Frontiers of Information Technology & Electronic Engineering
www.zju.edu.cn/jzus; engineering.cae.cn; www.springerlink.com
ISSN 2095-9184 (print); ISSN 2095-9230 (online)
E-mail: jzus@zju.edu.cn

Storage wall for exascale supercomputing*

Wei HU‡1, Guang-ming LIU1,2, Qiong LI1, Yan-huang JIANG1, Gui-lin CAI1
(1College of Computer, National University of Defense Technology, Changsha 410073, China)
(2National Supercomputer Center in Tianjin, Tianjin 300457, China)
E-mail: {huwei, liugm}@nscc-tj.gov.cn; qiong_joan_li@aliyun.com; yhjiang@nudt.edu.cn; cc_cai@163.com

Received June 15, 2016; Revision accepted Oct. 5, 2016; Crosschecked Oct. 25, 2016

Abstract: The mismatch between compute performance and I/O performance has long been a stumbling block as supercomputers evolve from petaflops to exaflops. Currently, many parallel applications are I/O intensive, and their overall running times are typically limited by I/O performance. To quantify the I/O performance bottleneck and highlight the significance of achieving scalable performance in peta/exascale supercomputing, in this paper, we introduce for the first time a formal definition of the 'storage wall' from the perspective of parallel application scalability. We quantify the effects of the storage bottleneck by providing a storage-bounded speedup, defining the storage wall quantitatively, presenting existence theorems for the storage wall, and classifying the system architectures depending on I/O performance variation. We analyze and extrapolate the existence of the storage wall by experiments on Tianhe-1A and case studies on Jaguar. These results provide insights on how to alleviate the storage wall bottleneck in system design and achieve hardware/software optimizations in peta/exascale supercomputing.

Key words: Storage-bounded speedup, Storage wall, High performance computing, Exascale computing
http://dx.doi.org/10.1631/FITEE.1601336 **CLC number:** TP338.6

1 Introduction

With the developments in system architecture, the supercomputer has witnessed great advances in system performance. In the TOP500 list of the world's most powerful supercomputers in 2015, Tianhe-2 was ranked as the world's No. 1 system with a performance of 33.86 petaflop/s (TOP500, 2015). It will not be long before supercomputers reach exascale performance (10^3 petaflop/s) (HPCwire, 2010).

Supercomputers provide important support for scientific applications; meanwhile, the demands of these applications stimulate the development of supercomputers. In particular, the rise in I/O-intensive applications, which are common in the fields of genetic science, geophysics, nuclear physics, atmospherics, etc., brings new challenges to the performance, scalability, and usability of a supercomputer's storage system.

In supercomputing, there exists a performance gap between compute and I/O. Actually, there are two causes of the mismatch between application requirements and I/O performance. First, since the 1980s, the read and write bandwidth of a disk has increased by only 10% to 20% per year, while the average growth in processor performance has reached 60%. Second, the compute system scales much faster than the I/O system. Thus, as I/O-intensive applications expand at a very fast pace, supercomputers have to confront a scalability bottleneck due to the competition for storage resources.

The storage bottleneck is well known in the supercomputer community. Many studies have explored the challenges of exascale computing including the storage bottleneck (Cappello *et al.*, 2009; Agerwala, 2010; Shalf *et al.*, 2011; Lucas *et al.*, 2014).

‡ Corresponding author
* Project supported by the National Natural Science Foundation of China (Nos. 61272141 and 61120106005) and the National High-Tech R&D Program (863) of China (No. 2012AA01A301)
ORCID: Wei HU, http://orcid.org/0000-0002-8839-7748

Some important and basic issues related to the storage bottleneck, e.g., quantitative description, inherent laws, and system scalability, have not been solved yet. This work seeks to solve these problems and to provide more insights in this area. To achieve this goal, we present the 'storage wall' theory to analyze and quantify the storage bottleneck challenge, and focus on the effects of the storage wall on the scalability of parallel supercomputing. Then, we present some experiments and case studies to provide insights on how to mitigate storage wall effects when building scalable parallel applications and systems at peta/exascale levels. Our main contributions can be summarized as follows:

1. Based on the analysis of I/O characteristics in parallel applications, we present a storage-bounded speedup model to describe the system scalability under the storage constraint.

2. We define the notion of a 'storage wall' formally for the first time to quantitatively interpret the storage bottleneck issue for supercomputers. We also propose an existence theorem and a variation law of the storage wall to reflect the degree of system scalability affected by storage performance.

3. We identify the key factor for the storage wall, called the 'storage workload factor'. Based on this factor we present an architecture classification method, which can reveal the system scalability under storage constraint for each architecture category. Then, we extend the storage workload factor and find the system parameters that affect scalability. We categorize mainstream storage architectures in supercomputers according to the above scalability classification method and reveal the scalability features by taking the storage wall into account. These works can help system designers and application developers understand and find solutions to current performance issues in hardware, software, and architecture.

4. Through a series of experiments on Tianhe-1A by using IOR benchmarks and case studies on Jaguar by using parallel applications with checkpointing, we verify the existence of the storage wall. These results and analyses can provide effective insights on how to alleviate or avoid storage wall bottlenecks in system design and optimization in peta/exascale supercomputing.

2 Related work

Speedup has been the exclusive measurement for performance scalability in parallel computing for a long time (Culler *et al.*, 1998). Various speedup models have been proposed.

In Amdahl (1967), a speedup model (Amdahl's law) for a fixed-size problem was advocated:

$$S_{\text{Amdahl}} = \frac{1}{f + (1-f)/P},$$

where P is the number of processors and f the ratio of the sequential portion in one program. Amdahl's law can serve as a guide to how much an enhancement will improve performance and how to distribute resources to improve cost-performance (Hennessy and Patterson, 2011). Nevertheless, the equation reveals a pessimistic view on the usefulness of large-scale parallel computers because the maximum speedup cannot exceed $1/f$.

Gustafson (1988) introduced a scaled speedup for a fixed-time problem (Gustafson's law), which scales up the workload with the increasing number of processors to preserve the execution time:

$$S_{\text{Gustafson}} = f + (1-f)\,P.$$

This speedup proves the scalability of parallel computers. Compared to Amdahl's law, Gustafson's law fixes the response time and is concerned with how large a problem could be solved within that time. It overcomes the shortcomings of Amdahl's law.

Sun and Ni (1993) presented a memory-bounded speedup model, which scales up the workload according to the memory capacity of the system, to demonstrate the relationship among memory capacity, parallel workload, and speedup. This speedup is suitable especially for the supercomputer with distributed-memory multiprocessors, because the memory available determines the size of the scalable problem.

Culler *et al.* (1998) proposed a speedup model taking the communication and synchronization overhead into account for the first time. Based on this model, researchers can improve the system performance by reducing the communication and synchronization overheads, such as overlapping communication and computing and load balancing.

In addition to the above speedup models, there are some other related studies. Wang (2009) provided a reliability speedup metric for parallel applications with checkpointing and studied the reliability problem of supercomputers from the view of scalability. Furthermore, Yang *et al.* (2011; 2012) introduced for the first time the concept of a reliability wall based on reliability speedup. Their work is more relevant to ours. They provided insights on how to mitigate reliability wall effects when building reliable and scalable parallel applications and systems at peta/exascale levels. Here, we establish the storage wall related theory to investigate storage challenges for current and future large-scale supercomputers.

Chen *et al.* (2016) provided a topology partitioning methodology to reduce the static energy of supercomputer interconnection networks. This is a new exploration to solve the energy problem for future exascale supercomputers.

In a supercomputer system, the performance of the storage system significantly affects the system scalability of a supercomputer. In this work, we develop a storage wall framework based on our previous work (Hu *et al.*, 2016) to analyze and quantify the effects of storage performance on the scalability of the supercomputer.

3 I/O characteristics of parallel applications

Large-scale scientific parallel applications running on supercomputers are mostly numerical simulation programs, and they have some distinctive I/O features as follows:

1. Periodicity: Scientific applications are usually composed of two stages, i.e., the compute phases and the I/O phases (Miller and Katz, 1991; Pasquale and Polyzos, 1993; Wang *et al.*, 2004; Liu *et al.*, 2014; Kim and Gunasekaran, 2015). For many I/O-intensive applications, write operations are usually the dominant operations in the I/O phase. For example, checkpointing can be seen as an I/O-intensive application and usually periodically uses write operations to save the state of the compute nodes (Hargrove and Duell, 2006; Bent *et al.*, 2009).

2. Burstiness: Scientific applications not only have periodic I/O phases, but also perform I/O in short bursts (Kim *et al.*, 2010; Carns *et al.*, 2011; Liu *et al.*, 2014; Kim and Gunasekaran, 2015), particularly with write operations (Wang *et al.*, 2004).

3. Synchronization: The parallel patterns of applications often determine parallel I/O access patterns (Lu, 1999). In single program multiple data (SPMD) or multiple program multiple data (MPMD) applications, more than 70% I/O operations are synchronous (Kotz and Nieuwejaar, 1994; Purakayastha *et al.*, 1995). For instance, applications in earth science and climatology (Carns *et al.*, 2011) consist of nearly identical jobs, which usually lead to a large number of synchronous I/O operations among processes (Xie *et al.*, 2012).

4. Repeatability: A number of jobs tend to run many times using different inputs, and thus they exhibit a repetitive I/O signature (Liu *et al.*, 2014).

According to the I/O characteristics, we will introduce a storage-bounded speedup model in Section 4 to pave the way for the storage wall definition.

4 Storage wall theory

Since processors are evolving rapidly, the increase in computing power has enabled the creation of important scientific applications. As parallel applications increase gradually in scale, their performance is often confined to the storage performance. This significantly affects supercomputer scalability. To analyze and quantify this effect, we first define the storage-bounded speedup, and then provide the storage wall theory to elaborate on supercomputer scalability based on the storage performance constraint.

4.1 Defining storage-bounded speedup

Storage-bounded speedup is built to quantify system scalability considering the storage performance constraint. Prior to defining storage-bounded speedup, we assume that one process runs on a single-core processor in a parallel system, and we treat a multi-core processor system as a system which consists of multiple single-core processors. Thus, given that a parallel system consists of P single-core processors, the running processes own the same number. For clarity, let each node denote each process. The given parallel system W has P nodes in total, and here we denote the ith node as N_i $(i = 0, 1, \cdots, P-1)$. All the symbols used in this paper are listed in Table 1.

For a given program Q, the execution process consists of two types of phases: the compute phase and the I/O phase (Miller and Katz, 1991; Pasquale and Polyzos, 1993; Wang *et al.*, 2004; Liu *et al.*, 2014;

Table 1 List of main symbols and the related specifications

Symbol	Description	Definition
N_i	The ith node in the parallel system ($i=0,1,\cdots,P-1$)	Section 4.1
P	The number of nodes with a single-core processor	Section 4.1
W	A parallel system with P nodes	Section 4.1
Q	A parallel program with P processes	Section 4.1
S_{Amdahl}	Amdahl's speedup	Section 2
$S_{\text{Gustafson}}$	Gustafson's speedup	Section 2
S_P	Compute speedup	Section 4.1
N^{s}	The number of I/O phases when Q runs on a single node	Eq. (1)
N^{c}	The number of compute phases when Q runs on a single node	Section 4.1
I^{sint}	Average time of all compute phases when Q runs on a single node	Eq. (2)
I^{sser}	Average time of all I/O phases when Q runs on a single node	Eq. (1)
T^{snon}	Execution time of all compute phases when Q runs on a single node	Eq. (1)
T^{sio}	Execution time of all I/O phases when Q runs on a single node	Section 4.1
$T^{\text{S}}_{\text{Sto}}$	Execution time of program Q running on a node serially	Eq. (1)
N^{P}_i	The number of I/O phases on the ith node	Section 4.1
I^{pint}_i	Average time of all the compute phases on the ith node	Section 4.1
I^{pser}_i	Average time of all the I/O phases on the ith node	Section 4.1
T^{pnon}_i	All the compute time on the ith node	Section 4.1
N^{P}	Average of N^{P}_i for each process when program Q is running on P nodes	Eq. (3)
I^{pint}	Average of I^{pint}_i for each process when program Q is running on P nodes	Eq. (4)
I^{pser}	Average of I^{pser}_i for each process when program Q is running on P nodes	Eq. (3)
T^{pnon}	The time for the compute parts when program Q is running on P nodes	Eq. (3)
T^{pio}_i	The I/O time on the ith node	Section 4.1
T^{pio}	The time for the I/O parts when program Q is running on P nodes	Section 4.1
$T^{\text{P}}_{\text{Sto}}$	Execution time of program Q on P nodes	Eq. (3)
$S^{\text{P}}_{\text{Sto}}$	Storage-bounded speedup	Eq. (5)
N	$I^{\text{pser}}/I^{\text{pint}}$	Eq. (6)
M	$I^{\text{sser}}/I^{\text{sint}}$	Eq. (6)
$O(P)$	Storage workload factor	Eq. (7)
$S^{\text{P}}_{\text{Amdahl-Sto}}$	Amdahl storage-bounded speedup	Eq. (9)
$S^{\text{P}}_{\text{Gustafson-Sto}}$	Gustafson storage-bounded speedup	Eq. (10)
$\text{supS}^{\text{P}}_{\text{Sto}}$	Storage wall	Section 4.2
α	Memory hit rate	Section 5.1
D	Average amount of data for the I/O phases for all processes	Section 5.1
V_{mem}	Average bandwidth for the memory access on each node	Section 5.1
$V_{\text{out-mem}}$	Average I/O bandwidth per node	Section 5.1
CDP architecture	Centralized, distributed, and parallel I/O architecture	Section 5.2.3
β	Absorbing ratio	Section 5.2.4
V_{burst}	Average bandwidth per node provided by burst buffers	Section 5.2.4
$S^{P}_{\text{G-Sto-fullckp}}$	Storage-bounded speedup for parallel application with full-memory checkpointing	Eq. (19)
$S^{P}_{\text{G-Sto-incrckp}}$	Storage-bounded speedup for parallel application with incremental-memory checkpointing	Eq. (20)
P_{opt}	Optimal parallel size at which parallel applications obtain the largest storage-bounded speedup	Section 6.2

Kim and Gunasekaran, 2015). In this study, the compute phases also denote I/O-free phases. In Fig. 1, we use shaded parts to denote compute phases and white parts to denote I/O phases. Fig. 1a shows a program Q running on a node serially, where N_0 denotes the process and $T^{\text{S}}_{\text{Sto}}$ stands for the execution time. Fig. 1b displays the case where program Q is running on a P-node system in parallel, where $T^{\text{P}}_{\text{Sto}}$ denotes the practical execution time.

In the following paragraphs we will analyze the execution time of a serial or parallel program Q running on one or multiple nodes.

In the serial program situation (Fig. 1a), the total execution time $T^{\text{S}}_{\text{Sto}}$ is obviously the time summation of all the compute phases and I/O phases, i.e., $T^{\text{S}}_{\text{Sto}} = T^{\text{snon}} + T^{\text{sio}}$ (T^{snon} denotes the total time of all compute phases and T^{sio} the total time of all I/O phases). Let N^{s} be the number of I/O

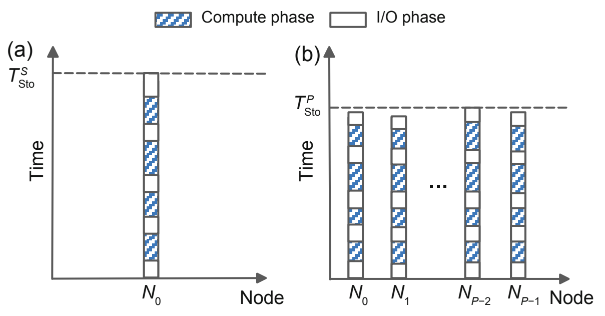

Fig. 1 Execution state of a program on a *P*-node system: (a) single process; (b) *P* parallel processes

phases and $I^{\rm sser}$ the average time of all I/O phases. We can obtain $T^{\rm sio} = N^{\rm s} I^{\rm sser}$, and thus $T^{\rm S}_{\rm Sto}$ can be described as

$$T^{\rm S}_{\rm Sto} = T^{\rm snon} + N^{\rm s} I^{\rm sser}. \tag{1}$$

Let $N^{\rm c}$ be the number of compute phases and $I^{\rm sint}$ be the average time of all compute phases. Thus, the total execution time of all compute phases is $T^{\rm snon} = N^{\rm c} I^{\rm sint}$. The I/O load of a program consists mainly of reading raw data, storing intermediate data, and writing final results, to name a few tasks. Both compute and I/O phases happen alternately, and the number of I/O phases $N^{\rm s}$ is usually one larger than that of the compute phases $N^{\rm c}$, i.e., $N^{\rm s} = N^{\rm c} + 1$ (Fig. 1). Therefore, $T^{\rm snon} = N^{\rm c} I^{\rm sint} = (N^s - 1) I^{\rm sint}$ and $N^{\rm s} = T^{\rm snon}/I^{\rm sint} + 1$. Then, we can rewrite Eq. (1) as

$$T^{\rm S}_{\rm Sto} = T^{\rm snon} + (T^{\rm snon}/I^{\rm sint}) I^{\rm sser} + I^{\rm sser}. \tag{2}$$

Before analyzing the execution time of a parallel program Q running on multiple nodes, two assumptions need to be described in detail. The first is that the load, especially the I/O load, is balanced among all nodes. The scalability of a large-scale parallel application can be affected by many factors, such as the application scaling model, architecture type, storage performance, and load imbalance (Culler *et al.*, 1998; Gamblin *et al.*, 2008). Due to the complexity of the problem, many studies generally focus on the impact of a certain factor, assuming that the other factors are fixed. Examples include Amdahl's law (Amdahl, 1967) and Gustafson's law (Gustafson, 1988), and they came to their conclusions in ideal conditions, including load balance. In this work, we discuss mainly the influence of storage performance on the scalability of a parallel application, assuming that the work load is balanced when the application scales.

The condition we study here is similar to Gustafson's law, in which the load (including the compute load and I/O load) increases continuously with application scaling (Gustafson, 1988). In this procedure, if the total load increases but the increase of I/O load is disproportionate to the increase of the compute load, the application scalability may suffer from additional effects. To be specific, if the I/O load increment is less than that of the compute load, it would not be sufficient to detect the impact of storage performance on application scalability; if the I/O load increment is more than the compute load, the application scalability will be further deteriorated by a rapid I/O load increase. The primary objective of this work is to discuss the influence of storage performance on the scalability of parallel applications. To exclude the influence of other factors, such as an irregular expansion of the application load, here we assume that the I/O load is in proportion to the compute load, and both loads increase with the increase of the number of compute nodes.

In the parallel program situation, it is nontrivial to analyze the execution time. In this work, we focus on the statistical significance of the execution time. That is, many parameters in the parallel running situation, such as the number of I/O phases, are the values on average. As a matter of fact, large-scale parallel applications may encounter barriers, and the slowest node determines the execution time in this condition. The difference between the average time and the worst case time due to the barrier is mainly the global synchronization overhead. For parallel applications, the synchronization overhead is affected mainly by load imbalance (Culler *et al.*, 1998). In the situation of load balance, all the processes can reach the barrier almost simultaneously.

As described by the first assumption above, this research is established under the assumption of load balance, and the global synchronization overhead is small and can be ignored here. From a statistical perspective, using the average time rather than a certain runtime can effectively reflect the trend in application execution time and help eliminate the bias caused by casual/random factors on the application execution time, thus improving the validity and reliability of this study. For these reasons, we use the average time to represent the time of the compute phase and I/O phase to simplify the storage-bounded speedup model, which is reasonable and in line with the assumptions above.

Here we analyze the execution time of the parallel program Q running on a P-node system W. Let $T^{\mathrm{P}}_{\mathrm{Sto}}$ denote the total execution time of the parallel program. Similar to the serial program situation, $T^{\mathrm{P}}_{\mathrm{Sto}}$ also consists of two parts, i.e., the time for the compute and I/O parts, denoted by T^{pnon} and T^{pio}, respectively. Thus, $T^{\mathrm{P}}_{\mathrm{Sto}} \approx T^{\mathrm{pnon}} + T^{\mathrm{pio}}$. Here we use approximation instead of equality. The approximation is reasonable and valid for the following theory because all parameters are defined on statistical values discussed above.

For the given parallel program Q that runs on P nodes, we use N^{p}_i to denote the number of I/O phases on the ith node, and I^{pser}_i to denote the average time of all the I/O phases on the ith node. So, the I/O time on the ith node is $T^{\mathrm{pio}}_i = N^{\mathrm{p}}_i I^{\mathrm{pser}}_i$. Further, we define $N^{\mathrm{p}} = \sum_{i=0}^{P-1} N^{\mathrm{p}}_i / P$ as the average of N^{p}_i for each process, and $I^{\mathrm{pser}} = \sum_{i=0}^{P-1} I^{\mathrm{pser}}_i / P$ as the average of I^{pser}_i for each process. So, we obtain $T^{\mathrm{pio}} = \sum_{i=0}^{P-1} (N^{\mathrm{p}}_i I^{\mathrm{pser}}_i)/P$. Since most parallel scientific applications have I/O features such as periodicity and synchronization, for the applications running on a large number of nodes, whose I/O load is balanced and scales in proportion to the compute load, the numbers of I/O phases are approximately equal among all processes and they are all approximately equal to N^{p}. This means $N^{\mathrm{p}}_i \approx N^{\mathrm{p}}$. So, we can obtain $T^{\mathrm{pio}} = \sum_{i=0}^{P-1} (N^{\mathrm{p}}_i I^{\mathrm{pser}}_i)/P \approx N^{\mathrm{p}} \sum_{i=0}^{P-1} (I^{\mathrm{pser}}_i / P) = N^{\mathrm{p}} I^{\mathrm{pser}}$.

Analogous with Eq. (1), the execution time of program Q running on P nodes is

$$T^{\mathrm{P}}_{\mathrm{Sto}} \approx T^{\mathrm{pnon}} + N^{\mathrm{p}} I^{\mathrm{pser}}. \tag{3}$$

Similarly, we use I^{pint}_i to signify the average time of all the compute phases on the ith node. Then we define $I^{\mathrm{pint}} = \sum_{i=0}^{P-1} I^{\mathrm{pint}}_i / P$ as the average of I^{pint}_i for each process. We use T^{pnon}_i to stand for all the compute time on the ith node, and define $I^{\mathrm{pint}} = \sum_{i=0}^{P-1} I^{\mathrm{pint}}_i / P$ as the average of I^{pint}_i for each process. As stated above, the number of I/O phases is one more than that of the compute phases on each node. Thus, we have $N^{\mathrm{p}} = T^{\mathrm{pnon}}/I^{\mathrm{pint}} + 1$ and Eq. (3) becomes

$$T^{P}_{\mathrm{Sto}} \approx T^{\mathrm{pnon}} + (T^{\mathrm{pnon}}/I^{\mathrm{pint}}) I^{\mathrm{pser}} + I^{\mathrm{pser}}. \tag{4}$$

In terms of the above analysis, we define the storage-bounded speedup to analyze the scalability variation under the storage performance constraint.

Definition 1 (Storage-bounded speedup) When a parallel program Q runs on a parallel system W with P nodes, the storage-bounded speedup can be defined as follows:

$$S^{\mathrm{P}}_{\mathrm{Sto}} = \frac{T^{\mathrm{S}}_{\mathrm{Sto}}}{T^{\mathrm{P}}_{\mathrm{Sto}}}. \tag{5}$$

According to Eqs. (2) and (4), suppose that S_P is the speedup of the compute part. Meanwhile, we define $N = I^{\mathrm{pser}}/I^{\mathrm{pint}}$ and $M = I^{\mathrm{sser}}/I^{\mathrm{sint}}$. Thus, the storage-bounded speedup can be described as

$$\begin{aligned} S^{\mathrm{P}}_{\mathrm{Sto}} &= \frac{T^{\mathrm{S}}_{\mathrm{Sto}}}{T^{\mathrm{P}}_{\mathrm{Sto}}} \approx \frac{T^{\mathrm{snon}} + (T^{\mathrm{snon}}/I^{\mathrm{sint}}) I^{\mathrm{sser}} + I^{\mathrm{sser}}}{T^{\mathrm{pnon}} + (T^{\mathrm{pnon}}/I^{\mathrm{pint}}) I^{\mathrm{pser}} + I^{\mathrm{pser}}} \\ &\le \frac{T^{\mathrm{snon}}(1 + I^{\mathrm{sser}}/I^{\mathrm{sint}})}{T^{\mathrm{pnon}}(1 + I^{\mathrm{pser}}/I^{\mathrm{pint}})} \\ &= S_P \frac{1+M}{1+N} = S_P \frac{1}{1 + \dfrac{N-M}{1+M}}. \end{aligned} \tag{6}$$

Here, we leave the derivations of Eq. (6) in Appendix. In Eq. (6), we set the upper bound of the right-hand side of the approximation as the value of the storage-bounded speedup. Since the storage-bounded speedup is used to quantify the scalability under the storage performance constraint, for long-running applications, this error caused by the upper bound is negligible and will not affect the correctness of the results analyzed below.

Let us define

$$O(P) = \frac{N-M}{1+M}. \tag{7}$$

$O(P)$ reflects the I/O execution time variation with the rise in the application load, and in this sense, we may call it the ‘storage workload factor’.

Thus, the final form of the storage-bounded speedup becomes

$$S^{\mathrm{P}}_{\mathrm{Sto}} = S_P \frac{1}{1 + O(P)}. \tag{8}$$

Eq. (8) implies how the I/O performance variation affects the speedup with the increase of the number of processors. Thus, it can reflect the scalability of parallel systems under the storage constraint.

In Eq. (8), if S_P is instantiated by Amdahl's speedup and Gustafson's speedup, respectively, we can obtain the Amdahl storage-bounded speedup as

$$\begin{aligned} S^{\mathrm{P}}_{\mathrm{Amdahl\text{-}Sto}} &= S_{\mathrm{Amdahl}} \cdot \frac{1}{1 + O(P)} \\ &= \frac{P}{1 + f(P-1)} \cdot \frac{1}{1 + O(P)}, \end{aligned} \tag{9}$$

and the Gustafson storage-bounded speedup as

$$\begin{aligned} S^{\mathrm{P}}_{\text{Gustafson-Sto}} &= S_{\text{Gustafson}} \cdot \frac{1}{1+O(P)} \\ &= (f + P(1-f)) \cdot \frac{1}{1+O(P)}. \end{aligned} \tag{10}$$

In both Amdahl's law and Gustafson's law, there is only one parameter f; namely, the ratio of the sequential portion in one program is used to characterize the scalability. Similarly, in this study, Eqs. (8), (9), and (10) use f to characterize the scalability of the compute part of the application. The difference between earlier works, i.e., Amdahl's law and Gustafson's law, and ours, is that our storage-bounded speedup takes into account the storage workload factor $O(P)$, which reflects the effect of the storage system over the whole system scalability, and this has not been studied before. For instance, both laws yield the highest speedup when $f = 0$. Actually, both claims may not be true in practice because I/O effect may exist. Due to the competition for I/O resources, I/O service time is prolonged in parallel computing, resulting in $O(P) \neq 0$. From this perspective, the storage-bounded speedup is more reasonable for practical applications, which is our main motivation for this study. Ideally, if $O(P) = 0$, our two storage-bounded speedup models, i.e., Eqs. (9) and (10), are simplified into traditional Amdahl's and Gustafson's speedup models, respectively, and the system scalability is not affected by the storage performance.

4.2 Defining the storage wall

Definition 2 (Storage wall) When a program Q runs on a parallel system which satisfies $\lim\limits_{P\to\infty} S_P = \infty$, the storage wall is the supremum of the storage-bounded speedup $S^{\mathrm{P}}_{\mathrm{Sto}}$, denoted as $\sup S^{\mathrm{P}}_{\mathrm{Sto}}$.

Theorem 1 When a program Q runs on a parallel system which satisfies $\lim\limits_{P\to\infty} S_P = \infty$, the storage wall exists if and only if $\lim\limits_{P\to\infty} S^{\mathrm{P}}_{\mathrm{Sto}} \neq \infty$.

Proof According to Definition 2, if the storage wall exists, the supremum of the storage-bounded speedup $S^{\mathrm{P}}_{\mathrm{Sto}}$ exists, i.e., $\sup S^{\mathrm{P}}_{\mathrm{Sto}} = R$ (R is a positive constant). Then, $S^{\mathrm{P}}_{\mathrm{Sto}} \leq \sup S^{\mathrm{P}}_{\mathrm{Sto}}$. Therefore, $\lim\limits_{P\to\infty} S^{\mathrm{P}}_{\mathrm{Sto}} \neq \infty$. If $\lim\limits_{P\to\infty} S^{\mathrm{P}}_{\mathrm{Sto}} \neq \infty$, $\lim\limits_{P\to\infty} S^{\mathrm{P}}_{\mathrm{Sto}} = R$ ($R \in Q^+$) holds. Then, $\forall \varepsilon > 0$, $\exists E \in Z+$, which makes $|S^{\mathrm{P}}_{\mathrm{Sto}} - R| < \varepsilon$ when $P > E$. Thus, $R - \varepsilon < S^{\mathrm{P}}_{\mathrm{Sto}} < R + \varepsilon$. Therefore, $S^{\mathrm{P}}_{\mathrm{Sto}}$ has an upper bound. Based on the supremum principle (Rudin, 1976), the supremum of $S^{\mathrm{P}}_{\mathrm{Sto}}$ exists, and the storage wall exists for program Q owing to Definition 2.

An important implication of Theorem 1 is as follows. If $\lim\limits_{P\to\infty} S_P \neq \infty$, $\lim\limits_{P\to\infty} S^{\mathrm{P}}_{\mathrm{Sto}} \neq \infty$ when $S^{\mathrm{P}}_{\mathrm{Sto}} < S_P$. Thus, there always exists the storage wall. For example, Amdahl's speedup meets this condition and will always have the storage wall. So, in the rest of this paper, we consider the condition $\lim\limits_{P\to\infty} S_P = \infty$ and replace S_P with $S_{\text{Gustafson}}$.

4.3 System classification

According to Eq. (8), it is easy to observe that $O(P)$ is the key factor in determining the existence of the storage wall. Due to different $O(P)$, the system storage-bounded speedup has different trends. To better analyze the laws of the storage wall, we categorize parallel systems into two types according to the characteristic of $O(P)$.

Before the classification, we introduce the following symbol conventions:

1. $f(x) \succ g(x)$ if $\lim\limits_{x\to\infty} f(x)/g(x) = \infty$, and $f(x) \succeq g(x)$ if $\lim\limits_{x\to\infty} f(x)/g(x)$ is a positive constant or ∞.

2. $f(x) \prec g(x)$ if $\lim\limits_{x\to\infty} f(x)/g(x) = 0$, and $f(x) \preceq g(x)$ if $\lim\limits_{x\to\infty} f(x)/g(x)$ is a non-negative constant.

3. Suppose $\Theta(x)$ is a set of all functions over x, $f(x) \in \Theta(x)$ and $g(x) \in \Theta(x)$. If $\lim\limits_{x\to\infty} f(x)/g(x)$ is a positive constant, then $f(x) = \Theta(g(x))$ or $g(x) = \Theta(f(x))$.

Definition 3 (Constant and incremental systems) Suppose that a program Q satisfying $\lim\limits_{P\to\infty} S_P = \infty$ runs on a system W. If $O(P) \preceq \Theta(1)$, system W is considered to be a constant system. If $O(P) \succ \Theta(1)$, system W is considered to be an incremental system.

Fig. 2 shows three examples for the classification based on $O(P)$. If $O(P) = K \preceq \Theta(1)$, where K is a positive constant, system W is a constant system. According to Eq. (10), we find that

$$\lim_{P\to\infty} S^{\mathrm{P}}_{\text{Gustafson-Sto}} = \lim_{P\to\infty} (f + P(1-f)) \frac{1}{1+K} = \infty.$$

By Theorem 1, there is no storage wall in this system. If $O(P) = KP\ln P \succ \Theta(1)$, system W is an incremental system. Similarly, we find that

$$\begin{aligned} &\lim_{P\to\infty} S^{\mathrm{P}}_{\text{Gustafson-Sto}} \\ &= \lim_{P\to\infty} (f + P(1-f)) \frac{1}{1+KP\ln P} = 0. \end{aligned}$$

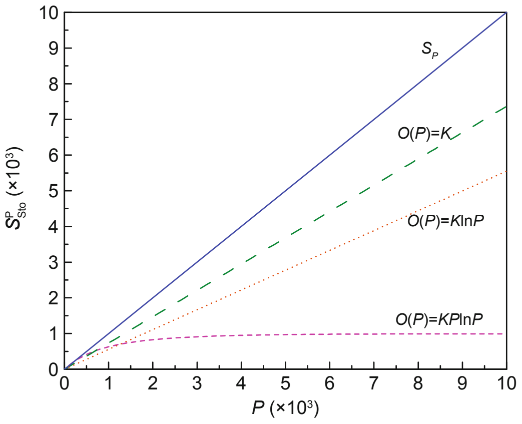

Fig. 2 Three examples for storage-bounded speedup and the storage wall of different types of systems based on $O(P)$ (K is a constant in each function). When $O(P) = K \preceq \Theta(1)$, W is a constant system and has no storage wall; when $O(P) = K \ln P \succ \Theta(1)$, W is an incremental system and has no storage wall; when $O(P) = KP \ln P \succ \Theta(1)$, W is an incremental system and has a storage wall

In this situation, the storage wall exists.

Now we introduce another existence theorem of the storage wall as follows:

Theorem 2 When a program Q that satisfies $\lim\limits_{P\to\infty} S_P = \infty$ runs on a system W, the storage wall exists if and only if $O(P) \succeq S_P$.

Proof According to Theorem 1, if the storage wall exists, $\lim\limits_{P\to\infty} S^{\mathrm{P}}_{\mathrm{Sto}} = \lim\limits_{P\to\infty} S_P/(1 + O(P)) \neq \infty$. Obviously, $O(P) \succeq S_P$ holds.

If $O(P) \succeq S_P$, then $\lim\limits_{P\to\infty} S^{\mathrm{P}}_{\mathrm{Sto}} = \lim\limits_{P\to\infty} S_P/(1 + O(P)) < \infty$. Thus, $\lim\limits_{P\to\infty} S^{\mathrm{P}}_{\mathrm{Sto}} \neq \infty$. So, the storage wall exists for program Q on system W by Theorem 1.

Thus, for a constant system which satisfies $\lim\limits_{P\to\infty} S_P = \infty$, the storage wall does not exist because $O(P) \preceq \Theta(1) \prec S_P$.

On the other hand, for an incremental system, there are two cases. If $O(P) \succeq S_P$, the storage wall exists by Theorem 2. However, if $\Theta(1) \prec O(P) \prec S_P$, the storage wall does not exist, which is similar to a constant system.

5 System analyses according to I/O architecture

The architecture determines the fundamentals of the supercomputer's hardware and software, and also the scalability of the supercomputer. In this section, we analyze the storage wall in terms of storage architecture, while we discuss the storage wall challenge based on large-scale parallel applications in Section 6. To be specific, the factors that impact the storage wall are analyzed in Section 5.1 and the storage wall related to different storage architectures are investigated in Section 5.2.

5.1 Factor analysis of the storage wall

In Section 3, we presented four I/O characteristics of scientific applications, i.e., periodicity, burstiness, synchronization, and repeatability. Based on these characteristics, we make the following assumptions to identify the key factors that influence the existence of the storage wall:

1. Suppose that the P processes of program Q running on system W are synchronized and analogous, and this is in line with the application characteristics detailed in Section 3. Meanwhile, the compute nodes of most supercomputers have the same configurations. So, the P processes running on each node have approximately the same I/O behavior.

2. Suppose that the parallel workload including compute and I/O loads will increase proportionally with the number of processes. Ideally, the load of compute phases approximates to be constant for each process even if the number of processes increases. So, we assume both I_i^{pint} and I^{sint} to be equivalent in the following analysis.

3. Suppose that the memory hit rate on every processor is approximately equal, noted as α. The average data amount of I/O phases for each process is considered to be approximately equal, denoted by D. The average bandwidth for memory access on each node is approximately equal, noted as V_{mem}. On the other hand, the I/O bandwidth on each compute node is approximately the same when the memory access misses, denoted by $V_{\text{out-mem}}$.

Many representative applications, e.g., those in earth science, nuclear physics, and climatology, are in line with the above assumptions. These applications consist of nearly identical jobs running on the compute nodes of identical architectures (Carns *et al.*, 2011). Besides, the checkpoint/restart mechanism as a synchronous I/O workload also fully satisfies the above assumptions. This mechanism is used to protect parallel applications against failure by storing the application state to persistent storage. Even if a failure occurs, the application still restarts from the most recent checkpoint (Bent *et al.*, 2009).

By the above assumptions, from the view of a compute node, we transform the storage workload factor into

$$\begin{aligned}O(P) &= \frac{N-M}{1+M} = \frac{I^{\text{pser}}/I^{\text{pint}} - I^{\text{sser}}/I^{\text{sint}}}{1+I^{\text{sser}}/I^{\text{sint}}}\\ &= \frac{I^{\text{sint}}(I^{\text{pser}}/I^{\text{pint}}) - I^{\text{sser}}}{I^{\text{sint}}+I^{\text{sser}}}\\ &= \frac{I^{\text{pser}} - I^{\text{sser}}}{I^{\text{sint}}+I^{\text{sser}}} = \frac{1}{C}(I^{\text{pser}} - I^{\text{sser}})\\ &= \frac{1}{C}\left(\frac{D}{V_{\text{mem}}} + \frac{(1-\alpha)D}{V_{\text{out-mem}}} - I^{\text{sser}}\right), \qquad (11)\end{aligned}$$

where $C = I^{\text{sint}} + I^{\text{sser}}$ is a positive constant.

Hence, we obtain

$$\begin{aligned}S_{\text{Sto}}^{\text{P}} &= S_P\frac{1}{1+O(P)}\\ &= S_P\frac{1}{\dfrac{1}{C}\left(\dfrac{D}{V_{\text{mem}}} + \dfrac{(1-\alpha)D}{V_{\text{out-mem}}} + I^{\text{sint}}\right)}. \qquad (12)\end{aligned}$$

Based on Eq. (12) and Theorems 1 and 2, we derive the following corollary:

Corollary 1 When a program Q that satisfies $\lim_{P\to\infty} S_P = \infty$ runs on a system W, if $(1-\alpha)D/V_{\text{out-mem}} \succeq \Theta(P)$, the storage wall exists.

Proof In Eq. (12), C, I^{sint}, and V_{mem} are constants. Let S_P be $S_{\text{Gustafson}} = \Theta(P)$. If $(1-\alpha)D/V_{\text{out-mem}} \succeq \Theta(P)$, then $P/((1-\alpha)D/V_{\text{out-mem}})$ has a supremum. Since Eq. (12) can be converted to

$$S_{\text{Sto}}^{\text{P}} = \Theta\left(\frac{P}{\dfrac{1}{C}\left(\dfrac{D}{V_{\text{mem}}} + \dfrac{(1-\alpha)D}{V_{\text{out-mem}}} + I^{\text{sint}}\right)}\right),$$

$S_{\text{Sto}}^{\text{P}}$ also has a supremum. According to Definition 2, when $(1-\alpha)D/V_{\text{out-mem}} \succeq \Theta(P)$ holds, the storage wall exists.

According to Corollary 1, when a program Q satisfying $\lim_{P\to\infty} S_P = \infty$ runs on a system W, the factors affecting the storage wall can be summarized as follows:

1. D and I^{sint}: D represents the average amount of data for the I/O phases for all processes. We use D_i to denote the average amount of data for the I/O phases on the ith node, and thus $D = \sum_{i=0}^{P-1} D_i/P$. According to assumptions, I^{sint} represents the average time of the compute phases no matter how the program runs (serially or in parallel). D and I^{sint} reflect the I/O characteristics of parallel programs, which distinguish compute-intensive programs with I/O-intensive ones. Table 2 shows the basic characteristics of compute-intensive and I/O-intensive programs related to D and I^{sint}. The larger the D and the smaller the I^{sint}, the more intensive the I/O workload.

Table 2 I/O characteristics of parallel programs

D	I^{sint}	Compute-intensive	I/O-intensive
Large	Large	√	√
Small	Small	√	√
Large	Small		√
Small	Large	√	

2. α and V_{mem}: α represents the average memory hit rate on every processor when the process performs I/O operations. It essentially implies the hierarchical storage architecture. V_{mem} represents the average bandwidth of the memory access on each node. According to the performance of dynamic random access memory (DRAM), memory provides better access bandwidth but confronts data volatility and small capacity due to the high price. Improving the memory hit rate is a hot topic for enhancing I/O performance. Many techniques such as data prefetching and cooperative caching are proposed to achieve this purpose. Meanwhile, a number of new media, such as FLASH and PCM, have been developed to replace DRAM or to join together with DRAM to compose hybrid memory. Both advanced media and advanced memory architecture can improve V_{mem}.

3. $V_{\text{out-mem}}$: It represents the average I/O bandwidth of each processor when the I/O requests miss in the memory. The average I/O access bandwidth is the ratio between the I/O data size and the I/O request time, which starts from initiation and ends with completion. $V_{\text{out-mem}}$ plays a fundamental role in the storage wall because it is closely associated with storage devices, storage architecture, and I/O access patterns. For different storage architectures, the impact factors of $V_{\text{out-mem}}$ contain metadata management, data layout, and data management strategies, to name a few. For different I/O access patterns, $V_{\text{out-mem}}$ is influenced by I/O request size, access density, burstiness, synchronization, and so on. As the processors scale up, the overhead for metadata access, communication collision, and data

access competition will increase remarkably, and this leads to a significant decrease of $V_{\text{out-mem}}$.

5.2 Storage wall analysis related to main supercomputer I/O architectures

According to the interconnecting relationship between compute and storage in supercomputer systems, the architecture of supercomputer storage systems can be divided into four categories: centralized architecture, distributed and parallel architecture, centralized, distributed, and parallel (CDP) architecture, and CDP architecture with burst buffers. Next, we will analyze the storage wall characteristics with each I/O architecture.

5.2.1 Centralized I/O architecture

Fig. 3 shows a centralized I/O architecture, such as network attached storage (NAS). This architecture is often used in small-scale supercomputers, and it is easily configured and managed, but has a poor scalability. It can merely scale up instead of scaling out because the storage system cannot horizontally expand to multiple servers but can vertically enhance the performance of a single server. Therefore, the I/O architecture severely limits the performance of the storage system. Suppose that the bandwidth provided by the storage system is M_{S}. Although M_{S} concerns different I/O access patterns, there always exists a positive constant M so that $M_{\text{S}} < M$. In synchronous I/O access mode, as P increases, we obtain $V_{\text{out-mem}} \preceq \Theta(M/P)$. Since M is a constant factor, we can omit it and simplify the above formula to $V_{\text{out-mem}} \preceq \Theta(1/P)$. According to Eq. (11), we obtain $O(P) \succeq \Theta(P)$. Then, we can find that the supercomputer with the centralized I/O architecture is an incremental system by Definition 3. Meanwhile, when $S_P = \Theta(P)$ (S_P is replaced by $S_{\text{Gustafson}}$), according to Theorem 2, we can obtain $O(P) \succeq S_P$ and this demonstrates that the storage wall exists in the supercomputer with centralized I/O architecture. Fig. 4 shows a case of storage wall for a supercomputer with a centralized I/O architecture.

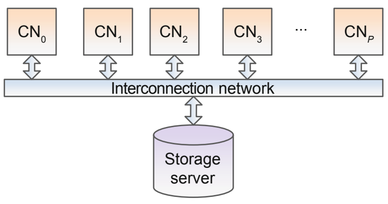

Fig. 3 Centralized I/O architecture (CN: compute node)

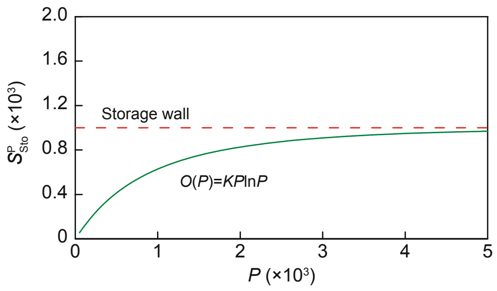

Fig. 4 A case of a supercomputer with a centralized I/O architecture in which the storage wall exists

5.2.2 Distributed and parallel I/O architecture

Fig. 5 shows a system with a distributed and parallel I/O architecture. This supercomputer storage system usually refers to the architecture in which each compute node has built-in storage or a direct-attached storage server. Each compute node has a file system, which can provide I/O service to itself and other compute nodes. Due to the problems of I/O scheduling and global data consistency, the implementations of the parallel file system are complicated. In addition, the storage servers have different positions for the compute nodes which are direct-attached or non-direct-attached; therefore, system load balance also becomes a problem. Hence, this architecture is usually adopted by supercomputers in small or medium scale.

For distributed and parallel I/O architecture, the storage nodes and compute nodes scale out equally. Thus, the storage server provides approximately fixed bandwidth for each compute node, that is, $V_{\text{out-mem}} = \Theta(1)$. According to Eq. (11), we obtain $O(P) = \Theta(1)$. As shown in Fig. 6, the supercomputer with a distributed and parallel I/O architecture belongs to the constant system and has a non-existent storage wall according to Definition 3.

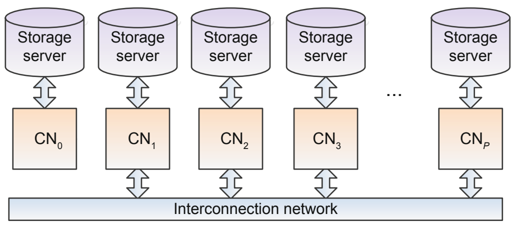

Fig. 5 Distributed and parallel I/O architecture (CN: compute node)

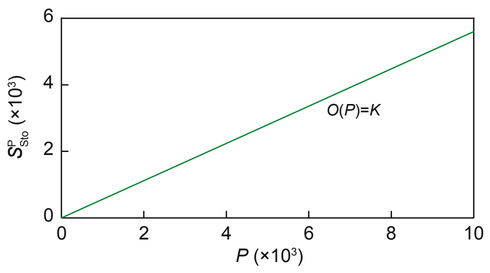

Fig. 6 A case of a supercomputer with a distributed and parallel I/O architecture in which the storage wall does not exist

5.2.3 Centralized, distributed, and parallel architecture

Fig. 7 shows the CDP architectures which dominate in TOP500 supercomputers (TOP500, 2015). Table 3 details the fastest 10 supercomputers in TOP500 whose storage architectures are all CDP architecture. For medium-sized supercomputers (the number of compute nodes reaches 10^3 to 10^4), compute nodes and storage servers are connected directly through an interconnection network (Fig. 7a). A parallel storage system consists of storage servers usually equipped with a parallel file system, such as Lustre and PVFS, while compute nodes access the storage system by the client of a parallel file system. The typical systems include Titan, Tianhe-1A, and so on.

With the development in supercomputers, I/O nodes (IONs) are inserted between compute nodes and parallel storage systems to provide the I/O forwarding and management functions (Fig. 7b). On the one hand, the number of compute nodes can increase continually without the limitation of the client number of parallel file systems. On the other hand, I/O performance can be improved by I/O scheduling and caching. All of these are attributed to the introduction of IONs in the systems. The typical systems are the IBM Bluegene series and Tianhe-2. Although the system has been improved by the I/O nodes, the essence of the I/O architecture remains unchanged. Thus, to simplify the model, the I/O node layer is omitted in subsequent analysis.

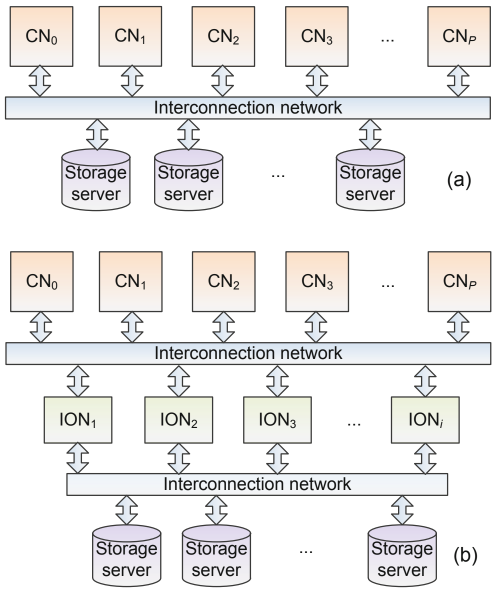

Fig. 7 Centralized, distributed, and parallel (CDP) architectures: (a) without I/O nodes; (b) with I/O nodes (CN: compute node; ION: I/O node)

In the CDP architecture, the storage system can not only scale up but also scale out. However, these systems still have some limitations: (1) For a specific supercomputer the scale of the storage system is fixed within a certain time period; (2) The parallel file system usually has ceilings on the number of storage nodes, aggregate performance, and storage capacity. Here we assume that the aggregate bandwidth of the parallel file system is M_S, and there always exists a positive constant M so that $M_S < M$. As P increases, the resource contention and access

Table 3 Storage architecture of the fastest 10 computers in the TOP500 (TOP500, 2015)

TOP500	System	Number of cores	Storage architecture	File system	I/O nodes used?
1	Tianhe-2	3 120 000	CDP	Hybrid hierarchy file system (H^2FS)	Yes
2	Titan	560 640	CDP	Lustre	No
3	Sequoia	1 572 864	CDP	Lustre + zettabyte file system (ZFS)	Yes
4	K computer	705 024	CDP	Fujitsu exabyte file system (FEFS)	No
5	Mira	786 432	CDP	General parallel file system (GPFS)	Yes
6	Piz Daint	115 984	CDP	Lustre	No
7	Shaheen II	196 608	CDP	Lustre	No
8	Stampede	462 462	CDP	Lustre	No
9	JUQUEEN	458 752	CDP	GPFS	Yes
10	Vulcan	393 216	CDP	Lustre + ZFS	Yes

conflict of the storage system are seriously aggravated. Thus, in synchronous I/O mode for one compute node, we obtain $V_{\text{out-mem}} \preceq \Theta(M/P)$. Since M is a constant factor, we can omit it and simplify the above formula to $V_{\text{out-mem}} \preceq \Theta(1/P)$. According to Eq. (11), we obtain $O(P) \succeq \Theta(P)$. Then we can find that the supercomputer with the CDP architecture is an incremental system by Definition 3. Meanwhile, when $S_P = \Theta(P)$, according to Theorem 2, we obtain $O(P) \succeq S_P$ and it demonstrates that the storage wall exists in the supercomputer with the CDP architecture. Fig. 8 shows a case for a supercomputer with the CDP architecture.

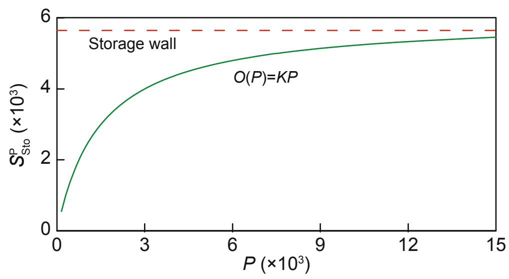

Fig. 8 A case for a supercomputer with a centralized, distributed, and parallel (CDP) architecture in which the storage wall exists

5.2.4 Centralized, distributed, and parallel architecture with burst buffers

The I/O workloads in scientific applications are often characterized by periodicity and burstiness, especially for write operations, and this provides an opportunity to use the write buffer. Burst buffers (Liu *et al.*, 2012; Sisilli, 2015; Wang *et al.*, 2014; 2015) are large high-performance write buffering spaces inserted as a tier of the supercomputer storage using solid state drives (SSDs), non-volatile random access memory (NVRAM), and so on. By using burst buffers, scientific applications can flush data to a high-performance temporary buffer, and then overlap computations that follow the I/O bursts with writing data from the burst buffer back to external storage. This buffering strategy can remove expensive parallel file system (PFS) write operations from the critical path of execution.

Under the CDP storage architecture, for the system without the I/O forwarding layer (like Titan), burst buffer nodes typically are located between the compute nodes and the PFS (Fig. 9a) (Wang *et al.*, 2014; 2015). For the system with an I/O forwarding layer (IBM Bluegene series) (Moreira *et al.*, 2006; Ali *et al.*, 2009), the burst buffers can be inserted into the I/O nodes, so this burst buffer layer is located in the same level as the I/O forwarding layer (Fig. 9b) (Liu *et al.*, 2012). Despite these different designs and implementations, the burst buffers have the same functionality and usage in absorbing the write activity. Below we use the work of Liu *et al.* (2012) as a case to study the burst buffer related architecture, and other works undergo the analogue analysis procedure.

According to the hierarchical CDP with a burst buffer architecture, Eqs. (11) and (12) can be rewritten as

$$\begin{aligned} O(P) &= \frac{1}{C}(I^{\text{pser}} - I^{\text{sser}}) \\ &= \frac{1}{C}\left(\frac{D}{V_{\text{mem}}} + \frac{\beta D}{V_{\text{burst}}} + \frac{(1-\alpha-\beta)D}{V_{\text{out-mem}}} - I^{\text{sser}}\right) \end{aligned} \tag{13}$$

and

$$\begin{aligned} S^{\text{P}}_{\text{Sto}} &= S_P \frac{1}{1+O(P)} \\ &= S_P \frac{1}{\frac{1}{C}\left(\frac{D}{V_{\text{mem}}} + \frac{\beta D}{V_{\text{burst}}} + \frac{(1-\alpha-\beta)D}{V_{\text{out-mem}}} + I^{\text{sint}}\right)}. \end{aligned} \tag{14}$$

Here, $C = I^{\text{sint}} + I^{\text{sser}}$ is a positive constant, β is defined as a data absorbing ratio, and thus βD represents the amount of data that can be absorbed by the burst buffers for each node. V_{burst} denotes the average bandwidth for each node provided by the burst buffers. Since the burst buffers form an intermediate layer with the I/O nodes, they scale out with compute nodes in a fixed proportion horizontally (for example, the ratio between the number of compute nodes and the number of I/O nodes is 64:1). So, we can assume that β and V_{burst} are approximately equal on every processor as the processor number increases.

Likewise, we can achieve a similar conclusion to Corollary 1:

Corollary 2 When a program Q that satisfies $\lim_{P\to\infty} S_P = \infty$ runs on a system W (CDP with burst buffers), if $D(1-\alpha-\beta)/V_{\text{out-mem}} \succeq \Theta(P)$, the storage wall exists.

The proof is omitted here because it is similar to the proof of Corollary 1. Based on Corollary 2, it can be easily found that the storage wall can be pushed forward or diminished for a storage system with CDP architecture with burst buffers .

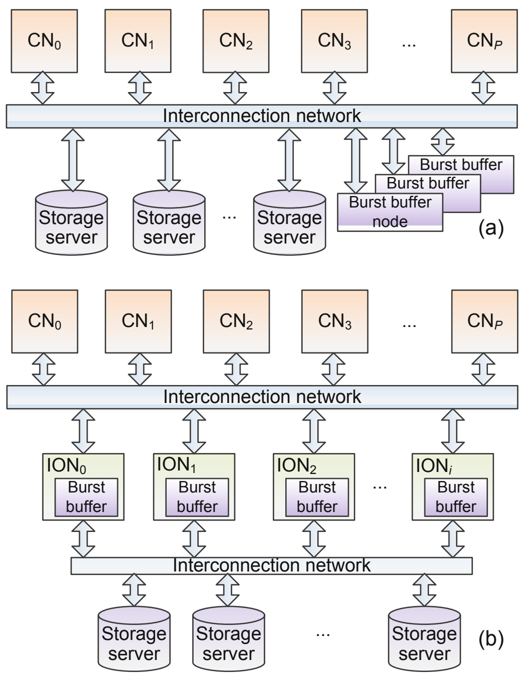

Fig. 9 Centralized, distributed, and parallel architecture with burst buffers without (a) and with (b) an I/O forwarding layer (CN: compute node; ION: I/O node)

The key factors to the effectiveness of burst buffers are V_{burst} and β. Burst buffers usually use nonvolatile media, which has better performance than disk-based storage. So, V_{burst} is a foundation such that burst buffers can play a role in dealing with the storage wall. This is closely related to the media itself. β is another key point, revealing how many write operations can be absorbed by burst buffers. It is closely related to the performance and capacity of burst buffers, and the I/O patterns of the applications. Burst buffers can accelerate only write operations rather than read operations.

Furthermore, they are also sensitive to the intensity of I/O write activity, the cost of flushing the burst buffers, and protocol interactions with the external storage system.

In the ideal case, if $\beta = 1 - \alpha$, an application Q has only write activity which can all be absorbed by the burst buffers. The storage-bounded speedup is not restrained by the external storage performance. The storage wall does not exist as the dashed line in Fig. 10 shows. However, in real circumstances, the storage wall can be pushed forward because a lot of write operations are absorbed and hidden by the burst buffer layer as the solid line in Fig. 10 shows.

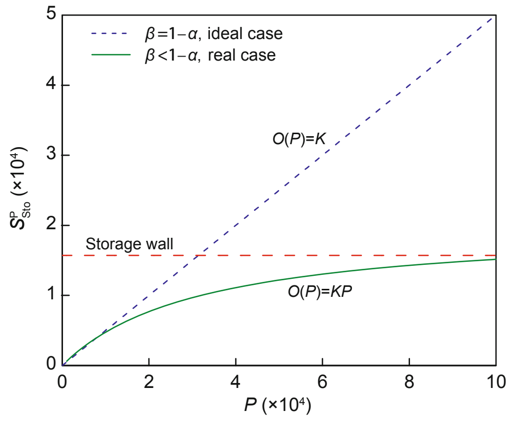

Fig. 10 Two cases for a supercomputer with a centralized, distributed, and parallel architecture with burst buffers. In the ideal case ($\beta = 1 - \alpha$), an application Q has only write activity which can all be absorbed by the burst buffers and the storage wall does not exist (dashed line); in real circumstances ($\beta < 1 - \alpha$), the storage wall can be pushed forward because a lot of write operations are absorbed and hidden by the burst buffer layer (solid line)

6 Experiments

In our experiments, we used typical benchmarks and applications to verify, analyze, and extrapolate the existence of the storage wall for the current representative systems and forthcoming exascale systems. The experiments have two goals: the first one is to obtain the performance laws of storage systems by using benchmarks and verify the existence of the storage wall in a specific I/O architecture (Section 6.1), and the second one is to analyze the parallel application with checkpointing to verify the storage wall existence and seek the best application scale (Section 6.2). Our analyses also provide insights on how to mitigate storage wall effects in system design and optimize large-scale parallel applications.

6.1 Storage wall existence verification using an I/O benchmark

In this section, we concentrate on the typical parallel I/O operations including the write and read operations, and obtain a further grip on the performance trends for storage systems along with the increase in program parallelism. Moreover, we preliminarily find the existence of the storage wall in the experiments.

The test platform of our experiment is the Tianhe-1A supercomputer (National Supercomputing Center in Tianjin, NSCC-TJ, Table 4), which has

Table 4 Specifications of the Tianhe-1A (YH MPP)

Item	Configuration
CPU	Xeon X5670 6C 2.93 GHz
Number of compute nodes	7168
Memory	229 376 GB
Interconnect	Proprietary (optic-electronic hybrid fat-tree structure, point-to-point bandwidth 160 Gbps)
Storage file system	Lustre (Lustre × 4, total capacity 4 PB)
Operating system	Kylin Linux

a typical CDP architecture with the Lustre parallel file system (Fig. 7a).

We used the interleaved-or-random (IOR) high performance computing (HPC) benchmark (University of California, 2007) to generate the I/O operations. IOR can be used for testing the performance of parallel file systems using various access patterns and interfaces such as the portable operating system interface (POSIX), message passing interface (MPI)-IO, and hierarchical data format 5 (HDF5). Due to the production system, our experiments were unable to extend to the whole system. Thus, we made a Lustre pool including four object storage targets (OSTs) on the Lustre4 (a Lustre system of the Tianhe-1A), and this Lustre pool had no jobs during the maintenance time. The Lustre pool had the same configuration as the whole storage system and was a microcosm, which can reveal the performance trends of the large-scale systems with larger program parallelism as well.

In our experiments, first we obtained the scalability of the I/O performance for the typical I/O operations with the number of processes increasing. Then, we revealed the existence of the storage wall and related factors.

Since I/O requests of parallel applications have different sizes in the data blocks, we chose three typical block sizes, 4 KB, 2 MB, and 512 MB, which represented small, medium, and large I/O requests.

In the experiments, we ran IOR using the POSIX application programming interface (API) and scaled the number of processes from 1 to 160 where there was one client per process. We set a few configurations to avoid the impact of the cache by setting an O_DIRECT flag for POSIX to bypass the I/O buffers, and setting a reorderTasks flag to avoid the reading after writing cache.

6.1.1 Scalability of I/O performance

Figs. 11–13 show the write and read performance variation trends along with the scaling of the number of processes in synchronous mode. Although the I/O block size is different in the three figures, they show the same trends in I/O performance with the number of processes increasing.

Figs. 11a, 12a, and 13a consistently show that the aggregate bandwidth of the storage system initially keeps up with the increase in the number of processes, and then starts growing slowly or declining due to the performance limits and resource competition. Figs. 11b, 12b, and 13b show that the I/O bandwidth for each process has a sharply declining trend with relatively low value. Figs. 11c, 12c, and 13c consistently demonstrate that the overhead time for read or write operations on each process always grows at a faster pace and achieves a large difference over the storage-bounded speedup.

From the results above, it is easy to see that a storage system with CDP architecture, like the Tianhe-1A, has poor storage performance scalability, and falls behind the development of the computing power of a supercomputer.

6.1.2 Storage wall and the related factors

In this subsection, we reveal the existence of the storage wall in the system with a CDP architecture, and verify the impact factors. Fig. 14 shows the presence of the storage wall of the Tianhe-1A storage system based on Lustre. Fig. 14 reveals the storage-bounded speedup variation along with the application parallelism P from three aspects: I/O block size (4 KB, 2 MB, or 512 MB), I^{sint}, and D. For a CDP architecture like Lustre, the storage wall is subject to the intensity and amount of data from the I/O requests, which are represented by I^{sint} and D, respectively.

For the same D, the smaller I^{sint} leads to the slower storage-bounded speedup increment, so the storage wall will appear faster, so does the case in which D is larger with the same I^{sint}. Thus, we find that the storage wall for the CDP architecture will appear quickly when the applications have intense I/O requests and a large amount of I/O data, and this type of application is generally called an 'I/O-intensive application'.

Fig. 15 shows the impact of α on the storage

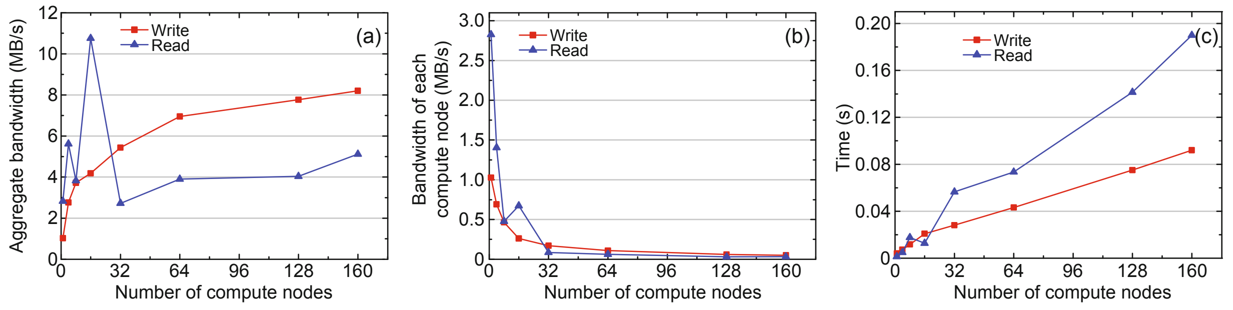

Fig. 11 Write and read operation performances when I/O block size is 4 KB: (a) aggregate bandwidth of the Lustre pool; (b) bandwidth per compute node; (c) time overhead per operation

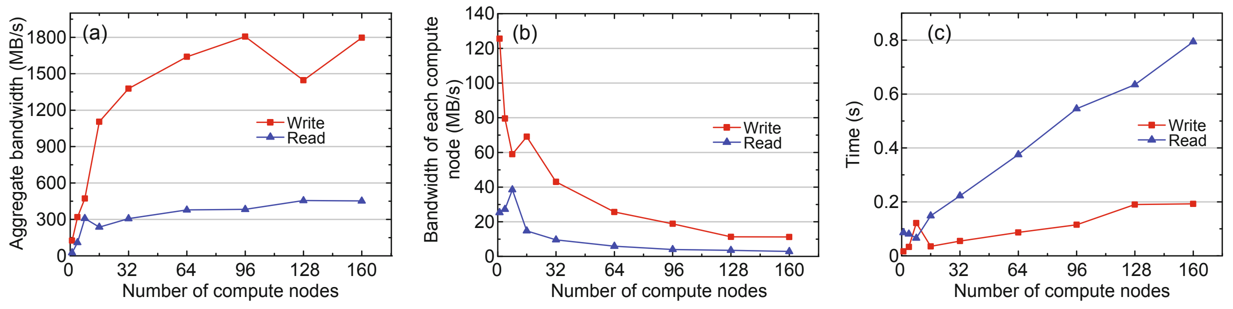

Fig. 12 Write and read operation performances when I/O block size is 2 MB: (a) aggregate bandwidth of the Lustre pool; (b) bandwidth per compute node; (c) time overhead per operation

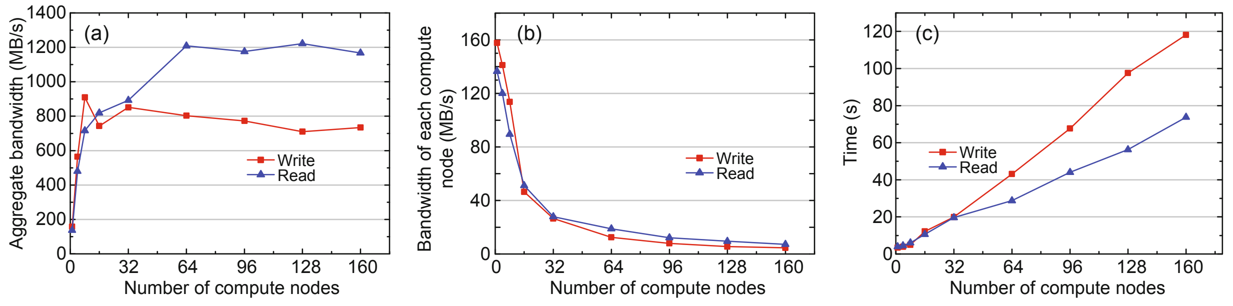

Fig. 13 Write and read operation performances when I/O block size is 512 MB: (a) aggregate bandwidth of the Lustre pool; (b) bandwidth per compute node; (c) time overhead per operation

wall. We can easily find that the storage wall can be alleviated by the improvement in memory hit rate, and this is in line with Eq. (12).

6.2 Application case study of storage wall

In this subsection, we use case studies to further detail how the application scalability is restrained by the storage performance and find the best application scale based on the storage wall. We use the representative cases that parallel applications with the checkpoint/restart mechanism run on the Cray XT4 Jaguar supercomputer. Jaguar is a typical supercomputer with a CDP architecture, equipped in the Oak Ridge National Laboratory. Although this supercomputer has been replaced by the Cray XK7 (renamed Titan), the storage architectures of these two systems are the same. The results based on Jaguar are still applicable to existing supercomputers with the CDP storage architecture.

Table 5 details the test-related configurations and Fig. 16 depicts the storage architecture of Jaguar (Alam *et al.*, 2007; Fahey *et al.*, 2008), in which OSS denotes the object storage server and MDS denotes the metadata server.

Checkpoint/restart is a widely used mechanism for fault-tolerance, and meanwhile it has become a driving I/O workload for supercomputer storage

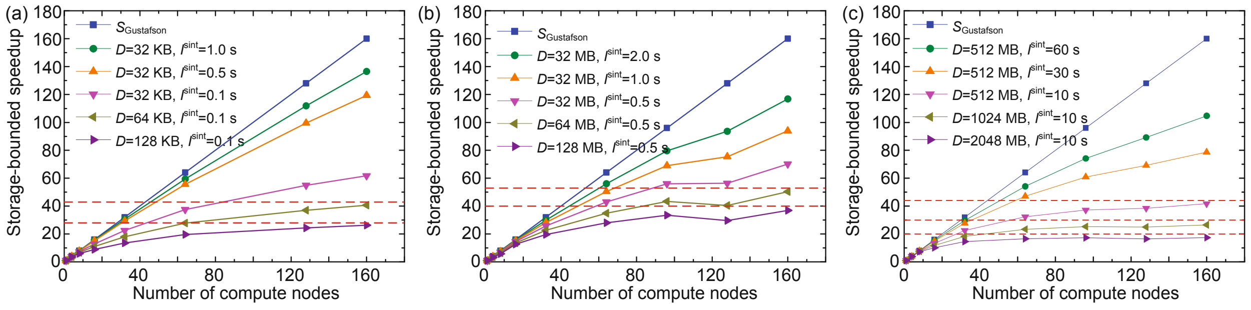

Fig. 14 Cases for the storage wall in centralized, distributed, and parallel (CDP) architecture in different block sizes, I^{sint}, and D: (a) block size = 4 KB; (b) block size = 2 MB; (c) block size = 512 MB

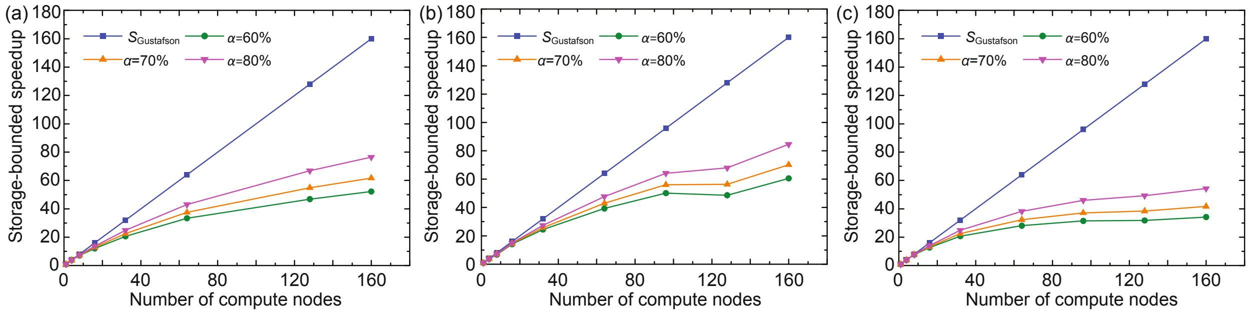

Fig. 15 The impact of α on the storage wall: (a) D=32 KB, I^{sint}=0.1 s, block size=4 KB; (b) D=2 MB, I^{sint}=0.5 s, block size=32 MB; (c) D=512 MB, I^{sint}=10.0 s, block size=512 MB

systems (Bent *et al.*, 2009).

For different checkpoint methods, parallel applications, and supercomputer configurations, the details of the checkpoint/restart mechanism are different, especially in the checkpoint data (Kalaiselvi and Rajaraman, 2000; Elnozahy *et al.*, 2002; Agarwal *et al.*, 2004; Elnozahy and Plank, 2004; Oldfield *et al.*, 2007; Bent *et al.*, 2009; Egwutuoha *et al.*, 2013). We consider parallel applications with system-level checkpointing techniques to evaluate system scalability under the storage performance constraint (Plank *et al.*, 1995; Agarwal *et al.*, 2004).

According to the checkpoint/restart mechanisms with different amounts of checkpoint data, we completed the analysis and comparison to obtain the storage wall laws.

In this subsection, we consider only the I/O caused by the checkpoint/restart mechanism and omit the I/O caused by the application itself, which may exacerbate the storage wall. As a production system, the test on the Tianhe-1A could not be extended to a larger scale. The Jaguar storage scaling tests were made specifically to a large scale by the researchers, and the follow-up work is based on the results from the Jaguar storage scaling tests (Alam *et al.*, 2007; Fahey *et al.*, 2008). The tests used the IOR benchmark in the MPI I/O mode with a constant buffer size (2 MB, 8 MB, 32 MB) per core, and

Table 5 Specifications of the Cray XT4 Jaguar system

Item	Configuration
CPU	2.6 GHz dual-core opteron
Number of processor sockets/cores	6296/12 592
Memory	2 GB/core
Interconnect	Cray SeaStar2
Disk controllers	18 DDN-9550 couplets
Number of OSSes/OSTs	72/144

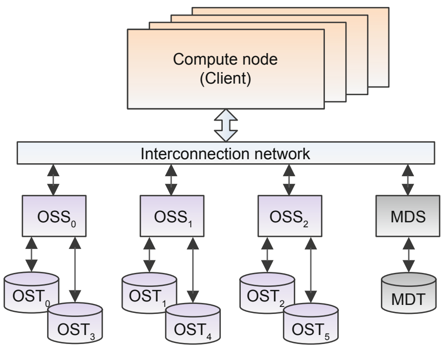

Fig. 16 Cray XT4 Jaguar storage architecture (OSS: objcet storage server; OST: object storage target; MDS: metadata server; MDT: metadata target)

they used two modes: one writing or reading with one file per client, and the other for a shared file. These results show the I/O aggregate bandwidth as the number of processes increases. Here we consider the $N-1$ checkpointing pattern (Bent *et al.*, 2009). Since the major operations of checkpointing are write operations, we use the results of the write experiments to a share file by multiple clients.

We use $V_{\text{out-mem-aggr}}$ to represent the aggregate bandwidth of the shared storage system. Since the data are transferred mainly between the memory and the shared storage system, both D/V_{mem} and α are zero. So, Eq. (12) can be simplified to

$$S_{\text{Sto}}^{\text{P}} = \frac{CS_P}{\dfrac{DP}{V_{\text{out-mem-aggr}}} + I^{\text{sint}}}, \tag{15}$$

where P stands for the number of parallel processes. Substituting $S_{\text{Gustafson}}$ for S_P, we obtain

$$S_{\text{Gustafson-Sto}}^{\text{P}} = \frac{C(f+(1-f)P)}{\dfrac{DP}{V_{\text{out-mem-aggr}}} + I^{\text{sint}}}. \tag{16}$$

For a large-scale parallel application, f is usually small and can be omitted. Then we can obtain

$$S_{\text{Gustafson-Sto}}^{P} = \frac{CP}{\dfrac{DP}{V_{\text{out-mem-aggr}}} + I^{\text{sint}}}. \tag{17}$$

To obtain the relationship between $V_{\text{out-mem-aggr}}$ and P when P increases, we use the Matlab curve fitting toolbox to analyze the storage scaling test results (Alam *et al.*, 2007; Fahey *et al.*, 2008). We find that two terms of the Gaussian function, i.e.,

$$\begin{aligned} &V_{\text{out-mem-aggr}} \\ &= a_1 \exp\left\{-\left(\frac{P-b_1}{c_1}\right)^2\right\} + a_2 \exp\left\{-\left(\frac{P-b_2}{c_2}\right)^2\right\}, \end{aligned} \tag{18}$$

can fit the $V_{\text{out-mem-aggr}}$ curve for different numbers of processes and give a small root mean square error (RMSE). Fig. 17 and Table 6 show the fitting curves of the scaling test data including three different buffer sizes (2 MB, 8 MB, 32 MB) per core. We find that the aggregate bandwidth of the storage system first rises to a maximum and eventually decreases to an asymptotic level. The curves in Fig. 17 accurately reflect the trend in the storage performance with a small error.

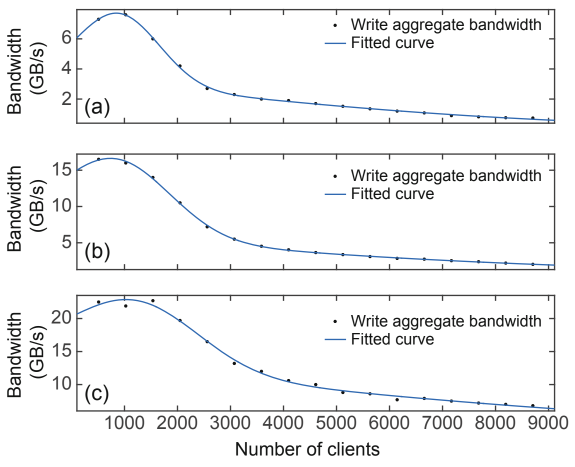

Fig. 17 Curve fitting of the storage system aggregate bandwidth by using two terms of the Gaussian function with the buffer size per core fixed at 2 MB (a), 8 MB (b), and 32 MB (c)

For different amounts of checkpoint data, parallel applications with a checkpoint/restart mechanism show a similar trend for the storage wall, but are to a different degree affected by the storage wall. Hereafter, we detail the storage wall of the cases for full- and incremental-memory checkpointing, while we use partial-memory (10% and 30% memory) checkpointing briefly for comparison.

1. Full-memory checkpointing. In this case, the entire memory context is saved at one checkpoint for a process. D is the size of the total memory. Since each dual-core node has 4 GB memory (Table 5), $D = 4\,\text{GB}$. Let P be the number of processes, I^{sint} the checkpointing interval, and I^{sser} the average I/O phase time of the process in serial mode. Here, we suppose that the I/O bandwidth is 500 MB/s for one node in serial mode, $I^{\text{sser}} = 4$ GB / 500 MB/s = 8 s. Then $S_{\text{Sto}}^{\text{P}}$ is

$$\begin{aligned} &S_{\text{G-Sto-fullckp}}^{\text{P}} \\ &= \frac{(I^{\text{sint}}+8)P}{\dfrac{4P}{a_1 \mathrm{e}^{-((P-b_1)/c_1)^2} + a_2 \mathrm{e}^{-((P-b_2)/c_2)^2}} + I^{\text{sint}}}. \end{aligned} \tag{19}$$

Fig. 18 shows the variation trends of $S_{\text{Sto}}^{\text{P}}$ in parallel applications with full-memory checkpointing on Jaguar. The results are based on three types of write aggregate bandwidth data (buffer sizes per core at 2 MB, 8 MB, and 32 MB, respectively). Fig. 19 displays the $S_{\text{Sto}}^{\text{P}}$ variation trends in three different checkpointing intervals (1, 3, and 6 h) under the write aggregate bandwidth data of buffer size per core at 32 MB.

Table 6 Parameters related to the curve fitting of storage system aggregate bandwidth using two terms of the Gaussian function

Buffer size	a_1	b_1	c_1	a_2	b_2	c_2	RMSE
2 MB	4.73	887.3	1094	4.342	-5.489×10^3	1.033×10^4	0.06902
8 MB	10.46	823	1471	6.991×10^{15}	-4.984×10^5	8.478×10^4	0.1212
32 MB	10.05	1191	1703	1.570×10^{16}	-7.925×10^5	1.346×10^5	0.4897

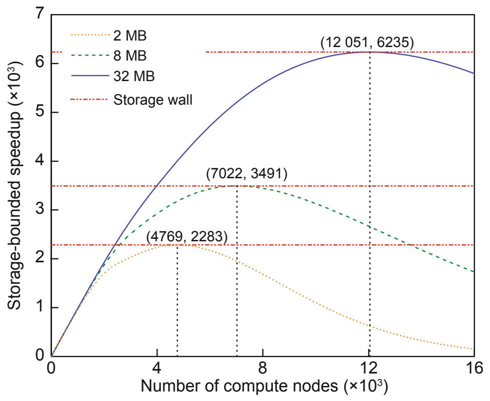

Fig. 18 $S^{\mathrm{P}}_{\mathrm{Sto}}$ variation trends in parallel applications with full-memory checkpointing on Jaguar for write aggregate bandwidths with buffer size per core at 2, 8, and 32 MB, respectively (checkpointing interval is 3 h)

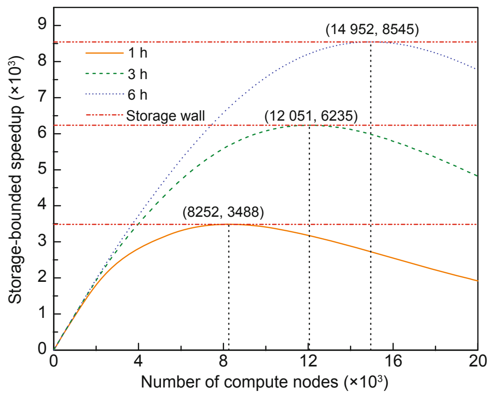

Fig. 19 $S^{\mathrm{P}}_{\mathrm{Sto}}$ variation trends in parallel applications with full-memory checkpointing on Jaguar, for checkpointing intervals of 1 h, 3 h, and 6 h, respectively (write aggregate bandwidth of buffer size per core at 32 MB)

2. Incremental-memory checkpointing. Incremental checkpointing attempts to reduce the amount of checkpoint data by saving only the differences from the last checkpoint (Agarwal *et al.*, 2004; Ferreira *et al.*, 2014). In this case, since the size of the first checkpoint data is the entire memory, the total size of all saved checkpoints is at least equal to that of the entire memory. Suppose that program Q runs on Jaguar for 20 days, and that the mean time to failure (MTTF) of the whole system is 10 days. We can obtain

$$D \geq \frac{\text{Fullmemory} + \left\lfloor \dfrac{\text{Runtime}}{\text{MTTF}} \right\rfloor \text{Fullmemory}}{\dfrac{\text{Runtime}}{I^{\text{sint}}} + \left\lfloor \dfrac{\text{Runtime}}{\text{MTTF}} \right\rfloor} = \frac{4 + \left\lfloor \dfrac{20}{10} \right\rfloor 4}{\dfrac{20}{I^{\text{sint}}} + \left\lfloor \dfrac{20}{10} \right\rfloor} = \frac{6}{\dfrac{10}{I^{\text{sint}}} + 1}.$$

According to Eqs. (17) and (18), and D described above, $S^{\mathrm{P}}_{\mathrm{Sto}}$ for incremental-memory checkpointing can be changed to

$$S^{\mathrm{P}}_{\text{G-Sto-incrckp}} \leq \frac{(I^{\text{sint}}+8)P}{\dfrac{\dfrac{6}{10/I^{\text{sint}}+1}P}{a_1\mathrm{e}^{-((P-b_1)/c_1)^2} + a_2\mathrm{e}^{-((P-b_2)/c_2)^2}} + I^{\text{sint}}}. \tag{20}$$

Fig. 20 shows the comparison of $S^{\mathrm{P}}_{\mathrm{Sto}}$ among applications with full-, partial-, and incremental-memory checkpointing under the write aggregate bandwidth data of buffer size per core at 32 MB. Applications with partial-memory checkpointing are the cases used to reveal the effect of checkpoint data size on the system scalability, and we use 10% and 30% of the entire memory as the checkpoint data size.

In Figs. 18–20, the storage-bounded speedup increases initially and starts to drop as soon as it hits the storage wall, i.e., the point P_{opt}. This point is called the 'optimal parallel scale' and implies that parallel applications can achieve the largest storage-bounded speedup in terms of the storage wall definition. For large-scale parallel applications on supercomputers with a CDP architecture, the storage wall may widely exist at the petascale level now and the exascale level in the future. By comparing different lengths of time intervals between I/O phases in Fig. 19, we find that the larger the compute proportion of the parallel application compared to I/O,

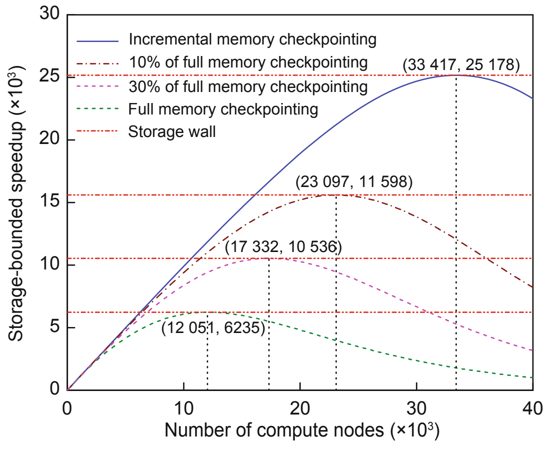

Fig. 20 Comparison of $S^{\mathrm{P}}_{\mathrm{Sto}}$ among applications with full-, partial-, and incremental-memory checkpointing with the write aggregate bandwidth data of buffer size per core at 32 MB (checkpointing interval is 3 h)

the more the appearance of the storage wall will be postponed.

By comparing applications with full-, partial-, and incremental-memory checkpointing, we find that reducing the data size of I/O operations can push the storage wall forward (Fig. 20). Based on the analysis results given in Eqs. (19) and (20), the system designer can increase $V_{\text{out-mem-aggr}}$ or I^{sint}, and decrease D, i.e., improve the bandwidth or optimize the computing and I/O ratio to push the storage wall forward.

All the analyses are in accordance with the impact factors of Section 5.1.

6.3 Mitigating storage wall effects

As demonstrated above, the storage wall may exist in large-scale supercomputers especially in forthcoming exascale supercomputing. Tackling this challenge will require a combined effort from storage system designers, file system developers, and application developers. Our storage wall theory can give researchers more insights into alleviating or even removing the challenge:

1. The storage-bounded speedup model clearly reveals the relationship between storage scalability and compute scalability. The storage wall quantifies the storage bottleneck as it describes the application scalability characteristics under the storage performance constraint.

2. Through the analysis of the storage wall, the key parameters that affect the system scalability, such as $V_{\text{out-mem}}$, D, I^{sint}, and α, should be taken into serious consideration. The system designer can push the storage wall forward by increasing the I/O bandwidth and the memory hit rate. For example, global shared cache across multiple compute nodes in memory can significantly increase the memory hit rate (Liao *et al.*, 2007; Frasca *et al.*, 2011; Sisilli, 2015).

3. Based on the analysis of I/O architecture using the storage wall theory, the new I/O architecture design and the usage method for new storage media can be guided by our theory to focus on solving the key factors that affect scalability. Hence, a scalable I/O system design, maybe a new kind of distributed I/O architecture, may overcome the storage wall limitation.

4. Now, the goal of parallel programming methods is to maximize the use of computing resources; however, this may engender the storage wall problem due to ignoring the storage constraint. It is imperative that new parallel programming models should focus on both the computing and I/O performances, and our storage wall theory is able to provide more recommendations.

7 Conclusions and future work

In this paper, we presented a storage-bounded speedup model to describe system scalability under the storage constraint and introduced for the first time the formal definition of a 'storage wall', which allows for the effects of the storage bottleneck on the scalability of parallel applications for large-scale parallel computing systems, particularly for those at peta/exascale levels.

We studied the storage wall theory through theoretical analyses, real experiments, and case studies using representative real-world supercomputing systems. The results verified the existence of the storage wall, revealed the storage wall characteristics of different architectures, and revealed the key factors that affect the storage wall. Based on our study, a balanced design can be achieved between the compute and storage system, and newly proposed storage architectures can be analyzed to determine whether the parallel scalability is constrained by the storage system or not. Our work enables researchers to push the storage wall forward by designing or improving storage architectures, applying I/O resource-oriented programming models, and so on.

In the future, our efforts will focus mainly on the following aspects. For the storage wall, we will refine the theory and explore alleviation parameters in detail. At the same time, we will present methods to optimize the I/O architecture and improve system flexibility.

References

Agarwal, S., Garg, R., Gupta, M.S., *et al.*, 2004. Adaptive incremental checkpointing for massively parallel systems. Proc. 18th Annual Int. Conf. on Supercomputing, p.277-286. http://dx.doi.org/10.1145/1006209.1006248

Agerwala, T., 2010. Exascale computing: the challenges and opportunities in the next decade. IEEE 16th Int. Symp. on High Performance Computer Architecture. http://dx.doi.org/10.1109/HPCA.2010.5416662

Alam, S.R., Kuehn, J.A., Barrett, R.F., *et al.*, 2007. Cray XT4: an early evaluation for petascale scientific simulation. Proc. ACM/IEEE Conf. on Supercomputing, p.1-12. http://dx.doi.org/10.1145/1362622.1362675

Ali, N., Carns, P.H., Iskra, K., *et al.*, 2009. Scalable I/O forwarding framework for high-performance computing systems. IEEE Int. Conf. on Cluster Computing and Workshops, p.1-10, http://dx.doi.org/10.1109/CLUSTR.2009.5289188

Amdahl, G.M., 1967. Validity of the single processor approach to achieving large scale computing capabilities. Proc. Spring Joint Computer Conf., p.483-485. http://dx.doi.org/10.1145/1465482.1465560

Bent, J., Gibson, G., Grider, G., *et al.*, 2009. PLFS: a checkpoint file system for parallel applications. Proc. Conf. on High Performance Computing Networking, Storage and Analysis, p.21. http://dx.doi.org/10.1145/1654059.1654081

Cappello, F., Geist, A., Gropp, B., *et al.*, 2009. Toward exascale resilience. *Int. J. High Perform. Comput. Appl.*, **23**(4):374-388. http://dx.doi.org/10.1177/1094342009347767

Carns, P., Harms, K., Allcock, W., *et al.*, 2011. Understanding and improving computational science storage access through continuous characterization. *ACM Trans. Stor.*, **7**(3):1-26. http://dx.doi.org/10.1145/2027066.2027068

Chen, J., Tang, Y.H., Dong, Y., *et al.*, 2016. Reducing static energy in supercomputer interconnection networks using topology-aware partitioning. *IEEE Trans. Comput.*, **65**(8):2588-2602. http://dx.doi.org/10.1109/TC.2015.2493523

Culler, D.E., Singh, J.P., Gupta, A., 1998. Parallel Computer Architecture: a Hardware/Software Approach. Morgan Kaufmann Publishers Inc., San Francisco, USA.

Egwutuoha, I.P., Levy, D., Selic, B., *et al.*, 2013. A survey of fault tolerance mechanisms and checkpoint/restart implementations for high performance computing systems. *J. Supercomput.*, **65**(3):1302-1326. http://dx.doi.org/10.1007/s11227-013-0884-0

Elnozahy, E.N., Plank, J.S., 2004. Checkpointing for peta-scale systems: a look into the future of practical rollback-recovery. *IEEE Trans. Depend. Secur. Comput.*, **1**(2):97-108. http://dx.doi.org/10.1109/TDSC.2004.15

Elnozahy, E.N., Alvisi, L., Wang, Y.M., *et al.*, 2002. A survey of rollback-recovery protocols in message-passing systems. *ACM Comput. Surv.*, **34**(3):375-408. http://dx.doi.org/10.1145/568522.568525

Fahey, M., Larkin, J., Adams, J., 2008. I/O performance on a massively parallel cray XT3/XT4. IEEE Int. Symp. on Parallel and Distributed Processing, p.1-12. http://dx.doi.org/10.1109/IPDPS.2008.4536270

Ferreira, K.B., Riesen, R., Bridges, P., *et al.*, 2014. Accelerating incremental checkpointing for extreme-scale computing. *Fut. Gener. Comput. Syst.*, **30**:66-77. http://dx.doi.org/10.1016/j.future.2013.04.017

Frasca, M., Prabhakar, R., Raghavan, P., *et al.*, 2011. Virtual I/O caching: dynamic storage cache management for concurrent workloads. Proc. Int. Conf. for High Performance Computing, Networking, Storage and Analysis, p.38. http://dx.doi.org/10.1145/2063384.2063435

Gamblin, T., de Supinski, B.R., Schulz, M., *et al.*, 2008. Scalable load-balance measurement for SPMD codes. Proc. ACM/IEEE Conf. on Supercomputing, p.1-12.

Gustafson, J.L., 1988. Reevaluating Amdahl's law. *Commun. ACM*, **31**(5):532-533. http://dx.doi.org/10.1145/42411.42415

Hargrove, P.H., Duell, J.C., 2006. Berkeley lab checkpoint/restart (BLCR) for Linux clusters. *J. Phys. Conf. Ser.*, **46**(1):494-499. http://dx.doi.org/10.1088/1742-6596/46/1/067

Hennessy, J.L., Patterson, D.A., 2011. Computer Architecture: a Quantitative Approach. Elsevier.

HPCwire, 2010. DARPA Sets Ubiquitous HPC Program in Motion. Available from http://www.hpcwire.com/2010/08/10/darpa_sets_ubiquitous_hpc_program_in_motion/.

Hu, W., Liu, G.M., Li, Q., *et al.*, 2016. Storage speedup: an effective metric for I/O-intensive parallel application. 18th Int. Conf. on Advanced Communication Technology, p.1-2. http://dx.doi.org/10.1109/ICACT.2016.7423395

Kalaiselvi, S., Rajaraman, V., 2000. A survey of checkpointing algorithms for parallel and distributed computers. *Sadhana*, **25**(5):489-510. http://dx.doi.org/10.1007/BF02703630

Kim, Y., Gunasekaran, R., 2015. Understanding I/O workload characteristics of a peta-scale storage system. *J. Supercomput.*, **71**(3):761-780. http://dx.doi.org/10.1007/s11227-014-1321-8

Kim, Y., Gunasekaran, R., Shipman, G.M., *et al.*, 2010. Workload characterization of a leadership class storage cluster. Petascale Data Storage Workshop, p.1-5. http://dx.doi.org/10.1109/PDSW.2010.5668066

Kotz, D., Nieuwejaar, N., 1994. Dynamic file-access characteristics of a production parallel scientific workload. Proc. Supercomputing, p.640-649. http://dx.doi.org/10.1109/SUPERC.1994.344328

Liao, W.K., Ching, A., Coloma, K., *et al.*, 2007. Using MPI file caching to improve parallel write performance for large-scale scientific applications. Proc. ACM/IEEE Conf. on Supercomputing, p.8. http://dx.doi.org/10.1145/1362622.1362634

Liu, N., Cope, J., Carns, P., *et al.*, 2012. On the role of burst buffers in leadership-class storage systems. IEEE 28th Symp. on Mass Storage Systems and Technologies, p.1-11. http://dx.doi.org/10.1109/MSST.2012.6232369

Liu, Y., Gunasekaran, R., Ma, X.S., *et al.*, 2014. Automatic identification of application I/O signatures from noisy server-side traces. Proc. 12th USENIX Conf. on File and Storage Technologies, p.213-228.

Lu, K., 1999. Research on Parallel File Systems Technology Toward Parallel Computing. PhD Thesis, National University of Defense Technology, Changsha, China (in Chinese).

Lucas, R., Ang, J., Bergman, K., *et al.*, 2014. DOE Advanced Scientific Computing Advisory Subcommittee (ASCAC) Report: Top Ten Exascale Research Challenges. USDOE Office of Science. http://dx.doi.org/10.2172/1222713

Miller, E.L., Katz, R.H., 1991. Input/output behavior of supercomputing applications. Proc. ACM/IEEE Conf. on Supercomputing, p.567-576. http://dx.doi.org/10.1145/125826.126133

Moreira, J., Brutman, M., Castano, J., *et al.*, 2006. Designing a highly-scalable operating system: the blue Gene/L story. Proc. ACM/IEEE Conf. on Supercomputing, p.53-61. http://dx.doi.org/10.1109/SC.2006.23

Oldfield, R.A., Arunagiri, S., Teller, P.J., *et al.*, 2007. Modeling the impact of checkpoints on next-generation systems. 24th IEEE Conf. on Mass Storage Systems and Technologies, p.30-46. http://dx.doi.org/10.1109/MSST.2007.4367962

Pasquale, B.K., Polyzos, G.C., 1993. A static analysis of I/O characteristics of scientific applications in a production workload. Proc. ACM/IEEE Conf. on Supercomputing, p.388-397. http://dx.doi.org/10.1145/169627.169759

Plank, J.S., Beck, M., Kingsley, G., *et al.*, 1995. Libckpt: transparent checkpointing under Unix. Proc. USENIX Technical Conf., p.18.

Purakayastha, A., Ellis, C., Kotz, D., *et al.*, 1995. Characterizing parallel file-access patterns on a large-scale multiprocessor. 9th Int. Parallel Processing Symp., p.165-172. http://dx.doi.org/10.1109/IPPS.1995.395928

Sisilli, J., 2015. Improved Solutions for I/O Provisioning and Application Acceleration. Available from http://www.flashmemorysummit.com/English/Collaterals/Proceedings/2015/20150811_FD11_Sisilli.pdf [Accessed on Nov. 18, 2015].

Rudin, W., 1976. Principles of Mathematical Analysis. McGraw-Hill Publishing Co.

Shalf, J., Dosanjh, S., Morrison, J., 2011. Exascale computing technology challenges. 9th Int. Conf. on High Performance Computing for Computational Science, p.1-25. http://dx.doi.org/10.1007/978-3-642-19328-6_1

Strohmaier, E., Dongarra, J., Simon, H., *et al.*, 2015. TOP500 Supercomputer Sites. Available from http://www.top500.org/ [Accessed on Dec. 30, 2015].

Sun, X.H., Ni, L.M., 1993. Scalable problems and memory-bounded speedup. *J. Parall. Distr. Comput.*, **19**(1): 27-37. http://dx.doi.org/10.1006/jpdc.1993.1087

University of California, 2007. IOR HPC Benchmark. Available from http://sourceforge.net/projects/ior-sio/ [Accessed on Sept. 1, 2014].

Wang, F., Xin, Q., Hong, B., *et al.*, 2004. File system workload analysis for large scale scientific computing applications. Proc. 21st IEEE/12th NASA Goddard Conf. on Mass Storage Systems and Technologies, p.139-152.

Wang, T., Oral, S., Wang, Y.D., *et al.*, 2014. Burstmem: a high-performance burst buffer system for scientific applications. IEEE Int. Conf. on Big Data, p.71-79. http://dx.doi.org/10.1109/BigData.2014.7004215

Wang, T., Oral, S., Pritchard, M., *et al.*, 2015. Development of a burst buffer system for data-intensive applications. arXiv:1505.01765. Available from http://arxiv.org/abs/1505.01765

Wang, Z.Y., 2009. Reliability speedup: an effective metric for parallel application with checkpointing. Int. Conf. on Parallel and Distributed Computing, Applications and Technologies, p.247-254. http://dx.doi.org/10.1109/PDCAT.2009.19

Xie, B., Chase, J., Dillow, D., *et al.*, 2012. Characterizing output bottlenecks in a supercomputer. Int. Conf. for High Performance Computing, Networking, Storage and Analysis, p.1-11. http://dx.doi.org/10.1109/SC.2012.28

Yang, X.J., Du, J., Wang, Z.Y., 2011. An effective speedup metric for measuring productivity in large-scale parallel computer systems. *J. Supercomput.*, **56**(2):164-181. http://dx.doi.org/10.1007/s11227-009-0355-9

Yang, X.J., Wang, Z.Y., Xue, J.L., *et al.*, 2012. The reliability wall for exascale supercomputing. *IEEE Trans. Comput.*, **61**(6):767-779. http://dx.doi.org/10.1109/TC.2011.106

Appendix: Derivation of Eq. (6)

In Section 4.1, the storage-bounded speedup is relaxed to an upper bound. Here we describe the derivation of the upper bound.

Suppose that

$$\begin{aligned} A &= \frac{T^{\mathrm{snon}} + N^{\mathrm{s}} I^{\mathrm{sser}}}{T^{\mathrm{pnon}} + N^{\mathrm{p}} I^{\mathrm{pser}}} \\ &= \frac{T^{\mathrm{snon}} + (T^{\mathrm{snon}}/I^{\mathrm{sint}}) I^{\mathrm{sser}} + I^{\mathrm{sser}}}{T^{\mathrm{pnon}} + (T^{\mathrm{pnon}}/I^{\mathrm{pint}}) I^{\mathrm{pser}} + I^{\mathrm{pser}}} \end{aligned}$$

and

$$\begin{aligned} B &= \frac{T^{\mathrm{snon}} + (N^{\mathrm{s}}-1) I^{\mathrm{sser}}}{T^{\mathrm{pnon}} + (N^{\mathrm{p}}-1) I^{\mathrm{pser}}} \\ &= \frac{T^{\mathrm{snon}} + (T^{\mathrm{snon}}/I^{\mathrm{sint}}) I^{\mathrm{sser}}}{T^{\mathrm{pnon}} + (T^{\mathrm{pnon}}/I^{\mathrm{pint}}) I^{\mathrm{pser}}}. \end{aligned}$$

The difference between B and A is

$$\begin{aligned} &\mathrm{Diff} = B - A \\ &= \frac{T^{\mathrm{snon}} + (N^{\mathrm{s}}-1) I^{\mathrm{sser}}}{T^{\mathrm{pnon}} + (N^{\mathrm{p}}-1) I^{\mathrm{pser}}} - \frac{T^{\mathrm{snon}} + N^{\mathrm{s}} I^{\mathrm{sser}}}{T^{\mathrm{pnon}} + N^{\mathrm{p}} I^{\mathrm{pser}}} \\ &= \frac{I^{\mathrm{pser}} T^{\mathrm{snon}} - I^{\mathrm{sser}} T^{\mathrm{pnon}} + (N^{\mathrm{s}} - N^{\mathrm{p}}) I^{\mathrm{pser}} I^{\mathrm{sser}}}{(T^{\mathrm{pnon}} + (N^{\mathrm{p}}-1) I^{\mathrm{pser}})(T^{\mathrm{pnon}} + N^{\mathrm{p}} I^{\mathrm{pser}})}. \end{aligned}$$

According to the assumptions in Section 4.1, the load is balanced on all nodes and the I/O load is in proportion to the compute load when the compute load increases with the increase of the number of processors.

Thus, N^{p} is approximately equal to N^{s} and T^{pnon} is less than or equal to T^{snon}. Due to storage performance degradation, I^{pser} is usually greater than I^{sser}, and this is verified by the experiments in Section 6. Thus, we have $(N^{\mathrm{s}}-N^{\mathrm{p}})I^{\mathrm{pser}}I^{\mathrm{sser}} \approx 0$ and $I^{\mathrm{pser}}T^{\mathrm{snon}} - I^{\mathrm{sser}}T^{\mathrm{pnon}} \geq 0$. Based on the analysis above, we find that the numerator of Diff is greater than or equal to zero, and the denominator is greater than zero. Therefore, B is greater than or equal to A, and B is an upper bound of A.

Suppose that $T_{\mathrm{Sto}}^{\mathrm{P}} = (T^{\mathrm{pnon}} + N^{\mathrm{p}}I^{\mathrm{pser}}) \approx (T^{\mathrm{pnon}} + (N^{\mathrm{p}} - 1)I^{\mathrm{pser}})$ and $T^{\mathrm{pnon}} \approx T^{\mathrm{snon}}$. We have

$$\mathrm{Diff} = B - A = \frac{(I^{\mathrm{pser}} - I^{\mathrm{sser}})T^{\mathrm{pnon}}}{(T_{\mathrm{Sto}}^{\mathrm{P}})^2}.$$

For long-term running applications,

$$(I^{\mathrm{pser}} - I^{\mathrm{sser}})T^{\mathrm{pnon}} \ll (T_{\mathrm{Sto}}^{\mathrm{P}})^2.$$

Hence, in Eq. (6), we set the upper bound as the value of storage-bounded speedup. Since Diff is negligible, this error will not affect the correctness of the results in this study.