# Parallel Erasure Coding: Exploring Task Parallelism in Erasure Coding for Enhanced Bandwidth and Energy Efficiency

Hsing-bung Chen
High Performance Computing-Design Group
Los Alamos National Laboratory
Los Alamos, New Mexico 87545, USA
hbchen@lanl.gov

Song Fu
Department of Computer Science and Engineering
University of North Texas
Denton, Texas 76203, USA
song.fu@unt.edu

***Abstract*— Very large data sets within the range of megabytes to terabytes generated daily from checkpoint-and-restart processes are seen in today's scientific simulations. Reliability and durability are two important factors to build an archive storage system. Erasure code based object storage systems are becoming popular choices for archive storage systems due to cost-effective storage space saving schemes and higher fault-resilience capabilities. Both erasure code encoding and decoding procedures involve heavy array, matrix, and table-lookup compute intensive operations. Current solutions of the erasure coding process are based on single process approach which is not capable of processing very large data sets efficient and effectively. In this paper, we address the bottleneck problem of single process erasure encoding by leveraging task parallelism offered by multi-core computers. We add parallel processing capability to the erasure coding process. More specifically, we develop a parallel erasure coding software, called *parEC*. It explores the MPI run time parallel I/O environment and integrates data placement process for distributing encoded data blocks to destination storage devices. We evaluate the performance of parEC in terms of both encoding throughput and energy efficiency. We also compare the performance of two task scheduling algorithms for parEC. Our experimental results show parEC can significantly reduce the encoding time (i.e., by 74.06%-96.86%) and energy consumption (i.e., by 73.57%-96.86%), and Demand-based Workload Assignment (DBWA) algorithm can a high system utilization (i.e., 95.23%).**

***Keywords*— *Erasure coding, Parallel I/O, Task parallelism, Workload distribution, Storage system, Power consumption.***

## I. INTRODUCTION

Erasure code based object storage systems are becoming popular choices for archive storage systems due to their cost effective storage space saving schemes and higher fault-resilience capabilities [1]. Customizable redundancy schemes implemented in erasure code are an storage-efficient alternative to the triple or multiple data replications solution implemented in most of today's large-scale data centers. Both erasure code encoding and decoding procedures involve heavy array, matrix, and table-lookup compute intensive operations. Research studies and reports from commercial systems have showed that software-based erasure coding can be deployed on distributed, production storage systems. Long processing time and slow coding bandwidth are major roadblocks to employing software based erasure coding techniques to large datasets. With today's advanced CPU design technologies such as multi-core, many-core, and streaming SIMD instruction sets we can effectively and efficiently adapt the erasure code technology in cloud storage systems and apply it to handle very large-scale date sets.

Our previous testing results have shown that SIMD based erasure code software can help to reduce erasure coding processing time and decrease power consumption [2] [3]. Multi-core processor design has been implemented in modern CPU technology. Each Multi-core processor has its own embedded SIMD sub-system. It is a challenging task to achieve erasure coding parallelism and maximize CPU utilization. Current solution of erasure coding process is based on single process/single node or multi-thread/single node approaches.

To relieve the performance bottleneck of single process erasure encoding approaches, we utilize the task parallelism [4] property from a multicore computing system and improve the erasure coding process with parallel processing capability. We have leveraged open source erasure coding software and implemented a concurrent and parallel erasure coding software, called *parEC*. The proposed parEC software is realized through MPI parallel I/O processing [5] [6] and achieves run-time adaptive workload balancing and scheduling [7]. We also compare the performance of two task scheduling algorithms for parEC. Our experimental results show parEC can significantly reduce the encoding time (i.e., by 74.06%-96.86%) and energy consumption (i.e., by 73.57%-96.86%), and Demand-based Workload Assignment (DBWA) algorithm can a high system utilization (i.e., 95.23%).

The rest of this paper is organized as follows. We present the related research works in Section II. In Section III, we present the architecture, design, and implementation of the proposed parEC software. In Section IV, we present scaling and intensive testing cases and results from performance evaluation and justification. In Section V, we conclude with the current parEC status and future development works.

## II. RELATED WORKS

In [8], authors adopted the transformative Parallel I/O features from PLFS [9] and PFTOOL [10], applied them on an OpenStack Swift object store, and turned a Swift object store into a scalable and parallel cloud object storage system. In [11],

978-1-5090-3315-7/16/$31.00 ©2016 IEEE

authors compared two different erasure coding approaches. They evaluated Gibraltar [12] run on NVIDIA's GPU/CUDA and Jerasure [13] run on Intel x86 based CPUs. They demonstrated the advantages of Gibraltar code using GPU's data vectorization and parallel processing capability. However, without ECC memory technology supported on GPU computing data reliability and integrity is still a concern. In [14], the author proposed an XOR based Cauchy-Reed/Solomon code and implemented it on multicore computing systems. Both data oriented and encoding equation oriented parallel coding approaches were discussed and evaluated in the paper. In [15], the author proposed a CPU and GPU based framework to calculate erasure code and translated encoding/decoding processes into OpenMP/CPU based and OpenCL/GPU based coding modules. The author identified factors that influenced the coding performance. In [16], authors proposed a Parallel Cauchy Reed-Solomon Coding Library for GPU based computing platform. A QoS-aware I/O scheduling framework for supporting software defined storage systems was proposed and evaluated in [17] and the proposed framework was limited to a single server operation mode.

## III. PARALLEL SOFTWARE ARCHITECTURE AND WORKLOAD DISTRIBUTION

Designing parallel software is always a challenging task no matter which parallel programming software is used. The key issue is to find a matched programming model to support the task and data characteristics of an application. We apply the proposed parEC software system on multicore based cluster machines [18]. Obviously MPI is the right selection to implement our proposed parallel encoding software system. We use MPI and MPI-IO library to develop this parEC software system. A diagram of the proposed parEC software system is shown in Figure 1. The parEC can handle two types of data encoding: single object encoding for small files and multi-part-object encoding for large files. The threshold of file size is determined based on our previous testing results [3]. We have designed four types of MPI processes in parEC.

The *Manager process* manages workload distribution from a data queue (DataQ) to available parEC processes, and finalizes job execution. DataQ is an event queue used to store source data information from results of a source file tree walk.

A *DataExplorer process* applies file-tree scans on source file trees, extracts each individual file, inserts file information to the data queue (DataQ), exports sub-directories, and maintains data queue status. The DataExplorer process also handles multi-part-object data partitioning. If a file size is too big, the DataExplorer process invokes a partitioning process, dividing this file into *n* sub-chunks, and inserting information of each sub-chunk into the data queue. By default, only one DataExplorer process is launched. More DataExplorers will be used if a very large data set is processed. We use the extended attributes of a file system to record chunking and encoding metadata information so that we can access and recover the original data objects from encoded data objects.

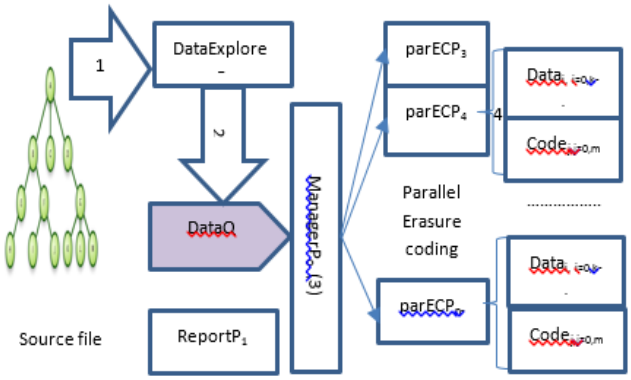


Figure 1: parEC software system diagram and components.

Each *parEC process* communicates with the Manager process, receives workload assignment from the Manager process, conducts erasure coding, generates encoding data (*K* chunks) and code (*M* chunks), and generates encoding metadata information. We reuse open source erasure coding software such as zfec [19] and Intel ISA-L [20] to implement the basic part of the parEC process. In this paper, we use the "Reed-Solomon-Vandermonde" coding method. We change the original Encoder program into an encoding library function and call this encoding function inside a parEC MPI process.

The *Report process* collects encoding process performance data and generates final statistic reports for a parallel erasure encoding job. We generate source file tree information (number of processing files, average size of processing file, file size distribution), erasure coding information (K and M ratio, number of data and code chunks generated) and Erasure coding performance information (processing time and encoding bandwidth).

To schedule parEC processes, we design two workload distribution methods in the manager process: round-robin workload assignment and demand based workload assignment.

- Round-robin workload assignment (RRWA): The Manager process assigns source data to each parEC process based on the round-robin or parEC process sequence. Using this workload assignment, each parEC is expected to receive the same number of source data objects. The object size factor is not considered in workload assignment.
- Demand based workload assignment (DBWA): Each parEC process asks the manager to assign the next source data object when this parEC finishes encoding the last assigned source data object.

## IV. PERFORMANCE EVALUATION

We set up a cluster test-bed with eight server nodes which are used to run MPI processes (Manager, DataExplorer(s), Report, and parECs). Each server has one eight-core CPU, 32GB DDR3 memory, and SIMD processors. The test-bed also has one storage server as the destination storage system. We install PCI-E based Flash storage devices on this storage server. Data chunks and code chunks generated from the coding process are store on the storage server. A Parallel File

System is used to support parallel read and concurrent access of source object files. A 10 Gigabit Ethernet Switch is used to provide data network connectivity.

The erasure encoding is parameterized by two integers, *K* and *M*. *K* is the total number of data chunks produced, and *M* is the total number of code chunks produced. Any *K* chunks of data and code are necessary to reconstruct the original data.

In the first set of experiments, we conduct six different test cases on one 40 GB data set. We evaluate six test cases, where the 40 GB file is evenly divided into 1, 2, 4, 8, 20 and 40 partitions. Each test case has used the same erasure coding ratio (2:1) or 50% extra storage overhead. We apply various data partitioning schemes to study the advantages of applying task parallelism on multicore computer systems. We also measure power consumption for each test case. WattsUp/.net is the power measurement equipment used in erasure code computing power consumption.

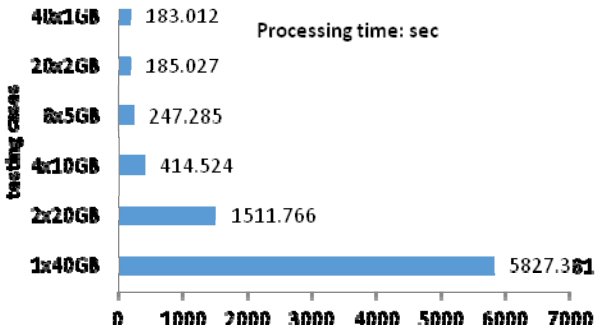


Figure 2: Processing time comparison

Figure 2 shows the processing time of erasure coding comparison among launched parEC processes. We clearly see that the processing time is reduced significantly when multiple parEC processes are performing erasure coding operations. The processing time is reduced by 74.1% and 92.9% when two and four partitions are processed by parEC, respectively. It is further reduced by 96.9% when 40 partitions are encoded.

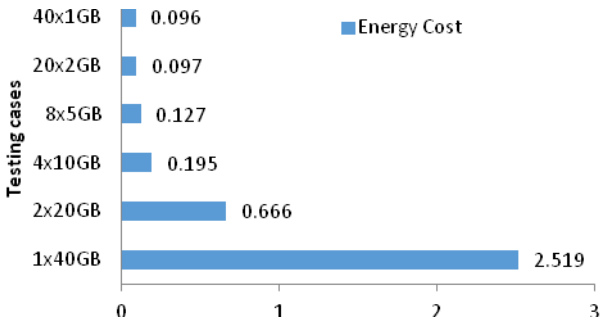


Figure 3: Energy cost comparison

The energy cost generated from WattsUP power meter is shown in Figure 3. The cases of (4x10GB, 8x5GB, 20x2GB, and 40x1GB) show a similar range of energy reduction but they all reduce significant portion of the energy cost when compared to the single process erasure coding approach by more than 92.3%.

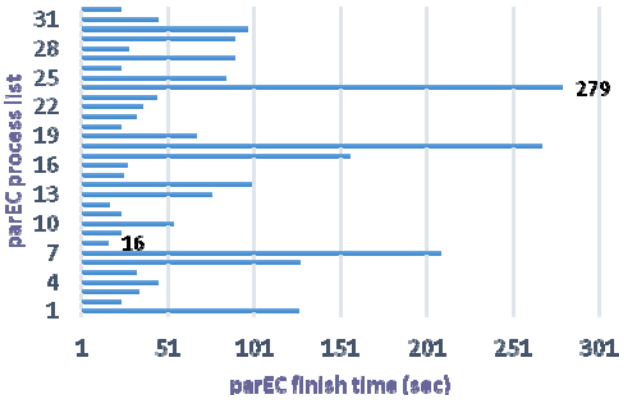


Figure 4: parEC finish time with RRWA workload assignment

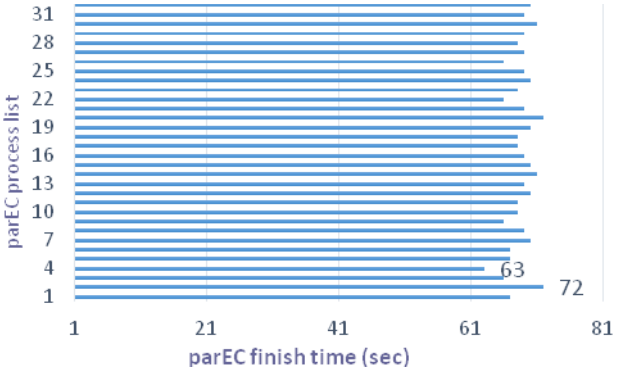


Figure 5: parEC finish time with DBWA workload assignment

In the second set of experiments, we compare the performance of the workload distribution methods, RRWA and DBWA. We launch four parEC processes on each server node. The 32 parEC processes perform erasure coding and we apply 600 various sizes of data objects in this test. Figures 4 and 5 show the results of finish time. The gap of finish time of using RRWA is 263 seconds and the gap of finish time using DBWA is 9 seconds.

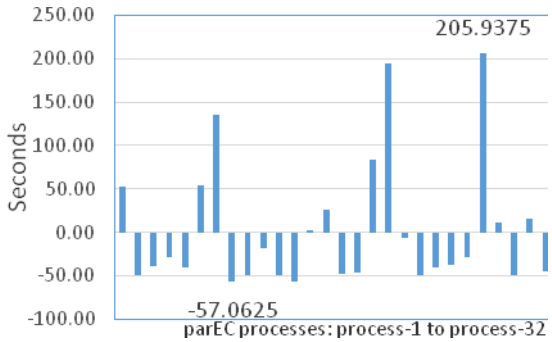


Figure 6: Difference between finish time and the average finish time using RRWA workload assignment

The finish time distribution of RRWA method is within the range [-69%, 282%] (Table 1) or [-57.0625secs, 205.9375secs] (Figure 6) of the average finish time, respectively. But when we use the DBWA method, the finish time distribution is within the range [-8%, 5%] (Table 1) or

[-5.5625secs, 3.4375secs] (Figure 7) of the average finish time respectively. Using DBWA workload assignment method can obtain better universalized finish–time distribution.

Then, we calculate the system utilization of the DBWA method and the RRWA method. DBWA's system utilization is 95.23% and RRWA's system utilization is 26.19% (Table 2).

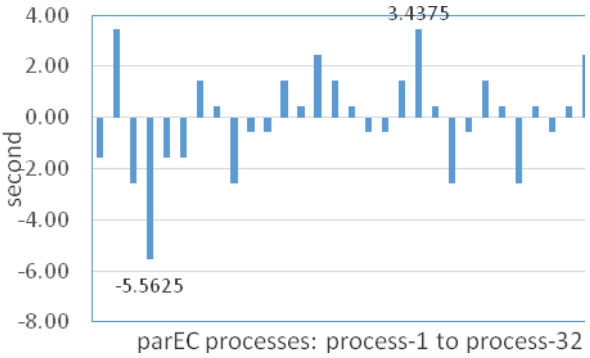


Figure 7: Difference between finish time and the average finish time using DBWA workload assignment

| Method | First Finish Time | Last Finish Time | Gap of Finish Time | Finish time distribution | System Utilization |
|---|---|---|---|---|---|
| RRWA | 16 | 279 | 263 | [-69%,282%] [-57s, 205s] | 26.19% |
| DBWA | 63 | 72 | 9 | [-8%,5%] [5.5s,3.4s] | 95.23% |

Table 1: Finish time distribution and system utilization

## V. Conclusions

In this paper, we leverage open source erasure coding software and MPI task parallelism capability to develop the parEC software using a scalable and parallel approach. We implement the parEC software using two different load balancing methods. Load balancing and system utilization are two major concerns in today's multicore computing systems. Our proposed Demand-Based Workload Assignment (DBWA) solves the problem of inefficient utilization of the current single process based erasure coding solution on multicore computing systems. Our testing results have proved that the parEC software can handle erasure coding efficiently on very large data sets.

## Acknowledgment

This work was supported in part by the U.S. Department of Energy and LANL ASC funding. We thank the technical supports from LANL's HPC-5 group and HPC-3 group. We also thank graduate students from the University of North Texas who helped on testing and verification of the parEC software. The publication has been assigned the LANL identifier LA-UR-16-21178.

## References

[1] James S. Plank, Erasure Codes for Storage Systems A Brief Primer, *USENIX .login*, Vol. 38 No. 6, 2013.

[2] Hsing-bung Chen, Ben McClelland, et al., An Innovative Parallel Cloud Storage System using OpenStack's Swift Object Store and Transformative Parallel I/O Approach, *Los Alamos National Lab Science Highlights*, 2013.

[3] Corentin Debains, Gael Alloyer, Evaluzation, Evaluation of Erasure-coding libraries on Parallel Systems, 2010.

[4] Peter Sobe, Parallel Reed/Solomon Coding on Multicore Processors, in *Proceedings of International Workshop on Storage Network Architecture and parallel I/O*, 2010.

[5] Babak Behzad, Improving parallel I/O auto tuning with performance modeling, in *Proceedings of ACM International Symposium on High-performance Parallel and Distributed Computing (HPDC)*, 2014.

[6] Hsing-bung Chen, parEC – A Parallel and Scalable of erasure coding support in Cloud Object Storage Systems, Los Alamos National Lab.

[7] A. Varbanescu , On the Effective Parallel Programming of Multi-core Processors, Ph.D Thesis, Technische Universiteit Delft , 2010.

[8] William Gropp Ewing Lusk, Anthony Skjellum, Using MPI: Portable Parallel Programming with the Message-Passing Interface, The MIT Press, 2014.

[9] Hsing-bung Chen, Parallel Workload Benchmark on Hybrid Storage EcoSystem, Los Alamos national Lab.

[10] Adam Manzanares, John Bent, Meghan Wingate, and Garth Gibson, The Power and Challenges of Transformative I/O, in *Proceedings of IEEE International Conference on Cluster Computing (CLUSTER)*, 2012.

[11] Hsing-bung Chen, Gary Grider, et al., Integration Experiences and Performance Studies of A COTS Parallel Archive System, in *Proceedings of IEEE International Conference on Cluster Computing (CLUSTER)*, 2012.

[12] Gibraltar: A Reed-Solomon coding library for storage applications on programmable graphics processors, *Concurrency and Computing: Practice and Experience*, 2011.

[13] Jerasure: Erasure Coding Library, http://jerasure.org/

[14] Zfec - a fast erasure codec which can be used with the command-line, https://pypi.python.org/pypi/zfec

[15] Jianzong Wang and Lianglun Cheng, qSDS: A QoS-Aware I/O Scheduling Framework towards Software Defined Storage, in *Proceedings of ACM/IEEE Symposium on Architectures for Networking and Communications Systems (ANCS)*, 2015.

[16] Peter Sobe, Parallel coding for storage systems - An OpenMP and OpenCL capable framework, *Lecture Notes in Informatics*, 2012.

[17] Xiaowen Chu, Chengjian Liu, et al., PErasure: A parallel Cauchy Reed-Solomon coding library for GPUs, in *Proceedings of IEEE Conference on Communications (ICC)*, 2015.

[18] Hsing-bung(HB) Chen and Song Fu, PASSI: A Parallel, Reliable and Scalable Storage Software Infrastructure for Active Storage System and I/O Environments, in *Proceedings of IEEE International Performance Computing and Communications Conference (IPCCC)*, 2015.

[19] Hsing-bung Chen, Gary Grider, et al., An Empirical Study of Performance, Power Consumption, and Energy Cost of Erasure Code Computing for HPC Cloud Storage Systems, in *Proceedings of IEEE International Conference on Networking, Architecture, and Storage (SC)*, 2015.

[20] The Intel Intelligent Storage Acceleration Library (Intel ISA-L), https://01.org/sites/default/files/documentation/isa-l_open_src_2.14.pdf