Compas'2022 : Parallélisme / Architecture/ Système
MIS/UPJV - Amiens France, 5-8 juillet 2022

IO-SETS : Simple and efficient approaches for I/O bandwidth management

Francieli Boito, Guillaume Pallez, Luan Teylo, and Nicolas Vidal*

Univ. Bordeaux, CNRS, Bordeaux INP, INRIA, LaBRI, UMR 5800, F-33400 Talence, France
{francieli.zanon-boito,guillaume.pallez, luan.gouveia-lima, nicolas.vidal}@inria.fr

Résumé
One of the main performance issues faced by high-performance computing platforms is the congestion caused by concurrent I/O from applications. When this happens, the platform's overall performance and utilization are harmed. From the extensive work in this field, I/O scheduling is the essential solution to this problem. The main drawback of current techniques is the amount of information needed about applications, which compromises their applicability. In this paper, we propose a novel method for I/O management, IO-SETS. We present its potential through a scheduling heuristic called SET-10, which requires minimum information and can be easily implemented.

Mots-clés : HPC ; Parallel I/O ; I/O scheduling ; Resource management

1. Introduction

As high-performance applications increasingly rely on data, the stress put on the I/O system has been one of the key bottlenecks of supercomputers. Indeed, processing speed has increased at a much faster pace than parallel file system (PFS) bandwidth. For example, the first supercomputer in the Top500 list [1] of November 1999 was ASCI Red, with peak performance of 3.2 TFlops and 1 GB/s of PFS bandwidth [9]. Twenty years later, in November 2019, the fastest machine was Summit, with 200.8 PFlops (over $62,000\times$ faster) and a PFS capable of 2.5 TB/s ($2,500\times$ faster) [16]. The I/O bottleneck becomes a bigger issue when multiple applications access the I/O infrastructure simultaneously. Congestion delays their execution, which means they hold compute resources for longer, hurting the utilization of the platform. Moreover, performance variability increases, since the application's execution time depends not only on what it does but on the current state of the machine.
In this context, solutions were proposed to try to manage the accesses to the shared I/O system. The main way of doing so, and the topic of this work, is I/O scheduling [12, 15, 7, 8, 21] : it consists in deciding algorithmically which applications get priority in this access.
In the majority of the I/O scheduling approaches in the related literature, the I/O operations are performed exclusively (i.e. one application at a time), or semi-exclusively when applications cannot use all the available bandwidth by themselves (i.e. some applications run concurrently using as much bandwidth as they can). An alternative method is to use *fair-sharing* : the bandwidth is shared equally by all applications in a best-effort fashion. With fair-sharing, when two applications perform I/O concurrently, each one takes twice the time it would take by itself.

*. The authors are presented in alphabetical order.

We propose a novel approach that is built on the intuition that both exclusive and fair-sharing have benefits. With this respect, we design a two-pronged I/O management approach for HPC systems : first, applications are sorted into *sets*. If applications from the same set try to perform I/O concurrently, that is done in an exclusive fashion (one at a time). Nonetheless, applications from different sets can do I/O accesses at the same time, and in this case they share the available bandwidth. The main contributions of this paper are the following :

— a novel method for I/O management in HPC systems ;
— an instantiation of this method with very simple heuristics that require little information about applications ;

The rest of this paper is organized as follows. In Section 2, we detail the problem we studied and our platform and application models, so that we can present our method and studied heuristics in Section 3. Results are presented in Section 4. The paper closes with related work in Section 5 and final remarks in Section 6.

2. I/O scheduling in HPC systems

In this work, we assume that we have a parallel platform, with a separated I/O infrastructure that is shared by all nodes and that provides a total bandwidth B. There are many ways to share the I/O bandwidth between concurrent I/O accesses [7]. *One of the novel ideas that we develop here is to manage I/O bandwidth using a priority-based approach*, which was inspired by a network protocol [20]. It consists in assigning priorities to I/O accesses. The idea is that, when an application (or job, in this paper we use both terms without distinction) performs an I/O phase by itself, it can use the full bandwidth of the system. However, when there are k concurrent requests for I/O, with respective priorities $p_1, p_2, \ldots, p_k$, then for $i \in \{1, \ldots, k\}$, the i^{th} request is allocated a share x_i of the total bandwidth such that :

$$x_i = \frac{p_i}{\sum_{j=1}^{k} p_j} \tag{1}$$

We consider the execution of workloads composed of N jobs. HPC applications have been observed to alternate between compute and I/O phases [7, 11]. Thus, for $j \leq N$, job J_j consists of n_j non-overlapping iterations. For $i \leq n_j$, iteration i is composed of a computing phase of length $t_{cpu}^{(i)}$ followed by an I/O phase of length $t_{io}^{(i)}$ [2]. These lengths (in time units) correspond to when the job is run in isolation on the platform (i.e. when there is no interference from other jobs). We consider here that an I/O operation that would take t_{io} units of time in isolation takes t_{io}/x units of time when using only a share x of the I/O bandwidth.

The precise I/O profile of an application, i.e. all its I/O accesses along with their volumes, start and finish time, can be extremely hard to predict. Obtaining this would involve a detailed I/O profiler that would impose a heavy overhead and generate a large amount of data, and still the profile would suffer from inaccuracy. I/O profilers used in practice focus on average or cumulative data. For instance, Darshan [17] is able to measure the total volume of I/O by an application. Therefore, we focus on average parameters.

The first parameter is the *characteristic time*, w_{iter}, of an application with n iterations. This parameter can be seen as the average time between the beginning of two consecutive I/O phases :

$$w_{iter} \stackrel{def}{=} \frac{\sum_{i \leq n} t_{cpu}^{(i)} + t_{io}^{(i)}}{n} \tag{2}$$

2. For simplicity, in the following and if there is no ambiguity, we relax some of the indexes and use n for the number of iterations, and t_{cpu} and t_{io} for the length of a compute or I/O phase

We also define a job's average portion of time spent on I/O. The I/O ratio of a job is : $\alpha_{io} = \frac{\sum_{i \leq n} t_{io}^{(i)}}{\sum_{i \leq n} t_{cpu}^{(i)} + t_{io}^{(i)}}$. Using this, we can define the average *I/O stress* of the platform with N applications at a given time :

$$\omega = \sum_{j \leq N} \alpha_{io}^{(j)} \tag{3}$$

The intuition behind this value is that it corresponds roughly to the expected average occupation of the I/O bandwidth. We can make the following important remarks :

— These average values do not depend on the number of used nodes or their performance. In practice, an application that runs on one node of 1 GFlops for 20 minutes and performs 20 iterations has the same w_{iter} value (1 minute) than another that runs on 2000 nodes with at 2 TFlops for 120 minutes and performs 120 iterations.

— The characteristic time and I/O ratio as defined above are average values that we expect could be evaluated easily, or approximated from previous runs. Due to being averages, they are more robust to variability than individual phases (Ergodic Theorem).

3. IO-SETS : a novel I/O management method for HPC systems

In this work we propose the IO-SETS method, which allows exclusive access for some applications, and (not fair) bandwidth sharing for others. Our proposition is a *set-based approach*, described below.

— when an application wants to do I/O, it is assigned to a set $S_i \in \{S_0, S_1, \ldots\}$.

— Each set S_i is allocated a bandwidth priority p_i.

— At any time, only one application per set is allowed to do I/O (exclusive access within sets). We use the first-come-first-served scheduling strategy within a set (i.e. we pick the application that requested it the earliest).

— If applications from multiple sets request I/O, they are allowed to proceed and their share of the bandwidth is computed using the sets' priorities and Equation (1).

Proposing a heuristic in the IO-SETS method consists therefore of answering two important questions : (i) how do we choose the set in which an application is allocated, and (ii) how do we define the priority of a set. We can illustrate the instantiation of the IO-SETS method by using it to represent the two discussed reference strategies.

EXCLUSIVE-FCFS : for this heuristic, all applications have exclusive access and are scheduled in a first-come-first-served fashion. This is the case where there is a single set S_1 with any priority (for instance $p_1 = 1$). All applications are scheduled in the set S_1.

FAIR-SHARE : in this heuristic, the bandwidth is shared equally among all applications requesting I/O access. This can be modeled by the case where there is one set by application $(S_0, \ldots, S_k)$, all with the same priority $p_i = 1$. Application id is scheduled in set S_{id}.

3.1. SET-10 algorithm

When two jobs have the same characteristic time, it seems more efficient to provide them with exclusive I/O access so that they can synchronize their phases. In this case, one of them would pay once a delay equal to the length of the other's I/O phase, but then for the remaining iterations, neither of them would be delayed. However, exclusive access does not bring benefits when the applications' characteristic times are very different. Based on this observation, we propose the SET-10 heuristic, which builds the sets depending on w_{iter}.

Given an application A_{id}, with a characteristic time w_{iter}^{id}, then the mapping allocation is π :

$$\pi : A_{id} \mapsto S_{\lfloor \log_{10} w_{iter}^{id} \rceil} \tag{4}$$

where $\lfloor x \rceil$ defines the nearest integer value of x.
To define priorities for the sets, we start with the following observation : if two applications start performing I/O at the same time and share the bandwidth equally, the one with the smallest I/O volume finishes first. Now if we increase the bandwidth of the smallest, it will finish earlier, but the largest one will not be delayed. Based on this, we want to provide higher bandwidth to the sets with the smallest I/O accesses. Intuitively, if the characteristic time w_{iter} are of different orders of magnitude, we assume that so are the I/O accesses. An advantage of using w_{iter} instead of the I/O volume is that it is easy to obtain, as previously discussed. Hence we define the priority p_i of set S_i (which corresponds to jobs such that $i = \lfloor \log_{10} w_{iter}^{id} \rceil$) :

$$p_i = 10^{-i} \tag{5}$$

We define SET-10 the strategy in the IO-SETS method consisting of :

- The list of sets with their priorities $\{\ldots, (S_{-1}, 10), (S_0, 1), (S_1, 0.1), (S_2, 0.01), \ldots\}$ (where the priorities are computed with Equation (5)) ;
- The π function that maps jobs to sets using Equation (4).

4. Evaluation and Results

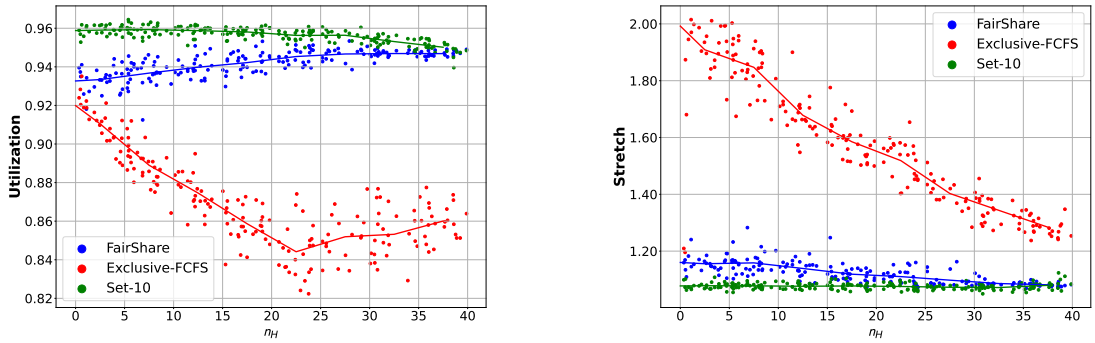
The evaluation in this paper represents the first exploratory work to show the importance of the IO-SETS method. For that, we use Simgrid v3.30 [4], an open-source simulator of distributed environments that also simulates I/O operations [13]. We chose to use simulation because we wanted to test many scenarios, to extensively assess our method's strengths and limitations (the extended version of the paper comprises tests with hundreds of different workloads). Moreover, test platforms are somewhat limited in size, but deploying I/O scheduling strategies in a production system (just to test them) is complex and usually not allowed. All code used for our evaluation, the details of implementation, and the extended version of the paper are available and documented at `https://gitlab.com/u693/iosets`.
In our experiments all applications start at the beginning of the simulation. It means that the early and final stages are not representative in terms of arriving I/O requests. To avoid such artifacts, we do our measurements within a time frame $[T_{begin}, T_{end}]$ with $T_{begin} > 0$ and $T_{end} < h$. Specifically, with a horizon $h = 20,000$s, we select : $T_{begin} = 1,000$s and $T_{end} = 9,000$s. That ensures that it starts after a possible de-synchronization of the different applications and stops before any application finishes. Additionally, for each task J_j, we define the effective completed work $e_j = e_j^{cpu} + e_j^{io}$ as the sum of the lengths of all compute and I/O phases executed by this task within the interval. We create 200 workloads each with $N = 60$ parallel applications, and we consider a saturated system, with an average I/O stress ω =0.80 (Equation 3).
We considered two metrics : i) the system `Utilization` (Equation 6), i.e., the proportion of time loss due to I/O congestion ; and ii) the application `Stretch` (Equation 7), i.e., how long does it take for a user to get a result. We aim to show that IO-SETS is indeed a relevant method when compared to the reference strategies EXCLUSIVE-FCFS and FAIR-SHARE. In order to focus on the importance of the use of sets in the IO-SETS method, we design a collection of applications that clearly belong to different groups of the heuristic SET-10. By doing that, we temporarily put aside the question of designing an efficient heuristic for IO-SETS.

$$\texttt{Utilization} = \frac{\sum_j e_j^{cpu}}{N.(T_{end} - T_{begin})} \quad (6) \qquad \texttt{Stretch} = \max_j \frac{T_{end} - T_{begin}}{e_j} \quad (7)$$

As discussed in Section 3, a job is fully described by its number of iterations n, and compute and I/O phases' length. In order to generate each job, we proceed as detailed below.

1. We decide on a characteristic time w_{iter} for the job. In practice, it is drawn from the normal distribution $\mathcal{N}(\mu, \sigma)$, truncated so that we consider only positive values. μ and σ are selected on an experiment basis.
2. The number of iterations of the job is $n = \frac{h}{w_{iter}}$.
3. $\alpha_{io}^{(j)}$ is defined as follows : for each application, we draw a value α_j uniformly at random in $\mathcal{U}[0,1]$, then we set $\alpha_{io}(j) = \frac{\omega}{\sum_{k \leq N} \alpha_k} \cdot \alpha_j$ to guarantee an I/O Load of ω.

α_{io} allows us to define the average values t_{cpu} and t_{io} : $t_{cpu} = (1 - \alpha_{io}) \cdot w_{iter}$ and $t_{io} = \alpha_{io} \cdot w_{iter}$.



(a) Machine utilization

(b) Maximum stretch

FIGURE 1 – Comparison of the policies for workloads with sets of different sizes. All 200 results obtained with each policy are presented. The y-axes are different and do not start at 0.

In Section 2, we explained the intuition that jobs should be grouped based on their characteristic time (w_{iter}). Therefore, we define three different job profiles (i.e. values for μ and σ) :

— n_H jobs with characteristic time $w_{iter} \sim \mathcal{N}(10, 1)$, also called high-frequency jobs in the following (the mean duration of an iteration is 10s) ;
— n_M medium-frequency jobs with $w_{iter} \sim \mathcal{N}(100, 10)$;
— n_L low-frequency jobs with $w_{iter} \sim \mathcal{N}(1000, 100)$;

We set $n_H + n_M + n_L = 60$, and specifically we study the impact of the algorithms when $n_M = 20$, $n_H \in \{0, \ldots, 40\}$, and $n_L = 40 - n_H$.

`Utilization` is presented in Figure 1a. The FAIR-SHARE and SET-10 strategies have a high and stable platform usage, the latter better by up to 4.4% when there are few high-frequency jobs. This difference disappears when n_H increases. EXCLUSIVE-FCFS has rather poor results compared to the other heuristics, with a higher utilization when I/O phases are mostly large.

Figure 1b presents the `Stretch` objective for each workload. We can observe the important gains provided by SET-10, a 15.3% improvement over FAIR-SHARE, especially when there are mostly low-frequency jobs (larger I/O phases). This difference disappears as n_H increases. Here again, EXCLUSIVE-FCFS has the worst results in all cases, confirming its expected shortcomings. Interestingly, the `Utilization` is not correlated to the `Stretch` : indeed, in an exclusive scenario, one of the high-frequency jobs can be penalized many times (which increases its `Stretch`), without hurting the `Utilization` significantly.

When there are diverse job profiles, EXCLUSIVE-FCFS is not a good strategy for scheduling I/O. While it is important to share the bandwidth, keeping some exclusivity has a positive impact on performance, showing the importance of the IO-SETS method.

5. Related Work

In this paper, we proposed a method for managing applications' accesses to the shared parallel file system in an HPC platform. Other efforts were put into scheduling I/O operations aiming at mitigating interference. Gainaru et al. [7] and Zhou et al. [22] schedule applications' operations at the I/O node level. In a more recent contribution, Gainaru et al. [8] propose a pattern-based schedule relying on the periodic I/O behavior. Dorier et al. [5] analyze the benefits of either delaying or interrupting an application when it is interfering with another one. ASCAR [14] uses controllers on storage clients to detect and reduce congestion using traffic rules. The scheduler AIS [15] identifies offline the I/O characteristics of the application and uses this information to avoid conflicts by issuing I/O aware scheduling recommendations. Alves and al. [2], Snyder and al. [18] and Dorier and al. [6] use different models to predict and avoid I/O interference. Our work is motivated by the observation that both exclusive and shared access to the bandwidth have advantages, as observed by Jeannot et al. [12]. **To the best of our knowledge, ours is the first work to propose classifying applications into sets and using that for managing the bandwidth.**
Others have improved the batch scheduler to consider I/O as one of the resources to arbitrate. Grandl et al. [10] and Tan et al. [19] enrich theoretical scheduling problems with additional dimensions such as I/O requirements. Bleuse et al. [3] attempt to schedule applications requesting both compute and I/O nodes. In their model, Herbein et al. [11] consider that jobs perform I/O proportionally to the number of nodes they need. They show that considering I/O helps reducing performance variability. Our approach is complimentary to those, because we focus on the situation where there already is a set of applications running and requesting I/O.

6. Conclusion

In this work we have introduced a novel method for I/O management in HPC systems : IO-SETS, intended to be easy to implement and light in overhead. This method consists in a smooth combination of exclusive and non-exclusive bandwidth sharing. We have proposed the SET-10 algorithm, based on this method, where jobs are categorized depending on the order of magnitude of their characteristic times. The latter is an average value that corresponds to the mean time between two consecutive I/O phases of an application. We shown the importance of IO-SETS, and then the excellent performance of the SET-10 algorithm.
We believe that the results from this paper open a wide range of avenues for I/O management. From a theoretical standpoint, optimal set mapping and bandwidth partitioning algorithms can be investigated. From a practical point of view, the implementation of IO-SETS will have to adapt to real applications that have more complex I/O behaviors that change over time. An important direction that we will consider is that jobs can change sets at each I/O phase.
For research on I/O monitoring tools, our results indicate that an average behavior may be enough information to obtain, and also cheaper. Finally, we believe our method could be successful in other parts of the I/O stack, for instance to manage the access to shared burst buffers.

Acknowledgments

Experiments presented in this paper were carried out using the PlaFRIM experimental testbed, supported by

Inria, CNRS (LABRI and IMB), Université de Bordeaux, Bordeaux INP and Conseil Régional d'Aquitaine (see https ://www.plafrim.fr). This work was supported in part by the French National Research Agency (ANR) in the frame of DASH (ANR-17- CE25- 0004), by the Project Région Nouvelle Aquitaine 2018-1R50119 "HPC scalable ecosystem" and by the "Adaptive multitier intelligent data manager for Exascale (ADMIRE)" project, funded by the European Union's Horizon 2020 JTI-EuroHPC Research and Innovation Programme (grant 956748).

Bibliographie

1. Top500. – https ://www.top500.org/.
2. Alves (M. M.) et de Assumpçao Drummond (L. M.). – "A multivariate and quantitative model for predicting cross-application interference in virtual environments". *Journal of Systems and Software*, vol. 128, 2017, pp. 150 – 163.
3. Bleuse (R.), Dogeas (K.), Lucarelli (G.), Mounié (G.) et Trystram (D.). – Interference-aware scheduling using geometric constraints. – In *European Conference on Parallel Processing*, pp. 205–217. Springer, 2018.
4. Casanova (H.), Giersch (A.), Legrand (A.), Quinson (M.) et Suter (F.). – Versatile, scalable, and accurate simulation of distributed applications and platforms. *Journal of Parallel and Distributed Computing*, vol. 74, n10, 2014, pp. 2899–2917.
5. Dorier (M.), Antoniu (G.), Ross (R.), Kimpe (D.) et Ibrahim (S.). – CALCioM : Mitigating I/O interference in HPC systems through cross-application coordination. – In *Parallel and Distributed Processing Symposium, 2014 IEEE 28th International*, pp. 155–164. IEEE, 2014.
6. Dorier (M.), Ibrahim (S.), Antoniu (G.) et Ross (R.). – Using formal grammars to predict I/O behaviors in HPC : The omnisc'io approach. *IEEE Transactions on Parallel and Distributed Systems*, vol. 27, n8, Aug 2016, pp. 2435–2449.
7. Gainaru (A.), Aupy (G.), Benoit (A.), Cappello (F.), Robert (Y.) et Snir (M.). – Scheduling the I/O of HPC applications under congestion. – In *2015 IEEE International Parallel and Distributed Processing Symposium*, pp. 1013–1022. IEEE, 2015.
8. Gainaru (A.), Le Fèvre (V.) et Pallez (G.). – I/O scheduling strategy for periodic applications. *ACM Transactions on Parallel Computing*, 2019.
9. Garg (S.). – Tflops pfs : Architecture and design of a highly efficient parallel file system. – In *SC '98 : Proceedings of the 1998 ACM/IEEE Conference on Supercomputing*, pp. 2–2, 1998.
10. Grandl (R.), Ananthanarayanan (G.), Kandula (S.), Rao (S.) et Akella (A.). – Multi-resource packing for cluster schedulers. *SIGCOMM Comput. Commun. Rev.*, vol. 44, n4, août 2014, pp. 455–466.
11. Herbein (S.), Ahn (D. H.), Lipari (D.), Scogland (T. R.), Stearman (M.), Grondona (M.), Garlick (J.), Springmeyer (B.) et Taufer (M.). – Scalable I/O-aware job scheduling for burst buffer enabled hpc clusters. – In *Proceedings of the 25th ACM International Symposium on High-Performance Parallel and Distributed Computing*, pp. 69–80, 2016.
12. Jeannot (E.), Pallez (G.) et Vidal (N.). – Scheduling periodic I/O access with bi-colored chains : models and algorithms. *Journal of Scheduling*, vol. 24, n5, 2021, pp. 469–481.
13. Lebre (A.), Legrand (A.), Suter (F.) et Veyre (P.). – Adding storage simulation capacities to the simgrid toolkit : Concepts, models, and api. – In *2015 15th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing*, pp. 251–260. IEEE, 2015.
14. Li (Y.), Lu (X.), Miller (E. L.) et Long (D. D. E.). – ASCAR : Automating contention management for high-performance storage systems. – In *2015 31st Symposium on Mass Storage Systems and Technologies (MSST)*, pp. 1–16, May 2015.
15. Liu (Y.), Gunasekaran (R.), Ma (X.) et Vazhkudai (S. S.). – Server-Side Log Data Analytics for I/O Workload Characterization and Coordination on Large Shared Storage Systems. – In

SC '16 : Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis, pp. 819–829, Nov 2016.

16. Oral (S.), Vazhkudai (S. S.), Wang (F.), Zimmer (C.), Brumgard (C.), Hanley (J.), Markomanolis (G.), Miller (R.), Leverman (D.), Atchley (S.) et Larrea (V. V.). – End-to-end I/O portfolio for the summit supercomputing ecosystem. – In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, *SC '19*, SC '19, New York, NY, USA, 2019. Association for Computing Machinery.
17. Snyder (S.), Carns (P.), Harms (K.), Ross (R.), Lockwood (G. K.) et Wright (N. J.). – Modular HPC I/O characterization with darshan. – In *2016 5th workshop on extreme-scale programming tools (ESPT)*, pp. 9–17. IEEE, 2016.
18. Snyder (S.), Carns (P.), Latham (R.), Mubarak (M.), Ross (R.), Carothers (C.), Behzad (B.), Luu (H. V. T.), Byna (S.) et Prabhat. – Techniques for modeling large-scale HPC I/O workloads. – In *Proceedings of the 6th International Workshop on Performance Modeling, Benchmarking, and Simulation of High Performance Computing Systems*, *PMBS '15*, PMBS '15, pp. 5 :1–5 :11, New York, NY, USA, 2015. ACM.
19. Tran (T. T.), Padmanabhan (M.), Zhang (P. Y.), Li (H.), Down (D. G.) et Beck (J. C.). – Multi-stage resource-aware scheduling for data centers with heterogeneous servers. *Journal of Scheduling*, vol. 21, n2, avril 2018, pp. 251–267.
20. Velho (P.), Schnorr (L. M.), Casanova (H.) et Legrand (A.). – On the validity of flow-level tcp network models for grid and cloud simulations. *ACM Transactions on Modeling and Computer Simulation (TOMACS)*, vol. 23, n4, 2013, pp. 1–26.
21. Zhang (X.), Davis (K.) et Jiang (S.). – IOrchestrator : Improving the performance of multi-node I/O systems via inter-server coordination. – In *SC'10 : Proceedings of the 2010 ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis*, pp. 1–11. IEEE, 2010.
22. Zhou (Z.), Yang (X.), Zhao (D.), Rich (P.), Tang (W.), Wang (J.) et Lan (Z.). – I/O-aware batch scheduling for petascale computing systems. – In *2015 IEEE International Conference on Cluster Computing*, pp. 254–263. IEEE, 2015.