





# Interference-Aware Scheduling Using Geometric Constraints

Raphaël Bleuse<sup>1,2</sup> , Konstantinos Dogeas<sup>1</sup>, Giorgio Lucarelli<sup>1</sup> ,  
Grégory Mounié<sup>1</sup> , and Denis Trystram<sup>1</sup> 

<sup>1</sup> Univ. Grenoble Alpes, CNRS, Inria, Grenoble INP, LIG, Grenoble, France  
{konstantinos.dogeas,giorgio.lucarelli,gregory.mounie,  
denis.trystram}@imag.fr

<sup>2</sup> FSTC/CSC, University of Luxembourg, Luxembourg City, Luxembourg  
raphael.bleuse@uni.lu

**Abstract.** The large scale parallel and distributed platforms produce a continuously increasing amount of data which have to be stored, exchanged and used by various jobs allocated on different nodes of the platform. The management of this huge communication demand is crucial for the performance of the system. Meanwhile, we have to deal with more interferences as the trend is to use a single all-purpose interconnection network. In this paper, we consider two different types of communications: the flows induced by data exchanges during computations and the flows related to Input/Output operations. We propose a general model for interference-aware scheduling, where explicit communications are replaced by external topological constraints. Specifically, we limit the interferences of both communication types by adding geometric constraints on the allocation of jobs into machines. The proposed constraints reduce implicitly the data movements by restricting the set of possible allocations for each job. We present this methodology on the case study of simple network topologies, namely the line and the ring. We propose theoretical lower and upper bounds under different assumptions with respect to the platform and jobs characteristics. The obtained results illustrate well the difficulty of the problem even on simple topologies.

## 1 Introduction

In High Performance Computing, the demand for computational power is steadily increasing [16]. To meet up with the challenge of greater performance the architecture of supercomputers also grows in complexity at the whole machine scale. This complexity arises from various factors: (i) the size of the machines (supercomputers now integrates millions of cores); (ii) the heterogeneity of the resources (various architectures of computing nodes, nodes dedicated to I/O); (iii) the interconnection topology. The evolution in the interconnection networks faces two main challenges: first, the community is proposing new topologies [12]; and second, the interconnection network is now unique within the machine (the

network is shared for various mixed data flows). Sharing such a single multi-purpose interconnection network creates complex interactions (e.g., network contention) between running applications, which have a strong impact on their performance [1, 5], and limits the understanding of the system by the users [3]. As the volume of processed data increases, so does the impact of the network.

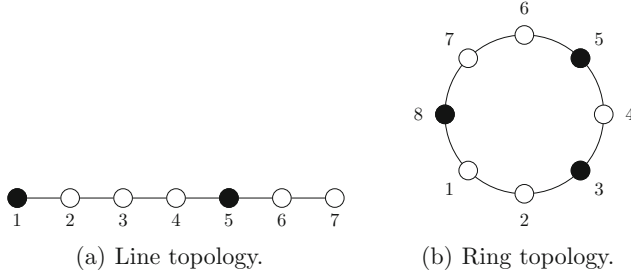
In this work, we introduce a generic framework for interference-aware scheduling. More precisely, we identify two main types of interleaved flows: the flows induced by data exchanges for computations and the flows related to I/O. Rather than explicitly taking into account these network flows, we address the issue of harmful interactions by constraining the shape of the allocations. Such an approach aims at taking into account the structure of the new platforms in a qualitative way that is more likely to scale properly. The scheduling problem is then defined as an optimization problem with the platform (nodes and topology) and the jobs' description as input. The objective is to minimize the maximum completion time while enforcing constraints on the allocations.

## 2 Problem Setting

In this work, we model a platform as a set  $\mathcal{V}$  of  $m$  nodes divided in two groups: a set  $\mathcal{V}^C$  of  $m^C$  nodes dedicated to computations, and a set  $\mathcal{V}^{I/O}$  of  $m^{I/O}$  nodes that are entry points to a high performance file system. As a consequence, we have  $m = m^C + m^{I/O}$ . We assume that the I/O nodes are exclusively used for communications with the file system and hence, there is no overlap between computing and I/O nodes, i.e.,  $\mathcal{V}^{I/O} \cap \mathcal{V}^C = \emptyset$ . Moreover, a computing or an I/O node is exclusively allocated to a job for its lifespan, i.e., any node cannot be used at the same time by more than one job.

The nodes can communicate using an interconnection network with a given *topology*, while the localization of every node within the topology is known. In this direction, we study here the instantiation of this framework with unidimensional topologies, namely the line (Fig. 1(a)) and the ring (Fig. 1(b)). Studying topologies of one dimension is a first step towards the more complicated state-of-the-art platforms, while these basic topologies provide lower bounds for the later ones. The line may indeed be seen as a degenerate tree. Fat-tree topologies are a common interconnect, and are for example used in the Curie and Oakforest-PACS platforms. On the other hand, the torus topologies, such as the one of Blue Waters and Titan (3D torus) or the K computer (6D torus), may be studied from the ring with classical embedding techniques.

Batch schedulers are a critical part of the software stack managing HPC platforms: their goal is to efficiently allocate resources (nodes from  $\mathcal{V}$  in our case) to the jobs submitted by the users of the platform. The jobs are queued in a set  $\mathcal{J}$ . The total number of jobs is  $n$ . Each job  $j$  requires  $q_j$  computing nodes and one I/O node. We distinguish two cases with respect to I/O requirements: in the *pinned* model each job asks for a specific I/O node, while in the *unpinned* model the jobs just need any arbitrary I/O node. The number of allocated computing nodes is fixed, i.e., the job is *rigid* [6]. We denote by  $\mathcal{V}(j)$  the set of nodes



**Fig. 1.** Example of platforms with unidimensional topologies. The nodes are numbered using the natural order. White nodes represent computing nodes, and black nodes represent I/O nodes.

allocated to the job  $j$ . If needed, we use  $\mathcal{V}^C(j)$  and  $\mathcal{V}^{I/O}(j)$  to distinguish among the computing and the I/O nodes assigned in  $j$ , respectively. Each job  $j$  requires a certain time  $p_j$  to be processed, and it is *independent* of every other job. Once a job starts being executed, it runs until completion, i.e., it is *not preemptive*.

As stated above, the goal of this paper is not to finely model the full context of execution. Instead, we propose to model the platform in such a way that the network interactions are *implicitly* taken into account. In this direction, we augment the scheduling problem with geometric constraints on the allocations of the jobs in the resources based on the platform topology and the application requirements. Before presenting these constraints, we need to precisely define the network flows we target. We distinguish two types of flows, directly deriving from the fact that we are dealing with two kinds of nodes:

**computational communications** which are induced by data exchanges during computations. Such communications occur between two computing nodes allocated to the same application.

**I/O communications** which are induced by data exchanges between computing and I/O nodes. Such communications occur when computing nodes read input data, checkpoint the state of the application, or save output results.

In order to avoid computational communication interactions, we consider the following constraint.

**Definition 1 (Contiguity [2,14]).** *An allocation is said to be contiguous if and only if the nodes of the allocation form a contiguous range with respect to the nodes' ordering.*

Note that the contiguity constraint relies on the nodes' ordering. For topologies such as lines or rings this ordering is natural (see Fig. 1).

The contiguity constraint is well suited to take into account the computational communications, but not the I/O communications. Indeed, the former type of communications may occur between any pair of computing nodes within an allocation: we usually describe this pattern as all-to-all communication. On

the other hand, I/O communications generate traffic towards few identified nodes in an all-to-one or one-to-all pattern. Hence, we propose the *locality* constraint, whose goal is to limit the impact of the I/O flows to the periphery of the job allocations. We must emphasize that the locality constraint proposed here is not related to the locality constraint described in [14].

**Definition 2 (Locality).** *A given allocation for a job  $j$  is said to be local iff it is contiguous, and the I/O node  $\mathcal{V}^{I/O}(j)$  is adjacent to computing nodes from  $\mathcal{V}^C(j)$ , with respect to the underlying topology.*

In this paper, we are interested in minimizing the maximum completion time among all jobs (i.e., the *makespan* of the schedule) while enforcing the *contiguity* and the *locality* constraints. Specifically, we aim at developing algorithms with performance guarantees by adding geometric constraints on the allocations of jobs into nodes.

### 3 Related Work

Most actual implementations of schedulers allocate resources greedily without any topological constraint in the allocation of the computing nodes. However, this naive solution has a bad impact on performances [5]. Constraining the allocations to enhance performance is however not a new idea. For example, Lucarelli et al. studied the impact of enforcing contiguity or locality in backfilling scheduling [14] (for fat trees). They showed that enforcing these constraints can be done at a small cost, and has minimum negative impact on usual metrics such as makespan, flow-time, or stretch.

Tackling the interactions arising from the context of execution, or, more specifically, network contention, can be done either by preventing these interactions from happening or by mitigating them. Still, the approaches discussed above require some knowledge about the application communication patterns (either compute or I/O communications). We review briefly related work in the prevention/mitigation of interactions before discussing monitoring techniques.

*Interactions Prevention.* Some steps have been taken towards integrating more knowledge about the communication patterns of applications into batch schedulers. For instance, Georgiou et al. studied the integration of TREEMATCH into SLURM [9]. Given the communication matrix of an application, the scheduler minimizes the load of the network links by smartly mapping the application's processes on the resources. This approach however does not consider the temporality of communications. Targeting the mesh/torus topologies, the works of Tuncer et al. [18] and Pascual et al. [15] are noteworthy. Another way to prevent interactions is to force the scheduler to use only certain allocation shapes with good properties: this strategy has been implemented in the Blue Waters scheduler [5]. The administrators of Blue Waters let the scheduler pick a shape among 460 precomputed cuboids. Yet, the works proposed above only target compute communications. HPC applications usually rely on highly tuned libraries such

as MPI-IO, parallel netCDF or HDF5 to perform their I/O. Tessier et al. propose to integrate topology awareness into these libraries [17]. They show that performing data aggregation while considering the topology allow to diminish the bandwidth required to perform I/O.

*Interactions Mitigation.* Given a set of applications, Gainaru et al. propose to schedule I/O flows of concurrent applications [7]. Their work aim at mitigating I/O congestion once applications have been allocated computation resources. To achieve such a goal, their algorithm relies on past I/O patterns of the applications to either maximize the global system utilization, or minimize the maximum slowdown induced by sharing bandwidth.

*Application/Platform Instrumentation.* A lot of effort have been put into developing tools to better understand the behavior of HPC applications. Characterizing I/O patterns is key as it allows the developers to identify performance bottlenecks, and allows the system administrator to better configure the platforms. A complementary path is to predict I/O performances during execution [4]. Such instrumentation efforts allow for a better use of the scarce communication resources. However, as they are application-centric, they fail to capture inter-application interactions. Monitoring of the platform is a way of getting insight on the inter-application interactions. We will not address this problem here.

## 4 Pinned I/O

In this section, we study the problem with respect to the *pinned* I/O model, according to which each job requests a specific I/O node. Such a model is representative of HPC platforms where the parallel file system is organized in stripes. For example, this is the case with the configuration of the Lustre file system in Blue Waters, where each I/O node is responsible for an address range (i.e., a stripe). Then, the jobs will request the I/O node corresponding to their data.

### 4.1 Complexity

We start by proving that the studied problem is  $\mathcal{NP}$ -complete even in the special case where all jobs require unit processing time to be executed, while the platform contains only three I/O nodes.

**Theorem 1.** *The problem of scheduling in the pinned model with respect to contiguity and locality constraints is strongly  $\mathcal{NP}$ -complete even in line topologies, with  $m^{I/O} = 3$  and  $p_j = 1$  for each job  $j \in \mathcal{J}$ .*

*Proof.* The problem clearly belongs to  $\mathcal{NP}$ . We give a reduction from a special case of the NUMERICAL 3-DIMENSIONAL MATCHING (N3DM) problem [8]. An instance of the classical N3DM problem consists of three disjoint sets  $W$ ,  $X$  and  $Y$ , each containing  $M$  positive integers, and a bound  $B \in \mathbb{Z}^+$ . The objective is to decide whether  $W \cup X \cup Y$  can be partitioned into  $M$  disjoint sets  $A_1, A_2, \dots, A_M$

such that each  $A_i$  contains exactly one element from each of  $W$ ,  $X$ , and  $Y$  and  $\sum_{a \in A_i} a = B$ , for  $1 \leq i \leq M$ .

Consider now SN3DM be the special case of N3DM in which all integers that belong to the set  $X$  are at least  $\frac{B}{2}$ . It is not hard to see that SN3DM is also strongly  $\mathcal{NP}$ -complete. Indeed, it suffices to transform an instance of N3DM to an instance of SN3DM by setting  $W' = W$ ,  $Y' = Y$ ,  $X' = \{x + B : \forall x \in X\}$  and  $B' = 2B$ . Then, any solution for N3DM corresponds to a solution for SN3DM, and vice versa.

We propose now a transformation from SN3DM to our problem as follows:

- $m^C = B$ ,  $m^{I/O} = 3$ ;
- the topology is a line starting with an I/O node, followed by  $\frac{B}{2}$  computing nodes, an I/O node,  $\frac{B}{2}$  computing nodes, and finishing with a third I/O node;
- for each  $a \in W \cup X \cup Y$ , we create a job  $j$  with  $q_j = a$ , and  $p_j = 1$ . All jobs derived from sets  $W$ ,  $X$ , and  $Y$  target the first, second, and third I/O node, respectively.

With respect to the ordering of the line topology, we refer to the computing nodes as  $1, 2, \dots, m^C$  and to the I/O nodes as  $1, 2, \dots, m^{I/O}$ .

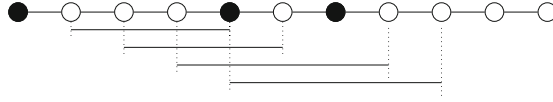
We will prove that a solution to SN3DM exists if and only if there is a schedule that satisfies all constraints and has a makespan at most  $M$ .

Assume that there is a solution for SN3DM. Then for each set  $A_i$ ,  $1 \leq i \leq M$ , we schedule the three jobs  $j_1 \in W$ ,  $j_2 \in X$  and  $j_3 \in Y$  corresponding to this set at time interval  $(i - 1, i]$ . Specifically,  $j_1$  will use the computing nodes  $1, \dots, q_{j_1}$ ,  $j_2$  the computing nodes  $q_{j_1} + 1, \dots, q_{j_1} + q_{j_2}$  and  $j_3$  the computing nodes  $q_{j_1} + q_{j_2} + 1, \dots, m^C$ . Note that each of these three jobs is adjacent to the targeted I/O node. Indeed, the  $j_1$  and  $j_3$  are adjacent to the leftmost and the rightmost I/O node, respectively, while  $j_2$  is always adjacent to the middle I/O node, since  $q_{j_2} \geq \frac{B}{2}$ . The makespan of the created schedule is equal to  $M$ .

Assume now that there exists a schedule of makespan at most  $M$ . As the total work is  $M \cdot B$ , no computing node is idle during the time interval  $(0, M]$ . Hence, the partition is directly derived by assigning jobs that start at time  $i - 1$  to  $A_i$ ,  $1 \leq i \leq M$ .  $\square$

## 4.2 Approximation Algorithm

In this section, we first propose a constant-factor approximation algorithm for line topologies and then we argue that it can be used even for ring topologies. The main idea of our algorithm is to first determine an allocation of each job to a specific set of computing nodes. We are interested in allocations that are simultaneously contiguous and local, while each job  $j$  requires a specific I/O node. As a consequence, there exist at most  $q_j + 1 = O(m^C)$  valid allocations for each job  $j$  (see Fig. 2). Given an allocation of all jobs to computing nodes, our problem coincides with the well-studied DYNAMIC STORAGE ALLOCATION (DSA) problem [10]. Then, we use a known approximation algorithm for the latter problem.



**Fig. 2.** Potential allocations for a job  $j$  requesting the middle I/O node with  $q_j = 3$ .

In order to decide the allocation of computing nodes we use an integer linear program. Let  $\mathcal{A}_j$  be the set of all potential allocations for each job  $j$ , where  $|\mathcal{A}_j| \leq q_j + 1$ . Each allocation  $a \in \mathcal{A}_j$  contains exactly  $q_j$  computing nodes as well as the required I/O node. Note that, an allocation may include more I/O nodes that will not be used during the execution of  $j$  neither by  $j$  nor by the other jobs due to the locality constraint. For example, in Fig. 2 the two rightmost allocations also cover the third I/O node in order to be able to include  $q_j = 3$  computing nodes. For each job  $j \in \mathcal{J}$  and allocation  $a \in \mathcal{A}_j$ , we introduce a binary indicator variable  $x_{j,a}$  which is equal to one if  $j$  is executed according to the allocation  $a$ , and zero otherwise. Moreover, for each node  $i \in \mathcal{V}$  we introduce a non-negative variable  $L_i$  which corresponds to the total load of jobs whose assigned allocation includes  $i$ . Finally, let  $\Lambda$  be the maximum load among all nodes. Then, we propose the following integer linear program which searches for the allocations that minimize the total load.

$$\text{minimize } \Lambda \tag{ILP}$$

$$\Lambda \geq L_i \quad \forall i \in \mathcal{V} \tag{1}$$

$$L_i \geq \sum_{j \in \mathcal{J}} \sum_{a \in \mathcal{A}_j} \sum_{i \in a} x_{j,a} p_j \quad \forall i \in \mathcal{V} \tag{2}$$

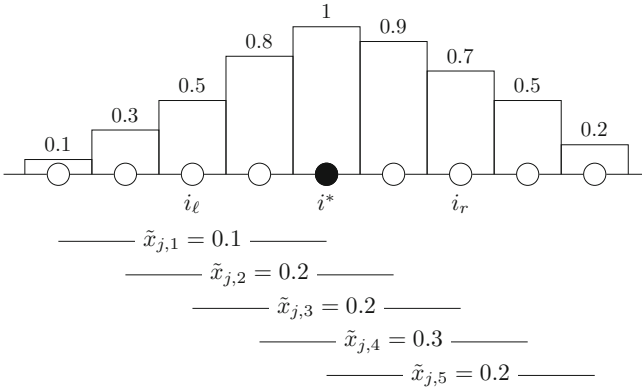
$$\sum_{a \in \mathcal{A}_j} x_{j,a} = 1 \quad \forall j \in \mathcal{J} \tag{3}$$

$$x_{j,a} \in \{0, 1\} \quad \forall j \in \mathcal{J}, a \in \mathcal{A}_j \tag{4}$$

Constraints (2) compute the total load for each node, while Constraints (3) ensure that each job is assigned an allocation. By relaxing the integrity Constraints (4), we can solve the corresponding linear program in polynomial time. Note that there are  $O(mn)$  variables and  $O(m + n)$  constraints. Moreover, an optimal solution to the above integer linear program is a lower bound to the makespan of an optimal solution for our problem, since it optimizes the maximum load without handling intersections of jobs in time, that is the scheduling phase.

Let  $\tilde{\Lambda}$ ,  $\tilde{L}_i$  and  $\tilde{x}_{j,a}$  denote the values of the variables in an optimal solution of the relaxed linear program. Then, the solution of this linear program is rounded to an integral feasible solution whose variables are denoted by  $\hat{\Lambda}$ ,  $\bar{L}_i$  and  $\bar{x}_{j,a}$ . Specifically, we round the indicator variables independently for each job  $j \in \mathcal{J}$

as follows: consider all possible allocations for the job  $j$  ordered with respect to the processors' ordering. The allocation chosen for  $j$  is the one with the smallest index  $k$  such that  $\sum_{a=1}^k \tilde{x}_{j,a} \geq \frac{1}{2}$ . Then, we set  $\bar{x}_{j,k} = 1$  and  $\bar{x}_{j,a} = 0$  for all  $a \neq k$ . Figure 3 gives an example of this rounding procedure.



**Fig. 3.** Rounding procedure for the variables that correspond to job  $j$ :  $\bar{x}_{j,3} = 1$  and  $\bar{x}_{j,1} = \bar{x}_{j,2} = \bar{x}_{j,4} = \bar{x}_{j,5} = 0$ .

The following lemma provides an upper bound to the integral solution  $\bar{A}$  obtained after the rounding procedure.

**Lemma 1.**  $\bar{A} \leq 2\tilde{A}$ .

*Proof.* Consider a job  $j$  and let  $k_j$  be the index of the allocation of  $j$  in the rounded solution, i.e.,  $\bar{x}_{j,k_j} = 1$ . Moreover, let  $\mathcal{V}(j)$  be the set of nodes (both computing and I/O) that are included in this allocation. We will first prove the following statement:

$$\sum_{a \in \mathcal{A}_j: i \in a} \tilde{x}_{j,a} \geq \frac{1}{2} \text{ for every } i \in k_j$$

For example, in Fig. 3 we have that  $k_j = 3$  and for each  $i \in \{3, \dots, 7\}$  the sum of the fractional variables that correspond to  $j$  and include  $i$  is at least 0.5. In order to prove the statement, let  $k_j = \{i_\ell, \dots, i_r\}$  be the set of nodes of the allocation  $k_j$  as these are ordered in the natural way. Recall that  $\mathcal{V}^{I/O}(j) \in k_j$  is the I/O node required by  $j$  and assume that  $\mathcal{V}^{I/O}(j)$  coincides with  $i^*$ , where  $i_\ell \leq i^* \leq i_r$ . By the definition of  $k_j$ , the statement is true for  $i = i_\ell$ . Moreover, the statement holds for each node  $i \in \{i_\ell, \dots, i^*\}$  since

$$\sum_{a \in \mathcal{A}_j: i \in a} \tilde{x}_{j,a} \geq \sum_{a \in \mathcal{A}_j: i_\ell \in a} \tilde{x}_{j,a} \geq \frac{1}{2}$$



It remains to prove it for  $i \in \{i^* + 1, \dots, i_r\}$ . We focus first on  $i_r$ . Observe that by the definition of  $k_j$  it holds that  $\sum_{a=1}^{k_j-1} \tilde{x}_{j,a} < \frac{1}{2}$ . Then, we have that

$$\sum_{a \in \mathcal{A}_j: i_r \in a} \tilde{x}_{j,a} = \sum_{a \in \mathcal{A}_j} \tilde{x}_{j,a} - \sum_{a=1}^{k_j-1} \tilde{x}_{j,a} > 1 - \frac{1}{2} = \frac{1}{2}$$

Finally, the statement holds for each node  $i \in \{i^* + 1, \dots, i_r\}$  since

$$\sum_{a \in \mathcal{A}_j: i \in a} \tilde{x}_{j,a} \geq \sum_{a \in \mathcal{A}_j: i_r \in a} \tilde{x}_{j,a} \geq \frac{1}{2}$$

In order to finalize the proof of the lemma, consider the load  $\bar{L}_i$  of a node  $i$  in the rounded solution. We have that

$$\bar{L}_i = \sum_{j \in \mathcal{J}} p_j \cdot \mathbf{1}_{\{\text{if } i \in k_j\}} = \sum_{j \in \mathcal{J}} p_j \sum_{a \in \mathcal{A}_j: i \in a} \tilde{x}_{j,a} \leq \sum_{j \in \mathcal{J}} p_j 2 \sum_{a \in \mathcal{A}_j: i \in a} \tilde{x}_{j,a}$$

where the last inequality holds by the proven statement and since by Constraint (3) we have that  $\sum_{a \in \mathcal{A}_j: i \in a} \tilde{x}_{j,a} \leq 1$ . Hence,

$$\bar{L}_i \leq 2 \sum_{j \in \mathcal{J}} p_j \sum_{a \in \mathcal{A}_j: i \in a} \tilde{x}_{j,a} = 2 \sum_{j \in \mathcal{J}} p_j \sum_{a \in \mathcal{A}_j} \sum_{i \in a} \tilde{x}_{j,a} = 2 \sum_{j \in \mathcal{J}} \sum_{a \in \mathcal{A}_j} \sum_{i \in a} \tilde{x}_{j,a} p_j = 2\tilde{L}_i$$

The lemma follows by considering the node of maximum load in the rounded solution, i.e.,  $\bar{L} = \max_i \{\bar{L}_i\} \leq 2 \max_i \{\tilde{L}_i\} = 2\tilde{L}$ . □

As mentioned before, given the allocations of all jobs, our problem coincides with the DSA problem [10]. An instance of the DSA problem consists of a set of  $n$  triples. Each triple  $(\ell_j, r_j, s_j)$  corresponds to a rectangle parallel to  $x$ -axis of size  $(r_j - \ell_j) \times s_j$ . Specifically,  $\ell_j$  and  $r_j$  are the projections of its leftmost and rightmost points, respectively, in the  $x$ -axis while  $s_j$  is its size projected in the  $y$ -axis. In other words, the position of the rectangle is fixed with respect to  $x$ -axis, but it can be shifted in any position in  $y$ -axis. The objective is to pack all rectangles without intersections in a strip of minimum height.

In our scheduling context, each job corresponds to a rectangle whose  $\ell_j$  and  $r_j$  values are defined by a given allocation as the leftmost and the rightmost computing nodes respectively, while  $p_j = s_j$ . Moreover, the makespan coincides with the height of the strip.

Gergov [10] presented a greedy 3-approximation algorithm for the DSA problem. The important property of this algorithm is that it uses as lower bound the maximum load over all  $x$ -coordinates, which allows as to use it in our analysis. The following theorem describes this property in scheduling terms.

**Theorem 2** [10]. *There is an algorithm which computes a feasible schedule whose makespan is at most three times the maximum load of every node.*

---

**Algorithm 1.**

---

- 1 Solve the relaxed version of (ILP)
  - 2 **for** each job  $j \in \mathcal{J}$  **do**
  - 3     Find the smallest index  $k$  such that  $\sum_{a=1}^k \tilde{x}_{j,a} \geq \frac{1}{2}$
  - 4     Set  $\tilde{x}_{j,k} = 1$  and  $\tilde{x}_{j,a} = 0$  for all  $a \neq k$
  - 5 Create a feasible schedule by applying the algorithm proposed in Theorem 2 using the allocations determined by the  $\tilde{x}_{j,a}$  variables
- 

Due to the equivalence of our problem with DSA, we can apply the algorithm mentioned in Theorem 2 and get a final solution to our problem. A high-level description of the above described procedure is given in Algorithm 1.

**Theorem 3.** *Algorithm 1 achieves an approximation ratio of 6 for the line topology in the pinned I/O model.*

*Proof.* Consider a schedule created by Algorithm 1 and let  $C_{\max}$  be the makespan of this schedule. Due to the allocation phase, we know that the maximum load over all nodes is equal to  $\bar{A}$ . Then, by Theorem 2 and Lemma 1, we have that  $C_{\max} \leq 3\bar{A} \leq 6\bar{A}$ . Hence, the theorem follows by the fact that the optimal solution to (ILP) is a lower bound to the optimal solution for our problem.  $\square$

We observe that Gergov's algorithm remains a 3-approximation even in the case of rings. Moreover, the allocation procedure based on the rounding of (ILP) can be also applied for rings; we just need to define an ordering of the possible allocations of each job. Thus, by considering an clockwise ordering, we can apply Algorithm 1 and get the following theorem.

**Theorem 4.** *Algorithm 1 achieves an approximation ratio of 6 for the ring topology in the pinned I/O model.*

## 5 Unpinned I/O

In this section, we study the *unpinned* I/O model according to which each job requires any arbitrary I/O node.

### 5.1 Complexity

We start by proving that the studied problem is NP-complete even in the special case where all jobs require unit processing time to be executed, while the platform contains only three I/O nodes. The proof is similar with the proof of Theorem 1 with the difference that the reduction is done by the classical 3-PARTITION problem [8]. For this reason, it is omitted.

**Theorem 5.** *The problem of scheduling in the unpinned model with respect to contiguity and locality constraints is strongly NP-complete even in line topologies, with  $m^{I/O} = 3$  and  $p_j = 1$  for each job  $j \in \mathcal{J}$ .*

## 5.2 An Approximation Algorithm for Equidistant I/O Nodes

In this section, we consider both line and ring topologies and we propose an approximation algorithm in the case where the I/O nodes are uniformly distributed. In other words, the I/O nodes are equidistant from each other. We denote by  $\delta$  the distance separating two consecutive I/O nodes. Note that, given any instance, in line topologies  $\delta$  can be either  $\lfloor \frac{m^C}{m^{I/O}} \rfloor$  or  $\lceil \frac{m^C}{m^{I/O}} \rceil$  while the first value is always the case in ring topologies.

We need some additional notation. We call a job *small* if it requires fewer computing nodes than the distance between two consecutive I/O nodes, i.e.,  $q_j^C < \delta$ . In a similar way, we call a job *big* if  $q_j^C \geq \delta$ . Let  $\mathcal{J}_{\leq \delta}$  and  $\mathcal{J}_{\geq \delta}$  be the sets of small and big jobs, respectively. Our algorithm handles these sets separately.

A small job cannot be adjacent to more than one I/O nodes in any feasible schedule. Moreover, an I/O node along with  $\delta$  consecutive computing nodes adjacent to it can be considered as a processing unit that can execute a small job. Based on this, we partition the set  $\mathcal{V}^C$  into  $\lfloor \frac{m^C}{\delta} \rfloor$  groups of consecutive computing units, each one of size at least  $\delta$ . Assume that these groups as well as the I/O nodes are numbered from left to right and we consider the  $i$ -th such group and the  $i$ -th I/O node to compose a processing unit. Note that, by the definition of  $\delta$ ,  $m^{I/O}$  can be either  $\lfloor \frac{m^C}{\delta} \rfloor$  or  $\lfloor \frac{m^C}{\delta} \rfloor + 1$ . In the second case, which can happen only in line topologies, the last I/O node is not used. Then, we can transform our problem for small jobs to an instance of the classical  $P \parallel C_{\max}$  problem with  $\lfloor \frac{m^C}{\delta} \rfloor$  machines [11]. Specifically, each machine corresponds to one processing unit, while each small job has a processing time as in the initial instance and requires only one processing unit. Then, we solve the created instance of  $P \parallel C_{\max}$  by using any known approximation algorithm for it. The following lemma, whose proof is omitted, summarizes the above procedure. The additional 2-factor in the line case is due to parity issues.

**Lemma 2.** *Any  $\rho_1$ -approximation algorithm for the  $P \parallel C_{\max}$  scheduling problem, can be used to obtain a  $2\rho_1$ -approximation algorithm to schedule small jobs in a line and a  $\rho_1$ -approximation algorithm to schedule small jobs in a ring.*

Due to the contiguity constraint, the big jobs are structurally guaranteed to be adjacent to at least one I/O node, i.e., we can then ignore the existence of I/O nodes when scheduling big jobs. Hence, the objective is to pack the big jobs and our problem reduces to the strip-packing problem [13]. The following lemma, whose proof is omitted, summarizes the above reduction. The additional 2-factor in the ring case is due to the degeneration of the ring to a line.

**Lemma 3.** *Any  $\rho_2$ -approximation algorithm for the strip-packing problem, can be used to obtain a  $\rho_2$ -approximation algorithm to schedule big jobs in a line and a  $2\rho_2$ -approximation algorithm to schedule big jobs in a ring.*

By combining Lemmas 2 and 3 the following theorem follows.

**Theorem 6.** *For the unpinned model, there is a  $(2\rho_1 + \rho_2)$ -approximation algorithm for line topologies and a  $(\rho_1 + 2\rho_2)$ -approximation algorithm for ring topologies, where  $\rho_1$  and  $\rho_2$  are the approximation ratios for the  $P \parallel C_{\max}$  and the strip-packing problems, respectively.*

Note that a PTAS exists for both  $P \parallel C_{\max}$  and strip-packing problems [11, 13], leading for  $(3 + \epsilon)$ -approximation algorithms for line and ring topologies.

## 6 Conclusions

We studied the makespan minimization problem on line and ring topologies, when the allocations are constrained to be both contiguous and local. We proved that both the pinned and unpinned models are  $\mathcal{NP}$ -complete and we presented constant-factor approximation algorithms for them. The proposed algorithms can be also applied in different settings (the proofs will be developed in an extended version of this work). For example, in the case where the I/O nodes can be shared by more than one jobs, then the 6-approximation algorithm of Sect. 4.2 can be simply adapted by excluding the requested I/O node from the allocation of the job in the definition of the indicator variables of (ILP). Note that due to the locality constraint an I/O node cannot be shared by more than two jobs. Another example is the case where each job requires more than one I/O nodes. However, this assumption in conjunction with the locality constraint could lead to several unused nodes, limiting its interest.

As future steps, one could implement the proposed algorithms, and study their performances through simulation. From a theoretical point of view, the tightness results show the limits of the two-phase approach in Sect. 4.2. The approximation ratios might be improved by scheduling the problem in a single phase. Finally, the study of more enhanced topologies, like two-dimensional ones, is a very interesting direction. In this case, contiguity could be replaced by more general constraints implying the convexity of the shape of the allocations.

## References

1. Bhatele, A., Mohror, K., Langer, S.H., Isaacs, K.E.: There goes the neighborhood: performance degradation due to nearby jobs. In: SC, pp. 41:1–41:12. ACM, November 2013
2. Błażdek, I., Drozdowski, M., Guinand, F., Schepler, X.: On contiguous and non-contiguous parallel task scheduling. *J. Sched.* **18**(5), 487–495 (2015)
3. Chen, N.-C., Poon, S.S., Ramakrishnan, L., Aragon, C.R.: Considering time in designing large-scale systems for scientific computing. In: CSCW, pp. 1533–1545. ACM, February 2016
4. Dorier, M., Ibrahim, S., Antoniu, G., Ross, R.B.: Using formal grammars to predict I/O behaviors in HPC: the Omnisc’IO approach. *IEEE Trans. Parallel Distrib. Syst.* **27**(8), 2435–2449 (2016)
5. Enos, J., et al.: Topology-aware job scheduling strategies for torus networks. In: Cray User Group, May 2014

6. Feitelson, D.G., Rudolph, L., Schwiegelshohn, U., Sevcik, K.C., Wong, P.: Theory and practice in parallel job scheduling. In: Feitelson, D.G., Rudolph, L. (eds.) JSSPP 1997. LNCS, vol. 1291, pp. 1–34. Springer, Heidelberg (1997). [https://doi.org/10.1007/3-540-63574-2\\_14](https://doi.org/10.1007/3-540-63574-2_14)
7. Gainaru, A., Aupy, G., Benoit, A., Cappello, F., Robert, Y., Snir, M.: Scheduling the I/O of HPC applications under congestion. In: IPDPS, pp. 1013–1022. IEEE, May 2015
8. Garey, M.R., Johnson, D.S.: Computers and Intractability: A Guide to the Theory of NP-Completeness. W. H. Freeman, New York (1979)
9. Georgiou, Y., Jeannot, E., Mercier, G., Villiermet, A.: Topology-aware resource management for HPC applications. In: ICDCN, pp. 17:1–17:10. ACM (2017)
10. Gergov, J.: Algorithms for compile-time memory optimization. In: SODA, pp. 907–908. ACM/SIAM, January 1999
11. Hochbaum, D.S., Shmoys, D.B.: Using dual approximation algorithms for scheduling problems: theoretical and practical results. *J. ACM* **34**(1), 144–162 (1987)
12. Kathareios, G., Minkenberg, C., Prisacari, B., Rodríguez, G., Hoefler, T.: Cost-effective diameter-two topologies: analysis and evaluation. In: SC, pp. 36:1–36:11. ACM, November 2015
13. Kenyon, C., Rémila, E.: Approximate strip packing. In: FOCS, pp. 31–36 (1996)
14. Lucarelli, G., Mendonça, F.M., Trystram, D., Wagner, F.: Contiguity and locality in backfilling scheduling. In: CCGRID, pp. 586–595. IEEE Computer Society, May 2015
15. Pascual, J.A., Miguel-Alonso, J., Antonio, L.J.: Application-aware metrics for partition selection in cube-shaped topologies. *Parallel Comput.* **40**(5), 129–139 (2014)
16. Strohmaier, E., Dongarra, J., Simon, H., Meuer, M.: TOP500 list, June 2018
17. Tessier, F., Malakar, P., Vishwanath, V., Jeannot, E., Isaila, F.: Topology-aware data aggregation for intensive I/O on large-scale supercomputers. In: COMHPC@SC, pp. 73–81. IEEE, November 2016
18. Tuncer, O., Leung, V.J., Coskun, A.K.: PaCMap: topology mapping of unstructured communication patterns onto non-contiguous allocations. In: ICS, pp. 37–46. ACM, June 2015