Journal of Computer Science

Original Research Paper

A Fast Access Big Data Approach for Configurable and Scalable Object Storage Enabling Mixed Fault-Tolerance

[1]Carlos Roberto Valêncio, [1]André Francisco Morielo Caetano, [2]Angelo Cesar Colombini, [3]Mário Luiz Tronco and [4]Márcio Zamboti Fortes

[1]Department of Computer Science and Statistics - DCCE, São Paulo State University (Unesp), Institute of Biosciences, Humanities and Exact Sciences (Ibilce), Campus São José do Rio Preto, São Paulo, Brazil
[2]Department of Computer Science and Statistics, Federal University of São Carlos (UFSCar) São Carlos, São Paulo, Brazil
[3]Department of Mechanical Engineering – EESC, São Paulo University (USP) São Carlos, São Paulo, Brazil
[4]Department of Electrical Engineering – TEE, Fluminense Federal University (UFF), Niterói, Brazil

Article history
Received: 23-04-2017
Revised: 08-06-2017
Accepted: 20-06-2017

Corresponding Author:
Marcio Zamboti Fortes
Department of Electrical Engineering – TEE,
Fluminense Federal University (UFF), Niterói, Brazil
Email: mzf@vm.uff.br

Abstract: The progressive growth in the volume of digital data has become a technological challenge of great interest in the field of computer science. That comes because, with the spread of personal computers and networks worldwide, content generation is taking larger proportions and very different formats from what had been usual until then. To analyze and extract relevant knowledge from these masses of complex and large volume data is particularly interesting, but before that, it is necessary to develop techniques to encourage their resilient storage. Very often, storage systems use a replication scheme for preserving the integrity of stored data. This involves generating copies of all information that, if lost by individual hardware failures inherent in any massive storage infrastructure, do not compromise access to what was stored. However, it was realized that accommodate such copies requires a real storage space often much greater than the information would originally occupy. Because of that, there is error correction codes, or erasure codes, which has been used with a mathematical approach considerably more refined than the simple replication, generating a smaller storage overhead than their predecessors techniques. The contribution of this work is a fully decentralized storage strategy that, on average, presents performance improvements of over 80% in access latency for both replicated and encoded data, while minimizing by 55% the overhead for a terabyte-sized dataset when encoded and compared to related works of the literature.

Keywords: Erasure Coding, Data Storage, Cache, Object Storage, Big Data

Introduction

Large-scale data, or Big Data, resilient storage is one of the major problems addressed in terms of infrastructure support in computer science (Alnafoosi and Steinbach, 2013) (Hashem *et al*., 2015). This means that when it comes to valuable information, storage systems design needs planning in such a way that no data is ever lost, regardless of external faults or factors common to any computational environment, such as hard disk failures and server crashes. In this sense, many of the existing state-of-the-art technologies use a replication methodology, which consists of entirely copying and storing data at different locations, often geographically distant, thus adding a degree of redundancy (Gonizzi *et al*., 2015). Although this technique has proved to be reasonably efficient in several scenarios and is still pertinent in many contexts, pure replication has its disadvantages. The biggest and most obvious is the increase in the required disk capacity to store a given dataset, which also implies a greater overhead on each update to keep identical copies as well as increases in hardware time and resources costs (Weatherspoon and Kubiatowicz, 2002). In this sense, new techniques have been progressively studied and introduced in distributed environments, with emphasis on methods that use error correction codes, also known as erasure codes (Khan *et al*., 2012). Erasure coding

algorithms split a data object *d* into *n* parts, each part significantly smaller than *d*. Not being full copies, these parts are instead generated using algebraic and logical operations in order to store only a fraction of the original data. Given an arbitrary number $k < n$, optimal erasure coding algorithms provide that the data object *d* can be fully restored by combining any *k* parts, therefore tolerating up to any combination of possible *n*-*k* failures. As these parameters (*n*, *k*) are configurable, an erasure coding algorithm could provide the same fault-tolerance of replication methods, but instead providing much less storage overhead (Li and Li, 2013). As examples of method application we can cite the research of Hyun *et al*. (2017; Al-Awami and Hossanein, 2016; Gribaudo *et al*., 2016).

To avoid having to transfer all parts of an erasure coded data object through the network of a distributed system every time the object is accessed, which could stress the network and decrease access time, one alternative is to use some sort of caching mechanism. A very common caching technique is LRU, which implements a least recently used policy, swapping data in and out of the faster access storage area based on how recent was the last request to it (Li *et al*., 2014). There is, though, an alternative technique with far better hit ratio (Megiddo and Modha, 2004), which means better usage of the limited cache area, than LRU. Albeit having more complex implementation, the adaptive replacement cache, or ARC, policy utilizes the history of content removed from the cache and dynamically allocates data on the faster access storage based on both recency and frequency.

Ma *et al*. (2013) proposed the CAROM storage architecture in their work, which was one of the first in academic literature to suggest the combined use of replication and erasure codes. The work focuses on cloud environments and the tests performed at a datacenter level. There is a per-datacenter global caching strategy implemented, which however uses the very simple LRU algorithm. This caching strategy uses RAM as a media for faster access. More details on this cache are available, but the architecture used is not Peer-to-Peer (P2P), which may lead to single point of failure issues.

Later in related works we find Robot storage architecture (Yin *et al*., 2013), which relies on the sole usage of erasure codes for data storage and ignores replication as combined approach, which according to recent studies may be an error (Gribaudo *et al*., 2016). However, it still presents good results and proposes a mix of architectures, since in a general overview there is clearly the figure of the master computers, which are those responsible for encoding and decoding the stored data, as well as controlling the metadata. On a second part, the authors use a P2P ring network of computers for data storage only and no further processing. It does not present any caching strategy to optimize access to this data.

The work proposed as the HDFS-Xorbas architecture (Sathiamoorthy *et al*., 2013) relies on the HDFS distributed file system (Borthakur, 2008). For that reason, the architecture is similar or more likely an extension to this file system, which design is master-slave and therefore subject to single point of failure problems. The main contribution of the paper is to provide an erasure codes scheme for HDFS, which initially uses only three-way replication. The work's proposal is a new type of code and authors implement it in an integrated way to this already existing technology, with good results, but still forcing the use of erasure codes or replication, not both together. It also does not use any kind of caching mechanism.

The work proposed by Tang *et al*. (2015), the MICS architecture, is more recent and based partially on the previous work that brought the CAROM architecture, but with some notable differences. It uses a management model with multiple masters and proposes the storage in the form of objects, besides having as one of the main contributions the creation of an update function for the stored objects, since this function usually depends on removing the object and reinserting it, because there is no direct update. At the end of the article, the authors suggest that the use of cache could be desirable in works of this nature, but they chose not to implement in their case.

The work proposed as the HRSPC architecture (Li *et al*., 2016) focuses on directly improving some aspects of erasure codes in order to merge them into a mixed algorithm rather than using the two techniques separately as other works. This, however, makes the work much more theoretical than applied. Not many architectural details are given and although it does work well as other works, it does not use P2P architecture explicitly, but it does suggest some concepts in that sense. It also does not feature a cache system, though it does suggest that using such a technique can reduce disk read costs.

The outline of this paper is organized as follows: First, there will be a section to describe the developed work, along with the contributions related to previous works. The following sections will present materials, methods and experiments performed in order to evaluate our work. There will be a section dedicated to the discussion of results obtained and how these results relate to other works in literature. The final section will present conclusions on this paper.

Proposed Work and Contributions

This work contribution is comprised of a fully decentralized storage architecture with a cache mechanism to improve response time. The architecture was named Griddler - namesake to a Japanese numerical

puzzle in which the objective is to reconstruct data that at first is unknown.

The developed solution complements and expands the other storage architectures in order to study the possibility and benefits of a fully decentralized P2P storage system to mitigate single points of failure in a network of nodes. Li and Liao (2005) comment about load balancing problem in P2P systems in their research. There is no single point of failure, as we use the Chord P2P protocol (Stoica *et al*., 2001). While an evaluation determines on whether a fully distributed system favors performance in data encoding and decoding, we also consider and provide replication as an alternative available when necessary. By supporting simultaneously and not alternately, replication and erasure coding of data, it is unique amongst other architectures because it supports efficient storage for data both hot - frequently accessed, more suitable for replication - and cold - rarely accessed, more suitable for coding. Awareness of hot and cold data access patterns is imperative for Big Data systems and analytics, as shown by authors of previous works (Kambatla *et al*., 2014). In addition to those contributions, this work studies the behavior of an improved cache algorithm that overcomes the limitations of the LRU algorithm used in CAROM. For this work, the selected cache algorithm was ARC, an Adaptive Replacement Cache algorithm, which integrates the architecture. ARC well knowingly outperforms LRU and experimental results for disk-read large web requests have shown scenarios where the hit ratio for ARC is over 40% and LRU just about 27% (Megiddo and Modha, 2004).

On Fig. 1 it is possible to observe a simple read operation and the placement of the cache, which is also local to each node and not global, as in CAROM. The read, or GET, operation converts the requested input key, for example “video1.mp4”, into an unique string by means of a secure hashing algorithm. That key is used to exactly identify the data object, which was previously stored in the network and has the same hash. Such behavior is defined by the Chord protocol. The search goes through the adaptive replacement cache area, which has two lists for both recently and frequently used data and these lists size increase or decrease based on another two metadata lists, the ghost lists. All cached data is stored on RAM. Given a cache hit, data returns without further operations. After a chache miss, the request forwards to other peers of the network using a routing table, also known as finger table. Ultimately the request resorting to disk if not found in any cache areas of any nodes.

Figure 2 is a representation of the complete topology used in Griddler architecture, which is a ring, as necessary for the Chord protocol. The use of cache on each node is the great differential of this architecture.

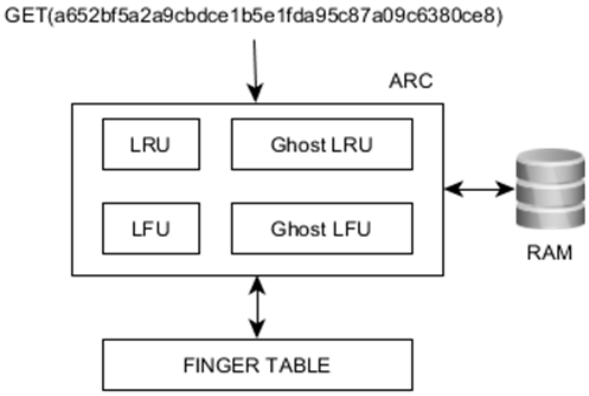

Fig. 1. Overview of access by means of ARC, in Griddler

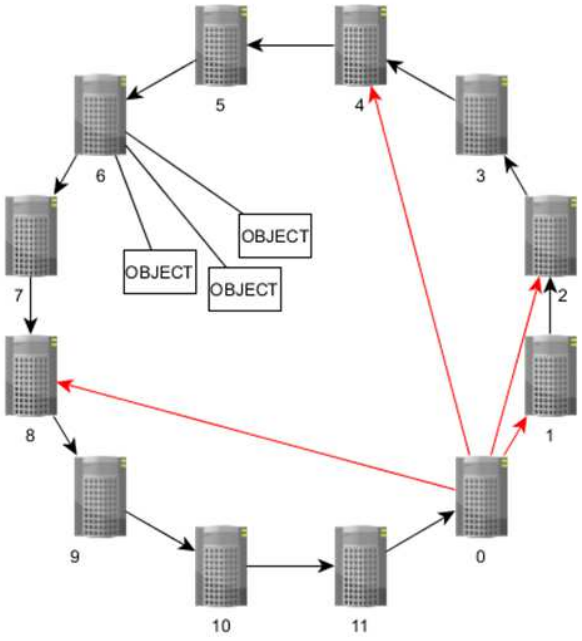

Fig. 2. Network topology for storage environment

Materials and Methods

There were four experiments conducted in order to produce this manuscript in order to measure latency of access with and without cache, processing time for encoding data, processing time for decoding data and total overhead of data when replicated or encoded. In order to write this paper the testing environment used hardware virtualization running on a single physical computer, although we are considering future experiments in larger distributed environments. The general specifications of the physical computer were an Intel Core i7 Haswell - 4700MQ 2.4 GHz, 6 MB Cache (3.40 GHz with Max Turbo) processor, 16 GB Corsair Vengeance DDR3 (1600 MHZ)/(2×8 GB)

RAM and a 1TB-7200 RPM HDD. Atop the hardware, there were several virtual servers with one virtual CPU, 2 GB of RAM and 50 GB storage space, connected to the same virtual network. All virtual nodes created use the 64-bit version of the Linux operating system. Our data varied between single-object sets with a few megabytes up to a 1 terabyte set with thousands of objects, thus not standing back when compared to other works.

Griddler consists entirely of C/C++ code, including programming libraries that implement the methods for performing the measurements described on the tests section. Experiments were conducted under low stress, nearly idle, operating systems, with dedicated resource usage and a short interval between each test. As it will be described on each test, we utilize standard methods for obtaining the results in each tests, always working with the average of a relevant amount of measurements, such as defined by the RFC 2544 (Bradner and McQuaid, 1999) for latency measurements. Reproducing the following tests would be as simple as to recompile the code on a different Linux environment. The source code is not yet available but full disclosure is something the authors are considering, even though it is under constant improvements.

Results

The first experiment intended to verify the latency of data access, with and without the use of the cache. For this experiment, latency measurement model followed the definitions of RFC 2544 therefore the average of 20 measurements performed for each data object, which in this case are large binaries. The measurements incurred from the variation of the size of objects stored with three-way replication or with erasure coding. For this and following tests, the erasure coding algorithm used was Liberation (Plank, 2008) with parameters (6,2). Results on Table 1 and Fig. 3 represent measurements for replicated data. Erasure coded data measurements follow on Table 2 and Fig. 4.

Second experiment intends to show that each node has enough processing power to contribute for erasure encoding operations. Therefore, storage architecture would benefit from a fully decentralized strategy. There would be no need for masters or nodes with specific functions of processing and storage like previous works suggested. In this experiment a varying number *n* of binary data objects, each with 10 MB of size, composed each dataset. Datasets size varied from 50 GB up to 1 TB of data and operations ran on a single node. Results follow on Table 3 and Fig. 5.

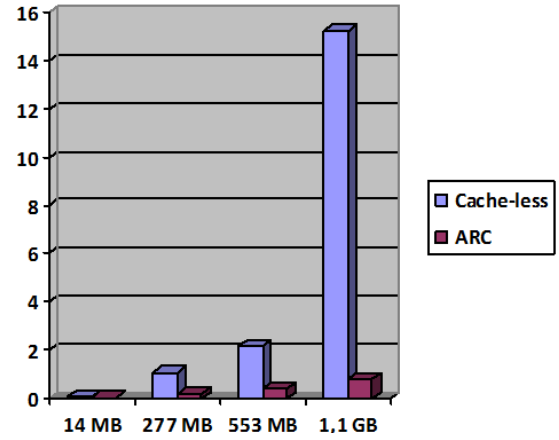

Fig. 3. Latency graphical comparison for replicated data (in seconds)

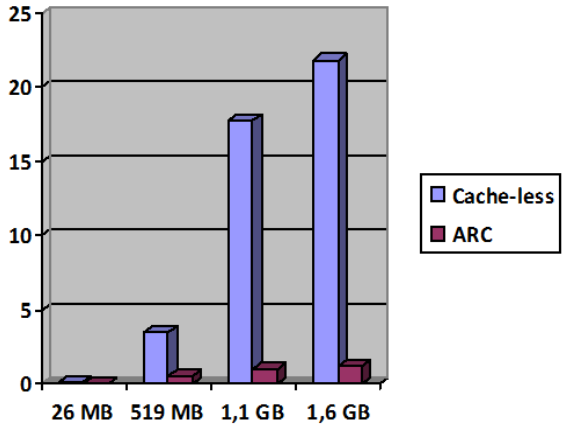

Fig. 4. Latency graphical comparison for encoded data (in seconds)

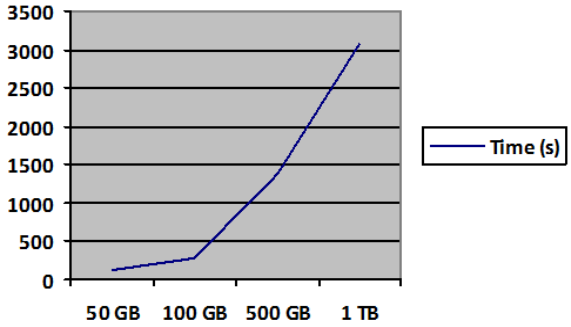

Fig. 5. Encoding time for different datasets (in seconds)

On a similar fashion, the third experiment intends to show that each node has enough processing power to contribute for erasure decoding operations. In this experiment a varying number *n* of binary data objects, each with 10 MB of size, composed each dataset. Datasets size varied from 50 GB up to 1 TB of data and operations ran on a single node. Results follow on Table 4 and Fig. 6.

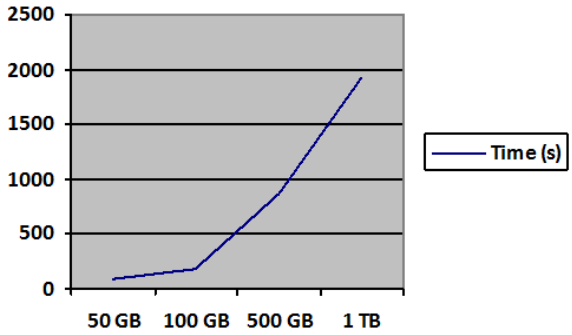

Fig. 6. Decoding time for different datasets (in seconds)

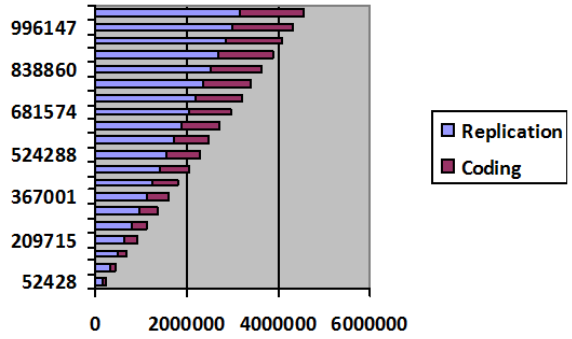

Fig. 7. Overhead comparison for both redundancy techniques

Table 1. Average latency for different objects, triple-replication

Size	Cache-less	ARC
14 MB	0.075427s	0.027292s
277 MB	1.069707s	0.196161s
553 MB	2.176905s	0.385806s
1,1 GB	15.241922s	0.808186s

Table 2. Average latency for different objects, erasure coded

Size	Cache-less	ARC
26 MB	0.203074s	0.058756s
519 MB	3.565754s	0.551705s
1,1 GB	17.730196s	1.054749s
1,6 GB	21.805975s	1.269461s

Table 3. Encoding time for different volumes of data

n	Total size	Time
5000	50 GB	131.390s
10000	100 GB	269.676s
50000	500 GB	1389.382s
100000	1 TB	3063.399s

Table 4. Decoding time for different volumes of data

n	Total size	Time
5000	50 GB	89.237s
10000	100 GB	183.615s
50000	500 GB	874.102s
100000	1 TB	1914.271s

Table 5. Overhead for binary object storage (in bytes)

Object size	Replication 3x	Erasure Coding
52428 B	157284 B	70651 B
104857 B	314571 B	140382 B
157286 B	471858 B	210112 B
209715 B	629145 B	279842 B
262144 B	786432 B	349573 B
314572 B	943716 B	420220 B
367001 B	1101003 B	489951 B
419430 B	1258290 B	559681 B
471859 B	1415577 B	629411 B
524288 B	1572864 B	699142 B
576716 B	1730148 B	769790 B
629145 B	1887435 B	839520 B
681574 B	2044722 B	909250 B
734003 B	2202009 B	978980 B
786432 B	2359296 B	1048711 B
838860 B	2516580 B	1119359 B
891289 B	2673867 B	1189089 B
943718 B	2831154 B	1258819 B
996147 B	2988441 B	1328549 B
1048576 B	3145728 B	1398280 B

On a ring P2P network, each node would be able to receive and process requests like the ones tested above simultaneously. Therefore, by testing the performance of a single node processing, it is possible to estimate that a larger number of nodes processing requests in parallel would decrease the overall encoding and decoding times. P2P is then the only model that allows for maximum node contribution in such situations.

The fourth experiment intends to show that although coding requires additional processing, it has much less overhead when compared to pure replication. Experiments measured varying object sizes and values represent total number of bytes, for better precision. Results follow on Table 5 and Fig. 7.

Discussion

Regarding the caching mechanism used, given that ARC is notably an improvement of LRU in terms of hit ratio, the first question at hand was whether a per-node cache system on a P2P networked storage system would provide any benefits for data access. As it was shown in the first experiment, our cache system dramatically reduces access times on each node, providing over 80% improvement on access latency for both replicated and encoded data when compared to a cache-less alternative. Since each node may actively process requests, on a real environment several users would benefit from faster responses. Since ARC uses implementation in RAM, a lower latency than the hard disk was expected. Thus, the usage of ARC instead of LRU is an improvement from previous works, specifically over the CAROM architecture, but also improving all other works who do not consider caching mechanisms to improve access

Second and third experiments have shown that the current algorithms for erasure coding have feasible execution time on each node. Therefore, it is natural to want to maximize the number of processing nodes and in that sense the P2P model provides an optimal solution, since all nodes actively contributes to the whole system. Assuming a network of *n* nodes in parallel, such as a cluster in a Cloud environment, previous works used only *n-k* of those nodes for coding, but with our approach the actual performance gains in terms of processing time could easily be reach up to 100% in comparison, given that our work uses all *n* nodes. Separating processing nodes and storage nodes would only add additional network transfer time and for that reason, Griddler uses nodes for both functions. This is also an improvement from previous works, such as MICS and HRSPC. Both these works have some sort of master-slave relation in distributed storage, therefore having possible problems with single points of failure.

The fourth experiment reinforced the importance of erasure codes in terms of storage overheads, thus surpassing previous works limited to replication. When compared to three-way replication, the same data, when coded, induces around 55% less overhead, in average. When it comes to Big Data, this characteristic would allow for larger volumes of data stored using the same hardware. When processing time is more of an issue than storage overhead, the proposed system is capable of storing data in replicated fashion as well, while some previous works were limited to erasure coding, such as Robot and HDFS-Xorbas.

Conclusion and Future Works

Efficient and secure storage of data is paramount in Big Data scenarios that rely on continuous access to information. At the same time as there is a considerable increase in data volume, which requires innovative technologies to increase storage capacity, it is even more important to use fault tolerance techniques for availability assurance such as replication and erasure codes. This scenario served as motivation for the work, which sought to develop a P2P data storage architecture with mixed fault tolerance, combining the two mentioned techniques, in an automated and configurable way. A fully decentralized topology lacks in similar works previously found in the literature and therefore this is an improvement. As an additional contribution, the proposed system implements a caching scheme, which led to an improvement in the speed of data access. For Griddler validation, our results were put in context with previous works shows another improvement, given that most works do not consider the usage o cache structures and the work that proposes the CAROM architecture uses an inferior cache algorithm than the one we use in our work. Access time of the data in the distributed system was measured, as well as other factors such as encoding and decoding time, with expressive results of over 80% response time gain for access and about 55% less general data overhead when encoding. It is evident that this work finds several applications in real situations and authors expect, when consolidating its development, to make it available in real situations that depend on large-scale distributed data storage. There is room for improvements, though, given that our work does not consider some relevant aspects of data storage, such as data security and privacy. We have yet to verify our architecture in larger computational environments for longer periods of time, with real user´s requests. There is also an interest in testing our caching mechanism with Solid-State Drives (SSD) instead of RAM in order to have larger cache areas available.

Acknowledgement

This work was financed by the Coordination for the Improvement of Higher Education Personnel (CAPES) and the National Counsel of Technological and Scientific Development (CNPq), both from Brazil.

Author's Contributions

Carlos Roberto Valêncio, Angelo Cesar Colombini, Mário Luiz Tronco and Márcio Zamboti Fortes: Participated in all the project decisions that defined the scientific contributions of this work, definition and analysis of the data, writing and reviewing of the article.

André Francisco Morielo Caetano: Participated in all the project decisions that defined the scientific contributions of this work, definition and analysis of the data, writing and reviewing of the article was responsible for most of the programming, testing and creation of the sub-products of this work.

References

Al-Awami, L. and H.S. Hossanein, 2016. Distributed data storage systems for data survivability in wireless sensor networks using decentralized erasure codes. Comput. Netw., 97: 113-127. DOI: 10.1016/j.comnet.2016.01.008

Alnafoosi, A.B. and T. Steinbach, 2013. An integrated framework for evaluating big-data storage solutions-IDA case study. Proceedings of the Science and Information Conference, Oct. 7-9, IEEE Xplore Press, London, UK, pp: 947-956.

Borthakur, D., 2008. HDFS architecture guide-Apache™ Hadoop.

Bradner, S. and J. McQuaid, 1999. Benchmarking methodology for network interconnect devices. RFC 2544.

Gonizzi, P., G. Ferrari, V. Gay and J. Leguay, 2015. Data dissemination scheme for distributed storage for IoT observation systems at large scale. Inform. Fus., 22: 16-25. DOI: 10.1016/j.inffus.2013.04.003

Gribaudo, M., M. Iacono and D. Manini, 2016. Improving reliability and performances in large scale distributed applications with erasure codes and replication. Future Generat. Comput. Syst., 56: 773-782. DOI: 10.1016/j.future.2015.07.006

Hashem, I.A.T., I. Yaqoob, N.B. Anuar, S. Mokhtar and A. Gani *et al*., 2015. The rise of "big data" on cloud computing: Review and open research issues. Inform. Syst., 47: 98-115. DOI: 10.1016/j.is.2014.07.006

Hyun, S., K. Sun and P. Ning, 2017. FEC-Seluge: Efficient, reliable and secure large data dissemination using erasure codes. Comput. Commun., 104: 191-203. DOI: 10.1016/j.comcom.2017.01.005

Kambatla, K., G. Kollias, V. Kumar and A. Grama, 2014. Trends in big data analytics. J. Parallel Distrib. Comput., 74: 2561-2573. DOI: 10.1016/j.jpdc.2014.01.003

Khan, O., R. Burns, J. Plank, W. Pierce and C. Huang, 2012. Rethinking erasure codes for cloud file systems: Minimizing I/O for recovery and degraded reads. Proceedings of 10th USENIX Conference on File and Storage Technologies, Feb. 14-17, San Jose, USA., pp: 1-14.

Li, J. and B. Li, 2013. Erasure coding for cloud storage systems: A survey. Tsinghua Sci. Technol., 18: 259-272. DOI: 10.1109/TST.2013.6522585

Li, J., J. Wu, G. Dan, A. Arvidsson and M. Kihl, 2014. Performance analysis of local caching replacement policies for internet video streaming services. Proceedings of IEEE 22nd International Conference on of Software, Telecommunications and Computer Networks, Sept. 17-19, IEEE Xplore Press, pp: 341-348. DOI: 10.1109/SOFTCOM.2014.7039112

Li, S., Q. Cao, S. Wan, L. Qian and C. Xie, 2016. HRSPC: A hybrid redundancy scheme via exploring computational locality to support fast recovery and high reliability in distributed storage systems. J. Netw. Comput. Applic., 66: 52-63. DOI: 10.1016/j.jnca.2015.12.012

Li, Z.J. and M.H. Liao, 2005. Modeling load balancing in heterogeneous unstructured P2P systems. J. Comput. Sci., 1: 323-331. DOI: 10.3844/jcssp.2005.323.331

Ma, Y., T. Nandagopal, K.P.N. Puttaswamy and S. Banerjee, 2013. An ensemble of replication and erasure codes for cloud file systems. Proceedings of IEEE INFOCOM, Apr. 14-19, IEEE Xplore Press, Turin, Italy, pp: 1276-1284. DOI: 10.1109/INFCOM.2013.6566920

Megiddo, N. and D.S. Modha, 2004. Outperforming LRU with an adaptive replacement cache algorithm. Computer, 37: 58-65. DOI: 10.1109/MC.2004.1297303

Plank, J.S., 2008. The RAID-6 liberation codes. Proceedings of 6th USENIX Conference on File and Storage Technologies, Feb. 26-29, San Jose, USA., pp: 1-14.

Sathiamoorthy, M., M. Asteris, D. Papailiopoulos, A.G. Dimakis and R. Vadali *et al*., 2013. XORing elephants: Novel erasure codes for big data. Proceedings of VLDB Endowment, Aug. 26-30, Trento, Italy, pp: 325-336. DOI: 10.14778/2535573.2488339

Stoica, I., R. Morris, D. Karger, M.F. Kaashoek and H. Balakrishnan, 2001. Chord: A scalable peer-to-peer lookup protocol for internet applications. IEEE/ACM Trans. Netw., 11: 17-32. DOI: 10.1109/TNET.2002.808407

Tang, Y., J. Yin, W. Lo, Y. Li and S. Deng *et al*., 2015. MICS: Mingling chained storage combining replication and erasure coding. Proceedings of IEEE 34th Symposium on Reliable Distributed Systems, Sep. 28-Oct. 1, Montreal, Canada, pp: 192-201. DOI: 10.1109/SRDS.2015.25

Weatherspoon, H. and J.D. Kubiatowicz, 2002. Erasure coding vs. replication: A quantitative comparison. Proceedings of the 1st International Workshop on Peer-to-Peer Systems, Mar. 07-08, Springer-Verlag, pp: 328-337. DOI: 10.1007/3-540-45748-8_31

Yin, C., J. Wang, C. Xie, J. Wan and C. Long *et al*., 2013. Robot: An efficient model for big data storage systems based on erasure coding. Proceedings of IEEE International Conference on Big Data, Dec. 6-9, IEEE Xplore Press, Silicon Valley, USA., pp: 163-168. DOI: 10.1109/BigData.2013.6691569