FLORIDA STATE UNIVERSITY

COLLEGE OF ARTS AND SCIENCES

EXPLORING NOVEL BURST BUFFER MANAGEMENT ON EXTREME-SCALE HPC SYSTEMS

By

TENG WANG

A Dissertation submitted to the
Department of Computer Science
in partial fulfillment of the
requirements for the degree of
Doctor of Philosophy

2017

Teng Wang defended this dissertation on March 3, 2017.
The members of the supervisory committee were:

Weikuan Yu
Professor Directing Dissertation

Gordon Erlebacher
University Representative

David Whalley
Committee Member

Andy Wang
Committee Member

Sarp Oral
Committee Member

The Graduate School has verified and approved the above-named committee members, and certifies that the dissertation has been approved in accordance with university requirements.

ACKNOWLEDGMENTS

First and foremost, I would like to express my special thanks to my advisor Dr. Weikuan Yu for his ceaseless encouragement and continuous research guidance. I came to study in U.S. with little research background. At the beginning, I had a hard time to follow the classes and understand the research basics. While everything seemed unfathomable to me, Dr. Yu kept encouraging me to position myself better, and spent plenty of time talking with me on how to quickly adapt myself to the course and research environment in the university. I cannot imagine a better advisor on those moments we talked with each other. In the meanwhile, Dr. Yu spared no efforts in steering me towards the correct research directions and took every opportunity to expose me to the excellent research scholars, such as my mentors during my three summer internships. It was under Dr. Yu's generous help that I quickly built up all the background knowledge on file system and I/O, learned how to identify the cutting-edge research topics and conduct quality-driven research. I am fortunate to have Dr. Yu as my advisor.

In addition, I gratefully acknowledge the support and instructions I received from Dr. Sarp Oral and Dr. Bradley Settlemyer during my summer internship in Oak Ridge National Laboratory. I am also deeply indebted to Dr. Kathryn Mohror, Adam Moody, Dr. Kento Sato and Dr. Tanzima Islam for their infinite help during my two summer internships at Lawrence Livermore National Laboratory. Those moments I studied with these excellent research scholars have been permanently engraved in my memory.

I would also like to thank my committee members Dr. Gordon Erlebacher, Dr. David Whalley and Dr. Andy Wang for their time and comments to improve this dissertation.

I also appreciate the friendship with all the members in the Parallel Architecture and System Research Lab (PASL). I joined PASL in 2012 and was fortunate to know most of the members since the establishment of PASL (2009). During my PhD study, we worked together as a family and helped each other unconditionally. I'm particularly grateful to Dr. Yandong Wang, Dr. Bin Wang and Yue Zhu for their substantial help on the burst buffer projects, and Dr. Zhuo Liu, Dr. Hui Chen and Kevin Vasko for their assistance on the GEOS-5 project. I will also cherish my friendship with Dr. Cristi Cira, Dr. Jianhui Yue, Xiaobing Li, Huansong Fu, Fang Zhou, Xinning

Wang, Lizhen Shi, Hai Pham, and Hao Zou. With this friendship, I never felt lonely in my PhD study.

My deepest gratitude and appreciation go to my parents, my parents-in-law and my wife Dr. Mei Li. It's their everlasting love, encouragement and support that always warm my heart and illuminate the long journey for me to pursue my dreams.

The research topics in this dissertation are sponsored in part by the Office of Advanced Scientific Computing Research; U.S. Department of Energy and performed at the Oak Ridge National Laboratory, which is managed by UT-Battelle, LLC under Contract No. DE-AC05-00OR22725 and resources of the Oak Ridge Leadership Computing Facility, located in the National Center for Computational Sciences at Oak Ridge National Laboratory. They are also performed in part under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under Contract DE-AC52-07NA27344. They are also funded in part by an Alabama Innovation Award and National Science Foundation awards 1059376, 1320016, 1340947, 1432892 and 1561041.

TABLE OF CONTENTS

List of Figures . . . vii
Abstract . . . ix

1 Introduction **1**
1.1 Overview of Scientific I/O Workloads . . . 2
1.1.1 Checkpoint/Restart I/O . . . 2
1.1.2 Multi-Dimensional I/O . . . 4
1.2 Overview of Burst Buffers . . . 5
1.2.1 Representative Burst Buffer Architectures . . . 5
1.2.2 Burst Buffer Use Cases . . . 6
1.2.3 Storage Models to Manage Burst Buffers . . . 7
1.3 Research Contributions . . . 10
1.3.1 BurstMem: Overlapping Computation and I/O . . . 10
1.3.2 BurstFS: A Distributed Burst Buffer File System . . . 10
1.3.3 TRIO: Reshaping Bursty Writes . . . 11
1.3.4 Publication Contributions . . . 11
1.4 Dissertation Overview . . . 13

2 Problem Statement **14**
2.1 Increasing Computation-I/O Gap . . . 14
2.2 Issues of Contention on the Parallel File Systems . . . 15
2.2.1 Analysis of Degraded Storage Bandwidth Utilization . . . 16
2.2.2 Analysis of Prolonged Average I/O Time . . . 17
2.3 Summary . . . 19

3 BurstMem: Overlapping Computation and I/O **20**
3.1 Introduction . . . 20
3.2 Memcached Based Buffering Framework . . . 21
3.2.1 Overview of Memcached . . . 21
3.2.2 Challenges from Scientific Applications . . . 22
3.3 Design of BurstMem . . . 23
3.3.1 Software Architecture of BurstMem . . . 24
3.3.2 Log-Structured Data Organization with Stacked AVL Indexing . . . 25
3.3.3 Coordinated Shuffling for Data Flushing . . . 27
3.3.4 Enabling Native Communication Performance . . . 29
3.4 Evaluation of BurstMem . . . 31
3.4.1 Methodology . . . 31
3.4.2 Ingress Bandwidth . . . 32
3.4.3 Egress Bandwidth . . . 34
3.4.4 Scalability of BurstMem . . . 35
3.4.5 Case Study: S3D, A Real-World Scientific Application . . . 37

3.5 Related Work 38
3.6 Summary 40

4 **BurstFS: A Distributed Burst Buffer File System** **41**
4.1 Introduction 41
4.2 Design of BurstFS 43
4.2.1 Scalable Metadata Indexing 44
4.2.2 Co-Located I/O Delegation 47
4.2.3 Server-Side Read Clustering and Pipelining 49
4.3 Evaluation of BurstFS 51
4.3.1 Testbed 51
4.3.2 Overall Write/Read Performance 53
4.3.3 Performance Impact of Different Transfer Sizes 55
4.3.4 Analysis of Metadata Performance 56
4.3.5 Tile-IO Test 57
4.3.6 BTIO Test 57
4.3.7 IOR Test 58
4.4 Related Work 59
4.5 Summary 61

5 **TRIO: Reshaping Bursty Writes** **63**
5.1 Introduction 63
5.2 Design of TRIO 64
5.2.1 Main Idea of TRIO 64
5.2.2 Server-Oriented Data Organization 66
5.2.3 Inter-Burst Buffer Ordered Flush 68
5.3 Implementation and Complexity Analysis 70
5.4 Evaluation of TRIO 70
5.4.1 Overall Performance of TRIO 71
5.4.2 Performance Analysis of TRIO 72
5.4.3 Alleviating Inter-Node Contention 73
5.4.4 Effect of TRIO under a Varying OST Count 74
5.4.5 Minimizing Average I/O Time 75
5.5 Related Work 77
5.6 Summary 79

6 **Conclusions** **80**

7 **Future Work** **83**

Bibliography 85

Biographical Sketch 95

LIST OF FIGURES

1.1 Checkpoint/restart I/O patterns (adapted from [32]). 3

1.2 I/O access patterns with multi-dimensional variables. 4

1.3 An overview of burst buffers on HPC system. BB refers to Burst Buffer. PE refers to Processing Element. CN refers to Compute Node. 6

2.1 Issues of I/O Contention. 15

2.2 The impact of process count on the bandwidth of a single OST. 17

2.3 Scenarios when individual writes are distributed to different number of OSTs. "N OST" means that each process's writes are distributed to N OSTs. 18

2.4 Bandwidth when individual writes are distributed to different number of OSTs. 19

3.1 Component diagram of Memcached. 22

3.2 Software architecture of BurstMem. 24

3.3 Data structures for absorbing writes. 25

3.4 Coordinated shuffling for N-1 data flushing. 29

3.5 CCI-based network communication. 30

3.6 Ingress I/O bandwidth with a varying server count. 33

3.7 Ingress I/O bandwidth with a varying client count. 34

3.8 Egress I/O bandwidth. 35

3.9 Dissection of coordinated shuffling. 36

3.10 Scalability of BurstMem. 37

3.11 I/O performance of S3D with BurstMem. 38

4.1 BurstFS system architecture. 44

4.2 Diagram of the distributed key-value store for BurstFS. 45

4.3 Diagram of co-located I/O delegation on three compute nodes P, Q and R, each with 2 processes. 48

4.4 Server-side read clustering and pipelining. 50

4.5 Comparison of BurstFS with PLFS and OrangeFS under different write patterns. . . 53

4.6 Comparison of BurstFS with PLFS and OrangeFS under different read patterns. . . . 54

4.7 Comparison of BurstFS with PLFS and OrangeFS under different transfer sizes. . . . 55

4.8 Analysis of metadata performance as a result of transfer size and process count. . . . 56

4.9 Performance of Tile-IO and BTIO. 58

4.10 Read bandwidth of IOR. 59

5.1 A conceptual example comparing TRIO with reactive data flush approach. In (b), reactive data flush incurs unordered arrival (e.g. B7 arrives earlier than B5 to Server1) and interleaved requests of BB-A and BB-B. In (c), *Server-Oriented Data Organization* increases sequentiality while *Inter-BB Flush Ordering* mitigates I/O contention. . . . 65

5.2 Server-Oriented Data Organization with Stacked AVL Tree. Segments of each server can be sequentially flushed following in-order traversal of the tree nodes under this server. 67

5.3 The overall performance of TRIO under both inter-node and intra-node writes. 71

5.4 Performance analysis of TRIO. 72

5.5 Flush bandwidth under I/O contention with an increasing process count. 74

5.6 Flush bandwidth with an increasing stripe count. 75

5.7 Comparison of average I/O time. 76

5.8 CDF of job response time. 77

5.9 Comparison of total I/O time. 78

ABSTRACT

The computing power on the leadership-class supercomputers has been growing exponentially over the past few decades, and is projected to reach exascale in the near future. This trend, however, will continue to push forward the peak I/O requirement for checkpoint/restart, data analysis and visualization. As a result, the conventional Parallel File System (PFS) is no longer a qualified candidate for handling the exascale I/O workloads. On one hand, the basic storage unit of the conventional PFS is still the hard drives, which are expensive in terms of I/O bandwidth/operation per dollar. Providing sufficient hard drives to meet the I/O requirement at exascale is prohibitively costly. On the other hand, the effective I/O bandwidth of PFS is limited by I/O contention, which occurs when multiple computing processes concurrently write to the same shared disks.

Recently, researchers and system architects are exploring a new storage architecture with tiers of burst buffers (e.g. DRAM, NVRAM and SSD) deployed between the compute nodes and the backend PFS. This additional burst buffer layer offers much higher aggregate I/O bandwidth than the PFS and is designed to absorb the massive I/O workloads on the slower PFS. Burst buffers have been deployed on numerous contemporary supercomputers, and they have also become an indispensable hardware component on the next-generation supercomputers.

There are two representative burst buffer architectures being explored: node-local burst buffers (burst buffers on compute nodes) and remote shared burst buffers (burst buffers on I/O nodes). Both types of burst buffers rely on a software management system to provide fast and scalable data service. However, there is still a lack of in-depth study on the software solutions and their impacts. On one hand, a number of studies on burst buffers are based on modeling and simulation, which cannot exactly capture the performance impact of various design choices. On the other hand, existing software development efforts are generally carried out by industrial companies, whose proprietary products are commercialized without releasing sufficient details on the internal design.

This dissertation explores the alternative burst buffer management strategies based on research designs and prototype implementations, with a focus on how to accelerate the common scientific I/O workloads, including the bursty writes from checkpointing and bursty reads from restart/analysis/visualization. Our design philosophy is to leverage burst buffers as a fast and intermediate storage layer to orchestrate the data movement between the applications and burst buffers, as well as the data movement between burst buffers and the backend PFS. On one hand, the performance benefit of burst buffers can significantly speed up the data movement between the applications and burst buffers. On the other hand, this additional burst buffer layer offers extra capacity to buffer and reshape the write requests, and drain them to the backend PFS in a manner catering to the most effective utilization of PFS capabilities. Rooted on this design philosophy, this dissertation investigates three data management strategies. The first two strategies answer how to efficiently move data between the scientific applications and the burst buffers. These two strategies are respectively designed for the remote shared burst buffers and the node-local burst buffers. The rest one strategy aims to speed up the data movement between the burst buffers and the PFS, it is applicable to both types of burst buffers. In the first strategy, a novel burst buffer system named BurstMem is designed and prototyped to manage the remote shared burst buffers. BurstMem expedites scientific checkpointing by quickly buffering the checkpoints in the burst buffers after each round of computation and asynchronously flushing the datasets to the PFS during the next round of computation. It outperforms the state-of-the-art data management systems with efficient data transfer, buffering and flushing. In the second strategy, we have designed and prototyped an ephemeral burst buffer file system named BurstFS to manage the node-local burst buffers. BurstFS delivers scalable write bandwidth by having each process write to its node-local burst buffer. It also provides fast and temporary data sharing service for multiple coupled applications in the same job. In the third strategy, a burst buffer orchestration framework named TRIO is devised to address I/O contention on the PFS. TRIO buffers scientific applications' bursty write requests, and dynamically adjusts the flush order of all the write requests to avoid multiple burst buffers' competing flush on the same disk. Our experiments demonstrate that by addressing I/O contention, TRIO not only improves the storage bandwidth utilization but also minimizes the average I/O service time for each job.

Through systematic experiments and comprehensive evaluation and analysis, we have validated our design and management solutions for burst buffers can significantly accelerate scientific I/O for the next-generation supercomputers.

CHAPTER 1

INTRODUCTION

The astonishing growth of top 500 supercomputers suggests that, around the time frame of 2023, the computation power of “leadership-class” systems is likely to surpass 1 exaflop/s (10^{18} flop-s/s) [74]. Scientific applications on such large-scale computers often produce gigantic amount of data. For example, the CHIMERA [37] astrophysics application outputs 160TB of data per check-point, writing these data takes around an hour on Titan [14] supercomputer hosted in Oak Ridge National Laboratory [66]. Titan has 18,688 compute nodes. It is currently the third fastest supercomputer in the world [24]. An exascale supercomputer is expected to have millions of nodes with a checkpoint frequency of less than one hour [59, 36]. With such rapid growth in computing power and data intensity, I/O remains a challenging factor in determining the overall applications' execution time.

Over the past two decades, system designers have been bridging the computation-I/O gap by expanding the backend PFS. The basic storage unit on PFS is the hard drive, which is cheap for capacity but expensive for I/O bandwidth. Currently, the provisioned storage bandwidth[1] of the leadership-scale supercomputers is a few hundreds of GB/s. In the exascale computing era, the peak bandwidth demand will become two orders of magnitude higher [65, 38]. Expanding PFS for this level of bandwidth requirement becomes a cost-prohibitive solution.

To make things worse, conventional HPC system generally have separated compute nodes and PFS. Contention on PFS happens when computing processes concurrently issue the bursty write requests to the shared PFS. Since the number of compute nodes is typically 10x∼100x more than the number of storage nodes on PFS, each storage node suffers from heavy contention during a typical checkpoint workload [55]. Contention has been considered as a key issue that limits PFS's bandwidth scalability [55, 72, 109].

Recently, the idea of *burst buffer* has been proposed to shoulder the exploding data pressure on the PFS. Many consider burst buffer as a promising storage solution for the exascale I/O

[1]The storage bandwidth is defined as the aggregate bandwidth of all the disks in a parallel file system.

workloads. Burst buffers refer to a broad category of high-performance storage device, such as DRAM, NVRAM and SSD. They are positioned between the compute nodes and PFS, offering much higher aggregate bandwidth than the PFS. This additional burst buffer layer can efficiently absorb the heavy I/O workloads on PFS. Moreover, buffering the bursty writes inside burst buffers gives plenty of opportunities to reshape the write requests and avoid the contention on PFS.

Numerous existing studies on burst buffers are based on modeling and simulation of burst buffers [65, 82, 109, 87]. Though several companies have also announced their ongoing development efforts and early products [33, 13], few design details have been made publicly available. In contrast to these works, this dissertation explores various design choices of a burst buffer system and assesses their benefits by prototyping burst buffers for the general scientific I/O workloads. Three burst buffer management strategies are proposed and studied, each highlighting a distinct contribution (see Section 1.3). In the rest of this chapter, we provide a high-level description of the characteristics of scientific I/O workloads, the representative burst buffer architectures and their use cases that underpin this dissertation, and the storage models adopted for our burst buffer management. We then discuss our research contributions.

1.1 Overview of Scientific I/O Workloads

In this section, we provide an overview of the representative I/O workloads on HPC systems, which are also the targeting workloads in this dissertation. These workloads include checkpoint/restart and multi-dimensional I/O. The two categories are not disjoint: checkpoints often contain multi-dimensional arrays, these arrays can be retrieved later for the purpose of crash recovery, data analysis and visualization. We classify multi-dimensional I/O in a separate category to emphasize the diversity of read patterns on a multi-dimensional dataset under the analysis/visualization workloads.

1.1.1 Checkpoint/Restart I/O

Checkpoint/restart is a common fault tolerance mechanism used by HPC applications. During a run, application processes periodically save their in-memory state in files called checkpoints, typically written to a PFS. Upon a failure, the most recent checkpoint can then be read to restart the job. Checkpointing operations are usually concurrent across all processes in an application, and occur at program synchronization points when no messages are in flight.

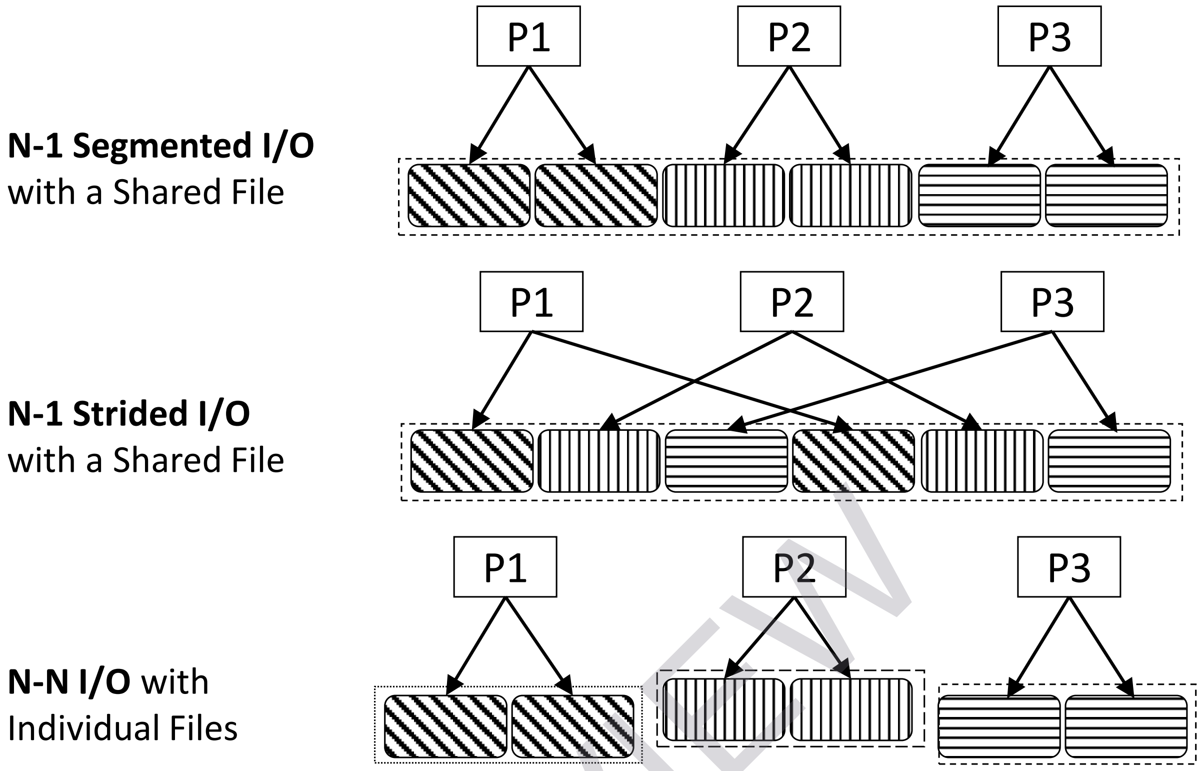
P1
P2
P3
N-1 Segmented I/O with a Shared File
N-1 Strided I/O with a Shared File
N-N I/O with Individual Files

Figure 1.1: Checkpoint/restart I/O patterns (adapted from [32]).

There are two dominant I/O patterns for checkpoint/restart, N-1 and N-N patterns as shown in Fig. 1.1. In N-N I/O, each process writes/reads data to/from a unique file. In N-1 I/O, all processes write to, or read from, a single shared file. N-1 I/O can be further classified into two patterns: N-1 segmented and N-1 strided. In N-1 segmented I/O, each process accesses a non-overlapping, contiguous file region. In N-1 strided I/O, processes interleave their I/O amongst each other.

On current HPC systems, checkpointing can account for 75%-80% of the total I/O traffic [83]. While there is ongoing debate on how checkpointing operations will change on exascale systems compared to today's systems, there is general consensus that the data size per checkpoint will increase due to larger job scales and the interval between checkpoints will decrease due to increased overall failure rates [36, 76]. The larger file sizes and shorter intervals for checkpointing will require orders of magnitude faster storage bandwidth [65].

1.1.2 Multi-Dimensional I/O

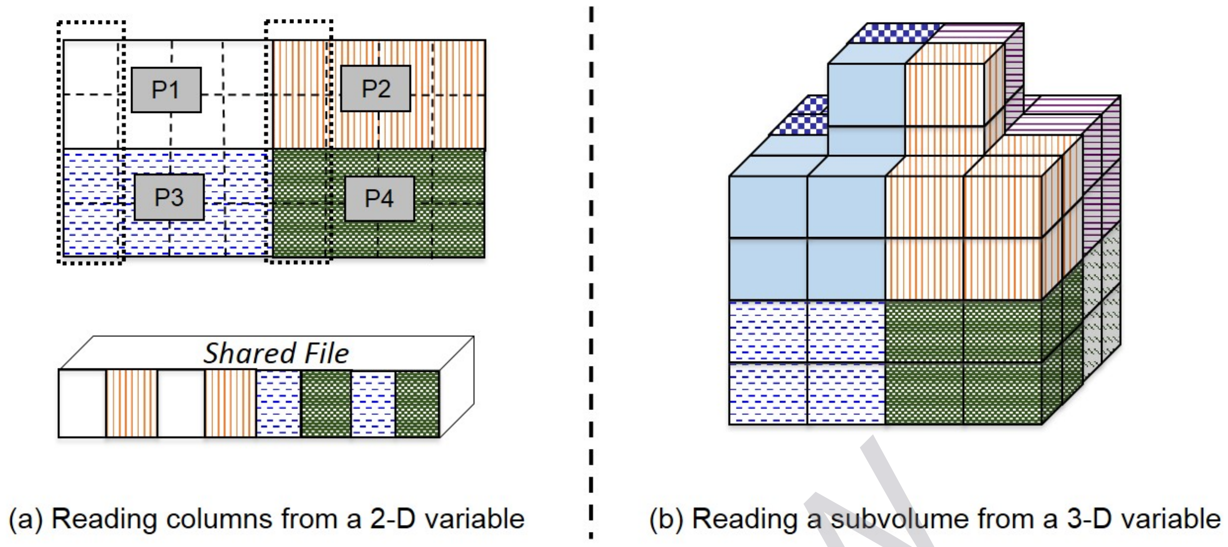
P1
P2
P3
P4
Shared File
(a) Reading columns from a 2-D variable
(b) Reading a subvolume from a 3-D variable

Figure 1.2: I/O access patterns with multi-dimensional variables.

Another common I/O workload on HPC systems is data access to multi-dimensional data variables in scientific applications. While multi-dimensional variables are written in one particular order, they are often read for analysis or visualization in a different order than the write order [95, 70, 61].

Fig. 1.2(a) shows a sample read pattern on a two-dimensional variable. This variable is initially decomposed into four blocks and assigned to four processes, which are written to a shared file on the backend PFS. When this variable is read back for analysis, one process may require only one or more columns from this variable. However, these two columns are stored non-contiguously across the data blocks. Therefore, this process needs to issue numerous small read requests on four different data blocks in order to retrieve its data for analysis. Fig. 1.2(b) illustrates a similar but more complex scenario with a three-dimensional variable. The 3-D variable is initially stored as eight different blocks across burst buffers. A process may only need a subvolume in the middle of the variable for analysis. This subvolume has to be gathered from eight different blocks to complete its data access, resulting in many small read operations. These small read operations are not favored by PFS [94, 93, 95].

1.2 Overview of Burst Buffers

In this section, we first outline the representative burst buffer architectures, and their use cases our dissertation is based on. We then discuss the alternative storage models adopted in our burst buffer management.

1.2.1 Representative Burst Buffer Architectures

Two representative burst buffer architectures are shown in Figure 1.3. In the node-local burst buffer architecture, burst buffers are located on the individual compute nodes. The benefit of this architecture is that the aggregate burst buffer bandwidth scales linearly with the number of compute nodes. Scientific applications can acquire scalable write bandwidth by having each process write its data to its local burst buffer. However, flushing buffered data to the backend PFS requires extra computing power on the compute nodes, which can incur computation jitters [31]. In the remote shared burst buffer architecture, burst buffers are placed on the dedicated I/O nodes deployed between the compute nodes and the backend PFS. Data movement from the compute nodes to the burst buffers needs to go through the network. This architecture achieves excellent resource isolation since flushing data to the backend PFS is done without interfering with the computation on the compute nodes. However, in contrast to the node-local burst buffers, its I/O bandwidth[2] is dependent on several factors, including the network bandwidth, the aggregate burst buffer bandwidth and the number of I/O nodes.

Both types of burst buffers have been widely deployed on existing high-end computing systems. They are also projected to come along with a majority of the next-generation supercomputers. For instance, node-local burst buffers have been equipped on DASH [53] at the San Diego Supercomputer Center [17], Catalyst [4] and Hyperion [12] at the Lawrence Livermore National Laboratory (LLNL) [15], TSUBAME supercomputer series [26] at the Tokyo Institute of Technology, and Theta [22] at the Argonne National Laboratory (ANL). They will also come with several next-generation supercomputers in a few years, such as Summit [21] at the Oak Ridge National Laboratory and Sierra [20] at the Lawrence Livermore National Laboratory. On the other hand, remote shared burst buffers have been set up on Tianhe-2 [23] at the Chinese National Supercomputer Center, Trinity [25] at the Los Alamos National Laboratory, and Cori [5] at the Lawrence

[2]I/O bandwidth is typically measured as a quotient of the total data size read/writtten by an application and its total I/O time.

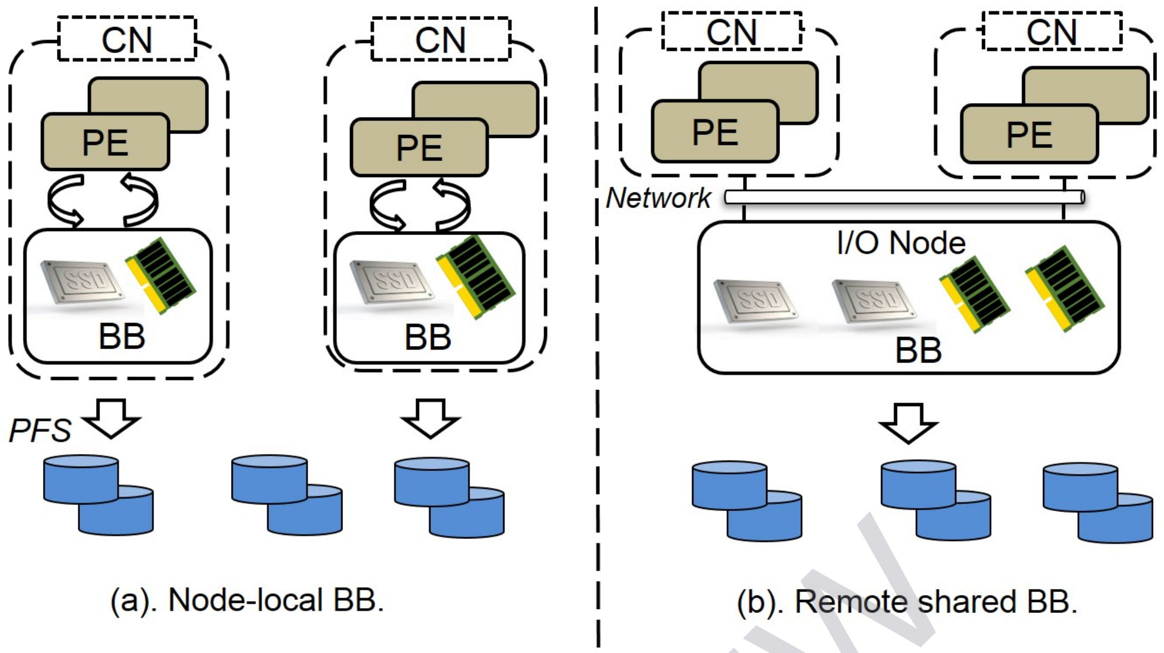

Figure 1.3: An overview of burst buffers on HPC system. BB refers to Burst Buffer. PE refers to Processing Element. CN refers to Compute Node.

Berkeley National Laboratory. As one of the next-generation supercomputers, Aurora [2] at the Argonne National Laboratory features a heterogeneous burst buffer architecture including both node-local burst buffers and remote shared burst buffers.

1.2.2 Burst Buffer Use Cases

Burst buffers are designed to accelerate the *bursty* write and read operations featured by most scientific applications. Although we are aware that burst buffers also have the potential to support the cloud-based applications that issue intermittent read/write requests, their use cases are beyond the focus of this dissertation. In general, burst buffers' support for scientific applications can be summarized into the following use cases.

- **Checkpoint/Restart:** Applications periodically write "checkpoints" that includes snapshots of data structures and state information. Upon a failure, they can load the checkpoints and rollback to a previous state. Applications checkpointing to/restarting from burst buffers can reap much higher aggregate bandwidth than PFS.
- **Overlapping Computation and Checkpointing to PFS:** Scientific applications' life cycle generally alternates between a phase of computation and a phase of checkpointing. Ap-

plications can hide the checkpointing latency by temporarily buffering the data in burst buffers after a phase of computation, and let burst buffers asynchronously flush the checkpoints to the PFS during the next phase of computation.

- **Reshaping Bursty Writes to PFS:** Burst buffers stand as a middle layer that absorbs the bursty writes to PFS. With all the buffered write requests, burst buffers have the global knowledge of how scientific data are laid out on the PFS. Based on this knowledge, they can reshape the write traffic to avoid contention on PFS.

- **Staging/Sharing Intermediate data:** Intermediate data such as checkpoints or plot files are staged for two purposes: out-of-core computation and data sharing. In the former case, applications with insufficient memory can swap out a portion of its in-memory data to burst buffers, and later read them back for computation. In the latter case, a job usually consists of multiple scientific workflows sharing data with each other. For instance, the output of a simulation program can be used as the input for an analysis program to perform post-analysis (after simulation) and in-situ/in-transit analysis (during simulation). In both cases, staging the intermediate data to burst buffers and loading the data from burst buffers are much faster than PFS.

- **Prefetching Data for Fast Analysis:** Scientific applications with the foreknowledge of future reads can hide the read latency on PFS by prefetching data to burst buffers.

Due to the architectural differences, node-local burst buffers and remote shared burst buffers have distinct preferences on these use cases. Since node-local burst buffers reside on the compute nodes, and each compute node is generally dedicated to an individual job, data on node-local burst buffers are temporarily available within the individual job's life cycle. Under this constraint, we can leverage node-local burst buffers to perform scalable checkpointing and enable temporary data sharing among coupled workflow applications in the same job. On the other hand, data on the remote shared burst buffers are independent of the life cycle of the individual job, we can harness remote shared burst buffers to provide the center-wide data service to all the jobs on the compute nodes. Moreover, remote shared burst buffers are jitter-free for data flushing. Therefore, it is tempting to overlap computation and checkpointing to PFS based on the remote shared burst buffers.

1.2.3 Storage Models to Manage Burst Buffers

A key consideration before structuring a burst buffer system is what type of storage models can be used to manage burst buffers. In particular, there are two representative storage models widely adopted under the cloud and HPC environments: distributed file systems and databases. Each offers an alternative approach to manage burst buffers. Although there are data management solutions that fall outside these two categories (e.g. DataSpaces [48], PGAS [92]), their architectures can be derived and synthesized from the two storage models. Therefore, we mainly focus on the discussion on the distributed file systems and databases. Our burst buffer-based solutions root in these two storage models.

Distributed File System vs. Distributed Database. In a file system, data are stored in the form of files. The format of each file is defined by the file system clients (i.e. applications and I/O libraries). For instance, applications using POSIX I/O [29] stores each file as a stream of binary bytes, data are accessed based on their sizes and byte addresses in the file. In order to provide richer data access semantics, application developers need to implement their own functions beyond POSIX I/O and define their own data format atop the binary stream in POSIX. High-level I/O libraries (e.g. HDF5 [11], NetCDF [58]) format each file as a container of multi-dimensional arrays, data are accessed based on their dimensionality, positions and sizes along each dimension. *However, these file formats are opaque to the file system.*

In contrast, a database is generally used for storing related, structured data (e.g. tables, indices) with well-defined data formats. Unlike a file system, the physical data formats of these structured data are defined by the database and they are opaque to the clients. Clients access databases using the database-specific semantics, such as SQL for relational databases and Put/Get APIs for key-value stores. Users using these interfaces can easily fulfill a wide variety of complex application logics without further effort to implement these logics. *Compared with a file system, the complicated implementation of these application logics are offloaded to the database.* Besides, a database allows indexing based on any attribute or data property (i.e. SQL columns for relational databases and keys for key-value stores). These indexed attributes facilitate fast query.

The choice of distributed file systems/databases largely depends on the burst buffer semantics to be exposed to applications. File system based burst buffer service is advantageous in two aspects. First, it can transparently support a large quantity of file system-based scientific applications that use POSIX and other POSIX-based I/O libraries, such as HDF5 and NetCDF. Second, it gives application developers more flexibility to define their own data formats, and implement their application logics. On the other hand, a database-based burst buffer service allows application developers to more easily implement their complex application logics by directly using the rich client-side interfaces. It also speeds up data processing with customizable data indexing. However, to support existing file system-based applications, application developers need to replace the POSIX/NetCDF/HDF5-based functions with database client's native functions, which demands non-trivial efforts.

Besides the choices of databases/file systems, another key consideration is what types of file system/database services can be harnessed to manage burst buffers. According to the data layout strategy, existing file systems can be classified into locality-based distributed file systems (e.g. HDFS [90]) and striping-based distributed file systems (e.g. Lustre [35]). There are also two types of databases: relational databases and non-relational databases. The non-relational databases can be further classified into key-value stores and graph stores. Both non-relational databases and key-value stores are applicable to manage burst buffers.

Locality-Based Distributed File Systems vs. Striping-Based Distributed File Systems. In the locality-based file systems, each process prioritizes writing data to its node-local storage. This design choice avoids data movement across network during a bursty writes. So it delivers scalable write bandwidth. *Because of this benefit, we design a locality-based distributed file system to manage the node-local burst buffers in Chapter 4.* However, a key challenge is how to read data: because file data are written by each process locally, in order to read the remote data, each process needs to look up the data sources for its read requests. A scalable metadata service is needed to serve the large quantity of lookup requests during bursty reads.

In the striping-based distributed file systems, data written by each process are striped across multiple data nodes. The locations of data sources are calculated based on a pre-determined hash function. Therefore, during read, each process can directly compute the locations of the requested data and retrieve the data from the data sources. Compared with the locality-based distributed file system, this type of file system delivers balanced read/write bandwidth. However, its write bandwidth is limited by contention when multiple processes concurrently write to the same data node.

Relational Database vs. Key-Value Stores.. A radical difference between relational database and key-value store is their data models exposed to users. Relational databases represent data in a form of one or more tables of rows and columns, with a unique key for each row. These tables are queried and maintained by SQL, which defines a rich set of semantics that allow users to easily fulfill complex application logics (e.g. insert/delete/update/query/join/transaction). However, the complex data model also makes the implementation of relational databases incredibly complicated. For example, a relatively simple SELECT statement could have dozens of potential query execution paths, which a query optimizer would evaluate at run time. In contrast, key-value stores represent data in the form of key-value pairs, and provide much simpler interfaces (e.g. Put/Get for storing/retrieving data). The simplicity of their data services afford key-value stores much faster speed and better scalability. *In Chapter 3, we leverage distributed key-value store to manage the remote shared burst buffers.*

1.3 Research Contributions

In this dissertation, we have thoroughly investigated the I/O challenges on HPC systems, and researched three approaches for burst buffer management. In particular, this dissertation has made the following three contributions.

1.3.1 BurstMem: Overlapping Computation and I/O

We have designed BurstMem, a burst buffer system to manage the remote shared burst buffers. It accelerates checkpointing by temporarily buffering applications' scientific datasets in burst buffers and asynchronously flushing the data to PFS. BurstMem is built on top of Memcached, a distributed key-value store widely adopted under the cloud environment. While inheriting the buffering management framework of Memcached, we have identified its major issues in handling the bursty checkpoint workload, including low burst buffer capacity and bandwidth utilization, unable to leverage the native transport of various high-speed interconnects on HPC system and lack of data indexing and shuffling support for fast data flush. Based on our analysis, we structure BurstMem to accelerate checkpointing with indexed and space-efficient buffer management, fast and portable data transfer, and coordinated data flush. Our evaluations demonstrated that BurstMem is able to achieve 8.5× speedup over the bandwidth of conventional PFS.

1.3.2 BurstFS: A Distributed Burst Buffer File System

We have analyzed the benefits and challenges of using node-local burst buffers to accelerate scientific I/O, and designed an ephemeral Burst Buffer File System (BurstFS) that supports scalable and efficient aggregation of I/O bandwidth from burst buffers while having the same life cycle as a batch-submitted job. BurstFS features several techniques including scalable metadata indexing for fast and scalable write with amortized metadata cost, co-located I/O delegation for scalable read and data sharing among coupled applications in the same job, and server-side read clustering and pipelining to optimize small and noncontiguous read operations widely used in a multi-dimensional I/O workload. Through extensive tuning and analysis, we have validated that BurstFS has accomplished our design objectives, with linear scalability in terms of aggregated I/O bandwidth for parallel writes and reads.

1.3.3 TRIO: Reshaping Bursty Writes

We have devised TRIO, a burst buffer orchestartion framework to efficiently drain data from burst buffers to the backend PFS. The strategy is applicable to both remote shared burst buffers and node-local burst buffers. TRIO buffers the applications' checkpointing write requests in burst buffers and reshapes the bursty writes to maximize the number of sequential writes to PFS. Meanwhile, TRIO coordinates the flushing orders among concurrent burst buffers to address two levels of contention on PFS, namely, competing writes from multiple processes on the same storage server, and the interleaving writes from multiple concurrent applications on the same storage server. Our experimental results demonstrated that TRIO could efficiently utilize storage bandwidth and reduce the average I/O time by 37% in typical checkpointing scenarios.

1.3.4 Publication Contributions

During my PhD study, I have contributed to the following publications (listed in the chronological order).

1. Yandong Wang, Yizheng Jiao, Cong Xu, Xiaobing Li, **Teng Wang**, Xinyu Que, Christian Cira, Bin Wang, Zhuo Liu, Bliss Bailey, Weikuan Yu. Assessing the Performance Impact of High-Speed Interconnects on MapReduce Programs. Third Workshop on Big Data Benchmarking, 2013 [107].

2. Zhuo Liu, Jay Lofstead, **Teng Wang**, Weikuan Yu. A Case of System-Wide Power Management for Scientific Applications. IEEE International Conference on Cluster Computing, 2013 [67].

3. Zhuo Liu, Bin Wang, **Teng Wang**, Yuan Tian, Cong Xu, Yandong Wang, Weikuan Yu, Carlos A Cruz, Shujia Zhou, Tom Clune, Scott Klasky. Profiling and Improving I/O Performance of a Large-Scale Climate Scientific Application, 2013 [69].

4. Yandong Wang, Robin Goldstone, Weikuan Yu, **Teng Wang**. Characterization and Optimization of Memory-Resident MapReduce on HPC Systems. IEEE 28th International Parallel and Distributed Processing Symposium, 2014 [106].

5. **Teng Wang**, Kevin Vasko, Zhuo Liu, Hui Chen, Weikuan Yu. BPAR: A Bundle-Based Parallel Aggregation Framework for Decoupled I/O Execution. International Workshop on Data Intensive Scalable Computing Systems, in Conjunction with SC 2014 [104].

6. **Teng Wang**, Sarp Oral, Yandong Wang, Bradley Settlemyer, Scott Atchley, Weikuan Yu. BurstMem: A High-Performance Burst Buffer System for Scientific Applications. IEEE International Conference on Big Data, 2014 [103].

7. **Teng Wang**, Sarp Oral, Michael Pritchard, Kento Vasko, Weikuan Yu. Development of a Burst Buffer System for Data-Intensive Applications. International Workshop on The Lustre Ecosystem: Challenges and Opportunities, 2015 [101].

8. **Teng Wang**, Kathryn Mohror, Adam Moody, Weikuan Yu, Kento Sato. Poster Presented at The International Conference for High Performance Computing, Networking, Storage and Analysis, 2015 [99].

9. **Teng Wang**, Sarp Oral, Michael Pritchard, Bin Wang, Weikuan Yu. TRIO: Burst Buffer Based I/O Orchestration. IEEE International Conference on Cluster Computing, 2015 [102].

10. **Teng Wang**, Kathryn Mohror, Adam Moody, Kento Sato, Weikuan Yu. An Ephemeral Burst Buffer File System for Scientific Applications. IEEE/ACM the International Conference for High Performance Computing, Networking, Storage and Analysis 2016 [98].

11. **Teng Wang**, Kevin Vasko, Zhuo Liu, Hui Chen, Weikuan Yu. Enhance Scientific Application I/O with Cross-Bundle Aggregation. International Journal of High Performance Computing Applications [105], 2016.

12. **Teng Wang**, Adam Moody, Yue Zhu, Kathryn Mohror, Kento Sato, Tanzima Islam, Weikuan Yu. MetaKV: A Key-Value Store for Metadata Management of Distributed Burst Buffers. IEEE 31th International Parallel and Distributed Processing Symposium [100], 2017.

1.4 Dissertation Overview

In the rest of this dissertation, we first elaborate on the problems that prevent scientific applications from achieving the satisfactory I/O performance. We then detail three burst buffer management strategies that address these issues. Every chapter focuses on presenting one approach, along with a comprehensive evaluations and comparisons between our solution and the state-of-the-art techniques.

In Chapter 2, we thoroughly investigate the key issues that constrain the scientific I/O performance. Namely, increasing computation and I/O gap, and I/O contention on PFS. The insights from this chapter are used to motivate our innovations.

In Chapter 3, we specify the design of BurstMem, a remote shared burst buffer based I/O burst buffer management framework. BurstMem complements Memcached for checkpointing with enhanced data services for fast data transfer, data buffering and flushing. Our performance evaluation demonstrates that our solutions can efficiently accelerate scientific checkpointing.

In Chapter 4, we explore the benefits and key considerations of architecting a distributed burst buffer file system to manage node-local burst buffers. Based on our exploration, we design BurstFS to speedup checkpointing/restart, analysis and visualization. BurstFS supports scalable writes and fast data sharing under various concurrent I/O workloads. Our comparison with the state-of-the-art solutions demonstrates that BurstFS carries great potential in handling the exascale I/O workloads.

In Chapter 5, we introduce TRIO, a burst buffer orchestration framework that buffers and reorganizes checkpointing write requests for sequential and contention-aware data flush to the PFS. This strategy can be used on both remote shared burst buffers and node-local burst buffers. Our experiments with different types of concurrent workloads demonstrate that by addressing the I/O contention on PFS, TRIO not only improves the bandwidth of a single application, but also minimizes the average I/O time of the individual application under a multi-application workload.

Finally, we conclude this dissertation and discuss opportunities for future work in Chapter 7.