Evaluating Information Dispersal Algorithms

Corentin Debains[1], Pedro Alvarez-Tabio[1], and Ioan Raicu[1,2]
[1]Illinois Institute of Technology, Chicago, IL, USA
[2]Argonne National Laboratory, Argonne, IL, USA

***Abstract*—The explosion in data acquisition and storage has led to the emergence of data-intensive applications that are used to process enormous quantity of information using methods such as the MapReduce paradigm. Data-Intensive Distributed File Systems (DI-DFS) have been designed to support these kinds of applications. These large-scale storage systems require fault-tolerance mechanisms to handle failures, which are a norm rather than an exception when working at a scale that will shortly reach exascale. A new trend among large-scale systems is the implementation of information dispersal algorithms, called erasure codes. The overhead introduced by the encoding and decoding can be a limiting factor for the integration at large-scale. This work compares two different approaches of erasure code computing on GPU and CPU. Our work in erasure coding serves as the foundation for the next step: integrating an information dispersal algorithm that eventually outperforms current state-of-the-art approaches in DI-DFSs.**

I. Introduction

The reliability of a computer system refers to the property that a system can run continuously without failure. Here "without failure" by no means indicates that failures do not happen. Rather, with data-intensive applications deployed on extreme-scale distributed systems, failures are the norm instead of the exception. The reliability a computer system could achieve becomes a problem of how well failures can be handled. Ideally, these failures should be completely transparent to the users, with a relatively low or even negligible cost. Keeping high reliability is one of the most important metrics for high performance computing (HPC) and cloud computing, and is often listed as mean-time-to-failure (MTTF) in the service-level agreement (SLA).

One of the most commonly used techniques to make data highly reliable is replication. For example, Google File System (GFS) [2] makes 3 replicas as the default. The Hadoop distributed file system [7] also uses replication to achieve high reliability. This technique is often sufficient: it is easy to implement and has excellent performance (assuming data are replicated asynchronously), at the cost of space efficiency. For example, with the original data and 3 replicas, the storage utilization rate is only $\frac{1}{1+3} = 25\%$. In this case the cost of storage is quadrupled when building a distributed system, which might not be economically acceptable in many applications. Another drawback of replication is that it consumes network bandwidth to migrate data across different nodes to maintain the consistency and reliability of replicas. Moreover, replicating the intact and non-encrypted data can potentially expose more security holes.

Other than replication, another important technique of data redundancy is erasure coding which is well known for its storage efficacy. Erasure coding partitions a file into multiple fragments which are encoded and stored on different nodes. Literature [6, 9] shows that erasure coding delivers a better space efficiency but, unfortunately, cannot meet the bandwidth requirement for a large-scale distributed file system because the encoding/decoding computation hits the bottleneck of CPU capacity, thus could not saturate the network bandwidth. With the state-of-the-art GPU/many-core technology, this computing bottleneck could potentially be alleviated.

In this work we systematically study the effectiveness of information dispersal algorithms (IDA) that operate using both CPUs and GPUs in a single-node basis. We believe this study will enlighten the design and implementation of the future generation of distributed systems.

II. Erasure Coding

Erasure coding, together with data replication, are the two major mechanisms to achieve data redundancy. Erasure coding has been studied by the computer communication community since the 1990's [4], as well as in storage and file systems [3]. Figure 1 describes what the encoding process looks like.

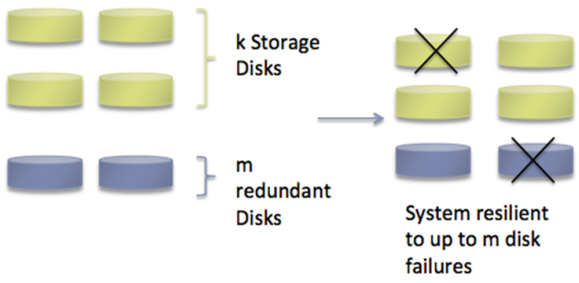
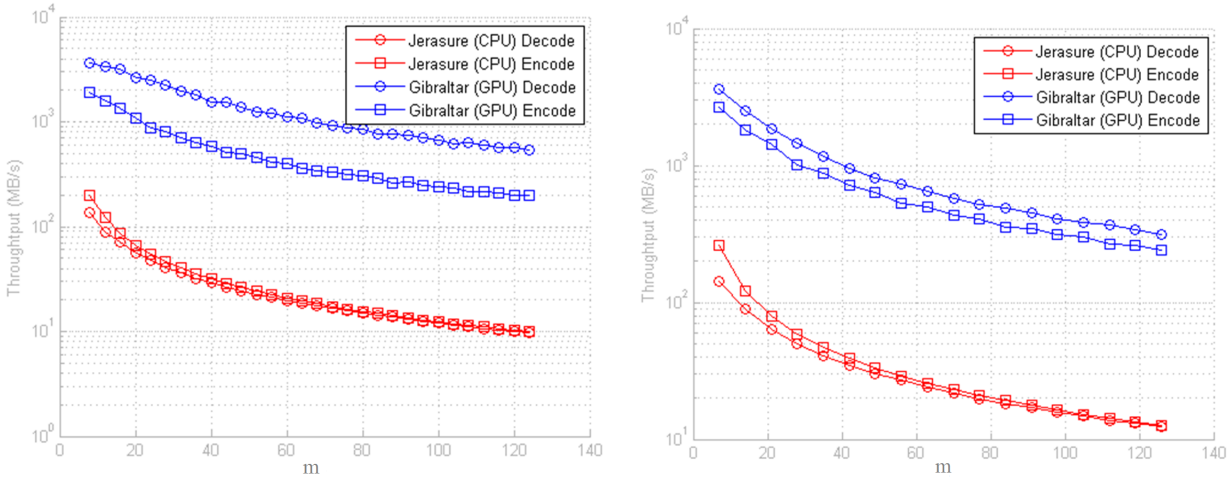


Fig. 1. Encoding k chunks into $n = k + m$ chunks so that the system is resilient to m failures.

The idea is straightforward: a file is split into k chunks and encoded into $n > k$ chunks, where any k chunks out of these n chunks can reconstruct the original file. We denote $m = n - k$ as the number of redundant or parity chunks. We assume each chunk is stored on a distinct storage disk. That assumption is supported by Weatherspoon et al. in [8],where they showed that, for N the total number of machines and M the number of unavailable machines then the availability of a chunk (or replica) A can be calculated as

$$A = \sum_{i=0}^{n-k} \frac{\binom{M}{i}\binom{N-M}{n-i}}{\binom{N}{n}}.$$

Compared to data replication, erasure coding has 3 important features. First, it offers a higher storage efficiency, denoted



(a) $E_{storage} = 33\%$ (b) $E_{storage} = 75\%$

Fig. 2. Encoding and decoding throughput with 1MB buffer size.

as $E_{storage}$, which is defined as $\frac{k}{n}$. The second advantage of erasure coding is the higher efficiency of storage, fewer copies of data exist in the system, which in turn saves the network bandwidth for data migration. The last and often underrated one is security. Rather than copying the intact and non-encrypted data from one node to another, erasure coding chops the data and encodes the chunks to disperse them into multiple nodes.

The drawback of erasure coding comes from its compute intensive nature. It placed an extensive burden on the CPU, which makes it impractical in the today's production distributed file systems that require high performance.

III. Libraries

Jerasure [5] is a C/C++ library that supports a wide range of erasure codes: RS coding, Minimal Density RAID-6 coding, CRS coding and most generator matrix coding. One of the most popular codes is the Reed-Solomon encoding method, which has been used for the RAID-6 disk array model. This coding can either use Vandermonde or Cauchy matrices to create generator matrices.

Gibraltar [1] is a CUDA Reed-Solomon coding library for storage applications. It has been demonstrated to be highly efficient when tested in a prototype RAID system. This library is known to be more flexible than other RAID standards; it is scalable with parity's size of an array.

IV. Preliminary Results

The encode and decode throughput of Jerasure and Gibraltar are plotted in figure 2(a) with m increasing from 2 to 128 while keeping $E_{storage} = 33\%$. The buffer size is set to 1MB. In all cases, with larger m values, the throughput decreases exponentially. This is because when the number of parity chunks increases, encoding and decoding take more time which reduces the throughput. A more interesting observation is the gap between Gibraltar and Jerasure for both encode and decode. There is more than 10X speedup with Gibraltar which suggests that GPU-based erasure coding would likely break through the CPU bottleneck in distributed file systems.

We then change the storage efficiency $E_{storage} = 75\%$ and measure the throughput with different m values in figure 2(b). Similar observations and trends are found just like the case $E_{storage} = 33\%$.

References

[1] M. L. Curry, A. Skjellum, H. Lee Ward, and R. Brightwell. Gibraltar: A Reed-Solomon coding library for storage applications on programmable graphics processors. *Concurr. Comput. : Pract. Exper.*, 23(18):2477–2495, Dec. 2011.

[2] S. Ghemawat, H. Gobioff, and S.-T. Leung. The Google file system. In *Proceedings of the nineteenth ACM symposium on Operating systems principles*, SOSP '03, 2003.

[3] O. Khan, R. Burns, J. Plank, W. Pierce, and C. Huang. Rethinking erasure codes for cloud file systems: minimizing I/O for recovery and degraded reads. In *Proceedings of the 10th USENIX conference on File and Storage Technologies*, FAST'12, pages 20–20, Berkeley, CA, USA, 2012. USENIX Association.

[4] A. J. McAuley. Reliable broadband communication using a burst erasure correcting code. In *Proceedings of the ACM symposium on Communications architectures & protocols*, SIGCOMM '90, pages 297–306, New York, NY, USA, 1990. ACM.

[5] J. S. Plank. Jerasure: A library in C/C++ facilitating erasure coding for storage applications. Technical report, University of Tennessee, 2007.

[6] R. Rodrigues and B. Liskov. High availability in DHTs: erasure coding vs. replication. In *Proceedings of the 4th international conference on Peer-to-Peer Systems*, IPTPS'05, 2005.

[7] K. Shvachko, H. Kuang, S. Radia, and R. Chansler. The Hadoop distributed filesystem: Balancing portability and performance. In *Proceedings of the IEEE 26th Symposium on Mass Storage Systems and Technologies (MSST)*, 2010.

[8] H. Weatherspoon and J. Kubiatowicz. Erasure coding vs. replication: A quantitative comparison. *Peer-to-Peer Systems*, 2002.

[9] H. Xia and A. A. Chien. RobuSTore: a distributed storage architecture with robust and high performance. In *Proceedings of the 2007 ACM/IEEE conference on Supercomputing*, SC '07, 2007.