Understanding the Performance of Erasure Codes in Hadoop Distributed File System

Jad Darrous, Shadi Ibrahim

Understanding the Performance of Erasure Codes in Hadoop Distributed File System

Jad Darrous*
Inria, IMT Atlantique, LS2N
Nantes, France

Shadi Ibrahim
Inria, Univ. Rennes, CNRS, IRISA
Rennes, France

Abstract

Replication has been successfully employed and practiced to ensure high data availability in large-scale distributed storage systems. However, with the relentless growth of generated and collected data, replication has become expensive not only in terms of storage cost but also in terms of network cost and hardware cost. Traditionally, erasure coding (EC) is employed as a cost-efficient alternative to replication when high access latency to the data can be tolerated. However, with the continuous reduction in its CPU overhead, EC is performed on the critical path of data access. For instance, EC has been integrated into the last major release of Hadoop Distributed File System (HDFS) which is the primary storage backend for data analytic frameworks (e.g., Hadoop, Spark, etc.). In this work, we measure and compare the performance of data accesses in HDFS under both replication and EC. Our analysis indicates that EC is a feasible solution for data-intensive applications and it can outperform replication in many scenarios. Furthermore, we demonstrate that it is the block placement algorithm in HDFS that mostly impacts the performance under EC.

Keywords: Erasure coding, HDFS, Performance evaluation

1 Introduction

Replication has been successfully employed and practiced to ensure high data availability in large-scale distributed storage systems [17, 27, 35]. Moreover, replication can be leveraged to improve data access performance under high load. While important, replication becomes very expensive [18, 30, 34, 41], not only in terms of storage cost (given the rapid growth in data size – IDS estimates data to grow from 64.2 ZettaByte (ZB) in 2020 to 181 ZB by 2025 [32]) but also in terms of network cost (half of all cross-rack traffic is attributed to replication traffic in production clusters at Facebook and Microsoft [11]) and hardware cost (expensive high-speed storage devices, i.e., SSDs and DRAMs, are increasingly deployed in storage systems to meet low latency data-intensive applications [22]).

While already popular for archived data in peer-to-peer systems [23, 33] and cold data [19, 28], in recent years, erasure coding (EC) has been progressively performed on the critical path of data access [31, 39]. For example, EC has been employed in in-memory storage systems on cached (hot) data [31]. The main reasons of such an adoption are that (1) EC can provide the same fault tolerance guarantee as replication but with much lower storage cost [33], and (2) the cost of data encoding and decoding is progressively reduced, thanks to the release of storage acceleration libraries, e.g., ISA-L [5], that allow EC operations to run at CPU speed (e.g., encoding throughput using only a single core is 5.3 GB/s for Intel Xeon Processor E5-2650 v4 [21]).

As a natural result of such an adoption, EC has been integrated in Hadoop Distributed File System since its last major release [18] (i.e., HDFS 3.0.0). HDFS is the primary storage back-end for data analytic frameworks (e.g., Hadoop [2], Spark [3], Flink [1], etc.). Several studies have been conducted to understand the performance of MapReduce applications under EC (with contiguous and striped data layouts [12, 13, 36, 40]), reduce the data reconstruction times under failures [26], and improve data locality [24, 25]. For example, Darrous et al. [13] provide an in-depth study on the impact of EC with striped data layout on the task runtimes and MapReduce application performances under different system configurations (e.g., HDD, DRAM, 1 Gbps and 10 Gbps network, etc.). Although important, the aforementioned studies target MapReduce applications – where the clients (i.e., the tasks) reside inside Hadoop cluster and are processing the data – and do not consider multi-tenant storage systems.

In an effort to complement existing efforts on understanding the performance of data-intensive applications under EC, in this paper, we study the performance of (concurrent) data accesses (write and read applications) under erasure coding and replication in a complementary and contrast approach. We experimentally study their behaviors through extensive micro-benchmarks on the Grid'5000 [9] testbed. We report our findings pertaining to the performance of write and read applications when varying the size of data and the number of concurrent accesses. Our results show that the write throughput under EC approaches that under replication and even outperforms it under high concurrency. Moreover, EC – by leveraging parallel reads from multiple disks – can deliver a 4.95x higher throughput than replication for a single read. However, under high concurrent reads, the performance of EC is impacted by the imbalanced load across nodes. We hope that the insights drawn from this paper will enable a better understanding of the performance

*Work done while at Inria, IMT Atlantique, LS2N of data-intensive applications under EC and motivate further research in adopting and optimizing EC in distributed storage systems.

The remainder of this paper is organized as follows. In Section 2, we present how EC is implemented in HDFS. The experimental methodology is explained in Section 3. Section 4 presents the experiments under write operation while Section 5 presents the different sets of experiments under read operation. Section 6 discusses the related work. Finally, Section 7 concludes this paper.

2 Erasure codes in HDFS

EC was officially integrated into HDFS in 2018, in the third major release of Hadoop ecosystem [18]. Hadoop community has adopted the XOR and Reed-Solomon erasure codes (RS) with *striped* layout (i.e., one logical block is represented physically by (n) chunks – usually distributed on multiple disks – and then used to compute (k) parity chunks. Accordingly, any (n) out of ($n+k$) chunks is sufficient to rebuild the original data block). EC with striped layout is efficient for small files [18] which are prominent in data-intensive clusters [16, 29], and requires less memory overhead when encoding and decoding data. To reduce the metadata overhead, every n blocks – belonging to the same file – are grouped into an *EC group*. Blocks of the same EC group are usually placed on the same set of nodes. EC policy is defined by a scheme and the size of striping cell. The scheme consists of the number of data and parity chunks alongside the code algorithms (XOR and Reed-Solomon are supported). The striping cell size determines the granularity of data access and buffer size (1 MB by default). The default policy under EC is "RS-6-3-1024k", which means that Reed-Solomon (RS) codes are used with 6 data chunks and 3 parity chunks and the encoding/decoding operations are performed with a striping cell of 1 MB.

3 Methodology Overview

We conducted a set of experiments to assess the impact of data access pattern and concurrent data access when HDFS operates under replication (REP) and erasure coding (EC).

3.1 Testbed

Our experiments were conducted on the French scientific testbed Grid'5000 [9] at the site of Nantes. Two clusters – Econome and Ecotype – are used for the experiments with 21 and 40 machines, respectively. Each machine is equipped with two Intel Xeon E5-2660 8-core processors, 64 GB of main memory, and one disk drive (HDD) at 7.2k RPM with 1 TB. The machines of each cluster are connected by 10 Gbps Ethernet network. The two TORs switches of both clusters are connected with four 40 Gbps links. The machines run 64-bit Debian stretch Linux with Java 8 and Hadoop 3.0.0 installed. All the experiments have been done in isolation on the testbed, with no interference originated from other users. Econome runs HDFS (one node hosts the NameNode (NN) and the remaining 20 nodes act as DataNodes (DNs)) while Ecotype hosts the clients. To exclude the impact of the local disks at the client side, we store the dataset in the main memory.

3.2 HDFS configuration

HDFS block size is set to 256 MB (similar to [15, 37]) and the replication factor is set to 3 (default value). For EC, we use the default EC policy in HDFS, i.e., $RS(6,3)$ scheme with a cell size of 1 MB.

3.3 Benchmarks

We used the read and write operations to measure the performance of HDFS. These operations are issued from the clients using the HDFS commands `hadoop fs -get` and `hadoop fs -put`, respectively. The test files are generated by the `dd` command from `/dev/urandom` as an input source.

3.4 Metrics

The throughput at the client side is the main metric used to measure the performance of read/write operations from/to HDFS. For one client, it is the amount of data read/written per second. In the case of concurrent reads and writes (by multiple clients), the average throughput per client is shown alongside the standard deviation of the clients' throughput.

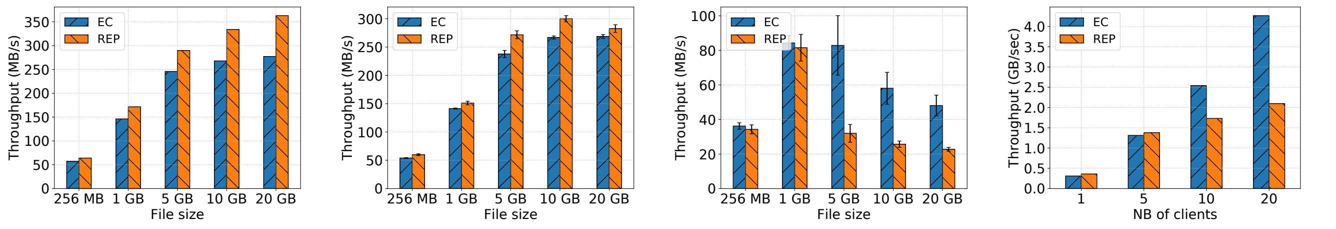
We also profile disk and network I/O of the nodes using the python library `psutil` [6] version 5.4.8.

4 Cost of Adding New Data

4.1 Write dataflow in HDFS

Why studying the write performance under EC? Data is generated at an extreme rate. To benefit from this data, it is usually stored - and then analyzed - in data-intensive clusters. Recent studies revealed that hundreds of terabytes of data are ingested every day in data-intensive clusters [11]. Those data, known as data inputs, vary significantly in their sizes. For example, traces from production Hadoop cluster [4] reveal that the size of input data varies from 3 GB to 13 TB. Furthermore, these input data are populated frequently by multiple concurrent users and applications. For example, one computation requires 150 pipelined jobs to complete [15]; hence, the output of a job is used as an input for the later one. This motivates us to study the cost of adding data under erasure coding compared to replication. We further study the impact of concurrency on the performances of both erasure coding and replication.

What is the impact of EC on write performance? Writes are pipelined under replication (i.e., the client sends the data block to the first node which, in turn, pipelines it to the second and then the third). However, under EC, the client encodes the data and then writes the data and parity chunks



(a) Write one file per client **(b)** Write 5 files by 5 clients **(c)** Write 40 files by 40 clients **(d)** Write 100 GB dataset (aggregated throughput)

Figure 1. Write under REP and EC.

directly to all the nodes in parallel. This parallelism can provide higher throughput but results in more data sent by the client stressing the link between the clients and the HDFS cluster. Regardless of the HDFS storage policy, the client splits the file into blocks equal to the HDFS configured block size. Under replication, and for each block, the client contacts the NN to get a list of DNs equal to the replication factor (e.g., 3 nodes for 3-way replication) to write the block to. The client streams each block to the first DN in the corresponding list which in turn pipelines the data to the second one and so on.

On the other hand, the write dataflow under EC is a bit different; for $RS(6, 3)$, after splitting the file into blocks, every 6 blocks are grouped into an EC group. For each EC group, the client contacts the NN to get a list of 9 DNs. To encode this group of blocks, the blocks are encoded sequentially; each block is split into cells (e.g., 1 MB), then the client encodes every 6 cells to generate 3 parity cells. The client sends these 9 cells to the 9 DNs, and then continues to do the same with the remaining cells, and repeats the same process for the remaining blocks in the group. Then, do the same for the remaining EC groups. In conclusion, three (number of replicas) DNs will be contributing to a write operation at a time under REP, while 9 DNs ($n + k$) will be simultaneously writing data under EC. It is important to mention that blocks are written sequentially to HDFS (the second block is sent once the first block is completely stored in a DN, persisted in disk or buffered in the memory). Moreover, when the data is stored completely in all DNs (at least buffered in the memory), the write operation is considered successful.

4.2 Results of single write

Figure 1a shows the write throughput of one client with different file sizes. The write throughput is 1.11x to 1.3x higher under REP compared to EC when increasing the file sizes from 256 MB to 20 GB, respectively.

When writing 20 GB file, REP achieves 363 MB/s write throughput, while EC obtains a throughput of 277 MB/s which is 76% of the throughput under REP. This can be explained due to the larger amount of data which is sent from the client to HDFS in EC (i.e., 30 GB accounts for the original and parity data) compared to REP (i.e., 20 GB) and the "low cost" data pipelining under REP. First, data stays in the buffers of the DNs when received from the client or pipelined from other DNs, hence, it is not synchronized to disks directly. As a result, data transfer time – to HDFS – strongly depends on the amount of data sent to the cluster. Given that the throughputs of inter-cluster traffic (network traffic between the clients and the HDFS cluster) are 444 MB/s and 399 MB/s under EC and REP respectively, the file under EC takes 35% more time to be transferred to HDFS compared to REP (the difference in transfer times between EC and REP is clearer when increasing the file size). Second, (buffered) data can be pipelined as soon as they reach the first DN because the intra-cluster (data transferred inside the HDFS cluster) throughput is almost two times the inter-cluster throughput (i.e., intra-cluster throughput is 801 MB/s), thus replicating (transferring) data inside the cluster does not introduce an extra cost. Finally, more data is persisted to disk under REP compared to EC (i.e., 21 GB under EC while it is 29 GB under REP); which in turn slightly slows down the write operation under REP.

Note that, by default, Hadoop client utilizes only one thread to send the data and therefore the inter-cluster throughput may not be fully utilized. In case of REP, data is communicated between two nodes only and thus the inter-cluster throughput is limited to 399 MB/s. In case of EC, despite that the client is sending data to 9 DNs, inter-cluster throughput is limited to 444 MB/s because only one core is responsible for encoding and sending data (in case of 20 GB, we observe that one core is always at 100% utilization).

`Observation 1.` Unlike under replication, more data (i.e., parity data) goes from the client to HDFS cluster under EC. This extra parity data (0.5x of the original data in our configuration) results in a loss of throughput compared to replication as data are pipelined (replicated) inside the cluster with minimal cost.

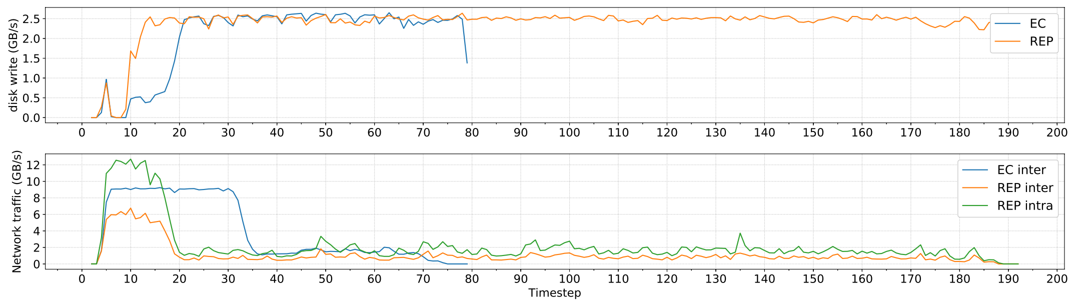


Figure 2. Disk, inter-network and intra-network throughput for concurrent writing of 5 GB files by 40 clients.

4.3 Results of concurrent writes

When increasing the number of concurrent clients to 5, EC starts to approach the throughput of REP (Figure 1b). Interestingly, with 40 clients, EC has a throughput 2 times that of REP for files equal and larger than 5 GB as depicted in Figure 1c.

In case of writing 5 GB files by 40 concurrent clients, REP achieves 32 MB/s write throughput, while EC obtains a throughput of 83 MB/s which is 2.59x higher than the throughput under REP. Disks are saturated under both EC and REP (the average disk throughput is 110 MB/s and 124 MB/s, respectively). As more data is persisted to disks under REP compared to EC (i.e., 464.4 GB under REP while it is 160.5 GB under EC), writing data to disks under EC is almost 2.5 times faster. On the other hand, the throughput of the intra-cluster network is low (i.e., 183 MB/s to 233 MB/s when the files are larger or equal to 5 GB), thus write performance will further degrade under REP. The low disk and intra-cluster performances will limit the arrival rate of data to HDFS cluster under REP (the arrival rate is 1162 MB/s under REP compared to 4535 MB/s under EC). As shown in Figure 2, the network performance (inter and intra) drops once the buffers are filled and data starts to be synchronized to disks while still not completely replicated (transferred) to other DNs.

To further explain the impact of concurrency on the performance of write under both EC and REP, Figure 1d shows the aggregated write throughput when writing a data set of 100 GB while varying the number of clients and therefore the files sizes. When the number of concurrent clients is 5, REP achieves a throughput of 1410 MB/s while it is 1340 MB/s under EC. The aggregated throughputs increase by 1.94x and 3.25x when increasing the number of clients to 10 and 20 clients under EC, respectively. This increase is mainly due to the increase in the inter-cluster throughputs; On the other hand, the aggregated throughputs increase 1.2x and 1.5x under replication when increasing the number of clients to 10 and 20 clients, respectively. Here, the high cost of data pipelining limits the scalability of HDFS under REP when increasing the number of clients. Specifically, buffers under replication are filled faster (original data and replicated data) compared to EC. Thus, the arrival rate of data to a DN is limited by the disk throughput (125 MB/s under EC and 128 MB/s under REP). However, as a DN under replication is receiving data from the clients and from other DNs, this will reduce the inter-cluster throughput and prolong the writing time under REP compared to EC.

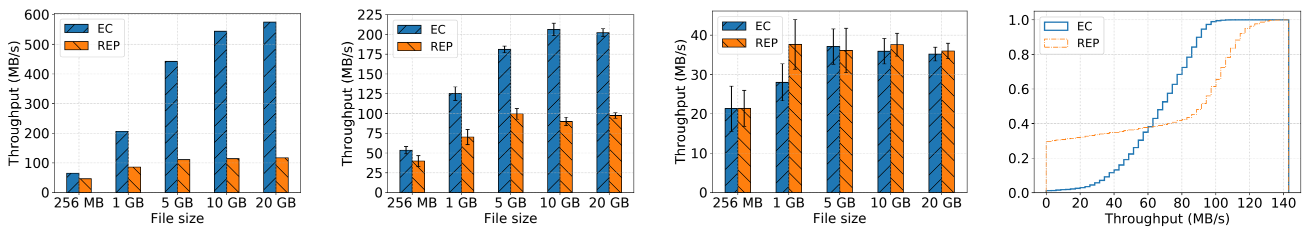
`Observation 2.` Under concurrent writes, in contrast to under EC, the performance of write workloads under replication is limited by the intra-cluster transfer and disk contention (as in total, the amount of written data is larger under REP compared to EC). Hence, due to the high network and memory cost of data pipelining, EC can even outperform replication – under heavy concurrent writes. This gap is more obvious for large files (e.g., in our experiments, the throughput under EC is at least 2x the throughput achieved under replication when 20 clients are writing 10 GB file each (Figure 1c).

4.4 Implications

Replication is not only costly in terms of storage requirements but also contributes to the problems of oversubscribed networks [15] and poor disk performance [15] in production clusters (e.g., traces from production clusters at Facebook and Microsoft pointed out that replication results in almost half of all cross-rack traffic [11]). Applying EC as an alternative to replication does not only reduce storage cost and disk overhead but also results in lower network traffic (up to 50%). In conclusion, given its performance, EC is feasible for write applications, more importantly, EC sustains high throughput for write operations under high concurrency.

5 Reading Data under EC

Why studying the read performance under EC? Reading input data is the first step for any data-intensive application. Many studies have discussed the importance of optimizing the read step (i.e., reading input data) to improve the job execution time - mainly through targeting high data locality [10, 14, 20, 38]. Therefore, it is important to understand read performance under EC. Typically, multiple jobs are



(a) Read one file per client **(b)** Read 5 distinct files by 5 clients **(c)** Read 40 distinct files by 40 clients **(d)** CDF of disk throughput per DN during the concurrent read of 40 files each of 20 GB

Figure 3. Read distinct files under REP and EC.

running concurrently (to achieve higher cluster utilization) therefore we need to evaluate concurrent reads. Also, it is possible that multiple jobs (could belong to different users) use the same files as input, thus we study concurrent reads of the same file under both storage policies.

What is the impact of EC on read performance? Clients reading data under replication can contact any replica and retrieve all the data from only one replica, this can provide load balance between the nodes hosting the replicas under concurrent reads. On the other hand, to read a block under EC, the client needs to contact the data nodes and read the data block in parallel. This might result in higher read throughput if the network bandwidth is higher than disk read bandwidth, but may cause an imbalance in the load as nodes hosting parity chunks may serve fewer requests.

5.1 Read dataflow in HDFS

To read a file in HDFS, the client contacts the NN to get the addresses of the DNs hosting each block of the file. For each block, the DNs are ordered by the network distance to the client and are randomized for the nodes that have the same distance. The client then reads the blocks in order i.e., sequentially, one by one. Reading one block of data (without any failure) has the same cost under both EC and REP, as the same amount of data (equal to the block size) will be read from disk and transferred over the network. However, under EC the data block can be read in parallel from multiple DNs (6 DNs in our configuration). This parallelization results in better performance especially when the network bandwidth is higher than the disk bandwidth [8].

5.2 Results of single read

The read throughput for different file sizes by one client is depicted in Figure 3a. As we expect, EC performs better than REP. Specifically, the read throughput is 1.4x to 4.95x higher under EC compared to REP when increasing the file size from 256 MB to 20 GB, respectively.

Table 1. Disk read throughput (MB/s) of active DNs in case of single client read.

	File size	mean	25%	50%	75%	max
EC	**256 MB**	35.42	26.62	42.00	42.75	43.00
	1 GB	71.28	57.00	71.25	85.50	86.00
	5 GB	74.59	65.51	85.33	85.51	85.67
	10 GB	83.76	85.35	85.45	85.61	102.80
	20 GB	83.39	77.44	85.39	86.85	102.80
REP	**256 MB**	85.33	85.33	85.33	85.33	85.33
	1 GB	85.42	85.33	85.33	85.42	85.67
	5 GB	85.19	85.46	85.67	85.67	102.40
	10 GB	88.44	85.33	85.55	89.40	102.80
	20 GB	86.63	83.19	85.50	92.86	102.60

When reading 20 GB file, REP achieves 116 MB/s read throughput, while EC obtains a throughput of 575 MB/s which is 4.95x higher than the throughput under REP. The reason behind this is that an EC block is read from 6 nodes, while under REP, a block is read from a single node. Therefore – as the performance of the read operation is limited by the disk – 6 disks are leveraged in parallel under EC, while a single disk is used under REP. Hence, the throughput of the read workload is strongly co-related with the performance of the disks. Therefore, the performance gap between EC and REP is clearer when increasing the file sizes, especially when the disk throughput under EC increases.

Under replication, a block will be read sequentially from the disk, hence, the disk will be exploited efficiently (i.e., the throughput of the disk is between 85 MB/s and 100 MB/s most of the time). On the other hand, under EC, a block will be read in parallel from multiple nodes (6 nodes in our configuration), therefore, each node reads a fraction of the block (one data chunk of 43 MB) and thus cannot leverage the full throughput of the disk. As we can see in Table 1, when the size of the file is 256 MB, the disk throughput is limited by the physical block size (43 MB/s) under EC. However, when the client reads all the blocks of an EC group (6 blocks in our configuration), the disk throughput of an individual node is

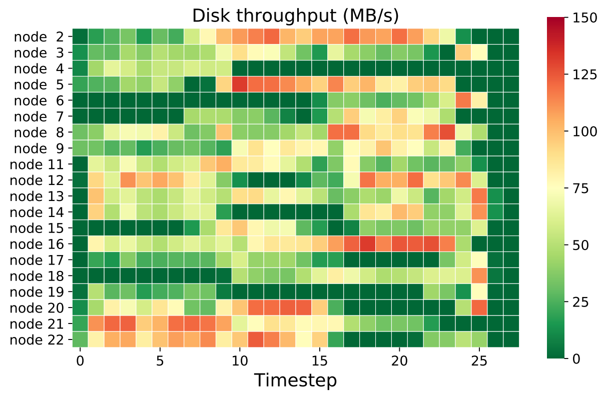


Figure 4. Disk throughput of the DNs during the concurrent read of 5 files by 5 clients each of 5 GB under EC.

fully utilized as 6 chunks belong to different blocks will be read sequentially from the disk. For instance, when the file size is 20 GB, the average disk throughputs (of active disks) for both EC and replication are 83.4 MB/s and 86.6 MB/s (the disk throughput is higher than 80 MB/s most of the time), respectively. Hence, this explains the 4.95x performance difference.

`Observation 3.` Compared to 3-way replication, a block (256 MB) is scattered on more DNs under EC. Therefore, reading a file under EC has a clear advantage compared to under REP, as multiple disks are leveraged in parallel.

5.3 Results of concurrent reads

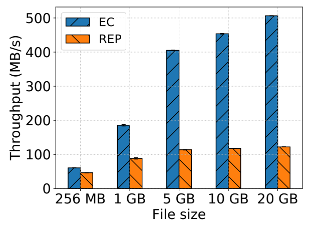
Reading distinct files. When increasing the number of concurrent clients to 5 – each reading a distinct file, the gap between EC and REP starts to decline as shown in Figure 3b. Interestingly, with 40 concurrent clients, REP achieves slightly higher throughput (clearer when the file size is 1 GB) as shown in Figure 3c. Moreover, we notice that the performance of read operations exhibits noticeable variation in-between different clients under both REP and EC.

When 5 clients are reading 5 files, each has a size of 20 GB, REP achieves 97 MB/s read throughput, while EC obtains a throughput of 202 MB/s which is 2.08x higher than that under REP. The average disk throughput of active DNs is 60 MB/s and 86 MB/s under EC and REP, respectively. The average disk throughput under EC decreases – compared to the one in case of 1 client (i.e., 83.4 MB/s) – which is due to data skew. As shown in Figure 4, although most of the DNs are serving data, some of them (around 6 DNs) serve more clients a time which in turn results in exploiting the full capacity of the disk (i.e., 125 MB/s) on those nodes. On the other hand, the majority of nodes were serving 1 client most of the time, thus low utilization of the disk. We observe that disk throughputs are above 100 MB/s during 50% and 18% of the read workload time under REP and EC, respectively. Hence, this, in addition to the increase in the number of contributing nodes (active disks), explains the decrease in the performance difference from 4.95x to 2.08x between EC and REP.

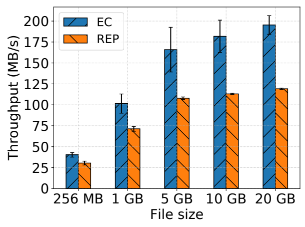
When 40 clients are reading 40 files, each has a size of 20 GB, we observe that HDFS suffers from obvious load imbalance under both REP and EC. Under REP, at least one disk was not active during 25% of the read workload (see Figure 3d). Under EC, some DNs still serve more clients compared to other DNs which in turn results in high contention on the disks and thus low disk throughputs; disk throughputs are above 100 MB/s during 40% and 2% under REP and EC, respectively. As a result, the aggregated disk throughput under EC is 1352 MB/s and is 1350 MB/s under REP, which explains the 2% difference. In addition, this load imbalance results in high variation in the latency of the read workload in-between clients, therefore, clients which are served by nodes under heavy load need more time to read their files.

Reading the same file. Figure 5a shows the average read throughput per client when 5 clients are reading the same file concurrently. With 20 GB file, the average read throughput is 506 MB/s under EC while it is 122 MB/s under REP, thus 4.14x faster under EC. To explain the difference, we inspect the amount of read data and sent data at the cluster level. Under EC, the cluster has an aggregated disk throughput of 537.6 MB/s and network throughput of 2840 MB/s: each block under EC is read once from the same nodes, cached and then sent to the 5 clients; hence, network throughput is 5.2x higher than the disk throughput. However, under REP, the cluster disk throughput is 322.5 MB/s and the network has a throughput of 652.8 MB/s, 1.7x higher than the disk. Here, a block is served by the 3 DNs at the same time: as a block is available on 3 DNs, the DNs receive requests – from different clients – to read the block, accordingly, the 3 DNs will read the data from the disk, cache it and then sent it to 5 clients (one DN serves 1 or 2 clients). This explains why the total amount of read bytes is 52.1 GB (2.6x the size of the file) under REP and 20 GB under EC.

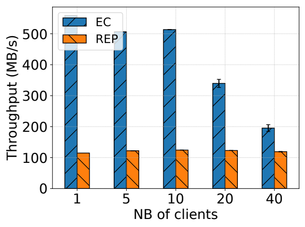
When increasing the number of concurrent clients to 40, we notice that clients maintain their average throughput under REP (119.2 MB/s) while it drops to 195.3 MB/s under EC, as shown in Figure 5b. Under REP, each block is served by 3 different DNs, so a block will be served by a new set of DNs (3 DNs) while the previous block is still being served to clients (new block is requested once a client out of the 40 clients finishes fetching the current block). Consequently, disk throughput increases under REP from 322 MB/s with 5 clients to 367.6 MB/s and thus the network throughput increases to 5079 MB/s. On the other hand, 6 DNs will be continuously serving the 40 clients: each DN is responsible for one original chunk. This will saturate the network bandwidth of the DNs and cause variations in-between clients.



(a) Read by 5 clients



(b) Read by 40 clients



(c) Read 20 GB file

Figure 5. Read the same file under REP and EC.

Table 2. Average aggregated cluster network and disk throughput (MB/s) in case of concurrent read of the same 20 GB file by the number of clients.

Number of clients	**Network throughput (MB/s)**		**Disk throughput (MB/s)**	
	EC	**REP**	**EC**	**REP**
5	2840.3	652.8	537.6	322.5
10	5836.8	1331.5	552.8	374.2
20	7009.7	2648.2	384.1	376.1
40	8048.7	5079.1	229.2	367.6

Moreover, as the reading of a new block can start when all the chunks are received by at least one client, the same 6 DNs will be serving two blocks (belong to the same EC group) simultaneously and this will reduce the number of disk read requests for the new block and at the same time reduce the amount of data sent to clients through network and further increase the variation. Even worse, the performance degradation at the disk-level and the network-level (and the resulted variation in the throughput in-between clients) will be amplified when DNs are serving two successive EC groups. We observe that the number of disk requests drops by almost 50% (from 50 to 24) when increasing the number of clients from 5 to 40 clients. Consequently, the disk throughput drops from 537.6 MB/s to 229.2 MB/s– the disk throughputs of active DNs are less than 50 MB/s for almost 80% of the read workload time. This, in addition to the contention at the network due to the high concurrency, results in a network throughput of only 8048 MB/s and a variation of 5.7% in the throughput in-between clients.

Figure 5c depicts the read throughput of the same file when increasing the number of clients. REP maintains the same throughput per client, however, we can clearly see how the throughput under EC drops with increasing the number of concurrent clients. Table 2 shows the cluster-level network and disk throughput. REP maintains a stable disk throughput when increasing the number of clients and relative increase of network throughput with the number of clients. However, under EC, the disk throughput drops while the network throughput increases sub-linearly.

`Observation 4.` The goal of presenting data as EC groups is to reduce metadata overhead in the NN, but it results in a high imbalance in data distribution across DNs compared to replication. This, in turn, causes stragglers when high concurrent reads are performed to either distinct files or the same file, and therefore, it reduces the advantage of parallel chunks reads. Specifically, DNs which are serving data continuously to multiple clients will exhibit low disk throughput due to high contention on disk (distinct files) or high contention at the network (same file).

5.4 Implications

The advantage of striped block layout under EC is to enable parallel data reads. However, the use of EC groups (to reduce metadata overhead) reduces this advantage under concurrent reads (by different clients). Because it introduces imbalance in the data access across DNs which, in turn, causes stragglers. Therefore, mitigating stragglers under concurrent access is essential. While *late binding* can be employed (as in EC-Cache [31]), it comes with the cost of extra network transfer. Moreover, *intelligent* data access can be used (as in EC-Store [7]), but this method requires historical analysis of data access. On the other hand, reads under EC benefit more from OS caches than replication as data blocks are always read from the same node, which might be beneficial for iterative applications. In conclusion, the previous experiments demonstrate the feasibility and the advantages of using EC for read operations in large-scale clusters.

6 Related Work

Zhang et al. [40] show the performance benefits of EC when the intermediate and output data of MapReduce applications are encoded compared to 3-ways replication. Yao et al. [36] demonstrate that encoding the intermediate data of MapReduce like applications can provide faster recovery after failures especially for multi-stage applications. Darrous et al. [13] study the impact of EC with striped data layout on the performance of Sort, Wordcount and Kmeans applications under different system configurations (e.g., failures, different back-end storage devices and network configurations, etc.). Among their findings, they also show that disk contention caused by chunks distribution affects the performance of data-intensive applications under EC. Li et al. [24, 25] propose to collocate original data chunks and parity chunks on the same nodes to improve data locality (increase the number of local map tasks). This work is complementary to the aforementioned studies in that it specifically targets understanding the performance of (concurrent) write and read operations when the clients reside outside Hadoop cluster and provides a more detailed analysis of the network and disk performance under EC.

7 Conclusion

Efficient data access in large-scale storage systems is an important problem with the current rate of data generation. Erasure codes are increasingly deployed in many storage systems as cost-efficient alternative to replication. In this paper, we experimentally evaluate the performance of data read and write operations in HDFS under both replication and erasure coding. Our findings can be summarized as follows: (1) Write operations under replication have higher throughput than under EC for a single client. However, when increasing the number of concurrent clients, the throughput under EC approaches that under replication and even outperforms it under high concurrency. In addition, applying EC as an alternative to replication does not only reduce storage cost and disk overhead but also results in lower network traffic. (2) When reading data, EC can leverage parallel reads from multiple disks and deliver a 4.95x higher throughput than replication for a single client. However, under high concurrency, the performance of EC is impacted by the imbalanced load across nodes, which is caused by the distribution of chunks and the design of EC group.

As future work, we plan to implement a new data placement strategy that targets balancing the data load across nodes and study its impact on the performance of read/write operations and MapReduce applications in shared Hadoop clusters.

Acknowledgments

This work was supported in part by the ANR KerStream project (ANR-16-CE25-0014- 01), the Stack/Apollo connect talent project, and the Inria Project Lab program Discovery (see https:// beyondtheclouds.github.io). Experiments presented in this paper were carried out using the Grid'5000 testbed, supported by a scientific interest group hosted by Inria and including CNRS, RENATER, and several Universities as well as other organizations (see http://www.grid5000.fr/).

References

[1] 2022. Apache Flink. https://flink.apache.org
[2] 2022. Apache Hadoop. http://hadoop.apache.org
[3] 2022. Apache Spark. https://spark.apache.org
[4] 2022. CMU Hadoop traces. https://www.pdl.cmu.edu/HLA/
[5] 2022. Intel Intelligent Storage Acceleration Library Homepage. https://software.intel.com/en-us/storage/ISA-L
[6] 2022. psutil: Cross-platform lib for process and system monitoring in Python. https://github.com/giampaolo/psutil
[7] Michael Abebe, Khuzaima Daudjee, Brad Glasbergen, and Yuanfeng Tian. 2018. EC-Store: Bridging the Gap between Storage and Latency in Distributed Erasure Coded Systems. In *2018 IEEE 38th International Conference on Distributed Computing Systems (ICDCS)*. 255–266. https://doi.org/10.1109/ICDCS.2018.00034
[8] Ganesh Ananthanarayanan, Ali Ghodsi, Scott Shenker, and Ion Stoica. 2011. Disk-Locality in Datacenter Computing Considered Irrelevant. In *Proceedings of the 13th USENIX Conference on Hot Topics in Operating Systems* (Napa, California) *(HotOS'13)*. USENIX Association, USA, 12.
[9] Daniel Balouek, Alexandra Carpen Amarie, Ghislain Charrier, Frédéric Desprez, Emmanuel Jeannot, Emmanuel Jeanvoine, Adrien Lèbre, David Margery, Nicolas Niclausse, Lucas Nussbaum, Olivier Richard, Christian Pérez, Flavien Quesnel, Cyril Rohr, and Luc Sarzyniec. 2013. Adding Virtualization Capabilities to the Grid'5000 Testbed. In *Cloud Computing and Services Science*. Communications in Computer and Information Science, Vol. 367. Springer International Publishing, 3–20. https://doi.org/10.1007/978-3-319-04519-1_1
[10] Yanpei Chen, Sara Alspaugh, and Randy H Katz. 2012. *Design insights for MapReduce from diverse production workloads*. Technical Report. CALIFORNIA UNIV BERKELEY DEPT OF ELECTRICAL ENGINEERING AND COMPUTER SCIENCE.
[11] Mosharaf Chowdhury, Srikanth Kandula, and Ion Stoica. 2013. Leveraging Endpoint Flexibility in Data-Intensive Clusters. In *Proceedings of the ACM SIGCOMM 2013 Conference on SIGCOMM* (Hong Kong, China) *(SIGCOMM '13)*. Association for Computing Machinery, New York, NY, USA, 231–242. https://doi.org/10.1145/2486001.2486021
[12] Jad Darrous and Shadi Ibrahim. 2019. Enabling Data Processing under Erasure Coding in the Fog. Poster at ICPP 2019 - the 48th International Conference on Parallel Processing. https://hal.inria.fr/hal-02388835
[13] Jad Darrous, Shadi Ibrahim, and Christian Perez. 2019. Is it Time to Revisit Erasure Coding in Data-Intensive Clusters?. In *2019 IEEE 27th International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS)*. 165–178. https://doi.org/10.1109/MASCOTS.2019.00026
[14] Jeffrey Dean and Sanjay Ghemawat. 2008. MapReduce: Simplified Data Processing on Large Clusters. *Commun. ACM* 51, 1 (jan 2008), 107–113. https://doi.org/10.1145/1327452.1327492
[15] Florin Dinu and T.S. Eugene Ng. 2014. RCMP: Enabling Efficient Recomputation Based Failure Resilience for Big Data Analytics. In *2014 IEEE 28th International Parallel and Distributed Processing Symposium*. 962–971. https://doi.org/10.1109/IPDPS.2014.102
[16] Bin Fan, Wittawat Tantisiriroj, Lin Xiao, and Garth Gibson. 2009. DiskReduce: RAID for Data-Intensive Scalable Computing. In *Proceedings of the 4th Annual Workshop on Petascale Data Storage* (Portland, Oregon) *(PDSW '09)*. Association for Computing Machinery, New York, NY, USA, 6–10. https://doi.org/10.1145/1713072.1713075
[17] Sanjay Ghemawat, Howard Gobioff, and Shun-Tak Leung. 2003. The Google File System. In *Proceedings of the Nineteenth ACM Symposium on Operating Systems Principles* (Bolton Landing, NY, USA) *(SOSP '03)*. Association for Computing Machinery, New York, NY, USA, 29–43. https://doi.org/10.1145/945445.945450
[18] Hadoop. 2017. HDFS Erasure Coding. https://hadoop.apache.org/docs/r3.0.0/hadoop-project-dist/hadoop-hdfs/HDFSErasureCoding.html
[19] Andreas Haeberlen, Alan Mislove, and Peter Druschel. 2005. Glacier: Highly Durable, Decentralized Storage despite Massive Correlated Failures. In *Proceedings of the 2nd Conference on Symposium on Networked Systems Design & Implementation - Volume 2 (NSDI'05)*. USENIX Association, USA, 143–158.

[20] Shadi Ibrahim, Hai Jin, Lu Lu, Bingsheng He, Gabriel Antoniu, and Song Wu. 2012. Maestro: Replica-Aware Map Scheduling for MapReduce. In *Proceedings of the 2012 12th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (Ccgrid 2012) (CCGRID '12)*. IEEE Computer Society, USA, 435–442. https://doi.org/10.1109/CCGrid.2012.122

[21] Intel. 2017. ISA-L Performance report. https://01.org/sites/default/files/documentation/intel_isa-l_2.19_performance_report_0.pdf

[22] Jaeho Kim, Donghee Lee, and Sam H. Noh. 2015. Towards SLO Complying SSDs through OPS Isolation. In *Proceedings of the 13th USENIX Conference on File and Storage Technologies* (Santa Clara, CA) *(FAST'15)*. USENIX Association, USA, 183–189.

[23] John Kubiatowicz, David Bindel, Yan Chen, Steven Czerwinski, Patrick Eaton, Dennis Geels, Ramakrishna Gummadi, Sean Rhea, Hakim Weatherspoon, Westley Weimer, Chris Wells, and Ben Zhao. 2000. OceanStore: An Architecture for Global-Scale Persistent Storage. *SIGARCH Comput. Archit. News* 28, 5 (nov 2000), 190–201. https://doi.org/10.1145/378995.379239

[24] Jun Li and Baochun Li. 2017. On Data Parallelism of Erasure Coding in Distributed Storage Systems. In *2017 IEEE 37th International Conference on Distributed Computing Systems (ICDCS)*. 45–56. https://doi.org/10.1109/ICDCS.2017.191

[25] Jun Li and Baochun Li. 2018. Parallelism-Aware Locally Repairable Code for Distributed Storage Systems. In *2018 IEEE 38th International Conference on Distributed Computing Systems (ICDCS)*. 87–98. https://doi.org/10.1109/ICDCS.2018.00019

[26] Runhui Li, Patrick P.C. Lee, and Yuchong Hu. 2014. Degraded-First Scheduling for MapReduce in Erasure-Coded Storage Clusters. In *2014 44th Annual IEEE/IFIP International Conference on Dependable Systems and Networks*. 419–430. https://doi.org/10.1109/DSN.2014.47

[27] Bunjamin Memishi, Shadi Ibrahim, María S. Pérez, and Gabriel Antoniu. 2016. *Fault Tolerance in MapReduce: A Survey*. Springer International Publishing, Cham, 205–240. https://doi.org/10.1007/978-3-319-44881-7_11

[28] Subramanian Muralidhar, Wyatt Lloyd, Sabyasachi Roy, Cory Hill, Ernest Lin, Weiwen Liu, Satadru Pan, Shiva Shankar, Viswanath Sivakumar, Linpeng Tang, and Sanjeev Kumar. 2014. F4: Facebook's Warm BLOB Storage System. In *Proceedings of the 11th USENIX Conference on Operating Systems Design and Implementation* (Broomfield, CO) *(OSDI'14)*. USENIX Association, USA, 383–398.

[29] Salman Niazi, Mikael Ronström, Seif Haridi, and Jim Dowling. 2018. Size Matters: Improving the Performance of Small Files in Hadoop. In *Proceedings of the 19th International Middleware Conference* (Rennes, France) *(Middleware '18)*. Association for Computing Machinery, New York, NY, USA, 26–39. https://doi.org/10.1145/3274808.3274811

[30] K.V. Rashmi, Nihar B. Shah, Dikang Gu, Hairong Kuang, Dhruba Borthakur, and Kannan Ramchandran. 2014. A "Hitchhiker's" Guide to Fast and Efficient Data Reconstruction in Erasure-Coded Data Centers. In *Proceedings of the 2014 ACM Conference on SIGCOMM* (Chicago, Illinois, USA) *(SIGCOMM '14)*. Association for Computing Machinery, New York, NY, USA, 331–342. https://doi.org/10.1145/2619239.2626325

[31] K. V. Rashmi, Mosharaf Chowdhury, Jack Kosaian, Ion Stoica, and Kannan Ramchandran. 2016. EC-Cache: Load-Balanced, Low-Latency Cluster Caching with Online Erasure Coding. In *Proceedings of the 12th USENIX Conference on Operating Systems Design and Implementation* (Savannah, GA, USA) *(OSDI'16)*. USENIX Association, USA, 401–417.

[32] David Reinsel, John Rydning, and John F. Gantz. IDC Report 2021. Worldwide Global DataSphere Forecast, 2021 – 2025: The World Keeps Creating More Data - Now, What Do We Do with It All?

[33] Rodrigo Rodrigues and Barbara Liskov. 2005. High Availability in DHTs: Erasure Coding vs. Replication. In *Peer-to-Peer Systems IV*, Miguel Castro and Robbert van Renesse (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 226–239.

[34] Maheswaran Sathiamoorthy, Megasthenis Asteris, Dimitris Papailiopoulos, Alexandros G. Dimakis, Ramkumar Vadali, Scott Chen, and Dhruba Borthakur. 2013. XORing Elephants: Novel Erasure Codes for Big Data. *Proc. VLDB Endow.* 6, 5 (mar 2013), 325–336. https://doi.org/10.14778/2535573.2488339

[35] Konstantin Shvachko, Hairong Kuang, Sanjay Radia, and Robert Chansler. 2010. The Hadoop Distributed File System. In *2010 IEEE 26th Symposium on Mass Storage Systems and Technologies (MSST)*. 1–10. https://doi.org/10.1109/MSST.2010.5496972

[36] Xin Yao, Cho-Li Wang, and Mingzhe Zhang. 2019. EC-Shuffle: Dynamic Erasure Coding Optimization for Efficient and Reliable Shuffle in Spark. In *2019 19th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGRID)*. 41–51. https://doi.org/10.1109/CCGRID.2019.00014

[37] Orcun Yildiz, Shadi Ibrahim, and Gabriel Antoniu. 2017. Enabling fast failure recovery in shared Hadoop clusters: Towards failure-aware scheduling. *Future Generation Computer Systems* 74 (2017), 208–219. https://doi.org/10.1016/j.future.2016.02.015

[38] Matei Zaharia, Dhruba Borthakur, Joydeep Sen Sarma, Khaled Elmeleegy, Scott Shenker, and Ion Stoica. 2010. Delay Scheduling: A Simple Technique for Achieving Locality and Fairness in Cluster Scheduling. In *Proceedings of the 5th European Conference on Computer Systems* (Paris, France) *(EuroSys '10)*. Association for Computing Machinery, New York, NY, USA, 265–278. https://doi.org/10.1145/1755913.1755940

[39] Heng Zhang, Mingkai Dong, and Haibo Chen. 2016. Efficient and Available In-Memory KV-Store with Hybrid Erasure Coding and Replication. In *Proceedings of the 14th Usenix Conference on File and Storage Technologies* (Santa Clara, CA) *(FAST'16)*. USENIX Association, USA, 167–180.

[40] Zhe Zhang, Amey Deshpande, Xiaosong Ma, Eno Thereska, and Dushyanth Narayanan. 2010. *Does erasure coding have a role to play in my data center?* Technical Report. Microsoft research.

[41] Zhe Zhang, Weihua Jiang, Andrew Wang, Kai Zheng, Uma Maheswara G., and Vinayakumar B. 2015. Introduction to HDFS Erasure Coding in Apache Hadoop. https://blog.cloudera.com/blog/2015/09/introduction-to-hdfs-erasure-coding-in-apache-hadoop