



**HAL**  
open science

## What size should your Burst-Buffers be?

Guillaume Aupy, Olivier Beaumont, Lionel Eyraud-Dubois

► **To cite this version:**

Guillaume Aupy, Olivier Beaumont, Lionel Eyraud-Dubois. What size should your Burst-Buffers be?. 2017. hal-01623846v1

**HAL Id: hal-01623846**

**<https://inria.hal.science/hal-01623846v1>**

Preprint submitted on 25 Oct 2017 (v1), last revised 13 Jul 2018 (v2)

**HAL** is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

# What size should your Burst-Buffers be?

*Guillaume Aupy, Olivier Beaumont and Lionel Eyraud-Dubois*  
*Inria and University of Bordeaux, Talence, France*  
*surname.name@inria.fr*

## Abstract

Burst-Buffers are high throughput, small size intermediate storage systems typically based on SSDs or NVRAM that are designed to be used as a potential buffer between the computing nodes of a supercomputer and its main storage system consisting of hard drives. Their purpose is to absorb the bursts of I/O that many HPC applications experience (for example for saving checkpoints or data from intermediate results). In this paper, we propose a probabilistic model for evaluating the performance of Burst-Buffers. From a model of application and a data management strategy, we build a Markov chain based model of the system, that allows to quickly answer issues about dimensioning of the system: for a given set of applications, and for a given Burst-Buffer size and bandwidth, how often does the buffer overflow? We also provide extensive simulation results to validate our modeling approach.

## 1 Introduction

Solving the bottleneck of I/O is a major point in current HPC systems. This point is especially striking when observing their recent evolution. For instance, when Los Alamos National Laboratory moved from Cielo to Trinity, the peak performance moved from 1.4 Petaflops to 40 Petaflops ( $\times 28$ ) while the I/O bandwidth moved to 160 GB/s to 1.45TB/s (only  $\times 9$ ) [1]. The same kind of results can be observed at Argonne National Laboratory when moving from Intrepid (0.6 PF, 88 GB/s) to Mira (10PF, 240 GB/s) and to Aurora (expected 180PF and 1TB/s) [2]. Indeed, the main storage technology is still based on hard drive, that have shown a better capacity to scale up in terms of storage capacity than speed.

On the other hand, the usage of HPC systems makes I/O more and more important. First, in the framework on the convergence between HPC and Big-Data [3], HPC systems are more and more used to run BigData applications, that require much more I/O bandwidth than traditional HPC applications. The main characteristic of BigData workload is that they are dominated by read operations. Second, the MTBF (Mean Time Between Failures) of HPC systems is decreasing [4, 5] and Checkpoint/Restart strategies are used to ensure the reliability of computations over a failure prone system. Contrarily to BigData

applications, Checkpoint/Restart strategies consume a lot of I/O bandwidth for storing checkpoints, and are dominated by write operations. An important feature of checkpoints is that in general, each new checkpoint for an application can erase the previous one, so that not all checkpoints have to be ultimately saved to the slow disk. Third, HPC applications themselves consume a lot of I/O bandwidth (see Section 2.3). They typically follow a quasi-periodic pattern, with an alternance of compute and I/O phases, that cannot be overlapped. As in the case of checkpointing, such HPC applications mostly write data to the disk, and due to their bursty nature, they are known to generate interferences and idle times at the I/O level [6] when several applications attempt to store their results simultaneously.

In order to cope with the limited I/O bandwidth of HPC system, Burst-Buffers have emerged as promising solution [7, 8, 9]. On the technological side, the use of NVRAM or SSDs makes it possible to achieve much higher bandwidth than hard disks. Due to their high cost and limited capacity, Burst-Buffers are not expected to replace hard-drives, but rather act like a potential intermediate storage layer between the computing nodes and the hard drive storage. Such a layer can be used both to increase data locality and to cope with I/O bursts, using their higher I/O bandwidth to avoid interferences and slowdowns.

There is still no clear consensus on Burst-Buffer architecture (see Section 2.1) to know whether they should be centralized or distributed over the platform, and whether they should act as a cache between the computing nodes and the storage system or if they could be bypassed by the applications. In this paper, we consider the simplest model where the Burst-Buffer acts as a potential intermediate centralized layer, with a higher I/O bandwidth but a smaller capacity than the hard disk storage system.

Our goal is to propose a probabilistic model for applications, that is amenable to theoretical analysis based on Markov chains and that provides an estimation of the I/O contention as a function of the size of the Burst-Buffer at a very reasonable cost. The rest of this paper is organized as follows. In Section 2, we present the related work on Burst-Buffer architecture, bandwidth allocation and HPC applications models. Then, we present in Section 3 the probabilistic model that will be used throughout the paper. In Section 4, we propose two methods to evaluate I/O bandwidth overflows respectively based on the use of (weighted) Chernoff bounds and on an ad-hoc algorithm. Then, we prove in Section 5 that a simple Markov chain can be used to evaluate the idle time induced by Burst-Buffer overflow and we provide simulation results to study the important parameters of the Burst-Buffers. In Section 6, we validate the probabilistic model introduced in Section 3 by comparing the results of the Markov chain approach with those of a discrete event simulator based on a more sophisticated applications model. At last, we prove in Section 7 that the Markov chain approach can be used to evaluate other Burst-Buffer management strategies, such as those recently introduced to limit transfers between the Burst-Buffer and the storage system, and we propose concluding remarks in Section 8

## 2 Related Work

### 2.1 Burst-Buffer Architectures and models

There are many implementations of Burst-Buffers. The two most studied characteristics are the location of the buffers and whether the buffers are shared between multiple applications.

Typically, Burst-Buffers can be located between the compute nodes and the Parallel File System (PFS). This is the case of DDN IME [7, 10] and Cray DataWarp [8, 11, 9]. In this *pseudo-centralized* architecture, the Burst-Buffers are often colocated with the I/O nodes. The management strategy can then differ. Mubarak et al. [11] study the case where the buffers are shared between the different applications on the platform and used to accelerate transfers and to prevent I/O congestion. On the contrary, in Schenck et al. [10] and Daley et al. [9], applications decide the size of the buffer that should be dedicated to them.

Another solution is a *distributed* version of Burst-Buffers where the buffers are allocated close to the compute nodes [12, 13]. A solution consists in allocating the distributed buffers to the applications using the compute nodes close to buffers [14]. However, other strategies focus on how to share them between the different applications [11, 12, 13]. This is particularly true in the context of fault-tolerance, where using a buffer on a different node can allow the implementation of hierarchical checkpointing strategies that provide more resilience than in-node buffer strategies [13]. Furthermore, in the case where, because of their costs, the number of buffers in the machine has to be limited, one must choose on which node they should be deployed and between which subset of applications they should be shared.

In this paper, we focus on the *pseudo-centralized* model, where Burst-Buffers are shared between applications. We use a model of architecture similar to the one shown by Schenck et al. [10]. We consider that applications are running on *Compute Nodes*. They use or generate data (also called I/O in this work) that has to be sent on the *Parallel File System*. This is done by sending their data to the I/O nodes where Burst-Buffers are located.

More precisely, the architecture we consider is depicted in Figure 1.

- The I/O nodes communicate with the PFS using a maximum bandwidth  $B$ .
- The Burst-Buffers are located next to the I/O nodes, their total size is denoted by  $S$  (in GB) and they communicate with the I/O nodes with a bandwidth  $B_{BB}$ .

### 2.2 Algorithms to deal with Burst-Buffers

When it comes to using Burst-Buffers, several solutions have been proposed. We present and discuss the most common ones.

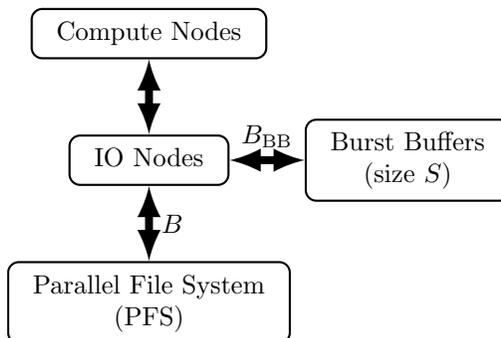


Fig. 1: Description of the pseudo-centralized platform model.

A natural idea is to use Burst-Buffers as a cache to improve the I/O-performance of applications [10]. For instance, DDN [7] announces bandwidth performance 10-fold that of PFS using their Burst-Buffers. The idea is to move the I/O to the Burst-Buffer as a temporary stage between compute nodes and the Parallel File System (whether the data is incoming or outgoing). Thanks to the higher bandwidth of the Burst-Buffers, this has the advantage of improving the I/O transfer time while pipelining the (slowest) phase of sending/receiving data from the PFS with the compute phase of the application. However, as was noted by Han et al. [15], this idea is not viable, (i) Burst-Buffers are based on technologies that are extremely expensive with respect to hard drives, (ii) they are currently based on SSD technology, that is known to have a limited rewrite lifespan [15]. Thus, the large number of I/O operations in HPC applications would decrease their lifespan too fast. This is why we do not consider this solution in this work.

The second natural idea proposed in the literature is to use Burst-Buffers to prevent I/O congestion [16, 17] while maintaining their lifespan. To achieve this goal, the applications use the direct link to the PFS (see Figure 1) when its bandwidth  $B$  is not exceeded. When the bandwidth is exceeded by the set of transfers, then the higher bandwidth of the Burst-Buffer is used to complement the bandwidth of the PFS. This is the solution advocated by DDN in [7]. The intuition behind this strategy is that the average use of PFS bandwidth is usually small enough, but that Burst-Buffers are crucial to deal with applications' (simultaneous) bursts. This corresponds to the model depicted in Figure 1 that will be used throughout this paper.

Finally, a large part of the literature on Burst-Buffers shows how to use them with a specific application workflow [9, 10]. Specifically, they consider systems where applications have dedicated and pre-allocated Burst-Buffers, and where the application can explicitly control its data transfers and the use of the Burst-Buffer. This requires to be done for each application and is very platform dependent. In practice, only few applications have the human-power to implement this. By opposition, our work is only architecture dependent and does not

require any additional work from applications developers. However, we believe our results can also be used by applications developers if they want to estimate the size of Burst-Buffers that they would need based on their application characteristics.

### 2.3 Application Model

Many recent HPC studies have independently observed patterns in the I/O behavior of HPC applications. The quasi-periodicity of HPC applications has been well documented [18, 19]. HPC applications alternate between computation and I/O transfer phases, this pattern being repeated almost identically over time.

Hu et al. [20] summarized the four key characteristics of HPC applications observed in the literature.

1. *Periodicity*: Applications alternate between compute phases and I/O phases. Furthermore they do so in a quasi periodic fashion.
2. *Burstiness*: In addition to the observed periodicity, sometimes, short I/O bursts occur.
3. *Synchronization*: I/O transfers of an application are performed in a synchronized way between the different parallel processes.
4. *Repeatability*: The same jobs are often run many times with different inputs, hence their compute-I/O pattern of an application can be reasonably predicted before execution.

When modeling applications, most Burst-Buffer related work use workload models based on these patterns [17, 15, 11]. In addition to this, Mubarak et al. [11] introduce a random background traffic representing HPC workloads such as graph computations and linear algebra solvers, based on the work of Yuan et al. [21].

## 3 Model

We consider a large computing platform, and we focus on modeling and analyzing the behavior of the storage system. We will thus ignore the computing nodes of the platform, and simply consider that we have a (fixed) set of  $n$  applications  $\mathcal{A}_i$  currently executing on the platform. We model the long-term storage system as a single file server with input bandwidth  $B$  (in a real system with several file servers,  $B$  would represent their aggregate bandwidth). In addition to this file server, the platform contains a Burst-Buffer, whose input bandwidth is significantly larger, and whose size is  $S$ . Figure 1 provides a schematic view of the model we consider.

Following the related work discussed in Section 2.3, application  $\mathcal{A}_i$  alternates computation and I/O phases. When it performs computations it does not induce any load on the storage system. When it sends intermediate results to the disks, it does so with a fixed bandwidth denoted by  $b_i$ . We do not assume that the applications follow a perfect periodic pattern and we rather model their data transfers with a random process. We consider a fixed time unit, which is close to

the characteristic time of the applications (the order of magnitude of the I/O burst duration), and we assume that during each of these time units, application  $\mathcal{A}_i$  sends data with probability  $p_i$  (with bandwidth  $b_i$ ). An important quantity in our model is the expected load `EXPECTEDLOAD`, defined by  $\text{EXPECTEDLOAD} = \sum_i p_i b_i$ , that represents the average bandwidth requirement over a long period of time.

If many applications send data at the same time and exceed the bandwidth  $B$  to the PFS, the excess can be sent to the Burst-Buffer, and later on sent back to the disk when some bandwidth is available. If the Burst-Buffer is full, this will induce some sort of contention and idle time, which is what we try to avoid. Section 5 presents a model of the system behavior to analyze the waste implied by these overflow events.

## 4 Instantaneous Load Estimation

With the model of Section 3, since the expected load `EXPECTEDLOAD` is given by  $\text{EXPECTEDLOAD} = \sum_i p_i b_i$  and the available bandwidth is given by  $B$ , we can immediately notice that if  $\text{EXPECTEDLOAD} > B$ , the I/O bandwidth will not be able to cope with all the demands and the Burst-Buffer, whatever its size, will eventually be saturated. To cope with this situation, we will introduce in Section 5 a delay mechanism where the applications freeze and stop sending data to the Burst-Buffer during one time unit so has to avoid Burst-Buffer overflow.

In this section, we are rather interested in estimating the probability distribution of the `INSTANTLOAD` of I/Os, in order to see if it exceeds the capacity of I/O bandwidth to the Burst-Buffer  $B_{\text{BB}}$ . We propose two different approaches to achieve this goal. We rely in Section 4.1 on generalizations of Chernoff bounds to the case of a weighted sum of independent variables, and we propose in Section 4.2 an ad-hoc strategy to estimate precisely the discretized probability distribution, and we compare both strategies in Section 4.3.

### 4.1 Chernoff based bounds

In order to analyze the probability distribution of `INSTANTLOAD`, let us denote by  $X_i$  the random variable indicating whether  $\mathcal{A}_i$  is sending I/Os. We assume that  $X_i$ s are independent random variables, what is in general the case if workflows and coupled codes are considered as distinct applications. Using the model described in Section 3, we have  $\text{Pr}(X_i = 1) = p_i$  and  $\text{Pr}(X_i = 0) = (1 - p_i)$  and the random variable  $X$  which represents the instant load is given by  $X = \sum_i b_i X_i$  and its expected value is  $E(X) = \sum_i b_i p_i$ . We cannot apply directly Chernoff bounds since  $X$  is a weighted sum, but several extensions have been proposed in the literature to deal with weighted sums (see [22, 23, 24, 25]). Theorem 1 provides a mean to estimate the probability that the instant load differs by an additive value  $\lambda$  from its expected value.

**Theorem 1** ([23]). *Let  $X_1, \dots, X_n$  be independent random variables such that  $\text{Pr}(X_i = 1) = p_i$  and  $\text{Pr}(X_i = 0) = (1 - p_i)$  and let  $X = \sum_i b_i X_i$ . Let us set*

$\nu = \sum_i b_i^2 p_i$  and  $b = \max\{b_1, \dots, b_n\}$ . Then,

$$\Pr(X \leq E(X) - \lambda) \leq \exp\left(-\frac{\lambda^2}{2\nu}\right) \text{ and}$$

$$\Pr(X > E(X) + \lambda) \leq \exp\left(-\frac{\lambda^2}{2(\nu + b\lambda/3)}\right).$$

## 4.2 Ad-Hoc Estimation

Chernoff bounds are known to model well the asymptotic behaviors, but due to their simplicity and generality, they fail to provide useful estimations in practical cases as they overestimate the probability that  $X$  differs from its expected value. Therefore, we propose in this section an ad-hoc way to estimate the probability distribution of  $X$ . To obtain this result, we discretize the possible values of the bandwidth with respect to  $B$ . More precisely, let us set  $B = 100$  and let us assume that all  $b_i$  values are integer values. This corresponds to rounding the value  $b_i$  to the closest integer and induces a loss of precision of at most 1%. Given the difficulty to precisely estimate the I/O bandwidth required by an application  $\mathcal{A}_i$  when transferring data to the disk, we consider that this approximation is smaller than the noise of the measurements.

We describe in Algorithm 1 the code used to compute the probability distribution  $\text{DIST}$  of  $X = \sum_i b_i X_i$ , where  $\text{DIST}(k)$  is the probability that  $\mathcal{A}_1, \dots, \mathcal{A}_n$  contribute to exactly  $k$ . To compute  $\text{DIST}$ , we add applications  $\mathcal{A}_i$  one after the other and we update at step  $i$  the probability distribution using only applications  $\mathcal{A}_1, \dots, \mathcal{A}_i$ . The size of vector  $\text{DIST}$  is  $\sum_i b_i$ , *i.e.* the maximal possible instant value which is upper bounded by  $nB = 100n$ , such that the overall complexity of Algorithm 1 is  $100n^2$ , where  $n$  denote the number of applications and is therefore expected to be very small ( $n = 20$  is already large in practical settings), so that computing  $\text{DIST}$  is immediate.

---

### Algorithm 1: $\text{FINDDISTRIBUTION}(\mathcal{A}_1, \dots, \mathcal{A}_n)$

---

**Input:** The characteristics of  $\mathcal{A}_1, \dots, \mathcal{A}_n$

**Output:**  $\text{DIST}$ , such that  $\text{DIST}(k) = \Pr(\sum b_i X_i = k)$

```

1  $\text{DIST}$  is a initialized with zeros and its size is  $\sum b_i$  ;
2 foreach  $\mathcal{A}_i$  do
3   foreach  $k$  do
4      $\text{DIST}(k) = (1 - p_i)\text{DIST}(k) + p_i\text{DIST}(k - b_i)$  if  $k \geq b_i$  ;
5      $\text{DIST}(k) = (1 - p_i)\text{DIST}(k)$  if  $k < b_i$  ;
6   end
7 end
8 return  $\text{DIST}$ 
```

---

### 4.3 Estimating the distribution

We can use the Chernoff bounds proposed in Section 4.1 to estimate the distribution of the instantaneous load. Indeed, let us set

- $f_u(x) = \exp\left(-\frac{(\sum b_i p_i - x)^2}{2\nu}\right)$  if  $x \leq \sum b_i p_i$  and  $f_u(x) = 1$  otherwise
- $f_l(x) = \exp\left(-\frac{(x - \sum b_i p_i)^2}{2(\nu + b(x - \sum b_i p_i)/3)}\right)$  if  $x \geq \sum b_i p_i$  and  $f_u(x) = 0$  otherwise.

Then,  $\forall x$ ,  $f_l(x) \leq Pr(X \leq x) \leq f_u(x)$ .

To estimate the precision of these inequalities, we have compared them to the result provided by Algorithm 1 on some randomly generated set of applications. We target an expected load of  $M = 100 (= B)$ , we fix a maximum  $p_i$  value  $p_{\max} \leq 1$ , and we generate  $n = 10$  applications in the following way. We first generate the values  $p_i$  uniformly between 0 and  $p_{\max}$  and we generate  $b_i$  values uniformly between 0 and 1. Then, we rescale the  $b_i$ s to obtain an expected load  $\sum_i p_i b_i$  equal to  $M$ . Figure 2 provides the comparison of the obtained distribution estimations for different values of  $p_{\max}$ . We can see that the Chernoff bounds is always correct, but very imprecise. Since computing the exact distribution with Algorithm 1 is in practice very fast, we will use this algorithm in the rest of the paper.

An important information that can be read from Figure 2 is that when applications have low  $p_{\max}$  values (applications that do not often perform I/O, but with higher bandwidth), then it leads to much more variance. Indeed, while most of the time `INSTANTLOAD` = 0, when `INSTANTLOAD`  $\neq$  0, then the probability that it is very high (two or three times  $B$ ) is not negligible.

## 5 Burst-Buffer Load Estimation

In Section 4.2, we proposed Algorithm 1 to compute the probability  $\text{DIST}(k)$  of having `INSTANTLOAD` =  $k \ \forall 0 \leq k \leq \sum b_i$ . As shown in Section 4.3,  $\text{DIST}(k)$  can be used directly to evaluate the probability of exceeding the I/O bandwidth to the disk. In Section 5.1, we use  $\text{DIST}$  in order to evaluate the probability of exceeding the capacity of the Burst-Buffer, using a Markov Chain to model the load of Burst-Buffer.

### 5.1 Markov Chain Model

In this section, we propose to model the occupation of the Burst-Buffer when using Algorithm 2.

Following the model described in Section 2.2, Algorithm 2 transfers as much data as possible to the disk. If `INSTANTLOAD`  $\leq B$ , then the Burst-Buffer is emptied as fast as possible, provided that the I/O bandwidth to the disk  $B$  is not exceeded (see Section 3 and Figure 1). If `INSTANTLOAD`  $> B$ , then the occupation of the Burst-Buffer increases. If the capacity of the Burst-Buffer is

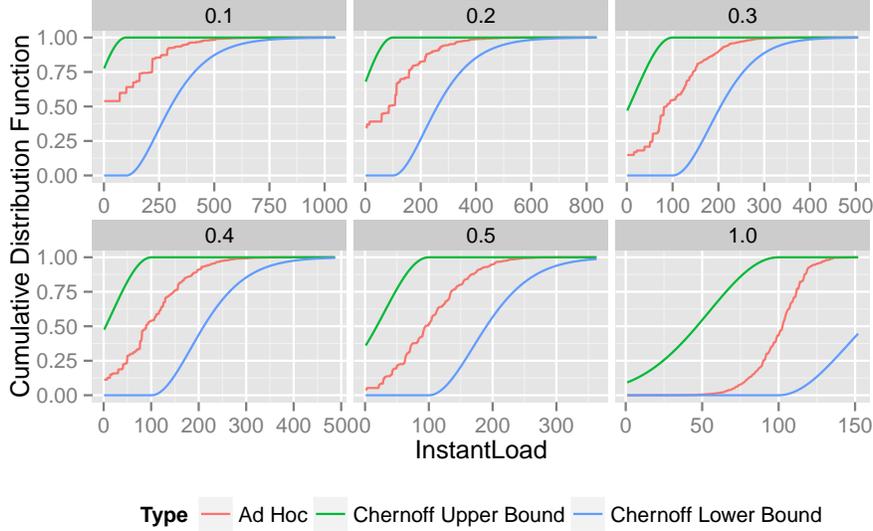


Fig. 2: Comparison of distribution estimations for different values of  $p_{\max}$ ,  $\text{EXPECTEDLOAD} = 100$ .

exceeded, then all transfers are stopped for one time unit so as to empty the Burst-Buffer at rate  $B$ .

The occupation of the Burst-Buffer when using Algorithm 2 can be easily modeled by a Markov Chain [26]. Let us denote by  $Y_k, 0 \leq j \leq S$  the “normal” state of the chain when the size of Burst-Buffer is exactly  $j$  and let us define  $\sum_i b_i$  additional “overflow” states  $Y_j, S+1 \leq j \leq S + \sum_i b_i$  that correspond to the states where the capacity of the Burst-Buffer is exceeded.

Using DIST, we can compute the probability to move from State  $Y_j$  to State  $Y_l, \forall 0 \leq j, l \leq S + \sum_i b_i$ .

When in normal state  $Y_i, i \leq S$ , if  $\text{INSTANTLOAD} = k$ , then

- If  $i + k \leq B$ , then the Burst-Buffer becomes empty and we jump to state  $Y_0$ .
- If  $i + k - B \leq S$ , then we jump to normal state  $Y_{i+k-B}$ .
- If  $i + k - B > S$ , then we jump to overflow state  $Y_{i+k-B}$ .

When the chain is in an overflow state  $Y_i, i \geq S$ , then applications are kept idle for one time unit and the Burst-Buffer is emptied at maximal rate, so that we jump to (normal or overflow) state  $Y_{i-B}$ . Therefore, the probability  $P_{j,l}$  of transition between state  $Y_j$  and state  $Y_l$  is defined as follows:

**Algorithm 2:** ALLOCATETRANSFERS

---

**Input:** The set of active applications  $\mathcal{A}_{k_1}, \dots, \mathcal{A}_{k_l}$ ,  
the load of the Burst-Buffer LOAD

**Output:** The set of transfers to the disk and to the Burst-Buffer

```

1 if  $\sum_1^l b_{k_i} \leq B$  then
2   Transfer  $\sum_1^l b_{k_i}$  from the computing nodes to the disk ;
3   Transfer  $\max(\text{LOAD} - (B - \sum_1^l b_{k_i}), 0)$  from the Burst-Buffer to the
   disk ;
4 end
5 else
6   Transfer  $B$  from the computing nodes to the disk ;
7   if  $\text{LOAD} + \sum_1^l b_{k_i} - B \leq S$  then
8     Transfer  $\sum_1^l b_{k_i} - B$  from the computing nodes to the
     Burst-Buffer;
9   end
10  else
11    Transfer  $S - \text{LOAD}$  from the computing nodes to the Burst-Buffer;
12    Stop all applications until the transfer is over ;
13    During each time unit, transfer  $B$  from the computing nodes to
    the disk ;
14    Complete with Burst-Buffer data at the end ;
15  end
16 end

```

---

- If  $0 \leq j \leq S$  and  $1 \leq l \leq S + \sum_i b_i$ , then  $P_{j,l} = \text{DIST}(l - j + B)$ .
- If  $0 \leq j \leq S$ , then  $P_{j,0} = \sum_{k=0}^{B-j} \text{DIST}(k)$ .
- If  $S + 1 \leq j \leq S + \sum_i b_i$ , then  $P_{j,j-B} = 1$ .

**5.2 Steady State Load Estimation**

The Markov chain  $Y$  introduced in Section 5.1 can be used to estimate the idle time induced by Algorithm 2. Let us first prove with Theorem 2 that  $Y$  has a stationary probability  $\pi$ .

**Theorem 2.** *There exists a unique stationary distribution  $\pi$  for the Markov chain  $Y$ .*

*Proof.* Since  $p_i < 1$  for all  $i$ , we have  $\text{DIST}(0) > 0$ . This implies that each state  $Y_i$  has a non-zero probability to jump to state  $Y_{\max(i-B,0)}$ , and thus that state 0 is accessible from any state  $Y_i$ . The Markov chain thus has only one final class (the one containing state 0). This implies that  $Y$  has one and only one stationary distribution. Additionally, since  $P_{0,0} > 0$ , this class is aperiodic and

this distribution can be obtained as the limit of  $\pi^l = P^l \pi^0$ , independently of  $\pi^0$ .  $\square$

The size of the transition matrix  $P$  is  $S + \sum b_i \times S + \sum b_i$  and  $\pi$  is the solution of the linear system  $\pi = \pi * P$ . In all the experiments that follow, since we set  $B = 100$  in Section 4.2, the size of  $P$  is therefore of a few hundreds. Moreover, in all our experiments, we use an iterative algorithm  $\pi^l = \pi^{l-1} P$  that always converged in less than 10 iterations. Therefore, the computation of the stationary distribution is extremely fast in practice.

**Theorem 3.** *The idle time induced by Algorithm 2 is given by  $\sum_{k>S} \pi(k)$ .*

*Proof.*  $\pi(k)$  represents the probability of being in state  $Y_k$ . For  $k \leq S$ , the load of the Burst-Buffer is smaller than its size  $S$ , and no idle time is induced. On the other hand, states  $Y_k$ ,  $k > S$  correspond to an overflow of the buffer. In this case, Algorithm 2 stops all transfers from the applications with probability 1 in order to empty the Burst-Buffer for one time-unit. Therefore, the fraction of idle time corresponds to the probability of being in a state  $Y_k$ ,  $k > S$  and is therefore given by  $\sum_{k>S} \pi(k)$ .  $\square$

### 5.3 Idle Time Prediction

This section presents the results obtained with the Markov chain model presented above. We generate applications in the same way as in Section 4.3, this time with  $n = 20$  applications, and we observe the resulting stationary distribution. We fix the bandwidth  $B$  of the disk to 100, and we vary the expected load  $M$  of the system, by considering different values of  $\alpha = \frac{M}{B}$ . As a reminder, once  $M$  is fixed, we scale the  $b_i$  values so that  $\sum_i b_i p_i = M$ . Small values of  $\alpha$  result in an underloaded system in which the buffer does not fill very often. On the other hand, values of  $\alpha$  larger than one mean that idle times are required even in presence of a very large buffer in order to allow enough time to write to the disks.

Figure 3 shows the resulting proportion of idle time (see Theorem 3) for different values of  $p_{\max}$  and  $\alpha$ . We can see that as expected, the idle time decreases with the buffer size, converging to 0 for values of  $\alpha$  below 1, and to a positive value for  $\alpha$  above 1. As mentioned in Section 4.3, lower values of  $p_{\max}$  induce larger values of  $b_i$  and model shorter bursts of higher intensity. Figure 3 indeed shows that lower values of  $p_{\max}$  require larger buffer sizes to obtain low idle times.

## 6 Model Validation and Simulations

### 6.1 Comments on the probabilistic model

In this section, we aim at evaluating the influence of the application model described in Section 3 on the quality of the prediction of the idle time induced by Algorithm 2. Indeed, we have proved in Section 5 that this model is theoretically

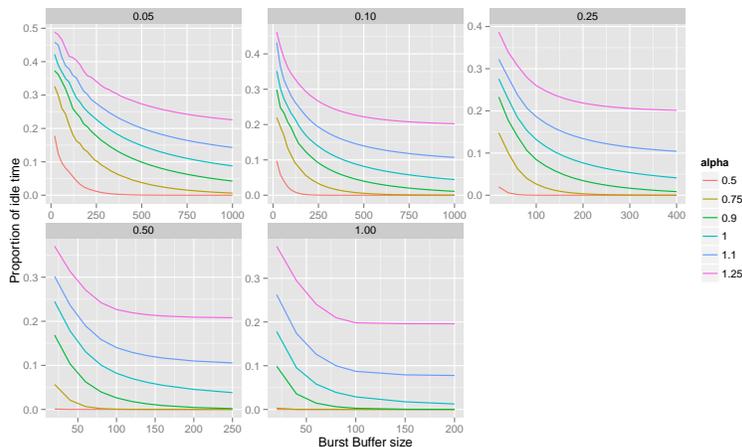


Fig. 3: Proportion of idle time as a function of buffer size, for different values of  $p_{\max}$  and  $\alpha$ .

and numerically tractable since a simple Markov chain enables to compute the idle time at low computational cost. Nevertheless, the model of Section 3 is not strictly equivalent to the models proposed in Section 2.3. Indeed, it is generally assumed that HPC applications follow a quasi periodic pattern alternating compute and communication phases and that the rate of emission is proportional to the number of I/O nodes and is therefore constant over time. Our model captures several crucial characteristics of the quasi periodic model, (i) alternating communication and computation phases, (ii) the quasi periodic pattern modeled by  $p_i$  probability of I/O communication (iii) the constant bandwidth  $b_i$  used by a given application  $\mathcal{A}_i$  when performing I/Os.

Nevertheless, several important characteristics are missing. In particular, in our model, all applications share a common time unit, and there is no correlation (no memory) between what happens at time step  $t$  and  $t + 1$ . This assumption is of course crucial to build a Markov chain model. Nevertheless, if the period of the pattern for  $\mathcal{A}_i$  is much longer than the time unit, the fact that  $\mathcal{A}_i$  is involved in a communication at time  $t$  strongly influences the probability that it is involved in same communication at time  $t + 1$ . On the other hand, if the period of the pattern for  $\mathcal{A}_j$  is much shorter than the time unit, then the I/O bandwidth consumed during one time unit with our model is underestimated (if  $X_j = 0$ ) or overestimated (if  $X_j = 1$ ) since during one time unit,  $\mathcal{A}_j$  would indeed perform several communication and computation phases.

It is therefore crucial to validate our probabilistic model against a more realistic (but theoretically intractable) model using a discrete event simulator that will be presented in Section 6.2.

## 6.2 Discrete Event Simulator

In order to validate the ability of the application model presented in Section 3 to predict the idle time, we build a discrete event simulator. The platform model is the same as the one introduced in [10] and described in Figure 1, and is characterized by the I/O bandwidth  $B$  between the processing nodes and the disks, that is shared by the transfers between the Burst-Buffer and the disk, so that the overall bandwidth to the disk is bounded by  $B$ .

On the application side, simulations rely on a more sophisticated model. Each application is characterized by a time period  $d_i$ , a probability of sending I/Os  $p_i$  and a used bandwidth  $b_i$ , and we add another parameter  $u$  that will be used to add noise to the pattern of  $\mathcal{A}_i$ . More precisely, a pattern for  $\mathcal{A}_i$  is built as follows

- $\mathcal{A}_i$  computes during  $(1 - p_i)d_i x$  where  $x$  is chosen uniformly at random in  $[1 - u, 1 + u]$  at each computing burst,
- $\mathcal{A}_i$  sends data at rate  $b_i$  during  $(1 - p_i)d_i x$  where  $x$  is chosen uniformly at random in  $[1 - u, 1 + u]$  at each burst of I/Os.

Using above model, we generate a pattern that is almost periodic (exactly periodic if  $u = 0$ ), and the quasi-period of  $\mathcal{A}_i$  is different for each application, so that the model is close to what has been observed in HPC systems (see Section 2.3 and references therein).

Our discrete event simulator is described in Algorithm 3.  $comp_i$  is a boolean value that indicates whether the next burst for  $\mathcal{A}_i$  is a I/O ( $comp_i = 0$ ) or a compute ( $comp_i = 1$ ) burst.

## 6.3 Model Validation

In order to compare the results obtained with the Markov chain model to those of the discrete event simulator, we once again generate applications for different values of  $p_{\max}$  and  $\alpha$  as was done in Section 5.3. In addition to  $p_i$  and  $b_i$  values, we also generate the values  $d_i$  uniformly at random between some value  $d_{\min}$  and  $2d_{\min}$ . This value  $d_{\min}$  influences the characteristic time of applications. Since we want our applications' I/O bursts have a duration close to the time-unit of the Markov model, we make  $d_{\min}$  depend on  $\frac{1}{p_{\max}}$ , which is the order of magnitude of their I/O duration. The results are depicted on Figure 4. For each scenario, we run Algorithm 3 15 times up to  $T_{\max} = 1000$ , and we show on the plots the 95% confidence intervals for the mean idle time with a filled ribbon. We can see that models and simulations are a close fit, provided that the correct value of  $d_{\min}$  is used.

**Algorithm 3:** Discrete Event Simulator

---

**Input:** The set of active applications  $\mathcal{A}_{k_1}, \dots, \mathcal{A}_{k_l}$ ,  
with their characteristics  $p_i, b_i$  and  $d_i$  and their random starting times  $t_i$   
a large time bound  $T_{\max}$ , the regularity of bursts length  $u$

**Output:** The total idle time IDLETIME

```

1  $\forall i, comp_i = 1$  NEXTEVENT =  $\min(t_1, \dots, t_n)$  ;
2 while NEXTEVENT  $\leq T_{\max}$  do
3    $i = \operatorname{argmin}(t_1, \dots, t_n)$  ;
4   Choose  $x$  uniformly at random in  $[1 - u, 1 + u]$  ;
5   if  $comp_i == 0$  then
6      $t_i = \text{NEXTEVENT} + (1 - p_i)d_i x$ ;
7      $comp_i = 1 - comp_i$ ;
8   end
9   else
10     $t_i = \text{NEXTEVENT} + p_i d_i x$ ;
11     $comp_i = 1 - comp_i$ ;
12  end
13  Update Burst-Buffer occupation at time  $t_i$ 
14  while Burst-Buffer occupation at time  $t_i \geq S$  do
15    Stop all transfers for one time unit;
16    Increase IDLETIME;
17  end
18 end
19 return IDLETIME

```

---

## 7 Lazy Emptying

### 7.1 Algorithm

To prevent from the phenomena of *Write Amplification* [27], Han et al. [15] consider lazy strategies for emptying the Burst-Buffer. In particular, they propose to empty the Burst-Buffer only when its load reaches a certain level. The rationale behind this is that performing a transfer between the Burst-Buffer and the disk has a fixed cost that does not only depend on the transferred data. Therefore, there is a clear interest in keeping the Burst-Buffer to disk link quiet as often as possible. Algorithm 4 is based on this idea. If the load of the Burst-Buffer is smaller than a given  $\text{THRESHOLD} < S$ , the Burst-Buffer is not emptied, even if some bandwidth remains available once the transfers between the computing nodes and the disk have been allocated.

In this context, we have two conflicting objectives. Indeed, we are interested in both maximizing the quiet time of the Burst-Buffer (denoted by QUIET) and to minimize the idle time induced by Burst-Buffer overflow (denoted by IDLETIME). Clearly, a high value of THRESHOLD is expected to make QUIET larger (since we keep the lazy strategy longer), but also to make IDLETIME

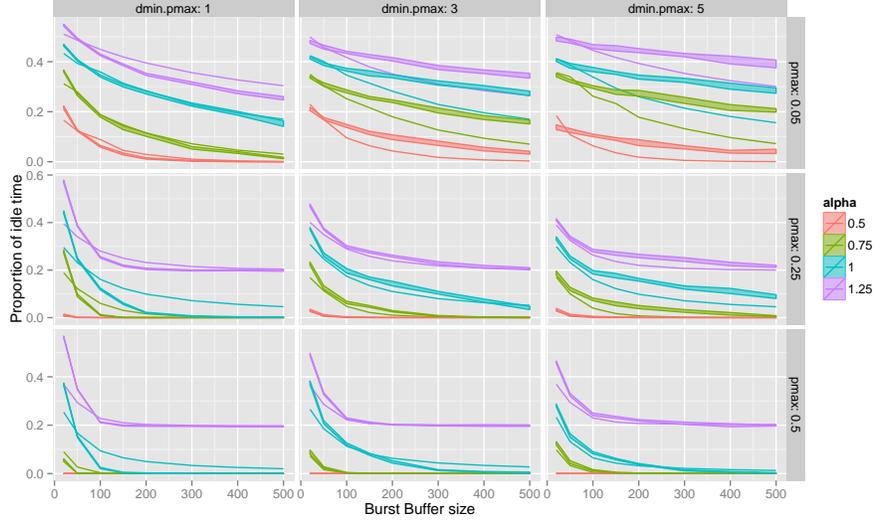


Fig. 4: Comparison of predicted idle times between Markov chain and discrete event simulator

larger (since we take more risks by not emptying the buffer). The details of the algorithm are presented in Algorithm 4. In order to enforce the properties of the associated Markov chain (see Section 7.2 and Theorem 4), we keep a (very small probability) of emptying the Burst-Buffer, even if  $\text{THRESHOLD} < S$ .

## 7.2 Markov Chain associated to Algorithm 4

Similarly to what has been proposed in 5.1, it is easy to associate a Markov chain to Algorithm 4, where state  $Y_j^{\text{LAZY}}$  corresponds to a load  $j$  of the Burst-Buffer and whose transition matrix  $P^{\text{LAZY}}$  is defined as follows

- If  $\text{THRESHOLD} \leq j \leq S$  and  $1 \leq l \leq S + \sum_i b_i$ , then  $P_{j,l}^{\text{LAZY}} = \text{DIST}(l - j + B)$ .
- If  $0 \leq j < \text{THRESHOLD}$ :
  - if  $1 \leq l < j$  then  $P_{j,l}^{\text{LAZY}} = 0.01 \times \text{DIST}(l - j + B)$ ;
  - $P_{j,j} = 0.99 \times \sum_{l=1}^{j-1} \text{DIST}(l - j + B) + \text{DIST}(B)$ ;
  - if  $j < l \leq S + \sum_i b_i$ , then  $P_{j,l}^{\text{LAZY}} = \text{DIST}(l - j + B)$ .
- If  $S + 1 \leq j \leq S + \sum_i b_i$ , then  $P_{j,j-B}^{\text{LAZY}} = 1$ .

With above definition, we can easily prove Theorem 4 and Theorem 5.

**Algorithm 4:** Lazy ALLOCATETRANSFERS

---

**Input:** The set of active applications  $\mathcal{A}_{k_1}, \dots, \mathcal{A}_{k_l}$ ,  
the load of the Burst-Buffer LOAD

**Output:** The transfers to disk and to the Burst-Buffer

```

1 if  $\sum_1^l b_{k_i} \leq B$  then
2   Transfer  $\sum_1^l b_{k_i}$  from the applications to the disk ;
3   if LOAD  $\geq$  THRESHOLD then
4     Transfer  $\max(\text{LOAD} - (B - \sum_1^l b_{k_i}), 0)$  from the Burst-Buffer to
     the disk ;
5   end
6   else
7     With probability 0.01, transfer  $\max(\text{LOAD} - (B - \sum_1^l b_{k_i}), 0)$  from
     the Burst-Buffer to the disk ;
8   end
9 end
10 else
11   Transfer  $B$  from the computing nodes to the disk ;
12   if  $\text{LOAD} + \sum_1^l b_{k_i} - B \leq S$  then
13     Transfer  $\sum_1^l b_{k_i} - B$  from the computing nodes to the
     Burst-Buffer;
14   end
15   else
16     Transfer  $S - \text{LOAD}$  from the computing nodes to the Burst-Buffer;
17     Stop all applications until the transfer is over ;
18     During each time unit, transfer  $B$  from the computing nodes to
     the disk ;
19     Complete with Burst-Buffer data at the end ;
20   end
21 end

```

---

**Theorem 4.** *There exists a unique stationary distribution  $\pi^{\text{LAZY}}$  for the Markov chain  $Y^{\text{LAZY}}$ .*

*Proof.* The difference between  $Y$  and  $Y_j^{\text{LAZY}}$  lies in the weights only, and not in the structure of the transitions. Therefore, the proof of Theorem 2 directly applies to Theorem 4 if we define  $\pi^{\text{LAZY}}$  as the solution of the linear system  $\pi^{\text{LAZY}} = \pi^{\text{LAZY}} \times P^{\text{LAZY}}$ .  $\square$

**Theorem 5.** *The idle time induced by Algorithm 4 is given by  $\text{IDLETIME} = \sum_{k>S} \pi^{\text{LAZY}}(k)$  and the quiet time induced by Algorithm 4 is given by  $\text{QUIET} = \sum_{k=0}^S \left( \pi^{\text{LAZY}}(k) \times \sum_{l \geq k} P^{\text{LAZY}}(k, l) \right)$ .*

*Proof.* The states  $Y_k^{\text{LAZY}}$  with  $k > S$  correspond to overflow states and systematically induce one time unit of idle time. To estimate QUIET, we sum, for all

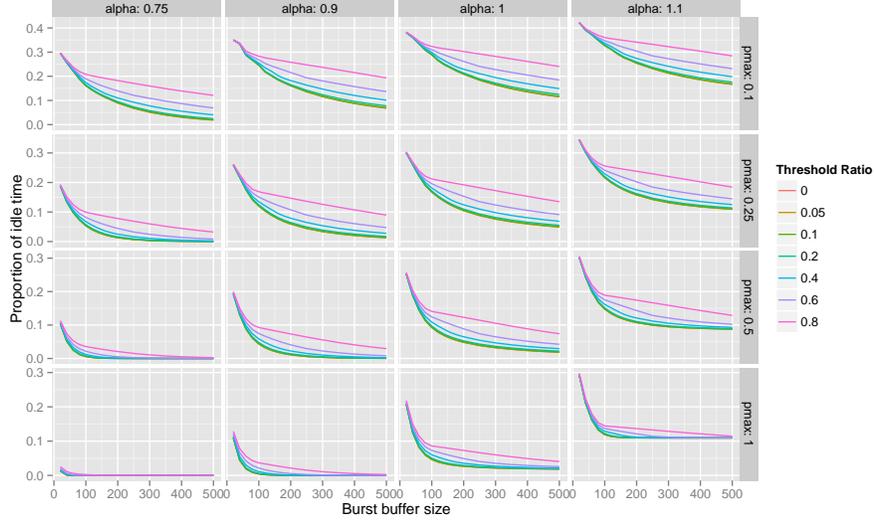


Fig. 5: IDLETIME times for different THRESHOLD values

possible “normal” states  $Y_k^{\text{LAZY}}$ ,  $k \leq S$ , the probability to not decrease the load of the Burst-Buffer, *i.e.* to jump to a state  $Y_l^{\text{LAZY}}$ ,  $l \geq k$ .  $\square$

### 7.3 IdleTime and Quiet times induced by Algorithm 4

To study the effect of the threshold value on the performance of the Burst-Buffer, we analyze the results obtained using the Markov chain model. We generate  $n = 20$  applications like previously, and we consider varying values for  $\alpha$ ,  $p_{\max}$ , and for the threshold ratio (*i.e.*, the ratio between the THRESHOLD value and the buffer size  $S$ ). A threshold ratio of 0 is equivalent to the basic case, and a threshold value of 1 means that the burst buffer is only emptied when it is full. Results for IDLETIME time are shown in Figure 5. In addition, we also measure QUIET time during which the burst buffer is not emptied (*i.e.*, its occupation either increases or stays the same). This is shown on Figure 6.

We can see that in most cases, even a small positive value of THRESHOLD allows to improve the QUIET, except when the load is high and  $p_{\max}$  is also high. In those cases, the Burst-Buffer is almost continuously kept full due to the constant arrival of application data, and both strategies behave similarly. In all other cases, the buffer has a non-zero probability of having a small enough load for the threshold to be useful. In many cases however, a too high value of THRESHOLD (above 20 or 40% of the Burst-Buffer size) induces significant cost on IDLETIME time. Since even a small THRESHOLD brings a benefit on the QUIET times (and for low load, increasing THRESHOLD further does not degrade IDLETIME time), we can see that this lazy strategy indeed has the

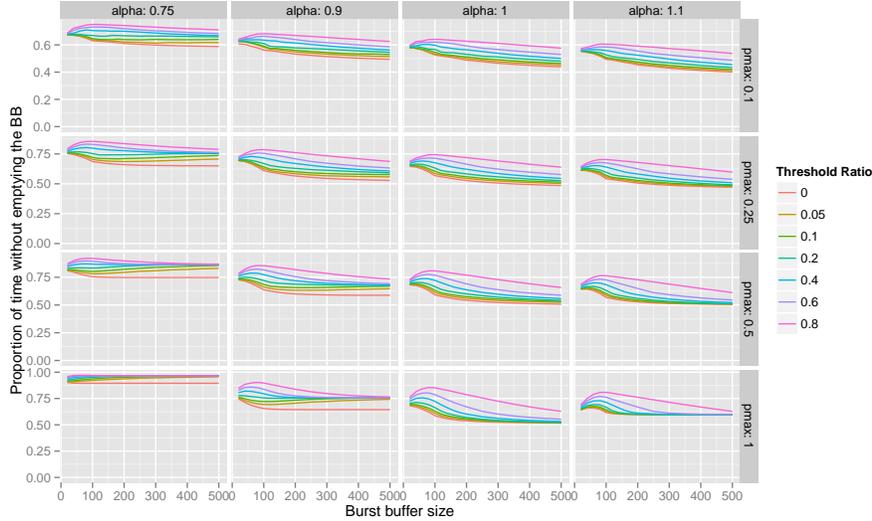


Fig. 6: QUIET times for different THRESHOLD values

potential to increase the performance of the Burst-Buffer. It is important to correctly dimension the THRESHOLD value, however a 20% value appears to be a good safe guess.

## 8 Conclusion

In this paper, we propose a probabilistic model for the use of Burst-Buffers in the context of organizing the transfers of a set of HPC quasi-periodic applications. We prove that this model enables to estimate at low computational cost the load of the Burst-Buffer under several classical management strategies. We also show by comparison with a discrete event simulator based on a more complex application model that despite its simplicity, the model is able to capture the main characteristics of the HPC load. Present work opens several perspectives. First, it would be of great interest to include in the probabilistic model the specific characteristics of BigData workload (*i.e.* the ability to prefetch data) and checkpointing strategies (*i.e.* the ability of removing old checkpoints without writing them eventually to the disk), that can improve the performance of the Burst-Buffer. Then, this work makes it possible to evaluate complex Burst-Buffer management strategies where the Burst-Buffer is used differently by the different applications. At last, extending this model to more distributed Burst-Buffer architectures is also an interesting perspective.

## References

- [1] “The trinity project,” <http://www.lanl.gov/projects/trinity/>, online; accessed 21 October 2017.
- [2] “I/o at argonne national laboratory,” [https://wr.informatik.uni-hamburg.de/\\_media/events/2015/2015-iode-argonne-isaila.pdf](https://wr.informatik.uni-hamburg.de/_media/events/2015/2015-iode-argonne-isaila.pdf), online; accessed 21 October 2017.
- [3] D. A. Reed and J. Dongarra, “Exascale computing and big data,” *Communications of the ACM*, vol. 58, no. 7, pp. 56–68, 2015.
- [4] F. Cappello, A. Geist, W. Gropp, S. Kale, B. Kramer, and M. Snir, “Toward exascale resilience: 2014 update,” *Supercomputing frontiers and innovations*, vol. 1, no. 1, pp. 5–28, 2014.
- [5] G. Bosilca, A. Bouteiller, E. Brunet, F. Cappello, J. Dongarra, A. Guermouche, T. Herault, Y. Robert, F. Vivien, and D. Zaidouni, “Unified model for assessing checkpointing protocols at extreme-scale,” *Concurrency and Computation: Practice and Experience*, vol. 26, no. 17, pp. 2772–2791, 2014.
- [6] O. Yildiz, M. Dorier, S. Ibrahim, R. Ross, and G. Antoniu, “On the root causes of cross-application i/o interference in hpc storage systems,” in *IPDPS’16*. IEEE, 2016, pp. 750–759.
- [7] DDN Storage, “BURST BUFFER & BEYOND, I/O & Application Acceleration Technology,” [https://www.ddn.com/download/resource\\_library/brochures/technology/IME\\_FAQ.pdf](https://www.ddn.com/download/resource_library/brochures/technology/IME_FAQ.pdf), 2014, online; accessed 20 October 2017.
- [8] D. Henseler, B. Landsteiner, D. Petesch, C. Wright, and N. J. Wright, “Architecture and design of cray datawarp,” *Cray User Group CUG*, 2016.
- [9] C. S. Daley, D. Ghoshal, G. K. Lockwood, S. S. Dosanjh, L. Ramakrishnan, and N. J. Wright, “Performance characterization of scientific workflows for the optimal use of burst buffers.” in *WORKS@ SC*, 2016, pp. 69–73.
- [10] W. Schenck, S. El Sayed, M. Foszczynski, W. Homberg, and D. Pleiter, “Evaluation and performance modeling of a burst buffer solution,” *ACM SIGOPS Operating Systems Review*, vol. 50, no. 1, pp. 12–26, 2017.
- [11] M. Mubarak, P. Carns, J. Jenkins, J. K. Li, N. Jain, S. Snyder, R. Ross, C. D. Carothers, A. Bhatle, and K.-L. Ma, “Quantifying i/o and communication traffic interference on dragonfly networks equipped with burst buffers,” in *Cluster Computing (CLUSTER), 2017 IEEE International Conference on*. IEEE, 2017, pp. 204–215.

- 
- [12] D. Kimpe, K. Mohror, A. Moody, B. Van Essen, M. Gokhale, R. Ross, and B. R. De Supinski, “Integrated in-system storage architecture for high performance computing,” in *Proceedings of the 2nd International Workshop on Runtime and Operating Systems for Supercomputers*. ACM, 2012, p. 4.
  - [13] L. Bautista-Gomez, S. Tsuboi, D. Komatitsch, F. Cappello, N. Maruyama, and S. Matsuoka, “Fti: high performance fault tolerance interface for hybrid systems,” in *Proceedings of 2011 international conference for high performance computing, networking, storage and analysis*. ACM, 2011, p. 32.
  - [14] L. Cao, B. W. Settlemyer, and J. Bent, “To share or not to share: comparing burst buffer architectures,” in *Proceedings of the 25th High Performance Computing Symposium*. Society for Computer Simulation International, 2017, p. 4.
  - [15] J. Han, D. Koo, G. K. Lockwood, J. Lee, H. Eom, and S. Hwang, “Accelerating a burst buffer via user-level i/o isolation,” in *Cluster Computing (CLUSTER), 2017 IEEE International Conference on*. IEEE, 2017, pp. 245–255.
  - [16] N. Liu, J. Cope, P. Carns, C. Carothers, R. Ross, G. Grider, A. Crume, and C. Maltzahn, “On the role of burst buffers in leadership-class storage systems,” in *Mass Storage Systems and Technologies (MSST), 2012 IEEE 28th Symposium on*. IEEE, 2012, pp. 1–11.
  - [17] S. Herbein, D. H. Ahn, D. Lipari, T. R. Scogland, M. Stearman, M. Grondona, J. Garlick, B. Springmeyer, and M. Taufer, “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*. ACM, 2016, pp. 69–80.
  - [18] Carns *et al.*, “24/7 characterization of petascale I/O workloads,” in *Proceedings of CLUSTER09*. IEEE, 2009, pp. 1–10.
  - [19] G. Aupy, A. Gainaru, and V. L. Fèvre, “Periodic i/o scheduling for super-computers,” in *International Workshop on Performance Modeling, Benchmarking and Simulation of High Performance Computer Systems*. Springer, Cham, 2017.
  - [20] W. Hu, G.-m. Liu, Q. Li, Y.-h. Jiang, and G.-l. Cai, “Storage wall for exascale supercomputing,” *Journal of Zhejiang University-SCIENCE*, vol. 2016, pp. 10–25, 2016.
  - [21] X. Yuan, S. Mahapatra, W. Nienaber, S. Pakin, and M. Lang, “A new routing scheme for jellyfish and its performance with hpc workloads,” in *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*. ACM, 2013, p. 36.

- 
- [22] F. Chung and L. Lu, “Connected components in random graphs with given expected degree sequences,” *Annals of combinatorics*, vol. 6, no. 2, pp. 125–145, 2002.
  - [23] —, “Concentration inequalities and martingale inequalities: a survey,” *Internet Mathematics*, vol. 3, no. 1, pp. 79–127, 2006.
  - [24] M. Habib, C. McDiarmid, J. Ramirez-Alfonsin, and B. Reed, *Probabilistic methods for algorithmic discrete mathematics*. Springer Science & Business Media, 2013, vol. 16.
  - [25] B. Vöcking, “Selfish load balancing,” *Algorithmic game theory*, vol. 20, pp. 517–542, 2007.
  - [26] W. J. Stewart, *Introduction to the numerical solutions of Markov chains*. Princeton Univ. Press, 1994.
  - [27] X.-Y. Hu, E. Eleftheriou, R. Haas, I. Iliadis, and R. Pletka, “Write amplification analysis in flash-based solid state drives,” in *Proceedings of SYSTOR 2009: The Israeli Experimental Systems Conference*. ACM, 2009, p. 10.