electronics

MDPI

Article

Adaptively Periodic I/O Scheduling for Concurrent HPC Applications

Benbo Zha and Hong Shen *

School of Computer Science and Engineering, Sun Yat-sen University, Guangzhou 510006, China; zhabb@mail2.sysu.edu.cn
* Correspondence: shenh3@mail.sysu.edu.cn

Abstract: With the convergence of big data and HPC (high-performance computing), various machine learning applications and traditional large-scale simulations with a stochastically iterative I/O periodicity are running concurrently on HPC platforms, which poses more challenges on the scarcely shared I/O resources due to the ever-growing data transfer demand. Currently the existing heuristic online and periodic offline I/O scheduling methods for traditional HPC applications with a fixed I/O periodicity are not suitable for the applications with stochastically iterative I/O periodicities, which are required to schedule the concurrent I/Os from different applications under I/O congestion. In this work, we propose an adaptively periodic I/O scheduling (APIO) method that optimizes the system efficiency and application dilation by taking the stochastically iterative I/O periodicity of the applications into account. We first build a periodic offline scheduling method within a specified duration to capture the iterative nature. After that, APIO adjusts the bandwidth allocation to resist stochasticity based on the actual length of the computing phrase. In the case where the specified duration does not satisfy the actual running requirements, the period length will be extended to adapt to the actual duration. Theoretical analysis and extensive simulations demonstrate the efficiency of our proposed I/O scheduling method over the existing online approach.

Keywords: I/O scheduling; periodic I/O scheduling; stochastic iterative application; high-performance computing

check for updates

Citation: Zha, B.; Shen, H. Adaptively Periodic I/O Scheduling for Concurrent HPC Applications. *Electronics* **2022**, *11*, 1318. https://doi.org/10.3390/electronics11091318

Academic Editor: David Defour

Received: 9 March 2022
Accepted: 19 April 2022
Published: 21 April 2022

Publisher's Note: MDPI stays neutral with regard to jurisdictional claims in published maps and institutional affiliations.

Copyright: © 2022 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (https://creativecommons.org/licenses/by/4.0/).

1. Introduction

High-performance computing (HPC) systems, especially supercomputers, play an unprecedentedly important role in modern scientific discovery, thanks to their enormous computing power and storage capacity. Large-scale numerical simulations from different fields, such as meteorology, aerospace, bio-pharmacy, and high-energy physics, are helping scientists to accelerate the progress of research and to save money by eliminating the need for real experiments [1]. With the era of the exascale supercomputer coming, more large-scale modeling, simulations, and other applications will be deployed and bring more challenges. I/O bottleneck is one of the most severe problems on HPC platforms.

Although computing power has increased dramatically, system I/O throughput cannot expand synchronously due to storage technology developments [2]. Larger-scale applications deployed on HPC will produce greater data transferring demands on the scarce I/O resource. Under the convergence trend of big data and HPC [3], certain big data applications have higher data requirements on the parallel file system (PFS). In addition, fault-tolerance technologies, such as checkpointing/restart, which are designed to resist the decreasing Mean Time between Failures (MTBF) also exacerbate I/O contention [4]. In order to meet these practical demands, data transferring and management must be more efficient.

Many studies have been conducted to mitigate the I/O bottleneck problem. In terms of system architecture, there are topology-aware methods [5], memory hierarchy-aware methods [6–8], burst-buffering methods [4,9,10], and so on. From the aspect of applications, many approaches, such as application coordinating [11,12], I/O scheduling [13–17], and data layouting [18,19], are proposed. The I/O scheduling method, which allocates I/O bandwidth to each application in order to optimize system utilization and applications efficiency, is widely used in HPC.

Nevertheless, the existing I/O scheduling approaches largely focus on traditional HPC applications that usually have a fixed I/O periocity. With the convergence of big data and HPC, many machine learning (ML)-based applications deployed on HPC exhibit a stochastically iterative I/O periodicity, the executions of which depend on specific input data to run in an iterative way to approximate an acceptable solution [20,21], such as the structural identification of orbital anatomy application [22]. Some scientific data analytic applications also present stochasticity, such as functional MRI quality assurance (fMRIQA) [22]. Furthermore, many traditional scientific applications based on solving large sparse linear systems with iterative methods, such as the randomized Kaczmraz method, also possess these properties [23]. The existing methods are either unable to fully exploit the characteristics of applications, such as online scheduling [13], or not suitable for applications with a stochastically iterative I/O periodicity, such as periodic I/O scheduling [14,15,24]. To simplify the expression, we refer to the application with a stochastically iterative I/O periodicity as a stochastic iterative application hereafter.

In order to utilize the stochastically iterative I/O periodicity of these emerging applications, we proposed an adaptively periodic I/O scheduling (APIO) to optimize application efficiency and system utilization. It first conducts a periodic scheduling to utilize the periodicity given the specified probabilistic distribution of applications, which allocates different specified bandwidths within different durations for each instance of applications in a period. In each period it then fine-tunes the allocation of bandwidth to resist the stochasticity of applications in run time. When the specified number of instances for some applications can not be scheduled within a period, it extends the period to adapt to the actual duration. Our proposed algorithm inherits the advantages of periodic I/O scheduling and adapts it for scheduling a wide range of HPC applications that have a stochastically itetative I/O periodicity.

The main contributions of this work include as follows:

- We propose an adaptively periodic I/O scheduling algorithm, which combines the advantages of periodic scheduling and online scheduling to leverage the iterativeness and stochasticity of the ever-growing stochastic iterative applications on HPC;
- We perform a theoretical analysis of the efficiency of the proposed scheduling;
- We conduct simulations to show the efficiency and effectiveness of our proposed method compared to the existing online scheduling.

The rest of this paper is organized as follows. Section 2 describes the related works on stochastic iterative applications and I/O scheduling. Section 3 introduces the models on platform and application, the I/O scheduling problems, and the existing I/O scheduling algorithms. In Section 4, the proposed adaptively periodic I/O scheduling algorithm is presented, and the related analysis on the efficiency is also shown. Section 5 shows the simulation experiments and Section 6 concludes this work.

2. Related Works

The enormous data-transferring requirements from a variety of applications pose a huge challenge for HPC storage systems, especially the ones with bandwidth-limited PFS. Several research studies have been conducted to study how to use such systems efficiently in different scenarios. In this work, the focus is on scheduling I/Os from stochastic iterative applications that share the aggregated bandwidth of PFS concurrently. Therefore, we discuss the three closest parts in this section.

2.1. Stochastic Iterative Applications

With the computing capacity of HPC systems rapidly increasing, there are a variety of applications originating from a wide range of fields that involve a lot of computation and large amounts of data transfer, and whose execution takes a lot of time (hours, and even days), deployed on such HPC platforms. Due to fault tolerance or visualization, these applications often store the intermediate results regularly into the persistent storage and then show the periodicity [14,25]. This periodicity might cause I/O bursts and then worsen the I/O bottleneck problem when many applications access the underlying PFS concurrently. An architecture solution for mitigating this I/O congestion is burst-buffering, which is widely discussed in the literature [4,9]. In addition, the applications running on HPC often show stochasticity, in which their execution time depends on the input data [26].

In our work, we define the stochastic iterative application as the application with a stochastically iterative I/O periodicity. The application executes I/O operations iteratively, but there is a random interval between two I/O operations to complete computing. The iterative I/O periodicity has many reasons, such as the iterative computing way and check-pointing/restart. The stochasticity of the computing phase comes from data characteristics, non-stationary iterative methods, and so forth.

The reasons why stochastic iterative applications are becoming more common mainly include the following points: First, the trend on the convergence of big data and HPC appeals to many ML-based applications to be deployed, which achieve an acceptable solution by the stochastic iterative algorithm [20,21]. The structural identification of an orbital anatomy application is such an ML-based data analysis example [22]. Second, some scientific data analytic applications, such as functional MRI quality assurance (fMRIQA) [22], show stochasticity and they execute on different instances iteratively. Third, many traditional scientific applications based on solving large sparse linear systems with popular iterative methods, such as the randomized Kaczmraz method, also possess stochasticity [23].

2.2. I/O Scheduling

Through controlling the execution procedure of I/O requests, I/O scheduling can be applied to many data-transferring scenarios to mitigate I/O-related problems. In terms of HPC, it schedules the I/O requests from different applications to access the underlying persistent storage. It can be implemented on different storage layers for different purposes [27]. For application-side optimizations, Liao et al. [28] proposed a dynamic file-domain-partitioning method according to the locking protocol of PFS to optimize the parallel I/O of one application. For server-side methods [29], Song et al. [30] presented a server-side I/O coordination for PFS to reduce the interference of different applications. For interaction between multiple layers [7,8], data compression and smart data movement are designed. In this work, we study coordinating I/O requests from many stochastic iterative applications on the I/O nodes.

I/O scheduling deployed on I/O nodes can utilize the data location information to optimize data access. In reference [31], the proposed IOrchestrator reorganizes the I/O requests by considering the data spatial locality. In reference [5], Tessier et al. provide a topology-aware data aggregation method to minimize the data conflict on the computing network. In reference [19], a randomness detection method, SSDup, is designed to improve the data transferring. In reference [18], a contention-aware scheduling is presented to balance the workload on each SSD server. In addition, this kind of I/O scheduling can obtain global application information and can easily integrate it into job scheduling to coordinate multiple applications. In order to resist the effects of I/O interference, Dorier et al. [12] propose a coordinating scheduling for two applications, CALCioM. In reference [32], Carretero et al. provided a bandwidth-aware mapping algorithm to consider job and I/O scheduling simultaneously.

The closest study to this work is the offline periodic scheduling proposed by Aupy et al. [14], which constructs a period to consider the periodicity of HPC applications. It achieves better performance on system efficiency and application dilation than another general online scheduling method [13]. In our prior study [17], we proposed a Markov-chain-based I/O scheduling, which improves the online scheduling by considering the state of burst-buffers. This type of I/O scheduling has wide applicability for applications on HPC.

2.3. Stochastic Scheduling

In order to consider the stochasticity of jobs, many stochastic job scheduling methods had been proposed in the book by Pinedo [33]. A speculative scheduling method proposed in reference [26] provides a solution for stochastic HPC applications in a reservation-based scheduling contexts, building a speculative reservation sequence to rerun the job when the prior reservation is unsatisfactory. In reference [22], Gainaru et al. continuously optimize the speculative scheduling by checkpointing the completed work.

For stochastic iterative applications, Du et al. [23] verified the robustness of the periodic checkpointing, which essentially is an I/O scheduling case to deal with stochasticity. In reference [34], the authors construct the optimal checkpointing strategies to decide which iteration performs checkpointing. These works place the research object on the stochastic iterative applications. This work for scheduling the I/O of such stochastic iterative applications is motivated by them.

3. Preliminaries and Motivations

In this section, we first describe the HPC platform model and the application execution model. Then, the I/O scheduling problem was formulated. Finally, we introduce the existing online and offline methods related to this work, and then describe the motivations.

3.1. Platform and Application Execution Model

3.1.1. HPC Platform Model

The HPC platform consists of lots of computing nodes and storage nodes to satisfy the requirements of large-scale scientific applications. The computing nodes are identical in terms of the computing capacity and the local bandwidth in common. A job scheduler assigns these computing resources to the applications in batch, and then each application has its own exclusive computing nodes.

We depict the platform model assumed for this work in Figure 1. There are many applications running on the platform concurrently and sharing the underlying PFS through I/O nodes (ION). The computation for each application is isolated on the specified computing nodes, but the I/O operations contend the shared I/O bandwidth of PFS, *B*. When the total I/O bandwidth requirement exceeds the aggregated bandwidth *B*, some applications have to be delayed, which is decided by the I/O scheduler.

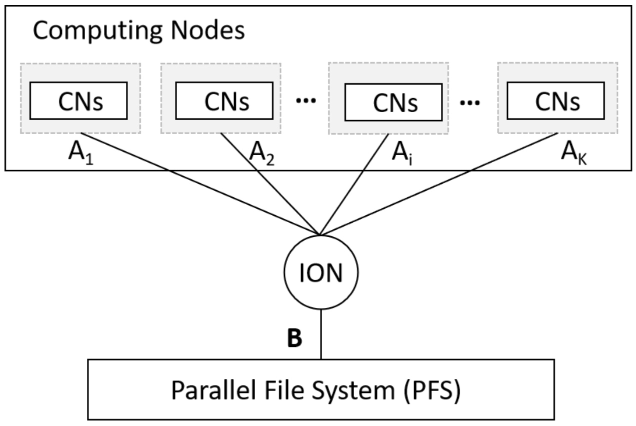


Figure 1. Schematic model of HPC platform.

3.1.2. Application Execution Model

In our considered execution model, there are *K* applications with stochastically iterative I/O periodicities running concurrently on the HPC platform illustrated above. Applications execute alternatively between the computing phrase and I/O phrase. The combination of a computing phrase and an I/O phrase refers to an instance. Each application consists of N_i such instances (because the applications running on HPC platform often last very long, so here we assume the number of instances is enough big to achieve a periodic scheduling such as in reference [14]). Unlike periodic scheduling [14], the length of the computing phrase in each instance is stochastic rather than fixed, which follows a distribution, $\mathcal{D}$, and the length of the I/O phrase is fixed, since the data structure of the intermediate results is designed as fixed in advance.

To clarify the execution procedure, an example for three stochastic iterative applications is illustrated in Figure 2. Three applications, A_1, A_2, and A_3, have their own execution characteristics. Each application A_i has N_i instances that have different computation lengths W_i^k and the same I/O volume IO_i. Due to the limitations of the platform bandwidth and the periodicity of the applications [25], I/O congestion might be happening during the execution. If the I/O execution procedure is disordered under the best-effort strategy, the caused I/O congestion would have dramatically degraded the I/O performance for the Write Amplification of SSD (solid-state drive) [13]. The aim of I/O scheduling is to ensure the order of I/O execution by arranging specific bandwidth for each application.

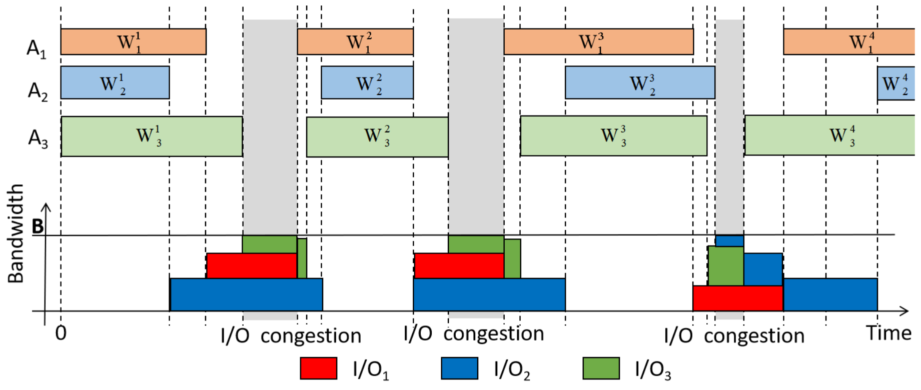


Figure 2. Execution example for three stochastic iterative applications.

Meanwhile, for each application A_i, it runs on β_i computing nodes, which are specified by the HPC batch job scheduler. The local bandwidth of a computing node is b. Thus, the maximum rate to transfer data for A_i is $B_i = min(B, \beta_i \cdot b)$. However, the real I/O rate of A_i at time t is the minimum between B_i and the remains of the PFS bandwidth, i.e., $b_i(t) = min\left(B_i, B - \sum_{i \neq j} b_j(t)\right)$.

3.2. Problem Description

The objectives of I/O scheduling are to achieve the maximum system efficiency and the minimum application dilation, the same as in the work in reference [13]. We define the application efficiency first for each application A_i at time t.

$$\tilde{\rho}_i(t) = \frac{\sum_{i \le n_i(t)} W_i}{t - r_i},$$

where $n_i(t) \le N_i$ is the number of instances of A_i that have been executed at time t, r_i is the release time of A_i. The optimal application efficiency ρ_i can be obtained in the dedicated mode: $\rho_i = \frac{\sum_{k \le N_i} W_i^k}{\sum_{k \le N_i}(W_i^k + \frac{IO_i}{B_i})}$ and $\tilde{\rho}_i(t) \le \rho_i$.

The system efficiency refers to the total performance of all processors in the platform. Additionally, the application dilation refers to the largest slowdown among all applications.

Therefore, we formulate two problems on these two objectives of I/O scheduling as follows:

Problem 1 (**MaxSysEfficiency**): Given K stochastic iterative applications, $A_i(\beta_i, N_i, r_i, W_i^k, IO_i)$, and a HPC platform that has a B PFS aggregate bandwidth and N computing nodes with

b local node bandwidth, find the I/O bandwidth assignment $b_i(t)$ for each application to maximize the total platform performance.

$$\begin{aligned} \max \quad & \frac{1}{N}\sum_{i=1}^{K}\beta_i\tilde{\rho}_i(d_i) \\ \text{s.t.} \quad & b_i(s_i^k+W_i^k)\le B_i,\ k\in\{1,\cdots,N_i\},\ i\in\{1,\cdots,K\} \\ & \sum_{i=1}^{K} b_i(s_j^k+W_j^k)\le B,\ k\in\{1,\cdots,N_j\},\ j\in\{1,\cdots,K\} \\ & \int_{s_i^k+W_i^k}^{s_i^{k+1}} b_i(t)dt = IO_i,\ k\in\{1,\cdots,N_i-1\},\ i\in\{1,\cdots,K\} \\ & \int_{s_i^{N_i}+W_i^{N_i}}^{d_i} b_i(t)dt = IO_i,\ i\in\{1,\cdots,K\} \end{aligned} \tag{1}$$

In Formula (1), $\tilde{\rho}_i(d_i) = \frac{\sum_{k=1}^{N_i} W_i^k}{d_i - r_i}$. The first and second constrained conditions are to satisfy the restriction of the application and platform bandwidth. In third and fourth constrained conditions, the I/O volume of each application is satisfied and the order of instances is promised implicitly.

Problem 2 (**MinDilation**): Find the I/O bandwidth assignment for each application $b_i(t)$ to minimize the largest slowdown among applications with the same parameters and consistent constraint conditions as Problem 1.

$$\min \max_{i=1\cdots K} \frac{\rho_i}{\tilde{\rho}_i(d_i)} \tag{2}$$

The rationale behind the *MinDilation* objectives is to provide fairness between all applications. It guides the scheduling to minimize the maximum of slowdowns to avoid starving some applications. All notations mentioned in these problem descriptions are listed in Table 1.

With the rapid growth of computing resources, HPC centers tend to rent spare computing resources to more users currently. Different applications have different I/O requirements for reasons such as the levels of services and the types of storage hardware [35]. The I/O scheduling problems can be generalized to take the applications' criticality into account. Here, we provide a simple enhanced model by introducing a weighted parameter for each application. The objective of the *MaxSysEfficiency* problem can be modified as $\max \frac{1}{N}\sum_{i=1}^{K}\beta_i w_i\tilde{\rho}_i(d_i)$, where w_i denotes the importance of application A_i. The *MinDilation* problem can also be modified in the same way. However, our proposed I/O scheduling in this work tends to make a global improvement while ignoring the demands of individual applications. The related weighted parameters are set to be one (i.e., all applications have the same importance).

3.3. Existing Methods and Motivations

Both problems described above have been proved as being NP-complete, even in a simple offline setting [13]. So we can just give some heuristics rather than an exact algorithm. These problems just have different optimization objectives. Thus, a unified method can deal with them with different strategies. Online I/O scheduling [13] is a greedy algorithm based on different heuristics. Periodic I/O scheduling [14] then improves the online one to exploit the periodicity. We describe both of them briefly and give the motivations of our work.

Table 1. Notations for problem description.

Notation	Description
N	The number of all the computing nodes in HPC platform
B	The aggregate bandwidth of PFS
b	The bandwidth of each local computing node
K	The number of all the stochastic iterative applications
A_i	The i-th application
β_i	The number of allocated computing nodes for A_i
N_i	The number of instances for A_i
r_i	The release time of A_i
d_i	The final complete time of A_i
B_i	The possible maximum bandwidth for A_i
W_i^k	The computing duration of the k-th instance of A_i
IO_i	The I/O volume of A_i
ρ_i	The optimal application efficiency for A_i
s_i^k	The start time of the k-th instance of A_i
$b_i(t)$	The assigned bandwidth at time t for A_i
$\tilde{\rho}_i(t)$	The real application efficiency at time t for A_i

3.3.1. Online I/O Scheduling

The rationale behind this is determining a priority queue of applications based on some strategies at each event. This greedy algorithm can adapt to many application settings. However, it is an online centralized method, which has heavy computation and a lack scalability. For different optimization objectives, there are different strategies [13] shown below that can be chosen.

- The *RoundRobin* strategy favors the application with the "first-come first-served" (FCFS) fashion. It ensures fairness and usually can be used for comparison;
- The *MinDilation* strategy favors the applications with low values of $\frac{\tilde{\rho}_i(t)}{\rho_i(t)}$. The application with low efficiency can be executed to improve the application efficiency that is user-oriented;
- The *MaxSysEff* strategy favors the applications with high $\beta_i \frac{\rho_i(t)}{\tilde{\rho}_i(t)}$. The application with a higher application efficiency represents that can utilize the system resources more efficiently. This objective is CPU-oriented;
- The *MinMax-γ* strategy is a balance between *MinDilation* and *MaxSysEff* . It favors the applications that have high values of $\beta_i \frac{\rho_i(t)}{\tilde{\rho}_i(t)}$, and dilation values of $\frac{\tilde{\rho}_i(t)}{\rho_i(t)}$ below a certain threshold γ.

3.3.2. Periodic I/O Scheduling

For the case with a fixed length of application instances, *periodic I/O scheduling* utilizes the periodicity of applications to assign the I/O bandwidth to each application offline [14]. It searches an appropriate period through an exponential search and inserts the *schedulable* application into the period based on some strategies, which is the same as the online method. This method obtains better performance than *online I/O scheduling* for this special case. It is decentralized, so it does not cause an additional overhead when applications run.

The method first sets the minimum possible period $T_{min} = \max_i(W_i + IO_i/B_i)$ and the maximum possible period $T_{max} = K' \cdot T_{min}$ with a specified parameter K'. It increasingly searches all possible periods between T_{min} and T_{max} by a factor of $(1+\epsilon)$. For each possible period T, it inserts the schedulable application A_i into the current bandwidth allocation by insert-in-pattern(P, A_i). If there is space to satisfy the I/O volume of A_i in the period, then A_i is *schedulable*. Finally, it chooses the optimal period T_{opt} to obtain the best system efficiency, SE. The detailed algorithm is shown in Algorithm 1.

Algorithm 1 Periodical I/O Scheduling (PerSched) [14]

```
Input: A set of applications $A_i(\beta_i, N_i, r_i, W_i, IO_i)$, PFS bandwidth $B$, local bandwidth $b$, $K'$, $\epsilon$
Output: The bandwidth allocation $P_{opt}$ for all applications and the period $T_{opt}$
1:  $T_{min} = \max_i(W_i + IO_i/B_i)$ and $T_{max} = K' \cdot T_{min}$
2:  $T = T_{min}$
3:  SE $= 0$, $T_{opt} = 0$, $P_{opt} = \emptyset$
4:  while $T \leq T_{max}$ do
5:      $P = \emptyset$
6:      while exists a schedulable application do
7:          $A = \{A_i$ is schedulable$\}$
8:          choose $A_i$ from $A$ by strategy $MaxSysEff$
9:          $P =$ Insert-In-Pattern($P$, $A_i$)
10:     end while
11:     if SE $<$ SysEfficiency($P$) then
12:         SE $=$ SysEfficiency($P$)
13:         $T_{opt} = T$, $P_{opt} = P$
14:     end if
15:     $T = T \cdot (1 + \epsilon)$
16: end while
```

3.3.3. Motivations

This work is motivated by three observations: First, due to some reasons, like the convergence of big data and HPC, there are many stochastic iterative applications deployed on the HPC platform, whose computing phrases obey some distributions. Second, the existing method can not utilize the characteristic information of applications adequately. The general online method ignores the periodicity and stochasticity of applications completely. Additionally, the periodic method is not able to adapt to the stochastic applications directly. Third, the effects of the lengths of different application instances getting longer or shorter can be counteracted. So, the adaptively periodic method is proposed to satisfy the requirement of stochasticity.

4. Adaptively Periodic I/O Scheduling

In this section, we describe the adaptively periodic I/O scheduling (APIO) in detail. First, we introduce the overall scheme and the related data structure. Then, we present the APIO algorithm and give some analysis results.

4.1. Scheme and Data Structures

In order to exploit the periodicity and stochasticity of applications with a stochastically iterative I/O periodicity, we construct a scheme based on the periodic I/O scheduling. For each stochastic iterative application A_i, the length of the computing phrase W_i^k of its each instance $\mathcal{I}_i^k$ is a random variable obeying a distribution $\mathcal{D}(\mu_i, \sigma_i)$. The practical length of W_i^k can be determined after the finish of that computing phrase.

The overall scheme includes two steps: In the first step, it sets W_i^k to be the same as μ_i and then utilizes the periodic I/O scheduling (Algorithm 1) to obtain a basic schedule P (periodic pattern). The schedule P can be expressed as $\cup_{i=1}^{K}(A_i\{\mathcal{I}_i^k\{< t_1, b_1 >, < t_2, b_2 >, \cdots\}\})$. For each instance $\mathcal{I}_i^k$, it includes a sequence of $< t_j, b_j >$ representing that the I/O operation of A_i starts at the time t_j with the bandwidth b_j. Then, we can construct an auxiliary array, $free$, to record the free space of PFS's I/O bandwidth. $free$ is also a sequence of $< t_j, b_j >$. In the second step, it adjusts the basic schedule P at each event when any computing phrase ends.

To clarify the algorithm in the second step, we introduce a list-data structure, L, which records all the start times of the first I/O part $< \mathcal{I}_i^k.t_1, p_i^k >$ of each instance $\mathcal{I}_i^k$. p_i^k is the pointer of the instance $\mathcal{I}_i^k$. L is a sorted array on $\mathcal{I}_i^k.t_1$ increasingly. The basic schedule P and the free space $free$ are also as input in the second step. These three main data structures are illustrated in Figure 3.

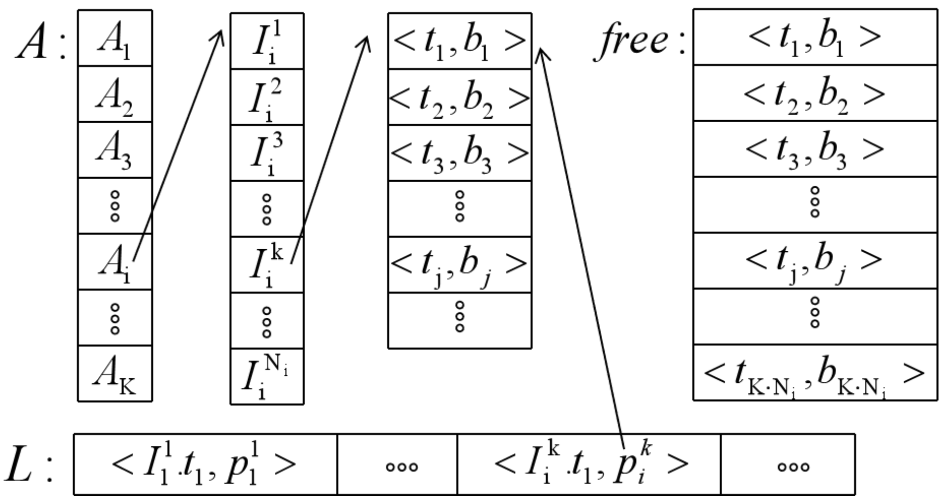


Figure 3. Data structure for APIO algorithm.

4.2. Adjusting the Periodic Schedule

Because the length of the computing phrase of each instance for the stochastic iterative application varies randomly, the periodic I/O scheduling pattern should be adjusted to achieve better performance or satisfy the extension of the computing. For an instance of an application, if its computing phrase ends in advance, its I/O phrase can be executed ahead. Otherwise, the execution of its I/O phrase would be postponed.

Specifically, when a computing phrase of an instance ends its execution on computing nodes, it will issue an event to notify that its I/O phrase can start. Let e_i^k be the event when the computing phrase of the k-th instance of the i-th application finished. If the time $e_i^k.t$ that the event is issued is less than the assigned time $\mathcal{I}_i^k.t_1$, the I/O transferring should be started earlier. Its periodic schedule, $\mathcal{I}_i^k\{< t_1, b_1 >, < t_2, b_2 >, \cdots\}$, should be modified. It gets the space from $free$ to execute IO_i. Similarly, when $e_i^k.t$ is greater than $\mathcal{I}_i^k.t_1$, its schedule also be adjusted.

In addition, when an event happens, there are some assigned I/O that have not been executed. We can assert that its execution time should be postponed. As such, its related schedule should be recalculated too. The detailed algorithm for the online execution of the stochastic iterative applications is described in Algorithm 2. The further explanation of the specific operations is also given.

Algorithm 2 Online Execution based on Adjusting (OnlineAdj)

```
Input: A set of applications $A_i(\beta_i, N_i, r_i, \mathcal{D}(\mu_i, \sigma_i), IO_i)$, PFS bandwidth $B$, local bandwidth
    $b$, the periodic schedule $P$
Output: The used time $T_{per}$ for the current period
1: gets the application set $A$, the remained bandwidth of PFS $free$ and the auxiliary list $L$
   from $P$
2: while exists an event $e_i^k$ do
3:     if $\mathcal{I}_i^k$ is marked as empty then
4:         Allocates bandwith for $\mathcal{I}_i^k$ and updates $free$
5:     else
6:         if $e_i^k.t < \mathcal{I}_i^k.t_1$ then
7:             cleans the assignment of $\mathcal{I}_i^k$
8:             allocates bandwith for $\mathcal{I}_i^k$ and updates $free$
9:         end if
10:        if $e_i^k.t > \mathcal{I}_i^k.t_1$ then
11:            cleans the assignment of $\mathcal{I}_i^k$
12:            allocates bandwith for $\mathcal{I}_i^k$ and updates $free$
13:        end if
14:        for each $L.\mathcal{I}_i^k.t_1 < e_i^k.t$ do
15:            empties $\mathcal{I}_i^k$ and updates $free$
16:            removes $\mathcal{I}_i^k$ term from $L$
17:        end for
18:    end if
19:    executes the current bandwidth assignment
20: end while
21: $T_{per} = \text{Time}(A)$
```

4.2.1. Cleaning Instances

When an event comes earlier or later, the bandwidth assigned previously is invalid and we need to recalculate the bandwidth assignment for the application issuing the event. We show an example in which an instance finished its computing phrase early in Figure 4. The solid line marked t_1^- represents the current time. t_1 denotes the expected time in the periodic I/O scheduling and the pre-assigned bandwidths should start at time t_1. However, the computing phrase of the instance $\mathcal{I}_1^1$ is finished early, so the pre-assigned bandwidths for $\mathcal{I}_1^1$ are invalid and then they are reassigned the bandwidth from the remaining bandwidth, $free$. The gray part in the figure represents the expected execution based on the pre-assignment of the periodic I/O scheduling. Similarly, if the computing phrase of the instance $\mathcal{I}_1^1$ is finished at a possible time t_1^+ that is greater than t_1, it will reassign the bandwidths too.

For an instance $\mathcal{I}_i^k$ that will be cleaned, we first release the bandwidths $\{< t_1, b_1 >, < t_2, b_2 >, \cdots\}$ assigned to it and then add to $free$. The algorithm then allocates bandwidths to $\mathcal{I}_i^k$ based on the *best-effort* strategy from the remaining bandwidth $free$. Among these operations, each item $< t_j, b_j >$ of $\mathcal{I}_i^k$ satisfies $b_j < B_i$ and each item $< t_j, b_j >$ of $free$ satisfies $0 \leq b_j \leq B$. Note that when the instance $\mathcal{I}_i^k$ issues the event e_i^k, a new bandwidth part $< e_i^k.t, b_1 >$ might be allocated. This will cause $free$ to add a new item with $e_i^k.t$, and remove the first bandwidth part $< \mathcal{I}_i^k.t_1, p_i^k >$ from the auxiliary list L for instance $\mathcal{I}_i^k$, which is executed instantly.

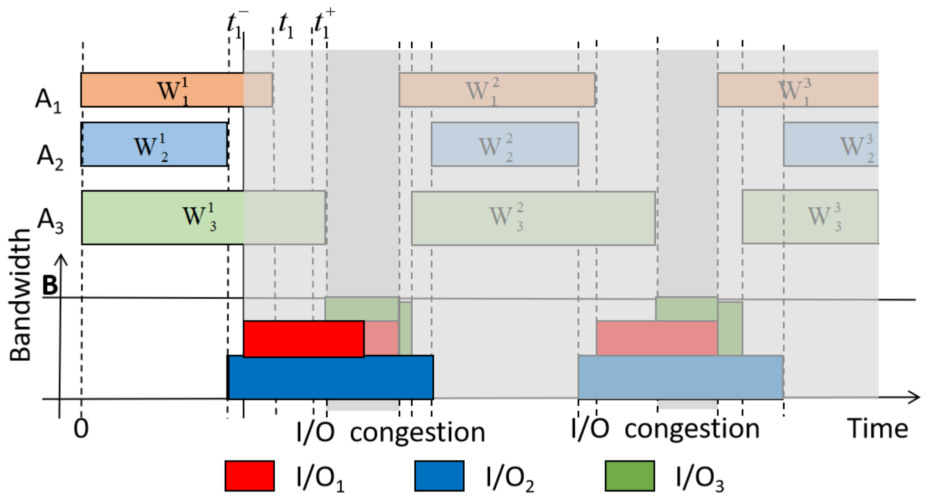


Figure 4. Example: instance finishes earlier in a period.

4.2.2. Emptying Instances

Assuming an event e_i^k comes, it will update the bandwidth assignment of instance $\mathcal{I}_i^k$ directly. However, if $e_i^k.t$ is greater than the start time of the I/O phrase of some instances, such as $\mathcal{I}_j^k$ with $j \neq i$, we can assert that the instance $\mathcal{I}_j^k$ will be postponed. To find such an instance, we maintain the auxiliary list L that records the first bandwidth part for each instance. From the beginning of the list, we find all the terms with $\mathcal{I}_i^k.t_1 < e_i^k.t$. Thus, we mark these instances $\mathcal{I}_j^k$ with a flag variable $empty$ and clean their assigned bandwidth to $free$. The record $< \mathcal{I}_j^k.t_1, p_j^k >$ of $\mathcal{I}_j^k$ in L is also removed.

When the event e_i^k comes, if the instance $\mathcal{I}_i^k$ is marked as $empty$, we just allocate the bandwidth for it from $free$ directly. The operation of cleaning instances is for the instance itself, and the operation of emptying instances is for other instances. Algorithm 2 adjusts the periodic bandwidth assignment through both operations, which reserves the advantage of the periodic I/O scheduling.

4.3. APIO Algorithm

In order to utilize the periodicity and stochasticity of the stochastic iterative applications, the adaptively periodic I/O scheduling (APIO) algorithm adjusts the bandwidth assignment of periodic I/O scheduling. It is composed of two basic modules: PerSched (Algorithm 1) and OnlineAdj (Algorithm 2). The complete description is shown in Algorithm 3.

APIO first calculates the total number of periods $N_{per} = \max_i(\frac{N_i}{N_i^{per}})$. N_i^{per} is the number of instances for application A_i in a period produced by algorithm PerSched. Then, for each period, it performs online scheduling by the OnlineAdj module, the input of which includes all the instance information of a period. Finally, it calculates the system efficiency SE and the dilation DI through the objectives shown in Formulas (1) and (2).

This algorithm can be seen as a combination of online and offline minds. It utilizes the periodic offline scheduling to obtain some prior information and then performs online scheduling to resist the stochasticity of the applications. It utilizes comprehensively the global information and the local information.

Algorithm 3 Adaptively Periodic I/O Scheduling (APIO)

```
Input: A set of applications $A_i(\beta_i, N_i, r_i, \mathcal{D}(\mu_i, \sigma_i), IO_i)$, PFS bandwidth $B$, local bandwidth $b$
Output: The system efficiency $SE$ and the dilation $DI$
1: gets the periodic schedule $P_{opt}$ and the period $T_{opt}$ by Algorithm 1
2: gets the number of period $N_{per} = \max_i(\frac{N_i}{N_i^{per}})$
3: $T_{tot} = 0$ , $p = 1$
4: while period $p \leq N_{per}$ do
5:     gets the used time $T_{per}$ by Algorithm 2
6:     $T_{tot} += T_{per}$, $p++$
7: end while
8: calculates $SE$ and $DI$ from $T_{tot}$
```

4.4. Performance Analysis

APIO is an online scheduling based on the pre-assignment of the periodic I/O scheduling to exploit the characteristics of stochastic iterative applications. Here, we analyze the advantages of this proposed method on the effectiveness and efficiency.

The key operations of APIO are the advance and delay of I/O transferring relative to the pre-assignment of the periodic method. Both operations do not worsen I/O congestion since there is enough space around the congestion area. In practice, the I/O overhead is less than one-third of the PFS aggregate bandwidth for most of the time [4].

The advance of I/O transferring can utilize the free space before the pre-assignment. As such, it does not worsen the schedule. Even if there is no free space, the pre-assignment of the application can satisfy the I/O requirement. When I/O transferring is postponed, some pre-assigned space might be wasted. However, in most cases, there is enough space to satisfy the postponed I/O requirement. With high probability ($p(x) \geq 0.95$), the length of the computing phrase is less than twice the mean length. Additionally, there is a pre-assigned space for the next instance that can be used. So Theorem 1 below is held.

Theorem 1. *The performance of APIO is within* two factors *of the online scheduling, with a high probability for stochastic iterative applications with Gaussian Distribution.*

Proof. Without the loss of generality, the performance considered here is the system efficiency for all the applications. For other objectives, we can achieve similar results.

In terms of application, the system efficiency is proportional to its completion time. This is assuming that the completion time of online scheduling proposed in reference [13] is T_{online} for the stochastic iterative applications, and the completion time of periodic I/O scheduling in reference [14] is $T_{periodic}$ for the applications that are generated by reducing the stochasticity of the stochastic iterative applications. Since the periodic I/O scheduling can utilize the periodicity of applications sufficiently, it obtains a global optimization and then $T_{periodic} < T_{online}$ with a high probability.

APIO adjusts the pre-assignment of periodic I/O scheduling. When the length of the computing phrase gets shorter, the completion time of the instance is less than the pre-assignment. However, when the length gets longer, the completion time would be longer. However, the length will be within *two factors* of the average length with a high probability. For the Gaussian distribution $\mathcal{D}(\mu, \sigma)$, the probability $p(x \leq \mu + 2\sigma)$ is 0.955. The pre-assigned space for the next instance can satisfy the I/O requirement of the current instance. So the completion time of APIO is within two factors of the periodic I/O scheduling with a high probability, which is $T_{APIO} < 2 \cdot T_{periodic}$. Then, $T_{APIO} < 2 \cdot T_{online}$. The theorem is proved. □

Moreover, APIO is more efficient than the existing online scheduling [13]. It just assigns the I/O bandwidth for each instance once with several computations, rather than for each application in each event. The pre-assignment of I/O bandwidth for a period is pre-calculated, which provides a performance basis of our method and makes the efficiency possible. The other computation overhead is searching the sorted auxiliary arrays, which can result in a constant complexity in run time.

5. Simulation Results

In this section, some simulation experiments have been designed to evaluate the performance of our proposed method, APIO. The experiments are conducted on stochastic iterative applications constructed by real applications with different I/O characteristics. We compared the performances on system efficiency and the application dilation of APIO and online scheduling [13] under different I/O congestion settings. All the simulations are implemented through a discrete event simulator introduced by reference [4], which maintains an event queue to mimic the execution of applications on an HPC platform.

5.1. Experiments Settings

The settings of the simulation experiments include system configuration and application configuration. Both configurations are built by simulating the parameters of the real system and application.

5.1.1. System Configuration

In this work, the run-time platform had been described by a very simple model illustrated in Figure 1. The related system parameters refer to the experiment settings in references [4,15], which originate from the real environment of the Intrepid Blue Gene/P supercomputer in Argonne National Laboratory, US.

The aggregate bandwidth of PFS, B, is set as 100GB/s. The peak bandwidth for each node, b, is 1 GB/s. The number of computing nodes is assumed to be sufficient. For this simple model, it does not need other parameters. The discrete event simulator getting these platform parameters can simulate the running of the entire HPC platform.

5.1.2. Application Configuration

Application settings used in this work also originated from the real applications that are reported in APEX's report (https://www.nersc.gov/assets/apex-workflows-v2.pdf, (accessed on 18 April 2022) for the LANL (Los Alamos National Laboratory) workflows [4]. We considered four real scientific applications: the Eulerian Application Project (EAP), Lagrangian Applications Project (LAP), Silverton, and the vector particle-in-cell (VPIC). The detailed characteristics of these applications are depicted in Table 2. Note that B_i rate (GB/s) implies the number of computing nodes assigned to the application A_i and the checkpoint time implies the volume of I/O transfers.

Table 2. Characteristics of the APEX applications [4].

Application	EAP	LAP	Silverton	VPIC
Number of instances	13	4	2	1
B_i rate (GB/s)	160	80	160	160
T_i Period (s)	5671	12,682	15,005	4483
Checkpoint time (s)	20	25	280	23.4

Then, we design a different I/O congestion to compare the performance of APIO and the basic online I/O scheduling (BIOS). The different combinations of these applications represent a different I/O congestion. The detailed configuration is shown in Table 3. From set #1 to #10, I/O contention is decreased with the decrease of the number of application LAP, because the I/O transferring at the same time is decreased.

Table 3. Application combinations for different I/O congestions [4].

Set #	1	2	3	4	5	6	7	8	9	10
EAP	0	0	0	0	0	0	1	0	0	1
LAP	10	8	6	4	2	2	2	0	0	0
Silverton	0	1	2	3	0	4	0	1	5	1
VPIC	0	0	0	0	1	0	0	1	0	0

In order to model the stochasticity of the application, we design three different distributions for the experiment. *Uniform* distribution is for simple situations, *Truncated Normal* and *LogNormal* distribution are more closer to the real situation. The parameters are derived from the characteristics of the APEX applications. The detailed distributions are shown in Table 4.

Table 4. Probability distributions of the lengths of the computing phrase.

(**a**) Probability distributions

Distribution	**PDF $f(x)$**
Uniform(a,b)	$\frac{1}{b-a}$
Truncated Normal(μ,σ,a,b)	$\frac{1}{\sigma}\frac{\frac{1}{\sqrt{2\pi}}\exp(-\frac{1}{2}(\frac{x-\mu}{\sigma})^2)}{\frac{1}{2}(1+erf(x/\sqrt{2}))}$
LogNormal(μ,σ)	$\frac{1}{x\sigma\sqrt{2\pi}}\exp(-\frac{1}{2}(\frac{\ln(x)-\mu}{\sigma})^2)$

(**b**) Distribution parameters

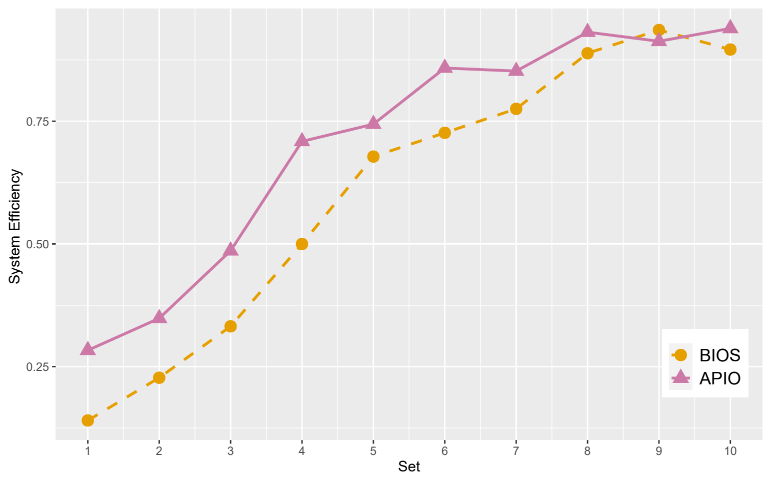
	EAP	**LAP**	**Silverton**	**VPIC**
μ	2	4	5	1
σ	0.5	1	1	0.5
a	1	2	3	0
b	3	6	7	2

From the parameters of these applications, we can construct the applications as the input of the discrete event simulator. The simulator eventually calculates the objective functions and obtains the system efficiency and application dilation for different I/O scheduling methods.

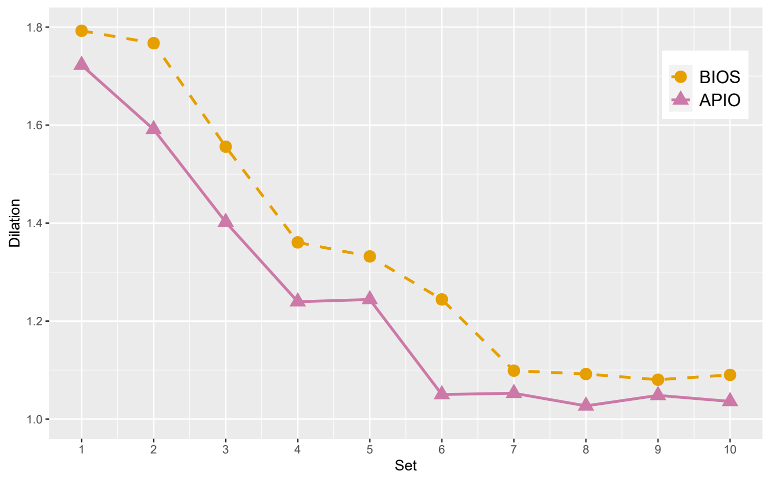
5.2. Results and Analysis

In this section, we show the experiment results by comparing the performance of APIO and the basic online I/O scheduling (BIOS [13]). Both methods are based on the *MinMax-γ* strategy with $\gamma = 0.5$, which is able to achieve higher than average performances compared to other strategies [13]. We conduct the simulation for each set of applications and different probability distributions, and then calculate the system efficiency and application dilation defined in Section 3.2. For each set, the test is repeated five times and we calculate the average.

Due to the simplicity of *Uniform* distribution, we first show the results of APIO and BIOS under the *Uniform* distribution. So, the length of the computing phrase of the application's instances is distributed in the interval $[a,b]$ with an equal probability. APIO can utilize the probabilistic characteristic to optimize the I/O scheduling. The detailed results are shown in Figure 5.



(**a**)



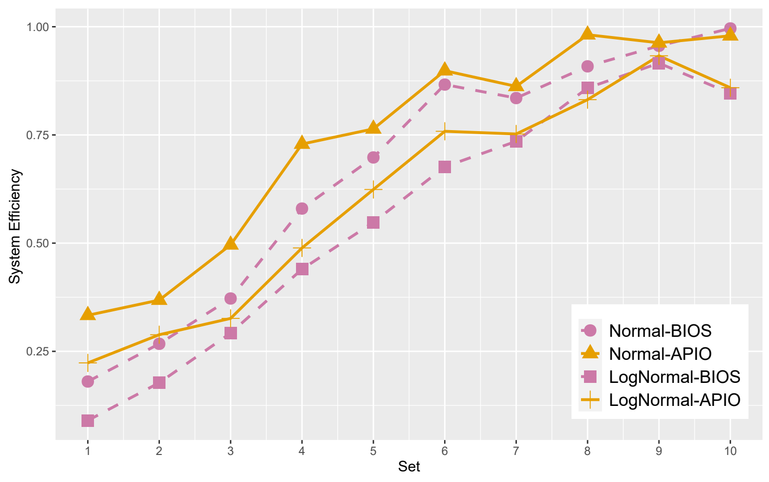
(**b**)

Figure 5. Performance of different scheduling for different I/O congestion. (**a**) System efficiency, (**b**) application dilation.

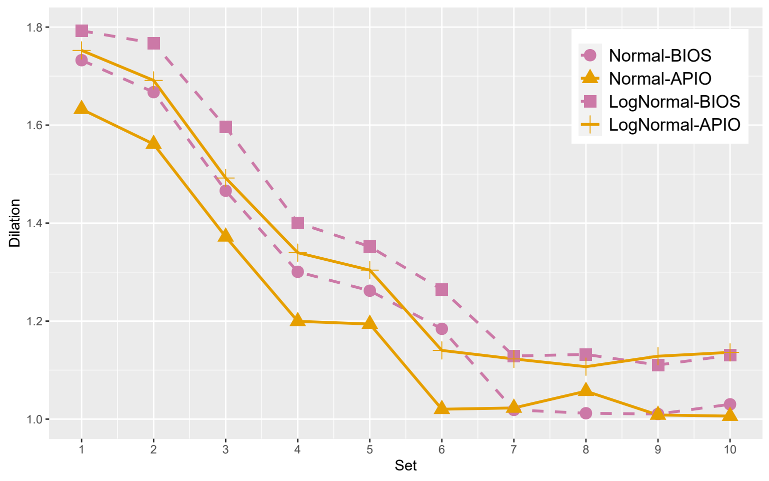
From set #1 to #10, the I/O congestion is increasing gradually. So, as Figure 5 shows, the system efficiency is also increasing and the application dilation is decreasing accordingly. For all the sets, the performance of APIO is superior to BIOS. At set #1, I/O congestion is the most serious, but APIO achieves the best relative performance. When I/O congestion disappeared, both methods obtained a similar performance.

Second, in order to show the influence of probability distributions, we conduct the simulation under the *Truncated Normal* and *LogNormal* distribution. The particular parameters of distribution are listed in Table 4. Other experiment's settings are the same as the *Uniform* distribution. The detailed results are shown in Figure 6.

As per the result shown, the trends of the system efficiency and application dilation are same as the results under the *Uniform* distribution. APIO obtains better performance than BIOS. However, the performance under the *Truncated Normal* distribution is better than the *LogNormal* distribution overall. The reason is that the proposed method favors the symmetric stochastic change of the computing phrase. Adaptively adjusting the periodic bandwidth allocation can counteract the effects of shrinking or expanding the computing phrase. The truncated normal distribution has better symmetry than the logNormal distribution and then obtains better performance. This result shows that the performance is seriously affected by the characteristics of the application.



(**a**)



(**b**)

Figure 6. Performance of different scheduling for different I/O congestion. (**a**) System efficiency, (**b**) application dilation.

In addition, when I/O congestion is serious, such as in set #1 and #2, the system efficiency of APIO for the *LogNormal* distribution even surpasses the performance of BIOS for the *Truncated Normal* distribution.

6. Conclusions

In this paper, we studied the I/O scheduling problem for applications with a stochastically iterative I/O periodicity to achieve the targeted objectives, such as system efficiency and application dilation. The existing methods did not utilize the stochasticity and periodicity presented in a wide range of applications, including particularly big data analytics. To take both characteristics of these applications into account, we proposed an online scheduling method, namely the adaptively periodic I/O scheduling (APIO) method, which dynamically adjusts the pre-assigned bandwidth online, which is provided by periodic I/O scheduling. APIO combines the advantages of the periodic I/O scheduling method to utilize the periodicity with the online adjustment to adapt the stochasticity. We provide the performance analysis to show the effectiveness and efficiency of the proposed method. The simulation experiment results show the superiority of the proposed method to the existing online scheduling method.

In our future work, a theoretical analysis based on computational complexity and probability theory will be done. Meanwhile, there are many research directions for new I/O scheduling methods that can be investigated in the future. A more sophisticated scheduling method based on more application properties can be studied. Weighted I/O scheduling with consideration for the applications' criticality will be explored, and energy-efficient I/O scheduling based on an HPC platform's energy model will also be studied in order to reduce energy consumption.

Author Contributions: Conceptualization, B.Z.; Methodology, B.Z.; software, B.Z.; Supervision, H.S.; Writing—original draft, B.Z.; Writing—review & editing, H.S. All authors have read and agreed to the published version of the manuscript.

Funding: This research was funded by the Key-Area Research and Development Plan of Guangdong Province, No. 2020B010164003 and the National Key Research and Development Plan's Key Special Program on the High-Performance Computing of China, No. 2017YFB0203201.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Hey, T.; Tansley, S.; Tolle, K. The Fourth Paradigm: Data-Intensive Scientific Discovery. *Proc. IEEE* **2011**, *99*, 1334–1337.
2. Boito, F.Z.; Inacio, E.C.; Bez, J.L.; Navaux, P.O.A.; Dantas, M.A.R.; Denneulin, Y. A Checkpoint of Research on Parallel I/O for High-Performance Computing. *ACM Comput. Surv.* **2018**, *51*, 1–35. [CrossRef]
3. Fox, G.C.; Qiu, J.; Jha, S.; Ekanayake, S.; Kamburugamuve, S. Big Data, Simulations and HPC Convergence. In *Proceedings of the Big Data Benchmarking—6th International Workshop, WBDB 2015, Toronto, ON, Canada, 16–17 June 2015 and 7th International Workshop, WBDB 2015, New Delhi, India, 14–15 December 2015*; Rabl, T., Nambiar, R., Baru, C.K., Bhandarkar, M.A., Poess, M., Pyne, S., Eds.; Revised Selected Papers; Lecture Notes in Computer Science; Springer: Berlin/Heidelberg, Germany, 2015; Volume 10044, pp. 3–17.
4. Aupy, G.; Beaumont, O.; Eyraud-Dubois, L. What Size Should Your Buffers to Disks be? In Proceedings of the 2018 IEEE International Parallel and Distributed Processing Symposium (IPDPS), Vancouver, BC, Canada, 21–25 May 2018; pp. 660–669.
5. Tessier, F.; Vishwanath, V.; Jeannot, E. TAPIOCA: An I/O Library for Optimized Topology-Aware Data Aggregation on Large-Scale Supercomputers. In Proceedings of the 2017 IEEE International Conference on Cluster Computing (CLUSTER), Honolulu, HI, USA, 5–8 September 2017; pp. 70–80.
6. Herbein, S.; Ahn, D.H.; Lipari, D.; Scogland, T.R.; Stearman, M.; Grondona, M.; Garlick, J.; Springmeyer, B.; Taufer, M. Scalable I/O-Aware Job Scheduling for Burst Buffer Enabled HPC Clusters. In Proceedings of the Proceedings of the 25th ACM International Symposium on High-Performance Parallel and Distributed Computing, HPDC '16, Kyoto, Japan, 31 May–4 June 2016; ACM: New York, NY, USA, 2016; pp. 69–80.
7. Dong, B.; Byna, S.; Wu, K.; Prabhat; Johansen, H.; Johnson, J.N.; Keen, N. Data Elevator: Low-Contention Data Movement in Hierarchical Storage System. In Proceedings of the 2016 IEEE 23rd International Conference on High Performance Computing (HiPC), Hyderabad, India, 19–22 December 2016; IEEE: Hyderabad, India, 2016; pp. 152–161.
8. Devarajan, H.; Kougkas, A.; Logan, L.; Sun, X.H. Hcompress: Hierarchical data compression for multi-tiered storage environments. In Proceedings of the 2020 IEEE International Parallel and Distributed Processing Symposium (IPDPS), New Orleans, LA, USA, 18–22 May 2020; pp. 557–566.
9. Liu, N.; Cope, J.; Carns, P.; Carothers, C.; Ross, R.; Grider, G.; Crume, A.; Maltzahn, C. On the role of burst buffers in leadership-class storage systems. In Proceedings of the 2012 IEEE 28th Symposium on Mass Storage Systems and Technologies (MSST), Pacific Grove, CA, USA, 16–20 April 2012; pp. 1–11.
10. Koo, D.; Lee, J.; Liu, J.; Byun, E.K.; Kwak, J.H.; Lockwood, G.K.; Hwang, S.; Antypas, K.; Wu, K.; Eom, H. An empirical study of I/O separation for burst buffers in HPC systems. *J. Parallel Distrib. Comput.* **2021**, *148*, 96–108. [CrossRef]
11. Thapaliya, S.; Bangalore, P.; Lofstead, J.; Mohror, K.; Moody, A. Managing I/O Interference in a Shared Burst Buffer System. In Proceedings of the 2016 45th International Conference on Parallel Processing (ICPP), Philadelphia, PA, USA, 16–19 August 2016; pp. 416–425.
12. Dorier, M.; Antoniu, G.; Ross, R.B.; Kimpe, D.; Ibrahim, S. CALCioM: Mitigating I/O Interference in HPC Systems through Cross-Application Coordination. In Proceedings of the 2014 IEEE 28th International Parallel and Distributed Processing Symposium (IPDPS), Phoenix, AZ, USA, 19–23 May 2014; pp. 155–164.
13. A. Gainaru.; Aupy, G.; Benoit, A.; Cappello, F.; Robert, Y.; Snir, M. Scheduling the I/O of HPC Applications Under Congestion. In Proceedings of the 2015 IEEE International Parallel and Distributed Processing Symposium (IPDPS), Hyderabad, India, 25–29 May 2015; pp. 1013–1022.
14. Aupy, G.; Gainaru, A.; Fèvre, V.L. Periodic I/O Scheduling for Super-Computers. In Proceedings of the High Performance Computing Systems. Performance Modeling, Benchmarking, and Simulation, Denver, CO, USA, 13 November 2017; Lecture Notes in Computer Science; Springer: Cham, Switzerland, 2017; pp. 44–66.
15. Aupy, G.; Gainaru, A.; Fèvre, V.L. I/O Scheduling Strategy for Periodic Applications. *ACM Trans. Parallel Comput. (TOPC)* **2019**, *6*, 1–26. [CrossRef]

16. Liang, W.; Chen, Y.; An, H. Interference-Aware I/O Scheduling for Data-Intensive Applications on Hierarchical HPC Storage Systems. In Proceedings of the 2019 IEEE 21st International Conference on High Performance Computing and Communications; IEEE 17th International Conference on Smart City; IEEE 5th International Conference on Data Science and Systems (HPCC/SmartCity/DSS), Zhangjiajie, China, 10–12 August 2019; pp. 654–661.
17. Zha, B.; Shen, H. Improved probabilistic I/O scheduling for limited-size Burst-Buffers deployed HPC. *Parallel Comput.* **2021**, *101*, 102708. [CrossRef]
18. Liang, W.; Chen, Y.; Liu, J.; An, H. CARS: A contention-aware scheduler for efficient resource management of HPC storage systems. *Parallel Comput.* **2019**, *87*, 25–34. [CrossRef]
19. Shi, X.; Liu, W.; He, L.; Jin, H.; Li, M.; Chen, Y. Optimizing the SSD Burst Buffer by Traffic Detection. *ACM Trans. Archit. Code Optim. (TACO)* **2020**, *17*, 1–26. [CrossRef]
20. Bu, Y.; Howe, B.; Balazinska, M.; Ernst, M.D. The HaLoop approach to large-scale iterative data analysis. *VLDB J.* **2012**, *21*, 169–190. [CrossRef]
21. Yildiz, O.; Zhou, A.C.; Ibrahim, S. Eley: On the effectiveness of burst buffers for big data processing in HPC systems. In Proceedings of the 2017 IEEE International Conference on Cluster Computing (CLUSTER), Honolulu, HI, USA, 5–8 September 2017; pp. 87–91.
22. Gainaru, A.; Goglin, B.; Honore, V.; Pallez Aupy, G.; Raghavan, P.; Robert, Y.; Sun, H. Reservation and Checkpointing Strategies for Stochastic Jobs. In Proceedings of the 2020 IEEE International Parallel and Distributed Processing Symposium (IPDPS), New Orleans, LA, USA, 18–22 May 2020; IEEE: New Orleans, LA, USA, 2020; pp. 853–863.
23. Du, Y.; Marchal, L.; Pallez (Aupy), G.; Robert, Y. Robustness of the Young/Daly formula for stochastic iterative applications. In Proceedings of the 49th International Conference on Parallel Processing (ICPP), Edmonton, AB, Canada, 17–20 August 2020; ACM: Edmonton, AB, Canada, 2020; pp. 1–11.
24. Jeannot, E.; Pallez, G.; Vidal, N. Scheduling periodic I/O access with bi-colored chains: Models and algorithms. *J. Sched.* **2021**, *24*, 469–481. [CrossRef]
25. Hu, W.; Liu, G.m.; Li, Q.; Jiang, Y.h.; Cai, G.l. Storage wall for exascale supercomputing. *Front. Inf. Technol. Electron. Eng.* **2016**, *17*, 1154–1175. [CrossRef]
26. Gainaru, A.; Aupy, G.P.; Sun, H.; Raghavan, P. Speculative Scheduling for Stochastic HPC Applications. In Proceedings of the Proceedings of the 48th International Conference on Parallel Processing (ICPP), Kyoto, Japan, 5–8 August 2019; ACM: Kyoto, Japan, 2019; pp. 1–10.
27. Boito, F.Z. Transversal I/O Scheduling: From Applications to Devices. Ph.D. Thesis, Universidade Federal do Rio Grande do Sul, Porto Alegre, Brazil, 2015.
28. Liao, W.K.; Choudhary, A. Dynamically Adapting File Domain Partitioning Methods for Collective I/O Based on Underlying Parallel File System Locking Protocols. In Proceedings of the 2008 ACM/IEEE Conference on Supercomputing, SC '08, Austin, TX, USA, 15–21 November 2008; IEEE Press: Piscataway, NJ, USA, 2008; pp. 3:1–3:12.
29. Boito, F.Z.; Kassick, R.V.; Navaux, P.O.; Denneulin, Y. AGIOS: Application-Guided I/O Scheduling for Parallel File Systems. In Proceedings of the 2013 International Conference on Parallel and Distributed Systems (ICPADS), Seoul, Korea, 15–18 December 2013; IEEE: Seoul, Korea, 2013; pp. 43–50.
30. Song, H.; Yin, Y.; Sun, X.H.; Thakur, R.; Lang, S. Server-side I/O Coordination for Parallel File Systems. In Proceedings of the Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis, SC '11, Seattle, WA, USA, 12–18 November 2011; ACM: New York, NY, USA, 2011; pp. 17:1–17:11.
31. Zhang, X.; Davis, K.; Jiang, S. IOrchestrator: Improving the Performance of Multi-node I/O Systems via Inter-Server Coordination. In Proceedings of the 2010 ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis, SC '10, New Orleans, LA, USA, 13–19 November 2010; pp. 1–11.
32. Carretero, J.; Jeannot, E.; Pallez, G.; Singh, D.; Vidal, N. Mapping and Scheduling HPC Applications for Optimizing I/O. In Proceedings of the 34th ACM International Conference on Supercomputing (ICS), Barcelona, Spain, 29 June–2 July 2020; pp. 1–12.
33. Michael, L.P. *Scheduling: Theory, Algorithms, and Systems*, 5th ed.; Springer: Berlin/Heidelberg, Germany, 2016.
34. Du, Y.; Marchal, L.; Pallez, G.; Robert, Y. Optimal Checkpointing Strategies for Iterative Applications. *IEEE Trans. Parallel Distrib. Syst.* **2022**, *33*, 507–522. [CrossRef]
35. Hua, Y.; Shi, X.; Jin, H.; Liu, W.; Jiang, Y.; Chen, Y.; He, L. Software-defined QoS for I/O in exascale computing. *CCF Trans. High Perform. Comput.* **2019**, *1*, 49–59. [CrossRef]