2017 IEEE 37th International Conference on Distributed Computing Systems

High-Performance and Resilient Key-Value Store with Online Erasure Coding for Big Data Workloads

Dipti Shankar, Xiaoyi Lu, Dhabaleswar K. (DK) Panda
Department of Computer Science and Engineering, The Ohio State University
Email: {shankard, luxi, panda}@cse.ohio-state.edu

***Abstract*—Distributed key-value store-based caching solutions are being increasingly used to accelerate Big Data applications on modern HPC clusters. This has necessitated incorporating fault-tolerance capabilities into high-performance key-value stores such as Memcached that are otherwise volatile in nature. In-memory replication is being used as the primary mechanism to ensure resilient data operations. However, this incurs increased network I/O with high remote memory requirements. On the other hand, Erasure Coding is being extensively explored for enabling data resilience, while achieving better storage efficiency. In this paper, we first perform an in-depth modeling-based analysis of the performance trade-offs of In-Memory Replication and Erasure Coding schemes for key-value stores, and explore the possibilities of employing Online Erasure Coding for enabling resilience in high-performance key-value stores for HPC clusters. We then design a non-blocking API-based engine to perform efficient Set/Get operations by overlapping the encoding/decoding involved in enabling Erasure Coding-based resilience with the request/response phases, by leveraging RDMA on high performance interconnects. Performance evaluations show that the proposed designs can outperform synchronous RDMA-based replication by about 2.8x, and can improve YCSB throughput and average read/write latencies by about 1.34x – 2.6x over asynchronous replication for larger key-value pair sizes (>16 KB). We also demonstrate its benefits by incorporating it into a hybrid and resilient key-value store-based burst-buffer system over Lustre for accelerating Big Data I/O on HPC clusters.**

I. Introduction

With the recent emergence of in-memory computing [1], [2], [3], [4], [5], [6] into mainstream Big Data analytics, distributed key-value stores have become more vital than ever for accelerating various data processing workloads, including, online analytical workloads and offline data-intensive workloads. For instance, leveraging a distributed and scalable key-value store like Memcached [4] over database systems such as MySQL is often invaluable to the application server's performance [7], for accelerating online data analytical workloads. Several research works have focused on exploiting advanced features like Remote-Direct-Memory-Access (RDMA) in key-value stores including RDMA-Memcached [8], [9], [10], RAMCloud [6], MICA [5], to improve the response time and throughput of the application servers deployed on HPC clusters. On the other hand, for offline data-intensive workloads, distributed key-value stores are being increasingly used to design efficient and high throughput I/O staging and caching solutions for alleviating the I/O bottleneck for Big Data analytics on modern HPC clusters. Specifically, for the high-performance computing (HPC) environment, solutions such as Boldio [11] present a high-performance and resilient burst-buffer system for Big Data I/O over Lustre, by mapping Hadoop I/O streams onto key-value pairs that are cached in the SSD-assisted RDMA-enabled Memcached [9]. Similarly, RDMA-based in-memory key-value stores are being employed for accelerating cluster computing frameworks, including Hadoop and Spark over HDFS [12], [13].

As we can see, irrespective of whether we are dealing with online or offline data processing, achieving high performance by taking advantage of in-memory key-value stores has proven to be invaluable. Additionally, this has necessitated incorporating fault-tolerance capabilities into high-performance key-value stores such as Memcached, that are otherwise volatile in nature. Currently, most of the key-value store based solutions employ data replication as their primary means of providing fault-tolerance. Data replication needs to store as many as N replicas in memory to tolerate N-1 node failures, which incurs a high memory overhead. This technique trades memory efficiency to guarantee that is resilience critical to Big Data workloads, and this limits the size of data that can be handled in memory. Thus, it may further degrade the performance when memory is used up. This motivates us to answer a broad question: **Can we design a viable alternative to enable resilience in key-value stores with better memory efficiency while delivering high performance?**

A. Motivation

In order to achieve resilience with better storage efficiency, erasure coding techniques such as Reed-Solomon codes [14], have been being extensively explored in the literature for different storage scenarios [15]. With erasure coding, each data entity is divided into K fragments, and encoded to generate M additional fragments. And these $K + M$ fragments are then distributed to N unique nodes. Any K fragments can be used to recover the original data object, which means erasure coding can tolerate up to M simultaneous node failures. Since erasure coding requires a storage overhead of N/K where (K<N), as compared to the high overhead of $M+1$ (i.e., replication factor) that is incurred by replication, it provides an attractive

*This research is supported in part by National Science Foundation grants #CNS-1513120, #CNS-1419123 and #IIS-1447804. It used the Extreme Science and Engineering Discovery Environment (XSEDE), which is supported by National Science Foundation grant number OCI-1053575.

1063-6927/17 $31.00 © 2017 IEEE
DOI 10.1109/ICDCS.2017.224

alternative to enable resilience in key-value stores with better memory efficiency.

However, recent studies [16] have shown that naively employing erasure coding in the critical path (i.e., online erasure coding) may not be beneficial for performance due to the computation overhead of encoding and decoding, especially when dealing with smaller data sizes (e.g., <1 MB). Unfortunately, the database online queries cached as key-value pairs typically range from 512 B to 32 KB [17] and the cached I/O operations in offline data analytics jobs typically manipulate chunked data as 512 KB to 1 MB key-value pairs [11], [13]. Such disparity brings a lot of difficulties to design a high-performance in-memory key-value store with online erasure coding.

On the other hand, since most HPC clusters are equipped with high-speed and multi-core CPUs, the computation of encoding and decoding in erasure coding can be processed much faster; it may be possibly used even in the critical path of data processing. Thus, this gives us the opportunity to trade the available compute resources for enabling resilience with better memory efficiency. Another overhead of erasure coding comes from the distributed storage of fragments into multiple nodes, which increases the communication overhead. The availability of high-performance interconnects such as InfiniBand on HPC clusters which provide high-bandwidth, low-latency, as well as Remote Direct Memory Access (RDMA) features to help design high-performance communication and overlapping schemes for erasure coding based key-value stores. Also, with the advent of RDMA over Converged Ethernet (RoCE) [18], the benefits of RDMA are being made available to data centers with high-speed TCP/IP Ethernet networks.

All these observations and inferences motivate us to explore: **Can Online Erasure-Coding be leveraged as a viable resilience technique to design a high-performance and resilient in-memory key-value store for accelerating various Big Data workloads on HPC clusters?** This broad question further brings more technical challenges:

- What are the performance characteristics and tradeoffs of online erasure coding and replication? Can we model them and analyze the tradeoffs in a theoretical manner?
- Can the model and theoretical analysis guide the actual design of an online erasure coding based key-value store?
- What could be the most efficient designs to alleviate the erasure coding bottlenecks when used in an online fashion?
- How much performance benefits and gains in memory efficiency can we exploit through online erasure coding over replication, for both online and offline data processing workloads?

B. Contributions

To address above challenges, in this paper, we first perform a detailed modeling-based analysis of the performance of in-memory replication and online erasure coding. Through the modeling analysis, we identify the major performance bottlenecks for the Set and Get operations with replication and erasure coding schemes. We further analyze the challenges and opportunities of employing online erasure coding for enabling resilience in a high-performance key-value store on modern HPC clusters with high-performance interconnects. We then design a non-blocking Memcached API and RDMA based engine to perform key-value store Set/Get operations with resilience via online erasure coding. We propose different schemes to overlap the encoding/decoding computation and data communication in the most optimized manner. We also find a real scenario to integrate our proposed design, and demonstrate its benefits by incorporating it into a hybrid and resilient key-value store-based burst-buffer system over Lustre for accelerating Big Data I/O. Our performance evaluations show that our proposed High-Performance and Resilient In-Memory Key-Value Store design with Online Erasure Coding can improvement the basic Set/Get latency by up to 2.8x over RDMA-based synchronous replication designs, and performs similar to asynchronous replication. With our enhanced asynchronous and pipelined request processing engine, we demonstrate that the aggregated throughput with YCSB update heavy workload (50:50 read:write) can be improved by about 1.34x and average latency by up to 2.3x over asynchronous replication in a multi-client environment, for key-value pair sizes greater than 16 KB. For Offline Analytical workloads, we demonstrate that our proposed Online Erasure Coding enabled designs can match the performance of Asynchronous Replication-based Burst-Buffer System (Boldio [11]), for accelerating Big Data I/O over Lustre on modern HPC clusters, while enabling better memory efficiency.

The rest of the paper is organized as follows. Section II presents the necessary background on erasure coding. Section III presents our modeling-based analysis for the challenges and opportunities. Section IV presents our proposed design, and Section V introduces the usage scenario of our designs. Section VI describes our detailed evaluation. Section VII discusses related studies in the literature. Finally, we conclude in Section VIII with some description of possible future work.

II. Background

In this section, we provide a brief overview of the two resilience techniques employed in Big Data storage systems.

A. Data Replication

Data replication is a technique for achieving fault tolerance through providing multiple identical copies of the distributed across the storage servers. This traditional approach requires data objects to be replicated over *M + 1* unique servers to tolerate *M* server failures. For instance, a 3x replication scheme is required in order to tolerate two node failures, which imposes a 200% overhead in storage space and other resources (e.g., network bandwidth for writing the data). Several database systems including MySQL employ replication for availability, while Big Data storage systems such as HDFS [19] employ replication as their primary mechanism for fault-tolerance.

B. Erasure Coding

Erasure coding [15] is a branch of information theory which extends a message with redundant data for fault tolerance.

Each data entity is broken into fragments, encoded and then stored in a distributed manner. With erasure coding, an N-node cluster can use K of *N* nodes for data and *M* nodes for codes (*K* + *M* = *N*). To store a data entity of size *D*, the object is divided into *K* chunks, and encoded to generate *M* additional fragments of the same size i.e., *D/K*, known as parity chunks, using a maximum distance separable (MDS) code [20]. These *K* + *M* data and encoded fragments are then distributed to *N* unique nodes. Any *K* data or parity chunks can be used to recover the original data object, i.e., up to *M* simultaneous node failures can be tolerated. This scheme allows a storage efficiency of up to *K/N* on an *N*-node cluster. A commonly used coding scheme is Reed-Solomon code (RS) [14], which computes parities according to its data over a finite field using the Vandermonde matrix. For instance, to tolerate two node failures on a 5-node cluster, we can employ Reed-Solomon erasure coding with *K = 3* and *M = 2*, as represented in Figure 1. In Figure 1(a), it shows the standard way of Reed-Solomon based encoding example. Figure 1(b) shows the decoding can recover the original data by any three chunks chosen from encoded data. This scheme provides a storage efficiency of 60%, which is significantly higher than the 33% efficiency enabled by the replication scheme.

III. Challenges and Opportunities for Online Erasure Coding Support in Key-Value Stores

As discussed in Section I-A, it is vital to understand the challenges and opportunities of enabling Erasure Coding over Replication in In-Memory Key-Value stores.

A. Modeling Latency for Erasure Coding and Replication

Let us consider T_{comm} as the communication time to access or update a key-value pair in a remote server cluster. Then, T_{comm} can be represented as a function of value size (*D*) of the key-value pair, network latency and bandwidth (*B*) as follows:

$$T_{comm}(D) = L + D/B \quad (1)$$

T_{comm} is the dominant factor (*Response-Wait*) that determines the time the Set/Get operation waits for the remote memory request to complete. Using this, we model the access latencies for key/value store operations, to understand the possible gains and overheads of the two resilient schemes. We assume that the key-value store client relies on underlying network's point-to-point communication capabilities.

1) Modeling Set Latency: In this section, we present analytical modeling of the Set operation.

Replication: This involves making multiple redundant copies distributed across the server cluster. Figure 2(a) represents an example where three-way data replication is employed on a 5-node cluster. Based on this, for an N-node key-value store server cluster with a replication factor *(F)*, to tolerate *(F-1)* node failures, the total time to write a key-value pair using Equation 1 can be given as:

$$T_{rep\_set} = F * (L + D/B) \quad (2)$$

From Equation 2, we observe that the *Response-Wait* (*L + D/B*) increases *F*-fold, making the network a critical factor for replicating key-value pairs.

Erasure Coding: This scheme basically splits data into *K* objects and generates parity chunks using Reed-Solomon encoding, and distributes these chunks across the server cluster. Figure 2(b) represents an example for RS(3,2) on a 5-node cluster. Since the computation mainly involves matrix multiplication we assume that the encoding time T_{encode} is a function of *D*. Based on this, for an N-node key-value store server cluster that employs a Reed-Solomon encoding of RS(*K, M*), to tolerate M node failures, the total time to write a key-value pair using Equation 1 can be given as (where N = K + M):

$$T_{era\_set} = T_{encode}(D) + N * (L + D/B * 1/K) \quad (3)$$

From Equation 3, we observe that the *Response-Wait* has been decreased by a factor of *N/K*, as compared to replication. However, an additional overhead of T_{encode} and multiple sends (*N * L*) are introduced that can affect the performance.

2) Modeling Get Latency: In this section, we present analytical modeling of the Get operation.

Replication: For the Get operation, data is just fetched from a designated primary server. If this server has crashed, then data is fetched from one of the replicas. Thus, it incurs a fixed server selection overhead and fetches the entire data object *D* in one round-trip, as represented in Figure 3(a). Based on this, for an N-node key-value store server cluster with a replication factor *(F)*, the total time to read a key-value pair using Equation 1 can be simply given as:

$$T_{rep\_get} = T_{check} + L + D/B \quad (4)$$

From Equation 4, we observe that the replication does not incur much overhead for reading data. Hence, this gives good Get latency in a fault-tolerant key-value store system.

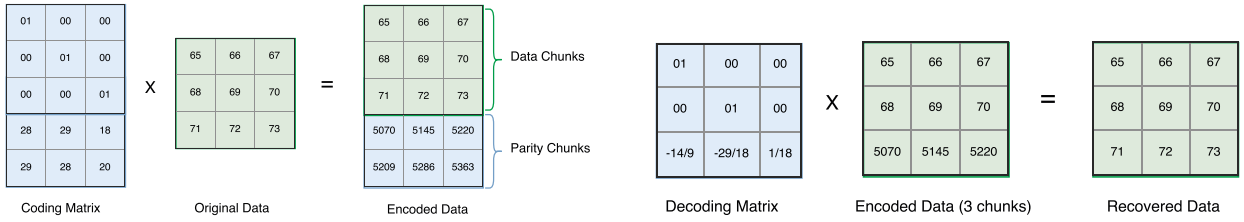
Erasure Coding: An RS(*K, M*) erasure coding scheme requires us to aggregate *K* data or parity chunks. In the case of failures, a re-compute overhead (T_{decode}) is incurred to decode the original data *D* from the available data and parity chunks, as represented in Figure 3(b). Based on this, for an N-node key-value store server cluster that employs RS(*K, M*), the total time to read a key-value pair using Equation 1 can be given as (where N = K + M):

$$T_{era\_get} = T_{decode}(D) + K * (L + D/B * 1/K) \quad (5)$$

From Equation 5, we observe that the erasure coding scheme requires issuing *K-1* additional read requests to fetch smaller chunks that make up the key-value pair, as compared to replication. It may incur additional compute overhead (T_{decode}) in the event of server failures. As a result, data recovery latencies are a major concern in comparison.

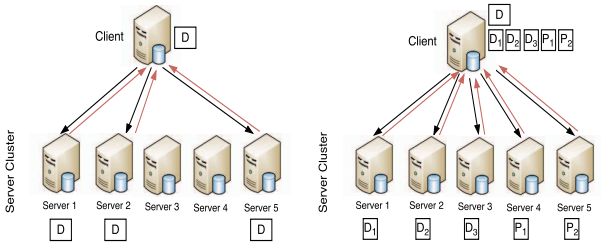
B. Challenges and Opportunities

To better understand the challenges posed by the above model, first, we study the performance of encoding and decoding key-value pairs. Then, we assess the opportunities



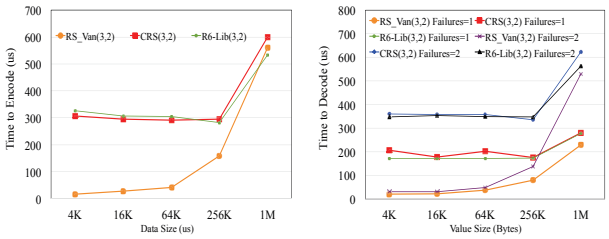
(a) Encoding Example (b) Decoding Example

Fig. 1. Reed-Solomon Erasure Coding for *K=3* and *M=2*



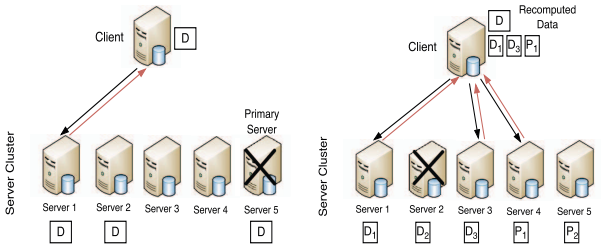
(a) Replication Scheme (b) Erasure Coding Scheme

Fig. 2. Set Operation with Replication and Erasure Coding



(a) RS T_{encode} (b) RS T_{decode}

Fig. 4. Jerasure Library Study for Key-Value Pair Sizes



(a) Replication Scheme (b) Erasure Coding Scheme

Fig. 3. Get Operation with Replication and Erasure Coding

for leveraging advanced HPC technologies to design a resilient key-value store with online erasure coding.

Reed-Solomon Codes Performance Study: We assume that key-value pair sizes supported in this work range from 1 KB – 1 MB, as discussed in Section I-A. With this as basis, we study the performance of erasure coding these key-value pair sizes using the state-of-the-art encoding/decoding library, known as Jerasure (v2.0) [21], to establish which encoding method is most suitable for leveraging in an in-memory key-value store. This library supports a wide variety of erasure codes, including Reed-Solomon coding with Vandermonde matrix (RS_Van) [14], Reed-Solomon coding with Cauchy matrices (CRS) [22], and RAID-6 Liberation codes (R6-Lib) [23].

For these experiments, we assume a 5-node cluster that can tolerate up to two concurrent node failures. Figure 4(a) presents the stand-alone times for encoding (T_{encode}) key-value pairs using the three schemes RS, CRS and R6-Liberation coding on the RI-DQR cluster (see Section VI-A that is equipped with 2.53 GHz Intel Xeon E5630 (Westmere) processors). Assuming that without failures no decoding is necessary, Figure 4(b) presents the corresponding decoding time for single and two node failures (T_{decode}). From these figures, we observe that the basic Reed-Solomon with Vandermonde matrix performs the best for data sizes in our range (1 KB - 1 MB) for both encoding and decoding. The CRS and R6-Liberation codes are optimized Reed-Solomon codes for better performance for large data sizes [24] (approx. 256 MB). Based on this, we choose **RS_Van** coding for performing resilient Set/Get operations pairs with erasure coding in this paper. We refer to this as **RS(K,M)** in the remaining sections (where N = K + M on an N-node cluster).

Opportunities for Leveraging HPC Clusters: From Figure 4, we observe that a noticeable overhead (few 100 microseconds) will be incurred for a key-value pair. Based on the above model (Equations 3 and 5), this affects the critical path of every Set/Get request. On the other hand, modern HPC clusters are equipped with high-performance interconnects such as InfiniBand [25], that enable the middleware to perform Remote-Direct-Memory-Access or RDMA [26]. This advanced HPC feature enables the incoming messages to be processed by the underlying network adapter without the involvement of the host CPU. By exploiting this one-sided communication semantics, it is possible to design a parallel and pipelined Key-Value Store design that can overlap the Reed-Solomon encoding/decoding computation overheads with the Set/Get Request and Response phases.

Based on this, our goal ideally is to explore designs that enable the following:

- For replication, the ideal Set latency, as compared to Equation 2, would be:

$$T_{rep\_set} = \max_{i=1}^{F}(L + D/B) \quad (6)$$

i.e., the maximum latency incurred by any of the replicas.

While much cannot be done for Get latency, any bulk Set/Get request access patterns can overlap the (*D/B*) factor in Equation 4, for reducing overall execution time.

- Similar, for erasure coding, we should be able to reduce the Set latency from Equation 3 to:

$$T_{era\_set} = T_{encode}(D) + \max_{i=1}^{K+M(=N)}(L + D/B * 1/K) \quad (7)$$

i.e., enable parallel communication of the N data/parity chunks. Similarly, the ideal Get latency as compared to Equation 5 should be able to achieve:

$$T_{era\_get} = T_{decode}(D) + \max_{i=1}^{K}(L + D/B * 1/K) \quad (8)$$

Here, as we can see, the erasure coding based scheme for Set and Get operations may be able to further improved by the reduced factor (*L + D/B * 1/K*) of communication. And another important thing is we should overlap the computation time of coding and decoding significantly with the communication part to make online erasure coding become viable.

With this as motivation, we discuss our proposed key-value store design in the following section (Section IV).

IV. Design Overview

To overcome the challenges and opportunities established in Section III, we propose an Online Erasure Coding-enabled In-Memory Key-Value Store that leverages RDMA and Non-blocking Key-Value Store API semantics to maximize performance via optimal computation/communication overlap.

A. Overall Architecture

We have implemented our proposed system on top of RDMA-enhanced Memcached Design [8], [9] that provides novel and high-performance Non-Blocking RDMA-based Libmemcached Set/Get APIs [10]. Figure 5 presents an overview of our proposed system. We design an Asynchronous Request Processing Engine (ARPE) that sits between the Memcached-based application and the RDMA-enhanced Libmemcached Client. It maintains a pool with a fixed number of pre-registered free buffers and a request queue (REQ_QUEUE). All new Set/Get requests are queued in the REQ_QUEUE using the `memcached_iset/memcached_iget` APIs. The ARPE interacts with an RS Encoder/Decoder Module appropriately, based on the type of Key-Value Store request (Set or Get). The ARPE maintains a tunable send/receive window that manages pending request completions using the asynchronous `memcached_wait` API, before proceeding to new requests. On the server-side, we also embed an ARPE engine (with Libmemcached Client and RS Encoder/Decoder), to enable Memcached servers to communicate with each other if necessary. These extensions were plugged into the RDMA-based Memcached server with minimal code changes.

To choose the servers for reading and writing key-value pairs, we leverage the consistent hashing inherent to Memcached to locate the originally designated server, and then

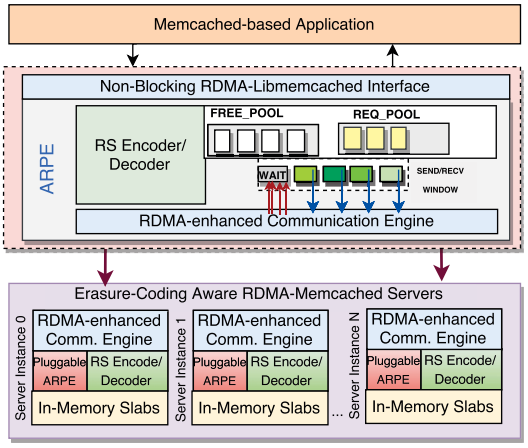


Fig. 5. Proposed Key-Value Store Framework

choose *N - 1* following servers in the Memcached server cluster list (available at each client) to determine the N servers needed to house the *K* data and *N* parity chunks. In this manner, we enable maximizing computation and communication overlap. This overlap is not restricted to each request, but leverages concurrent Set/Get requests-in-progress, for maximizing overall performance. In all our designs, the non-blocking RDMA-Libmemcached APIs enable us to separate and overlap the request and response phases of any Set or Get. Also, the request completion `memcached_test/wait` APIs can help us guarantee consistency semantics similar to that of the default blocking APIs.

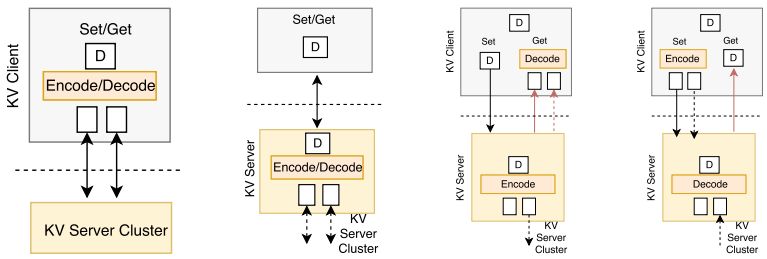
B. Design Choices and their Implications

Our primary goal is to reduce the Set and Get Latencies from Equation 3 and Equation 5 to the ideal Equations 7 and 8, respectively, as much as possible. Hence, we explore flexible key-value store designs that enable offloading Reed-Solomon encoding/decoding at either the Client or the Server, so it can exploit the available resources in the best way possible. We explore the following designs in our framework:

1) Client-Side Encode and Client-Side Decode (Era-CE-CD): In this design, the Key-Value Store Client performs both encoding/distribution of the data and parity chunks for Set, and aggregation/decoding for Get. As represented in Figure 6(a), the Client ARPE enables computation/communication overlap, while the default share-nothing Memcached architecture stays intact (no interaction between Memcached servers is required).

2) Server-Side Encode and Server-Side Decode (Era-SE-SD): In this design, the Server encodes and distributes the data and parity chunks for Set request received. For Get, it aggregates chunks from other servers, decodes and returns the value to the Client. As represented in Figure 6(b) result, all the computing overheads are incurred at the Server ARPE and Memcached Server interaction is required. This coordination overhead can increase T_{comm} considerably instead of benefiting from the parallel and pipeline designs.

3) Server-Side Encode and Client-Side Decode (Era-SE-CD): In this design, the Key-Value Store Client aggregates and



(a) Era-CE-CD (b) Era-SE-SD (c) Era-SE-CD (d) Era-CE-SD

Fig. 6. Design Choices for Enabling Online Erasure Coding in In-Memory Key-Value Store

decodes for Get operations, while the encoding and splitting are performed at the Server for each Set request. As shown in Figure 6(a), this hybrid design leverages compute capabilities at both the Client and the Servers. Since the Memcached Server employs a multi-threaded model, it can exploit its ARPE to improve its throughput. Also, as the Client ARPE incurs T_{decode} only during failures, it can work on optimizing T_{comm}, while the T_{encode} is overlapped at the Server.

4) Client-Side Encode and Server-Side Decode (Era-CE-SD): In this design, the Key-Value Store Client performs encoding/distribution of the data and parity chunks for Set, then Server side aggregates and decodes for Get operation and returns the value to the Client. As represented in Figure 6(d), this hybrid design also enables leveraging compute capabilities at both the Client and the Servers. However, this design still needs to do an extra communication for aggregating chunks from other servers, and incurs decode overhead in the server execution path in the event of server failures. As a result, this hybrid scheme is not the most suitable for optimal performance. Thus, in this paper, we mainly focus on the above three schemes for the evaluations.

Using the framework and design choices described above, we study the performance of the proposed high-performance and resilient In-Memory Key-Value Store with Online Reed-Solomon Erasure Coding.

V. Enabling Erasure-Coding Resilience in Boldio

Boldio [11] is a hybrid and resilient RDMA-enhanced Memcached-based burst-buffer system over global-shared parallel storage such as Lustre for accelerating I/O in Big Data workloads such as MapReduce, Spark, etc. Boldio maps the Hadoop I/O streams onto key-value pairs, and caches them in the remote RDMA-based Memcached server cluster; before asynchronously persisting the file data chunks to the underlying parallel filesystem such as Lustre [27]. It consists of an efficient burst-buffer client/server designs that enable optimal I/O request overlap via non-blocking RDMA-based Libmemcached API semantics and pipelined stages.

To enable the resilience that is critical to Big Data applications, in this infrastructure, Boldio leverages a Client-initiated Replication technique. The Boldio clients guarantee that redundant copies of I/O data exist before the application completes. The Boldio servers are designed to be resilient by being aware of the replicated data blocks that make up the Hadoop I/O files. To contrast our work with this real-world use-case, we study the performance of our proposed Online Erasure Coding-enabled designs we replace the Replication-Aware RDMA-enhanced Memcached Servers used as Key-Value Burst-Buffer layer with our proposed Online Erasure Coding-enabled RDMA-based In-memory Key-Value Store designs, as represented in Figure 7.

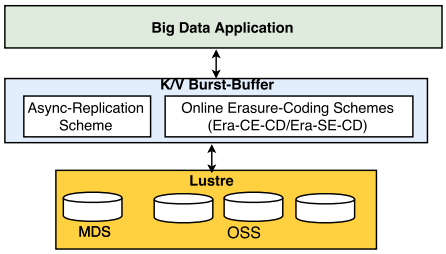


Fig. 7. Boldio Burst-Buffer Design for Big Data I/O on HPC Clusters

VI. Performance Evaluation

In this section, we present the results of our in-depth analysis of the proposed high-performance and resilient key-value store. We divide the evaluations into the following categories:

1) Evaluations with RDMA-based Memcached Latency Micro-benchmarks.
2) Evaluations of Online Cloud-Based Workloads with YCSB.
3) Case study for accelerating Big Data I/O over Lustre PFS with Boldio for HPC clusters.

A. Experimental Setup

We use the following three high-performance compute clusters for our evaluations:

(1) Intel Westmere Cluster (RI-QDR): This cluster is made up of 144 compute nodes, each with two quad-core Intel Westmere processors, 12GB DRAM, 160GB HDD and Mellanox InfiniBand QDR HCAs (32 Gbps). This cluster also consists of storage nodes equipped with 24 GB RAM, two 1TB HDDs, and a single 300GB PCIe-SSD. It consists of a Lustre setup with 1 TB capacity. We use up to 12 compute nodes and 5 storage nodes for our experiments.

(2) SDSC Comet (SDSC-Comet): The Comet supercomputing system at SDSC has 1,984 compute nodes. We use 15

nodes for our evaluations, each node is provisioned with Intel Haswell dual twelve-core processors, 128 GB of memory, 320 GB local SDD, connected via Mellanox IB FDR HCAs (56 Gbps). The cluster consists of a 4 PB Lustre setup.
(3) Intel Broadwell Cluster (RI2-EDR): This cluster is made up of 20 compute nodes, each provisioned with Intel Broadwell dual fourteen-core processors, 128 GB RAM, connected via Mellanox IB EDR interconnects (100 Gbps).

B. Memcached Latency Micro-benchmarks

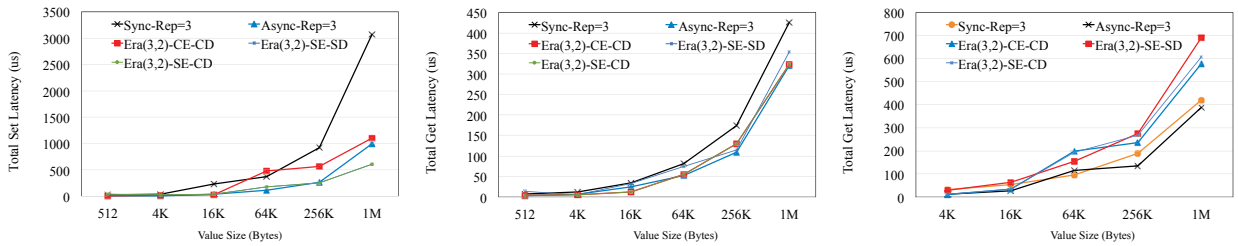
In this section, we perform basic Set/Get latency evaluations with the OHB Memcached Micro-benchmarks for Blocking and Non-Blocking RDMA-Libmemcached APIs [9]. To evaluate our proposed designs, we employ a 5-node RDMA-based Memcached Server cluster. A single client that issues 1K Set/Get operations, for value pair sizes varying from 512 B to 1 MB, and measures the total time taken to satisfy these requests. The key size is fixed to 16 B. We utilize 5 storage nodes on the RI-QDR cluster as servers, where each individual server is configured with 20 GB RAM and 8 worker threads.

We present evaluations with the *Era-CE-CD*, *Era-SE-SD* and *Era-SE-CD* designs with Reed-Solomon RS(3,2) code, and contrast this with two replication-based designs: (1) synchronous three-way replication employing RDMA-Libmemcached's `memcached_set/get` blocking APIs (*Sync-Rep=3*); each key-value pair and/or replicas are accessed synchronously, and, (2) asynchronous three-way replication using RDMA-Libmemcached's `memcached_iset/iget` non-blocking APIs to access the key-value pairs (*Async-Rep=3*); the request/response phases for accessing key-value pairs and their replicas are overlapped. All these configurations support two-node fault-tolerance (as described in Section III).
Write Latency: From Figure 8(a), we observe that *Async-Rep* overlaps concurrent requests to reduce the overall execution time (as per Equation 6), as compared to *Sync-Rep*. More interestingly, we observe that *Era-CE-CD* successfully leverages computation/communication overlap to improve total Set latency by about 1.6x to 2.8x over *Sync-Rep*, and performs close to *Async-Rep* as value sizes increase (as expected from Equation 7). We also observe that the server-side encoding *Era-SE-CD/SD* designs can perform the best on a low-load cluster, by overlapping both T_{comm} and T_{encode} on the server-side that also has the benefits of parallel executing server-side workers. It can improve performance by up to 38% over *Era-CE-CD*, for value sizes >64 KB; the client only needs to send one write request, as opposed to multiple smaller write requests issued by *Era-CE-CD*.
Read Latency and Recovery: Figures 8(b) and 8(c) present Get latency evaluations with no failures and two node failures, respectively. From Figure 8(b), we observe that Online Erasure Coding enabled designs can perform similar to *Async-Rep* as the both employ Non-Blocking Get Requests to overlap the request/response phases. Also, since there are no node failures, there is no decoding overhead. In the case of maximum tolerable server failures, as in Figure 8(c), we observe that *Era-SE-CD/Era-CE-CD* experience performance degradation of about 27%, as compared to *Async-Rep*. As per equation 8, this is an expected phenomenon as it has small scope for overlap ($K * (L + D / B)$, as compared to the Set operation ($N * (L + D / B)$). On the other hand, *Sync-Rep/Async-Rep* only incur a small constant overhead to identify a live server (as per Equation 3). We also observe that *Era-SE-SD* experiences a high degradation of 2.2x, as it has to incur both extra communication and computation overheads on the servers, which causes a high wait-response time at the client. On the other hand, *Era-CE-CD/Era-SE-CD* only incurs compute overheads for some of the request key-value pairs.
Time-Wise Breakdown Analysis: To obtain a clear view of computation/communication overlap exploited to bridge the performance gap between our baseline model and ideal goals, we study the client-side time-wise breakdown for the above Set/Get latency experiments (value sizes 64 KB to 1 MB). These results, as presented in Figure 9, show the three phases: (1) Response-Wait Time (Wait-Response), (2) Request Issue Time (Request), and, (3) Encode/Decode Computation Time. For Set latency, from Figure 9(a), we observe that the request issue phase is dominant for smaller value sizes, and the computation overheads are overlapped with the same. For larger value sizes, the T_{encode} is much more significant, and is overlapped with T_{comm}. For Get latency, from Figure 9(b), we observe that the wait time dominates the overall time, due to the skewed load caused by node failures. From these figures, we observe that *Era-CE-CD/Era-SE-CD* experience compute overheads at the client, whereas *Era-SE-CD* only experiences request/response times (the client only issues one request for each Set/Get and the server performs the encode/decode and distribute/aggregate phases required).
Memory Efficiency: To illustrate the memory efficiency benefits inherent in the proposed erasure coding-enabled designs, we run the OHB Set Micro-benchmark on the same 5-node cluster as above, and measure the percentage of total memory used by *Async-Rep* and *Era-CE-CD/-SE-CD* schemes (represented as Era-RS(3,2)). We vary the number of clients from 1 up to 40, with each client writing 1K key-value pairs of the 1 MB each. From Figure 10, we observe that the *Era-CE-CD/-SE-CD* designs can enable large memory savings (about 1.8x) over Replication-based schemes. For 40 concurrent clients writing 1 GB each, we observe that the *Era-CE-CD/-SE-CD* designs use up only 56% of the total 100 GB aggregated memory available; on the other hand, Async-Rep uses up 100% of the available aggregated memory along with suffering from data loss of about 2 GB.

Based on the above evaluations, it can be seen that *Era-CE-CD* and *Era-SE-CD* give the best overall performance and trade-offs among the proposed design choices. We use these two designs to further evaluate the proposed designs using YCSB [28], and also for our case study with Key-Value Store-based Burst-Buffer systems for Big Data I/O.

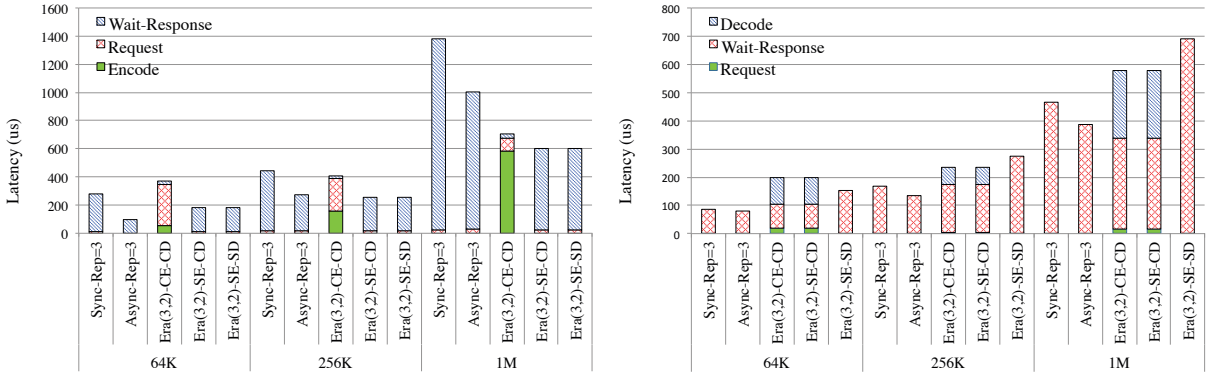
C. Evaluations with YCSB

To illustrate the applicability for Online Data Processing workloads, we study the performance of the proposed In-



(a) Set Latency (b) Get Latency (c) Get Latency with Two Node Failures

Fig. 8. RDMA-Memcached Micro-benchmark Latency on RI-QDR



(a) Set Latency (b) Get Latency with Two Node Failures

Fig. 9. Time-wise Breakdown for Set/Get Latency on RI-QDR

Memory Key-Value Store design that leverages Online Erasure Coding with the YCSB benchmark [28]. This Cloud-based Micro-benchmark suite generates key-value pairs based on the Zipfian distribution that demonstrates workload access patterns with skewed data popularity characteristics. We evaluate the proposed designs with different read/write ratios, including YCSB-A with read:write ratio of 50:50 and YCSB-B with 95:5 read:write ratio. We employ a 5-node Memcached Server Cluster with RS(3,2) encoding, with 64 GB memory per-server, for our proposed designs. For these experiments, we fix the key size to 16 B, and vary the value sizes from 1 KB to 32 KB. We load the Memcached server cluster with 250 K key-value pairs, and deploy 150 concurrent clients on 10 compute nodes with 2.5 K Set/Get requests per-client.

Using this environment, we study the throughput and latencies in a multi-client environment, and contrast with:
(1) default Memcached running over TCP/IP over InfiniBand (IPoIB) without any resilience (Memc-IPoIB-NoRep). This helps us to contrast the best-case performance that can be leveraged by synchronous and non-RDMA based Key-Value stores with Online Erasure Coding such as [16], that use the synchronous model.
(2) RDMA-based Memcached with no replication that employs Non-Blocking Set/Get APIs (Memc-RDMA-NoRep). This illustrates the upper-bound on the overall performance.
(3) Three-way Asynchronous Replication with RDMA-based Memcached (Async-Rep=3).

Figure 11(a) presents YCSB Write and Read Latencies for the 50:50 and 95:5 read/write patterns on the SDSC-Comet cluster, respectively. From this figure, we can observe that, by leveraging RDMA and Non-Blocking API engines in our request processing engine, Era-CE-CD can give up to 2.3x improvement over Async-Rep in a multi-client scenario for 32 KB value size, and has similar performance for smaller value sizes. Figures 12(a) and 12(b) present the throughput studies corresponding to the 50:50 and 95:5 read/write pattern experiments on the SDSC-Comet cluster, respectively. From these figures, we can observe that Era-CE-CD enables us to achieve a high throughput improvement of 1.9-3.01x compared to default Memcached over IPoIB without replication (Memcached-IPoIB-NoRep). Most importantly, Era-CE-CD can achieve up to 1.34x improvement over Async-Rep in throughput for update-heavy workloads (50:50) and average read/write latency by up to 2.3x for both update-heavy and read-heavy workloads, for value sizes of >16 KB. For read-heavy workloads, Era-CE-CD's throughput performs on par with Async-Rep. It can also be seen that the Era-SE-CD design performs similar to Async-Rep for both read/write patterns.

We also extend the above evaluations with 150 clients to the RI2-EDR cluster, that is equipped with higher-end interconnects. Figure 11(b) presents the YCSB Write and Read Latencies for the 50:50 and 95:5 read/write patterns on the RI2-EDR. Similarly, Figure 12(c) presents the corresponding YCSB throughput. From these figures, we can observe that Era-CE-CD design improves throughput by about 1.59x and average latency by over 2.6x over Async-Rep, for the update-

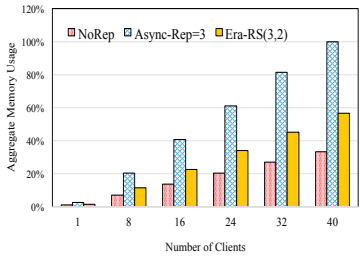
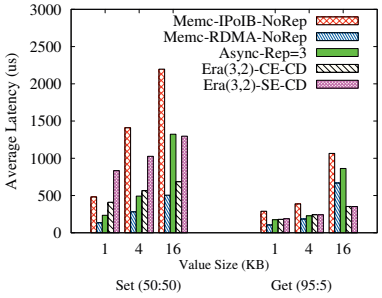
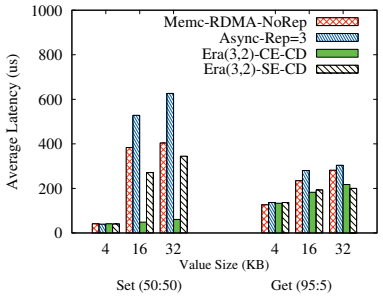


Fig. 10. Memory Efficiency

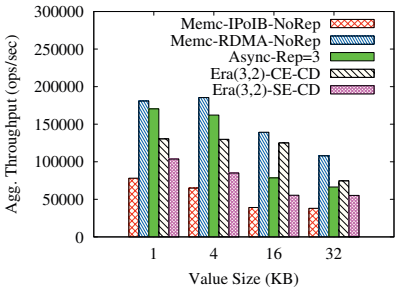


(a) Average Latency on SDSC Comet

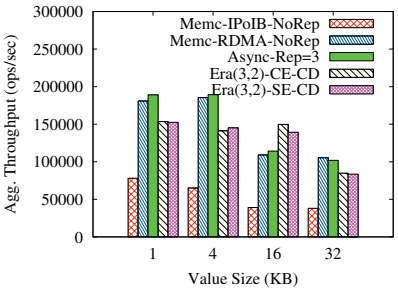


(b) Average Latency on RI2-EDR

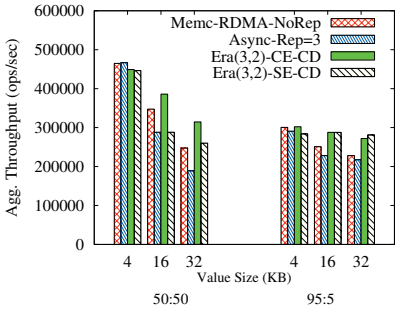
Fig. 11. YCSB Latency on SDSC Comet and RI2-EDR (150 Clients with Skewed Zipfian Pattern)



(a) Throughput 50:50 on SDSC Comet



(b) Throughput 95:5 on SDSC Comet



(c) Aggregated Throughput on RI2-EDR

Fig. 12. YCSB Throughput on SDSC Comet and RI2-EDR (150 Clients with Skewed Zipfian Pattern)

heavy 50:50 access pattern with >16 KB. We attribute this to the Broadwell CPUs and EDR bandwidth on RI2-EDR. Correspondingly, we also observe that Era-SE-CD can improve the performance of update-heavy workloads by about 1.9x (for similar key-value pair sizes). We observe that Era-CE-CD design can be a good fit for skewed cloud-based workloads like YCSB that deal with value sizes of 16 KB and larger.

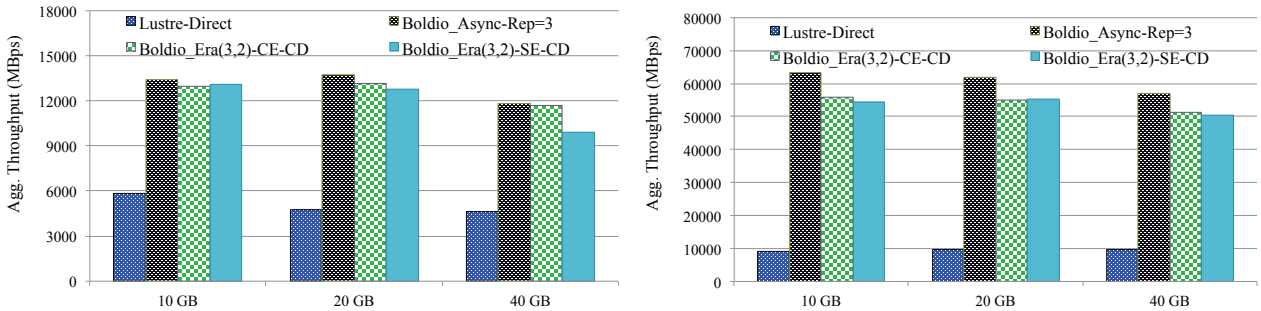
From further analysis, we understand that RDMA-based Memcached uses Eager protocol [8] for values <16 KB and Rendezvous protocol [8] for >16 KB. As a result, since Era-CE-CD splits the 4 KB-32 KB values into chunks <16 KB, it leverages higher performance via smaller latencies, as compared to Async-Rep and Era-SE-CD. Since Era-CE-CD/-SE-CD interacts uniformly with all five servers, it provides better load-balancing for the skewed pattern with high load on servers, as in [29]. This translates to the benefits we observed over asynchronous replication.

D. Evaluations with Burst-Buffer Systems for Big Data I/O

To illustrate the performance benefits of our design for Offline Analytical workloads, we present a case study with the real-world use-case. Boldio [11] is burst-buffer system designed to run over parallel file systems such as Lustre [27], and employs RDMA-based Client-Initiated Asynchronous Replication for fault-tolerance (Boldio_Async-Rep). As described in Section V, we introduced our proposed Online Erasure Coding enabled designs into the Boldio burst-buffer system. In this section, we present Hadoop I/O throughput studies with Boldio_Era-CE-CD and Boldio_Era-SE-CD. We contrast its performance with Boldio_Async-Rep design and Hadoop over Lustre (Lustre-Direct), which is the default HPC approach. We choose Boldio_Era-CE-CD and Boldio_Era-SE-CD as they give the best performance based on our micro-benchmark studies. We use 8 compute nodes on the RI-QDR cluster as Hadoop DataNodes, and a 5-node Boldio server cluster over Lustre. Each Boldio server uses 24 GB memory (total aggregated memory is 120 GB).

Similar to our previous experiments, we employ three-way replication for Boldio_Async-Rep and RS(3,2) for Boldio_Era-CE-CD and Boldio_Era-SE-CD. In order to have a fair distribution of resources, we use 12 DataNodes while running TestDFSIO over Lustre (Lustre-Direct). We run 4 maps-per-host and vary the data size from 10 GB to 40 GB. This refers to 48 concurrent Map tasks for Lustre-Direct, and 32 concurrent Map tasks for Boldio designs. Figures 13(a) and 13(b) present the TestDFSIO Write and Read throughput, respectively. From these figures, we can observe that Boldio_Era-CE-CD and Boldio_Era-SE-CD can achieve up to 2.6x over Lustre-Direct for Write Throughout, and up to 5.9x improvement for TestDFSIO Read throughput. We observe that Boldio_Era-CE-CD incurs no overhead for TestDFSIO writes and less than 9% for TestDFSIO read, as compared to Boldio_Async-Rep. On the other hand, Boldio_Era-SE-CD incurs a slightly larger performance overhead of 3-11% over Boldio_Async-Rep, especially as file output size increases.

Thus, Boldio_Era-CE-CD/-SE-CD can achieve up to 1.84x improvement in memory efficiency over Boldio_Async-Rep (40 GB Hadoop job uses up 120 GB aggregated memory



(a) Write Throughput (b) Read Throughput

Fig. 13. Performance with TestDFSIO Benchmark on RI-QDR (Uniform File Access Pattern)

for Boldio_Async-Rep while Boldio_Era uses up to 67 GB), while maintaining the performance equal to Boldio_Async-Rep. While above experiments demonstrate performance for the no-failure case, recovery overhead is of importance. Hence, we plan to undertake detailed recovery overhead analysis, for both offline and online workloads, as a part of our future work.

VII. Related Work

The idea of leveraging erasure coding for designing resilient systems has been thoroughly researched over the last decade, and it has recently paved its way into designing in-memory caching systems. In this section, we provide a brief overview of some of these related research works.

Erasure Coding support in Storage Systems: In recent years, erasure codes have become popular in several large-scale storage systems including Microsoft Azure [30], Facebook's HDFS cluster [31], Backblaze Vaults [32], Ceph [33], for storing data in a reliable and storage-efficient manner. Similarly, several efforts are being made to bring erasure coding to HDFS [34], [35], including the recent Apache Hadoop 3.x release [36]. These systems are designed for large-scale file storage that is usually made up of immutable data. On the other hand, our work focuses on smaller data objects that constitute both dynamic (online data processing) and immutable (offline analytics) data workloads, on HPC clusters.

Erasure Coding in Key-Value Stores: Several research works have focused on creating high-performance key-value stores suitable for HPC clusters [37], [9], [5], [38], [6], [29], [16]. While most of these works employ replication for fault-tolerance, a recent work namely Cocytus [16], proposes a high-available and space efficient key-value store by leveraging erasure-coding for Online Transactional workloads. On the other hand, EC-Cache [29] is a high-performance caching solution that employs online erasure coding and late binding for load balancing and reduced tail latencies. It is implemented over Alluxio [3] and mainly focuses on Big Data workloads where object sizes >1MB, that are atypical to key-value stores.

As compared to these works, our approach is unique in the sense that we enable Online Erasure Coding as a viable resilience technique for designing a high-performance in-memory key-value store for both Offline and Online Big Data workloads on modern HPC clusters with advanced features such as RDMA; thus, focusing on how to maximize the benefits with Erasure Coding as the primary resilience scheme. We also present a real-world use-case that requires such a memory-efficient and resilient key-value store design, i.e., burst-buffer systems for Big Data I/O.

VIII. Conclusion

In this paper, we present a high-performance and resilient key-value store with online erasure coding support for Big Data workloads. Our design is guided by a detailed modeling-based analysis of the performance tradeoffs of in-memory replication and online erasure coding schemes. The major achievements of our proposed design include: 1) Achieving optimal overlap between encoding/decoding computation with communication by taking advantage of non-blocking Memcached API-based design as well as RDMA feature available on modern HPC clusters; 2) Exploring different encoding/decoding schemes to hide computation and communication overhead as much as possible to make the online erasure coding become a viable alternative to deliver high-performance, resilience, as well as better memory efficiency.

Through extensive performance evaluations on three realistic HPC clusters, we show that our proposed Online Erasure Coding-enabled RDMA-based In-Memory Key-Value Store design can outperform synchronous RDMA-based replication by about 2.8x, and can improve YCSB throughput and average read/write latencies by about 1.34x–2.6x over asynchronous replication for larger key-value pair sizes (>16KB). We also demonstrate that our proposed key-value store designs can achieve up to 2.6x–5.9x improvement for TestDFSIO read/write throughput over Hadoop running directly over the Lustre on HPC clusters, while matching the performance of the Asynchronous Replication scheme employed in Big Data I/O Burst-Buffer designs like Boldio. In the future, we plan to evaluate our proposed designs with: (1) larger-scale Memcached workloads and (2) interactive and iterative Big Data workloads over Apache Spark. Further, we plan to minimize our recovery overheads by incorporating optimized erasure codes such as locally repairable codes, linear time fountain codes, etc., and explore hybrid erasure-coding/replication schemes with the goal of maximizing overall performance and storage efficiency for different workload data access patterns.

REFERENCES

[1] M. Zaharia, M. Chowdhury, T. Das, A. Dave, J. Ma, M. McCauley, M. J. Franklin, S. Shenker, and I. Stoica, "Resilient Distributed Datasets: A Fault-Tolerant Abstraction for In-Memory Cluster Computing," in *Proceedings of the 9th USENIX conference on Networked Systems Design and Implementation*. USENIX Association, 2012, pp. 2–2.

[2] Apache Spark, http://spark.apache.org/.

[3] Alluxio, http://www.alluxio.org/.

[4] "Memcached: High-Performance, Distributed Memory Object Caching System," http://memcached.org/.

[5] H. Lim, D. Han, D. G. Andersen, and M. Kaminsky, "MICA: A Holistic Approach to Fast In-Memory Key-Value Storage," in *Proceedings of the 11th USENIX Conference on Networked Systems Design and Implementation (NSDI 14)*, April 2014.

[6] J. Ousterhout, P. Agrawal, D. Erickson, C. Kozyrakis, J. Leverich, D. Mazières, S. Mitra, A. Narayanan, G. Parulkar, M. Rosenblum *et al.*, "The case for RAMClouds: scalable high-performance storage entirely in DRAM," *ACM SIGOPS Operating Systems Review*, vol. 43, no. 4, pp. 92–105, 2010.

[7] "Designing and Implementing Scalable Applications with Memcached and MySQL, A MySQL White Paper," Oracle Corporation, Tech. Rep., 2010.

[8] J. Jose, H. Subramoni, M. Luo, M. Zhang, J. Huang, M. Wasi-ur Rahman, N. S. Islam, X. Ouyang, H. Wang, S. Sur, and D. K. Panda, "Memcached Design on High Performance RDMA Capable Interconnects," in *Proceedings of the 2011 International Conference on Parallel Processing*, ser. ICPP '11, Washington, DC, USA, 2011.

[9] OSU NBC Lab, "High-Performance Big Data (HiBD)," http://hibd.cse.ohio-state.edu.

[10] D. Shankar, X. Lu, N. Islam, M. Wasi-Ur-Rahman, and D. K. Panda, "High-Performance Hybrid Key-Value Store on Modern Clusters with RDMA Interconnects and SSDs: Non-blocking Extensions, Designs, and Benefits," in *2016 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, May 2016, pp. 393–402.

[11] D. Shankar, X. Lu, and D. K. Panda, "Boldio: A Hybrid and Resilient Burst-Buffer Over Lustre for Accelerating Big Data I/O," in *2016 IEEE International Conference on Big Data (Big Data)*, Dec 2016, pp. 404–409.

[12] Y. Wang, L. Zhang, J. Tan, M. Li, Y. Gao, X. Guerin, X. Meng, and S. Meng, "HydraDB: A Resilient RDMA-driven Key-value Middleware for In-memory Cluster Computing," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, ser. SC '15, 2015, pp. 22:1–22:11.

[13] N. Islam, D. Shankar, X. Lu, M. W. Rahman, and D. K. Panda, "Accelerating I/O Performance of Big Data Analytics on HPC Clusters through RDMA-based Key-Value Store," in *International Conference on Parallel Processing (ICPP)*, Sept 2015.

[14] F. J. MacWilliams and N. J. A. Sloane, "The Theory of Error-Correcting Codes, Part I," *Journal of the society for industrial and applied mathematics*, vol. 8, no. 2, pp. 300–304, 1960.

[15] L. Rizzo, "Effective Erasure Codes for Reliable Computer Communication Protocols," *SIGCOMM Comput. Commun. Rev.*, vol. 27, no. 2, pp. 24–36, Apr. 1997.

[16] H. Zhang, M. Dong, and H. Chen, "Efficient and Available In-memory KV-Store with Hybrid Erasure Coding and Replication," in *14th USENIX Conference on File and Storage Technologies (FAST 16)*. Santa Clara, CA: USENIX Association, Feb. 2016, pp. 167–180.

[17] B. Atikoglu, Y. Xu, E. Frachtenberg, S. Jiang, and M. Paleczny, "Workload Analysis of a Large-Scale Key-Value Store," in *ACM SIGMETRICS Performance Evaluation Review*, vol. 40, no. 1. ACM, 2012, pp. 53–64.

[18] "RDMA over Converged Ethernet," http://www.roceinitiative.org/.

[19] K. Shvachko, H. Kuang, S. Radia, and R. Chansler, "The Hadoop Distributed File System," in *Proceedings of the 2010 IEEE 26th Symposium on Mass Storage Systems and Technologies (MSST)*, ser. MSST '10, Washington, DC, USA, 2010, pp. 1–10.

[20] I. S. Reed and G. Solomon, "Polynomial Codes over Certain Finite Fields," *Journal of the society for industrial and applied mathematics*, 1977.

[21] J. S. Plank, S. Simmerman, and C. D. Schuman, "Jerasure: A Library in C/C++ Facilitating Erasure Coding for Storage Applications - Version 1.2," University of Tennessee, Tech. Rep. CS-08-627, August 2008.

[22] J. Blmer, M. Kalfane, R. Karp, M. Karpinski, M. Luby, and D. Zuckerman, "An XOR-Based Erasure-Resilient Coding Scheme," 1995.

[23] J. S. Plank, "A New Minimum Density RAID-6 Code with a Word Size of Eight," in *2008 Seventh IEEE International Symposium on Network Computing and Applications*, July 2008, pp. 85–92.

[24] J. S. Plank, J. Luo, C. D. Schuman, L. Xu, and Z. Wilcox-O'Hearn, "A Performance Evaluation and Examination of Open-source Erasure Coding Libraries for Storage," in *Proccedings of the 7th Conference on File and Storage Technologies*, ser. FAST '09. Berkeley, CA, USA: USENIX Association, 2009, pp. 253–265.

[25] Infiniband Trade Association, http://www.infinibandta.org.

[26] Mellanox, "What is RDMA?" https://community.mellanox.com/docs/DOC-1963.

[27] Xyratex, "Lustre," http://wiki.lustre.org/index.php/Main_Page.

[28] B. F. Cooper, A. Silberstein, E. Tam, R. Ramakrishnan, and R. Sears, "Benchmarking Cloud Serving Systems with YCSB," in *The Proceedings of the ACM Symposium on Cloud Computing (SoCC '10)*, Indianapolis, Indiana, June 2010.

[29] K. Rashmi, M. Chowdhury, J. Kosaian, I. Stoica, and K. Ramchandran, "Efficient and Available In-memory KV-Store with Hybrid Erasure Coding and Replication," in *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)*. USENIX Association, 2016.

[30] C. Huang, H. Simitci, Y. Xu, A. Ogus, B. Calder, P. Gopalan, J. Li, and S. Yekhanin, "Erasure Coding in Windows Azure Storage," in *Proceedings of the 2012 USENIX Annual Technical Conference (USENIX ATC 12)*, 2012, pp. 15–26.

[31] Facebook's Erasure Coded Hadoop Distributed File System (HDFS-RAID), https://github.com/facebookarchive/hadoop-20.

[32] Backblaze ReedSolomon, https://github.com/Backblaze/JavaReedSolomon.

[33] Ceph - A Scalable Distributed Storage System, http://ceph.com/.

[34] M. Sathiamoorthy, M. Asteris, D. Papailiopoulos, A. G. Dimakis, R. Vadali, S. Chen, and D. Borthakur, "XORing Elephants: Novel Erasure Codes for Big Data," *Proceedings of the VLDB Endowment*, vol. 6, no. 5, pp. 325–336, Mar. 2013.

[35] M. Xia, M. Saxena, M. Blaum, and D. A. Pease, "A Tale of Two Erasure Codes in HDFS," in *13th USENIX Conference on File and Storage Technologies (FAST 15)*. Santa Clara, CA: USENIX Association, 2015, pp. 213–226.

[36] Apache Hadoop 3.0.0-alpha2, http://hadoop.apache.org/docs/r3.0.0-alpha2/.

[37] A. Dragojević, D. Narayanan, M. Castro, and O. Hodson, "FaRM: Fast Remote Memory," in *11th USENIX Symposium on Networked Systems Design and Implementation (NSDI 14)*, 2014, pp. 401–414.

[38] Y. Wang, X. Meng, L. Zhang, and J. Tan, "C-Hint: An Effective and Reliable Cache Management for RDMA-Accelerated Key-Value Stores," in *Proceedings of the ACM Symposium on Cloud Computing*. ACM, 2014, pp. 1–13.