i - \mathcal{LC}_m(1, n)$ . Similarly, the throughput of  $\mathcal{A}$  is given by  $\sum_i w_i - \mathcal{LR}_m(1, n)$ . Therefore, Equation (7) completes the proof of the theorem. ■

### 3.3 An Approximation Algorithm for the Dual Problem

This resource augmentation result can also be seen as an approximation for the problem MDGT (Minimize Degree for a Given Throughput). Indeed, if we are given a bound  $B \leq \min(\sum_j b_j, \sum_i w_i)$  on the throughput, a simple dichotomic search allows to find the minimum value  $\alpha_{\text{SEQ}}$  of  $\alpha$  such that the throughput of  $\text{SEQ}(I(\alpha))$  is at least  $B$  on the modified instance  $I(\alpha)$  in which server  $\mathcal{S}_j$  has degree  $d_j + \alpha$ . Theorem 3.4 states that if there is a solution  $\mathcal{A}$  of throughput  $B$  for instance  $I(\alpha - 1)$ , then  $\text{SEQ}(I(\alpha - 1))$  provides a valid solution for instance  $I(\alpha)$  of throughput at least  $B$ .

We have thus that  $\alpha_{\text{SEQ}} \leq \alpha^* + 1$ , where  $\alpha^*$  is the optimal value of the problem MDGT for instance  $I$ . Since MDGT is NP-complete, this is the best possible approximation result.

## 4 Simulation Results

### 4.1 Heuristics for Comparison

As already mentioned in Section 2.2, related work has been mostly done in the context of *Bin Packing*, where there is an infinite amount of identical bins, and the goal is to pack all items in as few bins as possible<sup>2</sup>. Interestingly, in this setting, the NEXT-FIT algorithm has a worst-case approximation ratio of  $2 - 1/k$  [9], but it can easily be observed that it does not exhibit a constant approximation ratio for the total packed size when the number of bins is fixed. Moreover, most of existing algorithms in this context are approximation schemes, with prohibitive running times. To provide a basis of comparison, we thus introduce three basic and natural greedy heuristics.

**LCLS (Largest Client Largest Server)** At each step, the client with largest  $w_i$  is associated with the server with largest available capacity  $b'_j = b_j - \sum_i w_i^j$ . The client is split if necessary, in which case the remaining  $w'_i = w_i - b'_j$  is inserted in the ordered list.

**LCBC (Largest Client Best Connexion)** In this heuristic, we also consider the largest client first, but servers are ordered according to their remaining *capacity per connexion*, which is defined as the ratio between the remaining capacity  $b'_j$  and the remaining available degree  $d'_j$ . The server with the largest capacity per connexion is selected. Here also, the client is split if necessary.

**OBC (Online Best Connection)** This heuristic is an online version of the previous one. All the servers are supposed to be known at the beginning of the execution, but the clients arrive at arbitrary time steps. To model this setting, the clients are considered in an arbitrary order, and we select the server with a remaining capacity per connexion as close as possible to the client's capacity. More precisely, we select the server with the largest  $b'_j/d'_j$  such that  $b'_j/d'_j \leq w_i$ .

<sup>2</sup>In our context, servers are bins and clients are items

## 4.2 Random Instance Generation

We generate instances randomly, trying to focus at the same time on realistic scenarios and difficult instances. Instances are more difficult to solve when the sum of the server capacities is roughly equal to the sum of the client capacities. Indeed, the minimum of both is a trivial upper bound on the total achievable throughput, and a large difference between them provides a lot of freedom on the largest component to reach this upper bound. Based on the same idea, we generate instances where the sum of the server degrees  $\sum_j d_j$  is roughly equal to the number  $n$  of clients.

In order to get a realistic distribution of server and client capacities, we have used information available from the volunteer computing project GIMPS [12] that provides the average computing power of all its participants. A simple statistical study shows that the computational power (based on the 7,000 largest participants) follows a power-law distribution with exponent  $\hat{\alpha} \approx 2.09$ . We have thus used this distribution and this exponent to generate the capacities of both clients and servers. The resulting values are then scaled so that their sums ( $\sum_i w_i$  and  $\sum_j b_j$ ) are roughly equal. Furthermore, the degree  $d_j$  of server  $\mathcal{S}_j$  is chosen proportional to its capacity  $b_j$  (it seems reasonable to assume that a server with larger capacity can accommodate more clients), with a gaussian multiplicative factor of mean 1 and variance 0.1.

## 4.3 Results

In the first set of experiments, we have measured the throughput of the solutions proposed by each algorithm. All values are normalized against the previously mentioned upper bound  $\min(\sum_j b_j, \sum_i w_i)$ . Figure 2(a) shows the average results on 250 instances when the number of servers varies from 20 to 140 (the number of clients is always 10 times the number of servers, and thus the average degree of the servers is 10), and Figure 2(b) shows the result for all the instances with 80 servers, which is a typical case. We can already make some remarks:

- For these instances, the SEQ algorithm performs consistently better than the others. In fact, it almost always reaches the upper bound.
- The performance of the LCBC algorithm is around 4% worse, and LCLS is around 10-12% worse than SEQ.
- OBC does not perform too badly on average, but it exhibits a much more higher dispersion than the others. This can be explained by the fact that it is an “online” algorithm, and thus its performance highly depends on the ordering in which clients arrive.

As we tried to investigate the variability of the results obtained, it appeared that LCBC performance strongly depends on the heterogeneity of the client capacities. We have thus plotted on Figure 3 the results for 1000 instances with  $m = 80$  against the relative mean difference<sup>3</sup> of the client capacities, which is a measure of their dispersion. For the sake of readability, we have separated the plots for LCBC and LCLS (the performance of SEQ is not sensitive to this heterogeneity measure, so the graph is not shown to save space).

<sup>3</sup>The *mean difference* of values  $\{y_i\}$  is the average absolute difference of all couples of values. The *relative mean difference* is the *mean difference* divided by the arithmetic mean.

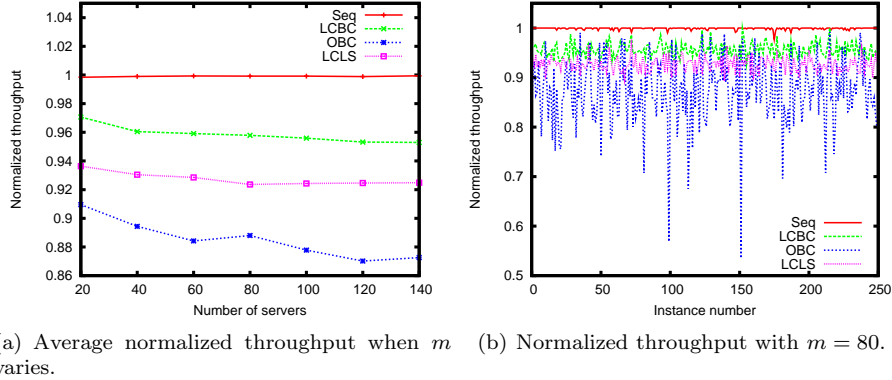


Figure 2: Simulation results

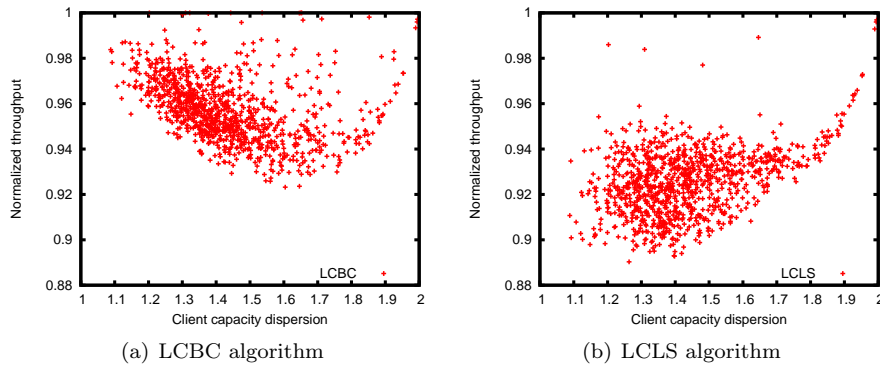
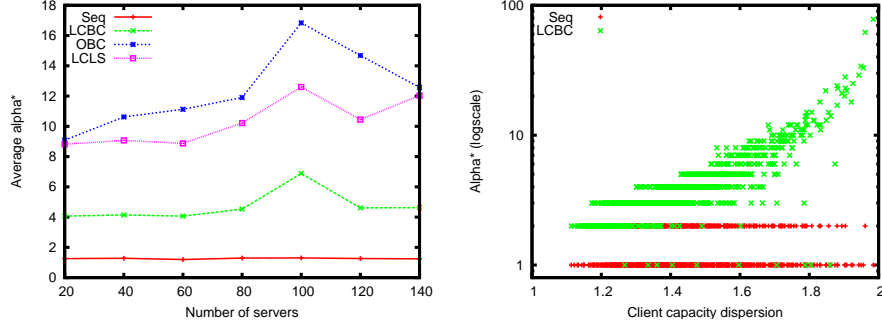


Figure 3: Normalized throughput against dispersion of client capacities, with  $m = 80$ .

We can observe on these plots that the performance of LCBC gets worse when the heterogeneity increases, at least up to a certain point. On the other hand, the performance of LCLS gets better when heterogeneity is high, and at some point both algorithms perform similarly well. This can be explained by the fact that when there is a very large client in the instance, it is more important to assign it to a large server than to the server with the largest capacity per connexion. Indeed, if the latter is a relatively small server, the large client is split, and the connexions of the smaller server are wasted.

In a second set of experiments, we have computed for each algorithm  $\mathcal{A}$  the minimum value  $\alpha^*$  that needs to be added to the degree of each server so that algorithm  $\mathcal{A}$  reaches the upper bound  $B = \min(\sum_j b_j, \sum_i w_i)$ . Note that the results of Section 3 do not imply that  $\alpha^* \leq 1$  for algorithm SEQ, since it may well be the case that the upper bound cannot be reached with the original degree sequence. Average results for all algorithms and for varying  $m$  are depicted in Figure 4(a), and the values of  $\alpha^*$  against the dispersion of client sizes for  $m = 80$  are depicted in Figure 4(b).

(a) Average  $\alpha^*$  for 250 instances for varying  $m$  (b)  $\alpha^*$  values against dispersion for  $m = 80$ Figure 4: Plots of  $\alpha^*$ , the smallest value such that  $\mathcal{A}(I/d_j \leftarrow d_j + \alpha) = B$ .

We can see that as expected, the SEQ algorithm makes very good use of the additional degree, and can almost always reach the upper bound with an increase of 1 or 2. As expected also, the ranking of algorithms observed for the total throughput is still the same when considering  $\alpha^*$ . We see that with LCBC, one needs about 4 more connexions to reach the bound, and that this number becomes 10 with LCLS and 12 with OBC. Remember that in all of the instances considered, the average degree of the servers is 10. Simply reasoning about the average values does not give much more information, but the second graph shows that most of the values for LCBC are between 2 and 5. However, it can be as high as 80 for instances with large dispersion in client capacities, and these high values tend to increase the average. Examination of the results for algorithms LCLS and OBC exhibit the same kind of behaviors, with larger values of  $\alpha^*$  for the most homogeneous instances, and this explains larger average values. Therefore, for these difficult heterogeneous instances, we can see the benefit of the guarantee proved in Section 3 for algorithm SEQ.

## 5 Conclusion

We considered the problem of allocating a large set of tasks to a fully heterogeneous platform made of servers and clients. We proved that if we add a bound on the maximal number of open connexions a server can handle simultaneously, the problem of maximizing the overall throughput becomes NP-Complete in the strong sense. Nevertheless, we also provided a polynomial time algorithm that reaches the optimal throughput using a very small resource augmentation on the number of connexions. More specifically, we proved that, if  $d_j$  denotes the maximal number of connexions that can be opened at node  $\mathcal{S}_j$ , then the throughput achieved using this algorithm and degree  $d_j + 1$  is at least the same as the optimal one with degree  $d_j$ . Finally, we also proposed extensive simulations to assess the performance of proposed algorithm.

The approach presented in this paper consists in determining a weak constraint that makes an allocation problem NP-Complete and then to perform resource augmentation on this parameter. We believe that this approach is very promising in the context of steady state scheduling, because it enables to con-

sider more realistic communication models without relying on approximation algorithms that limit the expected throughput.

A natural extension of the work presented in this paper would consist in considering the on-line case, where the set of clients is not known in advance<sup>4</sup>. Simulations performed with a natural greedy on-line algorithm tend to prove that the problem is more difficult in this case, but the questions of finding an appropriate resource augmentation or a satisfying approximation ratio are still open. Another interesting extension would consist in considering more complex virtual topologies (overlay networks) to organize the participating clients. Indeed, the clients have themselves some available outgoing bandwidth and may therefore be used both for processing tasks and for sending data to other clients. This is particularly desirable in the context where the number of opened connections at a node is bounded and therefore where it may not be possible to use all available resources, even if the overall throughput out of the servers is not exceeded.

## 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] D.P. Anderson, J. Cobb, E. Korpela, M. Lebofsky, and D. Werthimer. SETI@ home: an experiment in public-resource computing. *Communications of the ACM*, 45(11):56–61, 2002.
- [3] C. Banino, O. Beaumont, L. Carter, J. Ferrante, A. Legrand, and Y. Robert. Scheduling Strategies for Master-Slave Tasking on Heterogeneous Processor Platforms. *IEEE Transactions on Parallel and Distributed Systems*, pages 319–330, 2004.
- [4] O. Beaumont, A. Legrand, L. Marchal, and Y. Robert. Pipelining Broadcasts on Heterogeneous Platforms. *IEEE Transactions on Parallel and Distributed Systems*, pages 300–313, 2005.
- [5] D. Bertsimas and D. Gamarnik. Asymptotically optimal algorithm for job shop scheduling and packet routing. *Journal of Algorithms*, 33(2):296–318, 1999.
- [6] Martin A. Brown. Traffic Control HOWTO. Chapter 6. Classless Queuing Disciplines. <http://tldp.org/HOWTO/Traffic-Control-HOWTO/classless-qdiscs.html>, 2006.
- [7] F. Chung, R. Graham, J. Mao, and G. Varghese. Parallelism versus Memory Allocation in Pipelined Router Forwarding Engines. *Theory of Computing Systems*, 39(6):829–849, 2006.
- [8] L. Epstein and R. van Stee. Approximation Schemes for Packing Splittable Items with Cardinality Constraints. *Lecture Notes in Computer Science*, 4927:232, 2008.
- [9] Leah Epstein and Rob van Stee. Improved results for a memory allocation problem. In Frank K. H. A. Dehne, Jörg-Rüdiger Sack, and Norbert Zeh, editors, *WADS*, volume 4619 of *Lecture Notes in Computer Science*, pages 362–373. Springer, 2007.
- [10] 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.
- [11] M.R. Garey and D.S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-completeness*. WH Freeman San Francisco, 1979.
- [12] The great internet mersenne prime search (gimps). <http://www.mersenne.org/>.
- [13] B. Hong and V.K. Prasanna. Distributed adaptive task allocation in heterogeneous computing environments to maximize throughput. *International Parallel and Distributed Processing Symposium, 2004. Proceedings. 18th International*, 2004.

<sup>4</sup>It is worth noting that the algorithm we propose can be considered as on-line if clients are known at the beginning of the execution and servers are added on-line, even if it is not the most reasonable setting for the on line case.

- [14] B. Hubert et al. Linux Advanced Routing & Traffic Control. Chapter 9. Queueing Disciplines for Bandwidth Management. <http://lartc.org/lartc.pdf>, 2002.
- [15] 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.
- [16] T. Saif and M. Parashar. Understanding the Behavior and Performance of Non-blocking Communications in MPI. *Lecture Notes in Computer Science*, pages 173–182, 2004.
- [17] A. Schrijver. *Theory of linear and integer programming*. John Wiley & Sons, Inc. New York, NY, USA, 1986.
- [18] H. Shachnai and T. Tamir. Multiprocessor Scheduling with Machine Allotment and Parallelism Constraints. *Algorithmica*, 32(4):651–678, 2002.
- [19] Hadas Shachnai, Tami Tamir, and Omer Yehezky. Approximation schemes for packing with item fragmentation. *Theory Comput. Syst.*, 43(1):81–98, 2008.
- [20] 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.



---

Centre de recherche INRIA Bordeaux – Sud Ouest  
Domaine Universitaire - 351, cours de la Libération - 33405 Talence Cedex (France)

Centre de recherche INRIA Grenoble – Rhône-Alpes : 655, avenue de l'Europe - 38334 Montbonnot Saint-Ismier  
Centre de recherche INRIA Lille – Nord Europe : Parc Scientifique de la Haute Borne - 40, avenue Halley - 59650 Villeneuve d'Ascq  
Centre de recherche INRIA Nancy – Grand Est : LORIA, Technopôle de Nancy-Brabois - Campus scientifique  
615, rue du Jardin Botanique - BP 101 - 54602 Villers-lès-Nancy Cedex  
Centre de recherche INRIA Paris – Rocquencourt : Domaine de Voluceau - Rocquencourt - BP 105 - 78153 Le Chesnay Cedex  
Centre de recherche INRIA Rennes – Bretagne Atlantique : IRISA, Campus universitaire de Beaulieu - 35042 Rennes Cedex  
Centre de recherche INRIA Saclay – Île-de-France : Parc Orsay Université - ZAC des Vignes : 4, rue Jacques Monod - 91893 Orsay Cedex  
Centre de recherche INRIA Sophia Antipolis – Méditerranée : 2004, route des Lucioles - BP 93 - 06902 Sophia Antipolis Cedex

---

Éditeur  
INRIA - Domaine de Voluceau - Rocquencourt, BP 105 - 78153 Le Chesnay Cedex (France)  
<http://www.inria.fr>  
ISSN 0249-6399