A Methodology for Handling Data Movements by Anticipation: Position Paper

Raphaël Bleuse[1,2(✉)], Giorgio Lucarelli[1(✉)], and Denis Trystram[1]

[1] Univ. Grenoble Alpes, CNRS, Inria, Grenoble INP, LIG, 38000 Grenoble, France
{giorgio.lucarelli,denis.trystram}@imag.fr
[2] FSTC/CSC, University of Luxembourg, Luxembourg City, Luxembourg
raphael.bleuse@uni.lu

Abstract. The enhanced capabilities of large scale parallel and distributed platforms produce a continuously increasing amount of data which have to be stored, exchanged and used by various tasks allocated on different nodes of the system. The management of such a huge communication demand is crucial for reaching the best possible performance of the system. Meanwhile, we have to deal with more interferences as the trend is to use a single all-purpose interconnection network whatever the interconnect (tree-based hierarchies or topology-based heterarchies). There are two different types of communications, namely, the flows induced by data exchanges during the computations, and the flows related to Input/Output operations. We propose in this paper a general model for interference-aware scheduling, where explicit communications are replaced by external topological constraints. Specifically, the interferences of both communication types are reduced by adding geometric constraints on the allocation of tasks into machines. The proposed constraints reduce implicitly the data movements by restricting the set of possible allocations for each task. This methodology has been proved to be efficient in a recent study for a restricted interconnection network (a line/ring of processors which is an intermediate between a tree and higher dimensions grids/torus). The obtained results illustrated well the difficulty of the problem even on simple topologies, but also provided a pragmatic greedy solution, which was assessed to be efficient by simulations. We are currently extending this solution for more complex topologies. This work is a position paper which describes the methodology, it does not focus on the solving part.

Keywords: Scheduling · Affinity · Data movements · Heterogeneity
Topology · HPC

This work has been partially supported by a DGA-MRIS scholarship, and is partially funded by the joint research programme UL/SnT-ILNAS on Digital Trust for Smart ICT.

G. Mencagli et al. (Eds.): Euro-Par 2018 Workshops, LNCS 11339, pp. 134–145, 2019.
https://doi.org/10.1007/978-3-030-10549-5_11

1 Introduction

In High Performance Computing (HPC), the demand for computation power is steadily increasing [27]. To meet up the challenge of always more performances, while being constrained by ever growing energy costs, the architecture of supercomputers also grows in complexity at the whole machine scale. This complexity arises from various factors: firstly, the size of the machines (supercomputers now integrates millions of cores); secondly, the heterogeneity of the resources (various architectures of computing nodes, mixed workloads of computing and analytics, nodes dedicated to I/O, etc.); and lastly, the interconnection topology. The architectural evolutions of the interconnection networks at the whole machine scale pose two main challenges that are described as follows. First, the community proposed several types of topologies including hierarchies and heterarchies (which are based on structural well-suited topologies), the trend today is to create mixed solutions of tree-like machines with local structured toplogies [22]; and second, the interconnection network is usually unique within the machine (which means that the network is shared for various mixed data flows). Sharing such a single multi-purpose interconnection network begets complex interactions (e.g., network contention) between running applications. These interactions have a strong impact on the performances of the applications [4,15], and hamper the understanding of the system by the users [11]. As the volume of processed data increases, so does the impact of the network.

We propose in this work 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 or inefficient interactions by constraining the shape of the allocations. Such an approach aims at taking into account the complexity of the new HPC 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, maximize the throughput or optimize any other relevant objective while enforcing constraints on the allocations.

The purpose of this paper is to describe the methodology for interference-aware scheduling. The design of an algorithm and the corresponding simulations/experiments are another side of this subject. We are currently studying efficient solutions for assessing this methodology, but this paper does not focus on this point.

2 General Problem Setting

Modelization. A platform is of a set $\mathcal{V}$ of m nodes divided in two sets: m^{C} nodes dedicated to computations $\mathcal{V}^{\mathrm{C}}$, and $m^{I/O}$ nodes that are entry points to a high performance file system $\mathcal{V}^{I/O}$. The nodes are indexed by $i \in 0, \ldots, m-1$.

This numbering provides an *arbitrary ordering* of the nodes. We distinguish two interesting distributions of the nodes:

1. *coupled I/O*, where some compute nodes are also entry points for the I/O operations (i.e., $\mathcal{V}^{I/O} \subseteq \mathcal{V}^{\mathrm{C}} = \mathcal{V}$);
2. *separate I/O*, when there is no overlap between compute and I/O nodes (i.e., $\mathcal{V}^{I/O} \cap \mathcal{V}^{\mathrm{C}} = \emptyset$).

We also distinguish two ways of interacting with the I/O nodes, namely, *shared I/O* when any number of jobs can access an I/O node at any time, and *exclusive I/O* when an I/O node is exclusively allocated to a job for the job's lifespan. We further annotate node symbols with $\star^{I/O}$ ($\star^{\mathrm{C}}$, resp.) if there is a need to distinguish I/O nodes (compute nodes, resp.).

The nodes communicate thanks to an interconnection network with a given *topology* (i.e., the connected graph of the interconnection) or by a hierarchical topology (tree-like interconnection). The localization of every node within the topology is known. We define the distance that intrinsically derives from a topology as follows:

Definition 1 (Distance). *The distance* $\mathrm{dist}(i, i')$ *between two nodes* i *and* i' *(either compute or I/O) is defined as the minimum number of hops to go from* i *to* i'*. For hierarchical topologies, the distance is defined as the number of traversed levels (switches) to go from* i *to* i'*.*

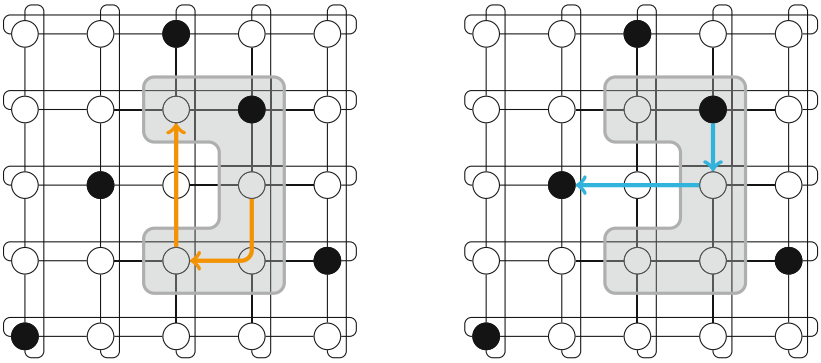
Batch schedulers are a critical part of the software stack managing supercomputers: their goal is to efficiently allocate resources (nodes from $\mathcal{V}$ in our case) to the jobs submitted by the users of the platform. The jobs are queued in a set $\mathcal{J}$ of n jobs. Each job j requires a number of compute nodes q_j^{C} and some I/O nodes $q_j^{I/O}$. The I/O nodes requirements can either be a number of nodes (*unpinned I/O*), or a dedicated subset of $\mathcal{V}^{I/O}$ (*pinned I/O*). The number of allocated nodes is fixed (i.e., the job is *rigid* [17]). We denote by $\mathcal{V}(j)$ the nodes allocated to the job j. Each job j requires a certain time p_j to be processed, and it is *independent* of every other jobs. Once a job starts executing, it runs until completion (i.e., it *cannot be preempted*). Finally, any compute node is able to process at most one job at any time.

Before presenting the constraints we consider in this work, 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.

Definition 2 (Communication types). *We distinguish two types of communications (see Fig. 1):*

***compute* communications** *are the communications induced by data exchanges for computations. Such communications occur between two compute nodes allocated to the same application.*

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



(a) *compute* communications (b) *I/O* communications

Fig. 1. Figuration of the two distinguished types of communications. Note that some communications stay within the allocation, while others do not. White nodes represent compute nodes, and black nodes represent I/O nodes.

As stated in the introduction, we do not aim at finely modeling the context of execution. We propose here to model the platform in such a way that network interactions are *implicitly* taken into account. We enrich the scheduling problem with alien geometric constraints on the allocations deriving from the platform topology or the application structure.

Most scheduler implementations are naive, in the sense that they allocate resources greedily. This is known to impact performances [15], and is the core difference between parallel machine scheduling and packing problems. Constraining the allocations to enhance performance is however no new idea. For example, Lucarelli et al. studied the impact of enforcing contiguity or locality constraints in backfilling scheduling [23]. They showed that enforcing these constraints can be done at a small computational cost, and has minimum negative impact on usual metrics such as makespan (i.e., maximum completion time), flow-time (i.e., absolute time spent in the system), or stretch (i.e., time spent in the system relative to each job size). One may refer to [9,14] for a detailed definition of classic optimization objectives in scheduling.

We go further with this model as we target heterogeneous machines, and distinguish network flows. We seek the following properties for the constraints:

- It *captures part of the execution context*: enforcing the constraint should help minimize nocuous effects arising from the execution context.
- It *derives from minimal reliable data*: constraints on the allocations are enforced ahead of the scheduling decisions. As a result, the proposed constraints only use the topology of the interconnection network and the size of the allocation as input data.
- It is *cheap to compute*: enumerating the list of allocations respecting some constraints cannot be a performance bottleneck for the scheduler.

We study in more detail in the two following sections how to consider these constraints for structured topologies and for hierarchical topologies.

3 Intrinsic Constraints for Structured Topologies

For the sake of clarity, we consider here a 2D-torus (the same constraints also hold for other regular topologies like higher-dimensional torus or hypercubes).

Avoiding Compute-Communication Interactions. Considering this classification of network flows, we first expose three constraints targeting compute communications.

Definition 3 (Connectivity). *An allocation π is said to be* connected *iff there exists a subset $\mathcal{V}_\pi$ of $\mathcal{V}^{I/O}$ such that $\left(\pi \cap \mathcal{V}^C\right) \cup \mathcal{V}_\pi$ is connected in the graph-theory sense. $\mathcal{V}_\pi$ may be empty.*

The *connectivity* constraint ensures, for a given allocation, that there exists a path without interference between any pair of compute nodes of the allocation. This however, with regard to the interconnection topology, can either require support for dynamic routing or demand to the application to implement its own routing policy. Moreover, it may lead to islets of isolated compute nodes. Hence, although satisfactory from the graph theoretical point of view, the connectivity constraint is not sufficient to ensure that compute communication do not interfere. We propose the *convexity* constraint with the goal of overcoming these limits.

Definition 4 (Convexity). *An allocation is said to be* convex *iff it is impossible for compute communications from any other potential allocation to share an interconnect link with respect to the underlying routing algorithm.*

By taking into account the effective routing policy, and by forbidding any potential sharing, the *convexity* constraint does forbid interactions.

Note that the convexity constraint dominates the connectivity constraint, as stated in the following Proposition.

Proposition 1. *Given any topology, any convex allocation is connected (Fig. 2).*

Definition 5 (Contiguity [6,23]). *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.*

One has to note that the contiguity constraint is intrinsically unidimensional as it relies on the nodes' ordering. For topologies such as trees, lines or rings the ordering is natural. On higher dimension topologies, no natural ordering exists, and an arbitrary mapping is needed. An usual strategy to order nodes is to use space-filling curves (e.g., Z-order curve [24], Hilbert curve [20], etc.) as they enforce a strong spatial locality. Albing proposes various orderings that may be more suited for HPC use cases, and a method to evaluate them [3]. Contiguity is an interesting relaxation of convexity as it offers good spatial locality properties for a reasonable computing cost. It is however unable to ensure that no jobs could interact.

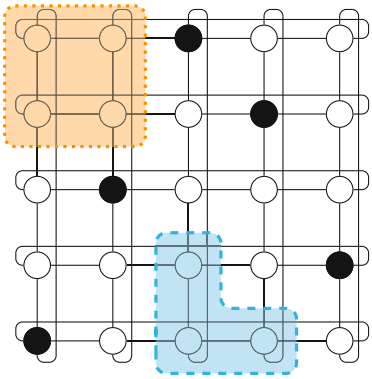


Fig. 2. Example of a convex allocation (dotted orange contour), and a non-convex, but connected allocation (dashed blue contour). The underlying topology is a 2D-torus, with dimension-order routing. White nodes represent compute nodes, and black nodes represent I/O nodes. (Color figure online)

Avoiding I/O-Communication Interactions. The constraints exposed so far are well suited to take into account the compute communications, but not the I/O communications. Indeed, the compute communications may occur between any pair of compute nodes within an allocation: we usually describe this pattern as all-to-all communications. I/O communications, on the other hand, 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 (see Fig. 3). We must emphasize that the locality constraint proposed here is not related to the locality constraint previously described by Lucarelli et al. [23].

Definition 6 (Locality). *A given allocation for a job j is said to be* local *iff it is connected, and every I/O nodes from $\mathcal{V}^{I/O}(j)$ are adjacent to compute nodes from $\mathcal{V}^{C}(j)$, with respect to the underlying topology. In other words, $\mathcal{V}^{I/O}(j)$ is a subset of the closed neighborhood of $\mathcal{V}^{C}(j)$.*

Interestingly, the locality constraint enforces a bound on the number of concurrent jobs that can target a given I/O node.

Proposition 2. *Given any topology, any I/O node i, at any time, the number of local jobs targeting i cannot exceed the number of adjacent compute nodes of i.*

As a consequence, if the I/O nodes can be shared, the number of concurrent jobs targeting a given I/O node is bounded by the degree of this I/O node. This identity obviously also holds for exclusive I/O, but has limited interest in this case.

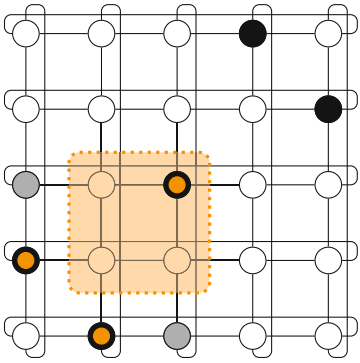


Fig. 3. Given an allocation (dotted orange contour) for a job j, the allocation is local iff j uses a subset of the I/O nodes marked with the orange dot. Foreign compute nodes potentially impacted by I/O communications of j are depicted in gray: these nodes can only be in the neighborhood of the allocation thanks to the locality constraint. The underlying topology is a 2D-torus, with dimension-order routing. White nodes represent compute nodes, and black nodes represent I/O nodes.

4 Intrinsic Constraints for Hierarchical Topologies

Hierarchical platforms are composed of computing nodes and communication switches. The interconnect is a tree where the leaves are the computing nodes, and the internal nodes correspond to the switches. A group of leaves connected by the same switch is a *cluster*. The communications inside a cluster are negligible while external communications require to cross all the switches along the unique path from a node to another. Figure 4 depicts a model of hierarchical platform.

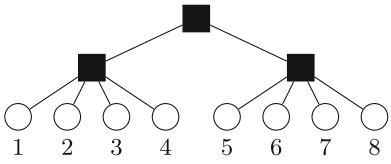


Fig. 4. Example of a hierarchical topology. White nodes represent compute nodes, and black nodes represent internal nodes (switches).

Avoiding Compute-Communication Interactions. In tree-like topologies the three constraints introduced for torus topologies should be revisited. Specifically, the convexity constraint is not relevant for hierarchical topologies since it implies that the internal nodes (switches) should be exclusively used by a single application, which significantly affects the platform utilization. On the other hand, the contiguity constraint can be naturally applied, by considering an arbitrary order of the children of any internal node and then numbering the leaves from

left to right. Finally, the definition of connectivity constraint does not directly apply to hierarchical topologies.

The main characteristic of the hierarchical topologies is that there is no reason to distinguish among nodes that are connected under a common switch. However, the distance among two nodes of the same allocation is very important. In what follows, we define two new constraints that are better suited to tree-like topologies.

Definition 7 (Proximity). *A given allocation π for a job j satisfies the* proximity *constraint iff the quantity* $\max_{i,i' \in \pi} dist(i, i')$ *is minimized.*

In other words, the maximum distance among any two computing nodes assigned to the job j should be minimum. Hence, the allocation affects the minimum number of levels of the tree (see Figs. 5b and c).

Definition 8 (Compacity). *A given allocation π for a job j is called* compact *iff the quantity* $\sum_{i,i' \in \pi} dist(i, i')$ *is minimized.*

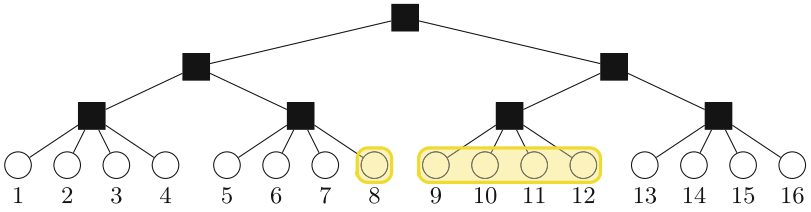
Intuitively, the compacity constraint intents not only to use the minimum number of levels in the tree, but also to consider two qualitative properties of the allocation (see Fig. 5c). First, compacity implies that an allocation spans as few clusters as possible. Second, if a cluster is used, the compacity constraint aims at maximizing the number of nodes allocated within this cluster.

Avoiding I/O-Communication Interactions. In the previous presentation of hierarchical topologies, the I/O nodes have been implicitly placed at the switch levels, as it is common in many existing architectures [1]. Let notice that our analysis also holds where the I/O nodes are located at the leaves level as it is the case in some architectures like in the interconnect of the private cloud [25].

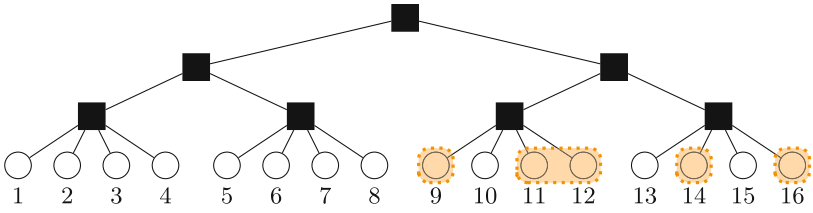
5 Related Work

Tackling the nocuous interactions arising from the context of execution—or, more specifically, network contention—can be seen as a scheduling problem with uncertainties. Within this framework, there exist two main approaches to abate the uncertainty: by either preventing some uncertainties from happening (proactive approach), or by mitigating the uncertainties impact (reactive approach) [5]. We start reviewing some related works in the prevention/mitigation of interactions before discussing monitoring techniques.

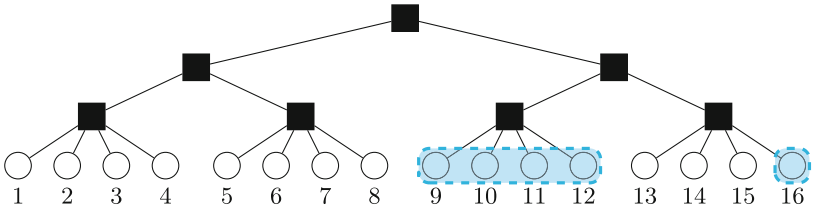
Interactions Prevention. Some steps have been taken towards integrating more knowledge about the communication patterns of applications into the batch scheduler. For example, Georgiou et al. studied the integration of TreeMatch into SLURM [19]. Given the communication matrix of an application, the scheduler minimizes the load of the network links by smartly mapping the application's processes on the resources. This approach however is limited to tree-like topologies, and does not consider the temporality of communications. Targeting the



(a) contiguous, non-proximate, non-compact allocation



(b) non-contiguous, proximate, non-compact allocation



(c) non-contiguous, proximate, compact allocation

Fig. 5. Figuration of the constraints on a hierarchical topology. The depicted allocations contain five compute nodes (i.e., $q_j = 5$). White nodes represent compute nodes, and black nodes represent internal nodes (switches).

mesh/torus topologies, the works of Tuncer et al. [29] and Pascual et al. [26] 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 [15]. 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 [28]. They show that performing data aggregation while considering the topology allow to diminish the bandwidth required to perform I/O. The CLARISSE approach proposes to coordinate the data staging steps while considering the full I/O stack [21].

Interactions Mitigation. Given a set of applications, Gainaru et al. propose to schedule I/O flows of concurrent applications [18]. Their work aim at mitigating I/O congestion within the interconnection 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. Deeper in the I/O stack, at the I/O node level, the I/O flows can be reorganized to better match the characteristics of the storage devices [8].

Application/Platform Instrumentation. The approaches discussed above require the knowledge of the application communication patterns (either compute or I/O communications). A lot of effort has 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. Some tools, such as Darshan [10], instrument the most used I/O libraries, and record every I/O-related function call. The gathered logs provide valuable data for postmortem analysis. Taking a complementary path, Omnisc'IO aims at predicting I/O performances during execution [13]. The predictions rely on a formal grammar to model the I/O behavior of the instrumented application.

These 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 [2,16]. For example, the OVIS/LDMS system deployed on Blue Waters collect 194 metrics on every 27648 nodes every minute [2]. Among the metrics of interest are the network counters: the number of stalls is a good indicator of congestion [12].

6 Conclusion and Future Work

The goal of this paper was to propose a methodology for handling data communications in modern parallel platforms for both structured topology and hierarchical interconnects. Our proposal was to identify relevant constraints that can easily be integrated into an optimization problem. We have successfully applied this methodology for a specific topology (line/ring of processors) [7].

Defining constraints that work well for any kind of topologies has been troublesome. This raises the question to know if a topology-agnostic heuristic can be designed at a reasonable cost with decent performances. If not, it would be interesting to classify the topologies, and propose class-specific constraints and heuristics. We are currently working on the design of a generic heuristic that can address several topologies.

The proposed constraints are strongly expected to have a positive impact on the performances as they implicitly emphasize data locality. We however did not verify through experiments that these constraints indeed have a positive impact on the network usage. The benefits from these experiments will be twofold: first, validate the proposed constraints; second, provide a feedback to design better suited constraints.

References

1. TGCC Curie Supercomputer. http://www-hpc.cea.fr/en/complexe/tgcc-curie.htm
2. Agelastos, A., et al.: The lightweight distributed metric service: a scalable infrastructure for continuous monitoring of large scale computing systems and applications. In: SC, pp. 154–165. IEEE, November 2014
3. Albing, C.: Characterizing node orderings for improved performance. In: PMBS@SC, pp. 6:1–6:11. ACM (2015)
4. 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
5. Billaut, J.C., Moukrim, A., Sanlaville, É.: Flexibility and Robustness in Scheduling. Control Systems, Robotics and Manufacturing, Wiley (2008)
6. Błądek, I., Drozdowski, M., Guinand, F., Schepler, X.: On contiguous and non-contiguous parallel task scheduling. J. Sched. **18**(5), 487–495 (2015)
7. Bleuse, R., Dogeas, K., Lucarelli, G., Mounié, G., Trystram, D.: Interference-aware scheduling using geometric constraints. In: Aldinucci, M., Padovani, L., Torquati, M. (eds.) Euro-Par 2018. LNCS, vol. 11014, pp. 205–217. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-96983-1_15
8. Boito, F.Z., Kassick, R.V., Navaux, P.O.A., Denneulin, Y.: Automatic I/O scheduling algorithm selection for parallel file systems. Concurr. Comput. Pract. Exp. **28**(8), 2457–2472 (2016)
9. Brucker, P.: Scheduling Algorithms, 5th edn. Springer, New York (2007)
10. Carns, P.H., Harms, K., Allcock, W.E., Bacon, C., Lang, S., Latham, R., Ross, R.B.: Understanding and improving computational science storage access through continuous characterization. ACM Trans. Storage **7**(3), 8:1–8:26 (2011)
11. Chen, N., 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
12. Deveci, M., et al.: Exploiting geometric partitioning in task mapping for parallel computers. In: IPDPS, pp. 27–36. IEEE, May 2014
13. 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)
14. Drozdowski, M.: Scheduling for Parallel Processing. Computer Communications and Networks. Springer, London (2009). https://doi.org/10.1007/978-1-84882-310-5
15. Enos, J., et al.: Topology-aware job scheduling strategies for torus networks. In: Cray User Group, May 2014. https://cug.org/proceedings/cug2014_proceedings/includes/files/pap182.pdf
16. Evans, R.T., Browne, J.C., Barth, W.L.: Understanding application and system performance through system-wide monitoring. In: IPDPS Workshops, pp. 1702–1710. IEEE, May 2016
17. 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
18. 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

19. Georgiou, Y., Jeannot, E., Mercier, G., Villiermet, A.: Topology-aware resource management for HPC applications. In: ICDCN, pp. 17:1–17:10. ACM (2017)
20. Hilbert, D.: Ueber die stetige Abbildung einer Line auf ein Flächenstück. Math. Ann. **38**(3), 459–460 (1891)
21. Isaila, F., Carretero, J., Ross, R.B.: CLARISSE: a middleware for data-staging coordination and control on large-scale HPC platforms. In: CCGrid, pp. 346–355. IEEE, May 2016
22. 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
23. Lucarelli, G., Machado Mendonça, F., Trystram, D., Wagner, F.: Contiguity and locality in backfilling scheduling. In: CCGRID, pp. 586–595. IEEE Computer Society, May 2015
24. Morton, G.M.: A computer Oriented Geodetic Data Base; and a New Technique in File Sequencing. Technical report, IBM Ltd., March 1966. https://domino.research.ibm.com/library/cyberdig.nsf/0/0dabf9473b9c86d48525779800566a39
25. Ngoko, Y.: Heating as a cloud-service, a position paper (industrial presentation). In: Dutot, P.-F., Trystram, D. (eds.) Euro-Par 2016. LNCS, vol. 9833, pp. 389–401. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-43659-3_29
26. 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)
27. Strohmaier, E., Dongarra, J., Simon, H., Meuer, M.: TOP500 list. https://www.top500.org/lists/
28. 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 (Nov 2016)
29. 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