RNCDDS - RANDOM NETWORK CODED DISTRIBUTED DATA SYSTEM

Hiroshi Nishida [1], *Thinh Nguyen* [2]

[1] ASUSA Corporation, [2] School of EECS, Oregon State University

ABSTRACT

Many Random Network Coding based storage systems have been proposed to increase robustness in terms of data preservation and storage efficiency. However, there are two major practical issues: (1) slow encoding and decoding speeds and (2) difficult access to the data. In this paper, RNCDDS solves problem (1) by introducing a new and efficient Galois Field arithmetic library and problem (2) by employing a new JavaScript program. As a result, RNCDDS is not only theoretically more robust and storage efficient than major distributed data systems such as Hadoop (HDFS) and GlusterFS, but also outperforms them in all speed measurements. Also, the proposed JavaScript program enables easy data fetch through a web browser including watching a video with an HTML5 video player and has potential for drastically reducing data storage amount in a cloud system or Content Delivery Network.

Index Terms— Distributed System, Data Storage, Random Network Coding, Galois Field, Multimedia Streaming, Content Delivery Network, Cloud Computing

1. INTRODUCTION

Distributed data systems are widely employed in cloud computing not only to store a huge amount of data but also to assure semi-permanent data preservation. Hadoop [1], a de facto standard distributed data system in cloud computing, is used by many enterprises and serves a vast amount of data in their data centers. However, due to their simple raw data distribution scheme, most distributed data systems do not always achieve both high robustness and storage efficiency at the same time. For instance in a Hadoop cluster, a file is split into some pieces and their replications are distributed to the servers as shown in Fig. 1. To guarantee the redundancy of R servers, i.e., the retrieval of data under concurrent failure of R servers, Hadoop requires $1 + R$ storage size in a cluster. In other words to store a 1GB file, 3GB storage is necessary in a Hadoop cluster consisting of original server plus two other fault-tolerant servers. This is also applicable to the case in which simple replications of the original file are placed on the storage servers such as GlusterFS [2] (Fig. 2). On the other hand, a distributed data system based on *Random Network Coding* (RNC) that for instance distributes encoded data with

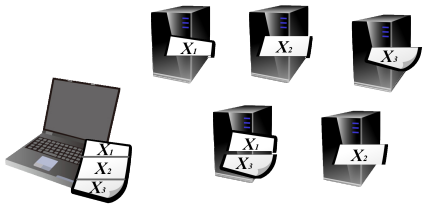


Fig. 1: Hadoop splits a file into pieces and distributes their replications to servers.

Fig. 2: GlusterFS places replications on servers.

$1/3$ the size of the original file to each server requires only $1 + R/3$ storage in a cluster, which saves as much as 44% storage for $R = 2$ compared to Hadoop or GlusterFS. However, current RNC based systems suffer from the following two major issues:

1. The slow processing speed in encoding and decoding
2. The difficulty in access to data from clients

In our investigation, fast processing in encoding and decoding turned out to be difficult to achieve due to lack in an efficient Galois Field (GF) arithmetic library with a lax license (see Sec. 3 and 4). Also, users may hesitate to use dedicated client programs employed in most RNC based systems to access the encoded data for security reasons and will prefer to watch a video on a web browser. *RNCDDS* solved those issues and made an RNC based system practical by introducing the following two programs:

1. *gf-nishida-16*: a new and efficient GF arithmetic library
2. *RNC.js*: a JavaScript program to download and decode RNC encoded data and to play a video with an HTML5 video player

Moreover, RNCDDS is planned to be open source and will be publicly downloadable in the future.

This paper presents the architecture of RNCDDS together with the fundamentals of RNC, a brief explanation of gf-nishida-16, its advantages over the other major cloud storage systems, and its efficiency in delivering multimedia contents.

2. RELATED WORK

Other than Hadoop and GlusterFS, many storage systems, such as OpenAFS [3], Ceph [4], MooseFS [5], Quantcast File System (QFS) [6] and Pyramid Codes [7], have been proposed and we believe they play an important role in cloud computing. Among those systems, Hadoop, GlusterFS, OpenAFS, Ceph and MooseFS are based on non-coded data distribution, while QFS and Pyramid Codes employs coded data distribution by Erasure Coding or similar techniques in order to increase the robustness and storage capacity efficiency. Erasure Coding is very similar to RNC but basically limits the number of encoded data. Therefore unlike RNCDDS, it cannot distribute as many different encoded data as possible to servers, which restricts the usage in multimedia streaming. RNC based storage systems are mostly proposed for used in P2P networks [8] [9] [10] and are proved to be robust, resilient and efficient in storage capacity. However to the best of our knowledge, there are few papers that refer to the encoding and decoding speeds [11]. In our research, we concluded that an RNC based storage system could not be fast enough without an efficient GF arithmetic library (see Sec. 3).

As for multimedia streaming with RNC, many papers are focused on the usage in P2P networks [12] [13] [14] or wireless networks [15] [16], and there are few practical systems based on the client-server style except for [17] which introduces a unique P2P assisted client-server system. Also, there is little information on the accessibility to RNC data, and we have not confirmed if there is a program to decode and directly access RNC data on a web browser. To the best of our knowledge, there is no practical RNC based multimedia streaming system in a pure client-server style with easy access from the clients.

3. FUNDAMENTALS OF RNC

RNC provides high robustness and storage efficiency by utilizing the characteristics of linear equations. Suppose we have three integers $x_1 = 3$, $x_2 = 1$, $x_3 = 2$ and randomly create many linear equations such as:

$$\begin{cases} 2x_1 + 5x_2 + x_3 = 13 \\ 7x_1 + 3x_2 + 8x_3 = 42 \\ 4x_1 + x_2 + 2x_3 = 19 \\ 5x_1 + 4x_2 + 9x_3 = 33 \\ \cdots \end{cases} \quad (1)$$

To obtain the original x_1, x_2, x_3, we basically need only three equations out of them, which is the fundamental technique employed by RNC. RNC splits a file into some pieces, say x_1, x_2, x_3, and creates linear equations (encode) with randomly generated coefficients as such as:

$$\begin{cases} a_{1,1}x_1 + a_{1,2}x_2 + a_{1,3}x_3 = b_1 \\ a_{2,1}x_1 + a_{2,2}x_2 + a_{2,3}x_3 = b_2 \\ \cdots \end{cases} \quad (2)$$

Then, each of $b_1, b_2, \cdots$ is distributed to each server together with associate coefficients as and composes one encoded file as shown in Fig. 3. To restore the original file, we need to collect only three encoded files from any combination of three servers and to obtain x_1, x_2, x_3 by solving the system of linear equations (decoding). In other words, as long as three servers that have the encoded files are alive in the system, the original file can be regained. Therefore in Fig. 3, the cluster guarantees the retrieval of data under the failure of any two servers.

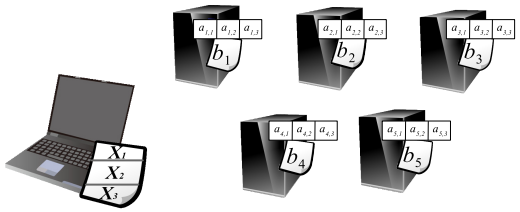


Fig. 3: RNC distributes encoded data to servers, where the size of each encoded file is approximately 1/3 of the original.

Also, note that the size of $b_n, \forall n$ is equal to that of x_1, x_2, x_3 (= 1/3 of the original file size) because the arithmetic computations in (2) are executed in GF. Since the size of as (16bits $\times$ 3) is usually negligibly small compared to that of b_n, we can safely regard the size of each encoded file as almost equal to 1/3 of the original file size. As a result, the storage size in Fig. 3 is only $5/3$ with the redundancy of two servers, while Hadoop and GlusterFS require a size of 3 storage with the same redundancy.

The only drawback of RNC will be the computation time; creating linear equations (encoding) and solving a system of linear equations (decoding) require certain amounts of time especially in GF. In our development of RNCDDS, we noticed the times taken for encoding and decoding were greatly affected by the computation speed in GF. We first started our project with one of open source GF arithmetic libraries without considering its speed, which later turned out to be very slow and to be impractical. We afterward employed another open source library called *GF-Complete* [18] that focused on the processing speed using SSE, and gained preferable results, though it has been removed by its author due to the possible patent infringement [19]. Using SSE for GF arithmetic computation seems to be caught on a patent. However, it encouraged us to develop an original GF arithmetic library and gave us the idea to create gf-nishida-16.

4. GF-NISHIDA-16

Though arithmetic computation in GF is indispensable for coded data systems, it costs expensive and therefore affects the system performance. Moreover, some computation techniques are patented and do not allow users to easily use them. Gf-nishida-16 [20], an open source $GF(2^{16})$ arithmetic library with a 2-Clause BSD license, resolves those issues and provides high speed arithmetic computation in $GF(2^{16})$. Due to space limitations, we skip the explanation on its details but note the high speed processing in gf-nishida-16 is achieved by optimizing two step memory lookup. Please see our technical paper [21] for the full description. In this section, we introduce its benchmark results and show its efficiency.

We measured the elapsed times for multiplication and division with the following open source GF libraries:

gf-nishida-8 An 8bit version of gf-nishida-16.

gf-nishida-16 Our main library.

gf-nishida-region-16 A region computation version of gf-nishida-16 employed by RNCDDS which speeds up the repeated computation of $a \times x$ or $a \div x$ in GF where a is fixed and x is variable (see [21]).

gf-complete-* An efficient $GF(2^n)$ library with SSE [18] [19].

gf-basic-8 A simple 8bit library [22].

gf-plank-* A library based on [23].

gf-plank-logtable-16 A 16bit library similar to gf-nishida-16 [23].

gf-clmul-128 A 128bit program retrieved from Solaris source code that uses Intel CLMUL instruction set specialized for $GF(2^n)$ multiplication.

gf-ff-* A library downloaded from [24].

gf-aes-gcm-128 A 128bit GF program retrieved from FreeBSD source code that uses no special techniques.

Note that the results in division exclude some libraries that do not have division functions.

The benchmark results are shown in Table 1. As a consequence, gf-nishida-16, especially gf-nishida-region-16 used by RNCDDS shows excellent performance in both multiplication and division. Considering its performance and risklessness of patent infringement, we believe gf-nishida-16 is the best library for the GF arithmetic computation in RNC.

5. RNCDDS

5.1. Encoding

Suppose a client distributes four encoded files to servers in a cluster, then it first decides three 16bit coefficients for each of them such as $a_{k,1}$, $a_{k,2}$, $a_{k,3}, \forall k \in \{1,2,3,4\}$. The client next sends them to each server together with other file information such as the file size, modification time. In encoding, a file is cut down every 6bytes and each 2bytes in the 6byte chunk is assigned to x_1, x_2, x_3 as illustrated in Fig. 4. Each chunk is encoded by calculating $b_k = a_{k,1}x_1 + a_{k,2}x_2 + a_{k,3}x_3, \forall k \in \{1,2,3,4\}$ with gf-nishida-16 and newly obtained 2byte data b_k is sent to server k, which is repeated till the end of the file. Note the size of the encoded data is basically 1/3 of the size of the original file. However as a header containing the coefficients $a_{k,1}$, $a_{k,2}$, $a_{k,3}$, file attributes, checksum, etc is appended to it, the final size of an encoded file stored on each server becomes slightly greater than it.

Table 1: Elapsed times (ms) in multiplication and division with $GF(2^n)$ arithmetic libraries (the shorter, the faster).

Library	Multiplication	Division
gf-nishida-region-16	**41583**	**41557**
gf-complete-64	55106	4424996
gf-nishida-8	61171	86159
gf-basic-8	61195	114292
gf-nishida-16	118850	118973
gf-complete-32	168429	10053109
gf-plank-logtable-16	244935	251010
gf-plank-32	314016	27664514
gf-plank-16	391792	369352
gf-plank-8	406299	360553
gf-clmul-128	1281013	-
gf-ff-64	5231520	5393946
gf-ff-32	10464125	105356429
gf-aes-gcm-128	14013111	-

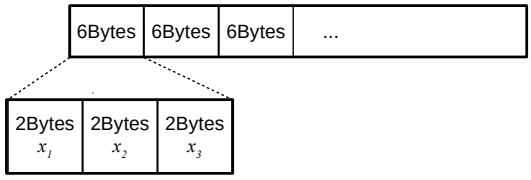


Fig. 4: RNCDDS splits a file every 6bytes and assigns each 2bytes to x_1, x_2, x_3.

5.2. Decoding

Decoding starts with searching for the servers that have the target files, where RNCDDS performs it by using Consistent Hashing [25] without a metadata server (see Sec. 5.3). If two or fewer servers have the files, the decoding fails. If three or more servers have the files, then the client chooses three servers and collects the coefficients $a_{k,1}$, $a_{k,2}$, $a_{k,3}$ from them. The client next retrieves the encoded data and solves the following system of linear equations for each 2byte chunk of b_k by the Gaussian Elimination:

$$\begin{cases} a_{1,1}x_1 + a_{1,2}x_2 + a_{1,3}x_3 = b_1 \\ a_{2,1}x_1 + a_{2,2}x_2 + a_{2,3}x_3 = b_2 \\ a_{3,1}x_1 + a_{3,2}x_2 + a_{3,3}x_3 = b_3. \end{cases} \quad (3)$$

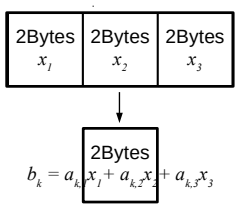
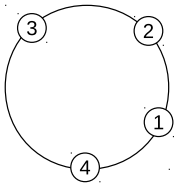
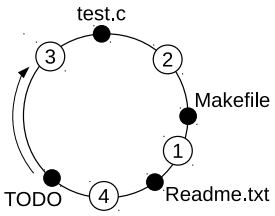


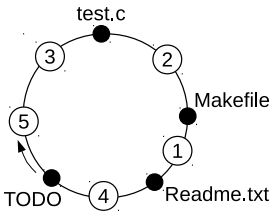
Fig. 5: RNCDDS encodes 6byte data to 2byte data.



(a) An initial state of a hash ring.



(b) Before adding a server.



(c) After adding server 5.

Fig. 6: An example of file management by Consistent Hashing.

Since x_1, x_2, x_3 hereby are 2bytes, the original file is restored every $2 \times 3 = 6$bytes. The decoding ends at the end of the file and the decoded file is checked with SHA256.

5.3. File Management by Consistent Hashing

Hadoop manages file information such as the attributes and locations by a metadata server. A metadata server easily scales the cluster but can be a single point of failure and its failure disables the access to all data in the cluster. In RNCDDS, on the other hand, files are managed by Consistent Hashing (CH) that provides resilient data distribution with minimum reorganization when servers are added or deleted without a metadata server. For instance, suppose that there are four servers and four files `test.c`, `Makefile`, `Readme.txt` and `TODO`. CH initially locates the servers as shown in Fig. 6 (a) by hashing their names and then locates the four files in a similar way as seen in 6 (b). Since the hash code or owner of a file in CH is the closest node (server) in the clockwise direction on the ring, the owner of `TODO` becomes server 3 and that of `test.c` becomes server 2. When server 5 is added to the ring as seen in Fig. 6 (c), the only file whose owner is affected is `TODO`, where its new owner becomes server 5. Thus, CH achieves resilient and scalable file management.

In RNCDDS, servers in a cluster maintain a common server list and a client retrieves it when it connects to one of them. The client then creates a hash ring based on it as shown in Fig. 6 (a) and looks up files in a way illustrated in Fig. 6 (b) (c). Once the hash ring is created, the time complexity for the file lookup is only $O(1)$. Note RNCDDS locates files by broadcasting messages to all servers even if there is a discrepancy with the server lists owned by servers and CH takes no effect. Thus, RNCDDS pays scrupulous attention to data preservation.

5.4. Programs

RNCDDS consists of three C programs tested with FreeBSD 11 and CentOS 7 and a JavaScript program. The C programs depend on only a few external libraries such as OpenSSL, FUSE, libevent2 (only for CentOS, the FreeBSD version uses kqueue) so that they can be easily ported to other platforms.

rncddsd is a server program that communicates with client programs `rncdds` and `rncfsd` and receives/sends, saves/reads encoded files. It is usually run as a daemon process.

rncdds, corresponding to Hadoop's `hadoop fs`, is a command line client program that encodes, uploads, decodes, downloads files and sends other messages to `rncddsd`. For example,

```
% rncdds put fileA /dirA/
```

uploads fileA under /dirA on the servers. Note `rncdds` is designed to achieve fast directory upload and download by multithreaded parallel pipeline processing, where encoding/decoding and reading/saving processes are executed simultaneously. Our benchmark results in Sec. 7 show its superior performance.

rncfsd mounts RNCDDS as a filesystem through FUSE userland filesystem library [26] so that one can access the files as if they were on the local host. This is very convenient though it does not perform as well as `rncdds` (see Table 2). However, using an SSD and/or a small size of HDD as cache to store the decoded files, `rncfsd` minimizes the overheads for decoding. Both `rncdds` and `rncfsd` check the consistency of encoded files by their timestamps and inspect the decoded files with SHA256.

RNC.js is a JavaScript program that downloads RNC encoded data from three servers via HTTP using XMLHttpRequest and decodes them. If the decoded data is a video, RNC.js plays it on a web browser with an HTML5 player. It is currently capable of playing segmented MP4 and WebM video data and appends a segment of data at a time to the buffer of the video player while playing it. Due to the overheads for decoding through JavaScript, video playback with RNC.js consumes slightly higher CPU power than directly playing MP4 or WebM data though the difference is very small. Fig. 7 compares the CPU usages in direct playback (left) and playback by RNC.js (right), where there is no re-

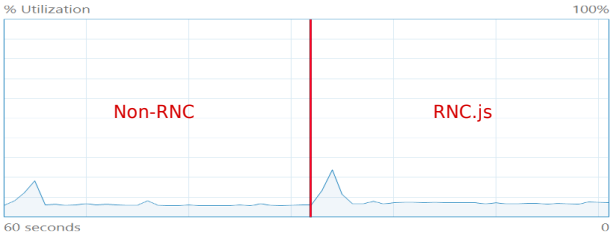
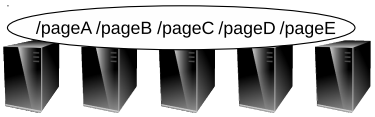


Fig. 7: Comparison in CPU usage between direct playback and playback by RNC.js. There is no remarkable difference.

(a) Five HTTP servers.



(b) An HTTP service system by five RNCDDS servers.

Fig. 8: An example of load distribution by RNCDDS.

markable difference between their CPU usages except for the spike in each playback due to page loading. Suppose the CPU usage for direct playback of MP4 data is 6 to 7%, RNC.js uses roughly 7 to 9% CPU power. If decoding functions written in C are incorporated into the web browsers, the CPU usage will be even lowered.

5.5. Load Distribution by RNCDDS

RNCDDS is designed not only to distribute data but also to distribute load on the servers. In this subsection, we introduce a unique HTTP service system that distributes the load on the servers by RNCDDS as its application example.

Suppose we have five regular HTTP servers as shown in Fig. 8 (a). If we replace these servers with an RNCDDS system consisting of five servers that store all the data the original HTTP servers owned as seen in Fig. 8 (b) and run a filesystem client program `rncfsd` (see Sec. 5.4) on each server, then all the data become retrievable from all the RNCDDS servers. If we also run an HTTP service program such as Apache or Nginx on each server, then all the data the original HTTP servers had can be served from any new RNCDDS server and herewith the load by HTTP access can be distributed to all RNCDDS servers. This system provides the following advantages:

1. Heavy access to a server in Fig. 8 (a) degrades the performance and sometimes causes inaccessibility to the server. The load distribution by a system in Fig. 8 (b) decreases the possibility of server inaccessibility.

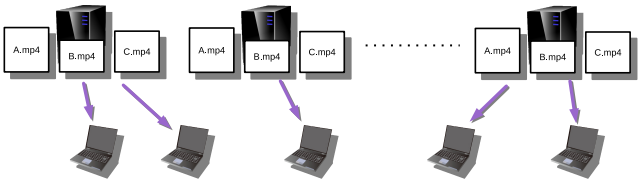


Fig. 9: Servers store the same files in a typical CDN.

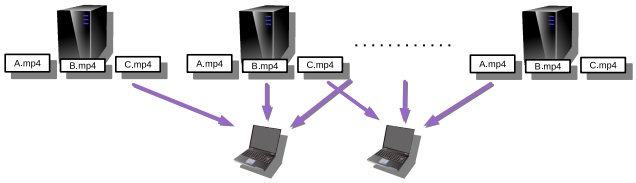


Fig. 10: RNCDDS reduces the data storage amount in a CDN.

2. The RNCDDS system with data redundancy such as $R = 2$ can be operated under the fault of two servers, which increases robustness and reliability of the system.

Note the overheads for decoding can be alleviated by `rncfsd`'s cache which temporarily stores decoded data on an SSD.

6. MULTIMEDIA STREAMING WITH RNCDDS

Many multimedia streaming services are provided through a Content Delivery Network (CDN) which is closely connected with cloud storage systems. In a typical CDN, servers store the same files as shown in Fig. 9 in order to distribute access from clients. However, it requires a vast amount of storage space and causes significant performance degradation for the following reasons:

1. The usage rate of HDDs over SSDs increases and the throughput drops in consequence.
2. The cache hit ratio decreases.

If we replace those files with RNC encoded files and let clients access them through RNC.js as seen in Fig. 10, the storage size becomes 1/3 and therefore not only a large amount of budget can be saved but also the throughput will be improved. This is a huge advantage and we expect RNCDDS to be employed by many video streaming CDNs. We also note that there is almost no need to re-configure the server settings; if the servers provide video data via HTTP with Nginx or Apache, all what the administrators have to do are to replace the original video files with the corresponding RNC encoded files and to add a few lines such as "add_header 'Access-Control-Allow-Credentials' 'true';" to their configuration files as a client needs a cross-origin request (CORS) to access three different RNC encoded file servers. Also, note the total amount of traffic does not change because a client

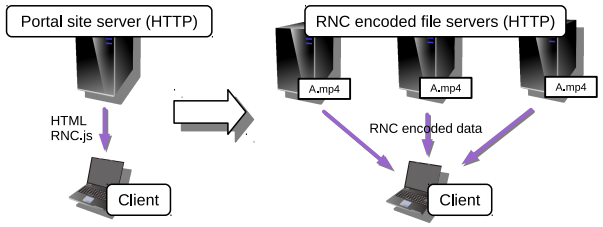


Fig. 11: A client first accesses a portal site server and then retrieves RNC encoded data from three RNC servers.

downloads three encoded files whose sizes are 1/3 of the original.

In our prototype system, a client first accesses a portal site and downloads RNC.js and the information on the three RNC encoded file servers. The client then fetches RNC encoded data from them via HTTP and plays a video with RNC.js (see Fig. 11). Our RNC demonstration video with three RNC encoded file servers can be watched at http://rnc01.asusa.net and the original video (no RNC) can be watched at http://rnc01.asusa.net/videos.

7. BENCHMARK RESULTS

To assure our system's performance, we measured the elapsed times for uploading and downloading a directory containing 5,533 sub-directories and 80,919 files and its 63GB tarball file with Hadoop, GlusterFS and RNCDDS (`rncdds` and `rncfsd`). Each measurement was repeated ten times and was averaged. The following is the hardware specification of our host machines:

CPU Intel Xeon E3-1225v5

RAM DDR4-2133 ECC Non-buffered 64GB

NIC Intel X550 10GBase-T (1Gbps was used in benchmark due to the capacity of our switch)

HDD 2TB 64MB Cache 7200RPM SATA3 $\times$ 2 as RAID1 for main storage, 1TB 64MB Cache 7200RPM SATA3 for `rncfsd`'s write cache

SSD 128GB SATA3 for `rncfsd`'s read cache

Four servers were used as data storage, that is, there were four slave nodes in Hadoop and four regular servers in GlusterFS and RNCDDS, and one client sent/received data to/from them. As for the the redundancy, we used $R = 1$, i.e., two replications for Hadoop and GlusterFS and four encoded files for `rncdds` and `rncfsd`. All the hosts were operated with CentOS 7. To upload/download a file/directory, we used

```
% hadoop fs -copyFromLocal/-copyToLocal
```

command for Hadoop and

```
% rncdds put/get
```

command for `rncdds`. As GlusterFS and `rncfsd` handle data through filesystems,

Table 2: Elapsed times for uploading/downloading 63GB file/directory (the shorter, the faster).

	Hadoop	GlusterFS	rncdds	rncfsd
File upload	10m30.2s	17m26.4s	**8m49.6s**	8m56.6s
Dir upload	48m34.0s	54m18.5s	**12m31.1s**	17m04.5s
File download	10m31.9s	10m51.3s	**8m40.0s**	9m12.8s
Dir download	20m51.0s	25m10.0s	**16m51.1s**	34m54.5s

```
% cp -a
```

was used towards the mounted directories.

The results are shown in Table 2. Overall, `rncdds` shows excellent performance and surpasses Hadoop and GlusterFS in all metrics. The directory upload with Hadoop and GlusterFS takes much longer than that with `rncdds` and `rncfsd`. The performance degradation in handling small files with Hadoop seems to be a known issue [27] [28]. The overheads by the filesystem interface reduces the performance of `rncfsd` compared to `rncdds`, which is conspicuous at the directory download. `rncdds` downloads, decodes and saves multiple files simultaneously, while `rncfsd` has to process the files one by one because the filesystem does not pass the order for the next file till the order for the current file has been completely processed. Note both `rncdds` and `rncfsd` consume more CPU power than Hadoop and GlusterFS. Therefore, use of a fast CPU is recommended for RNCDDS.

8. CONCLUSION

In this paper, we have illustrated the architecture of RNCDDS and have shown its robustness, storage efficiency, superb performance and high affinity with multimedia streaming. We believe RNCDDS will not only play an important role in the future cloud multimedia streaming but will also contribute to the innovation in the distributed storage technology. However, the optimization of RNC.js and research on the optimal allocation of RNC encoded files to servers in terms of load and storage size balance remain as our future work.

9. REFERENCES

[1] "Hadoop," `http://hadoop.apache.org/`.

[2] "Glusterfs," `https://www.gluster.org/`.

[3] "Openafs," `https://www.openafs.org/`.

[4] Sage A. Weil, Scott A. Brandt, Ethan L. Miller, Darrell D. E. Long, and Carlos Maltzahn, "Ceph: A scalable, high-performance distributed file system," in *Proceedings of the 7th Symposium on Operating Systems Design and Implementation*, Berkeley, CA, USA, 2006, OSDI '06, pp. 307–320, USENIX Association.

[5] "Moosefs," https://moosefs.org/.

[6] Michael Ovsiannikov, Silvius Rus, Damian Reeves, Paul Sutter, Sriram Rao, and Jim Kelly, "The quantcast file system," *Proc. VLDB Endow.*, vol. 6, no. 11, pp. 1092–1101, Aug. 2013.

[7] Cheng Huang, Minghua Chen, and Jin Li, "Pyramid codes: Flexible schemes to trade space for access efficiency in reliable data storage systems," *Trans. Storage*, vol. 9, no. 1, pp. 3:1–3:28, Mar. 2013.

[8] Kien Nguyen, Thinh Nguyen, Yevgeniy Kovchegov, and Viet Le, "Distributed data replenishment," *IEEE Trans. Parallel Distrib. Syst.*, vol. 24, no. 2, pp. 275–287, Feb. 2013.

[9] B. Li and D. Niu, "Random network coding in peer-to-peer networks: From theory to practice," *Proceedings of the IEEE*, vol. 99, no. 3, pp. 513–523, March 2011.

[10] Alexandros G. Dimakis, P. Brighten Godfrey, Yunnan Wu, Martin J. Wainwright, and Kannan Ramchandran, "Network coding for distributed storage systems," *IEEE Trans. Inf. Theor.*, vol. 56, no. 9, pp. 4539–4551, Sept. 2010.

[11] Ádám Visegrádi and Péter Kacsuk, *Efficient Random Network Coding for Distributed Storage Systems*, pp. 385–394, Springer Berlin Heidelberg, Berlin, Heidelberg, 2014.

[12] Kien Nguyen, Thinh Nguyen, and Sen-Ching Cheung, "Video streaming with network coding," *J. Signal Process. Syst.*, vol. 59, no. 3, pp. 319–333, June 2010.

[13] Anh Tuan Nguyen, Baochun Li, and Frank Eliassen, "Chameleon: Adaptive peer-to-peer streaming with network coding," in *Proceedings of the 29th Conference on Information Communications*, Piscataway, NJ, USA, 2010, INFOCOM'10, pp. 2088–2096, IEEE Press.

[14] B. Barekatain, D. Khezrimotlagh, M. Aizaini Maarof, H. R. Ghaeini, S. Salleh, A. A. Quintana, B. Akbari, and A. T. Cabrera, "MATIN: A Random Network Coding Based Framework for High Quality Peer-to-Peer Live Video Streaming," *PLoS ONE*, vol. 8, pp. e69844, Aug. 2013.

[15] Luísa Lima, Steluta Gheorghiu, João Barros, Muriel Médard, and Alberto Lopez Toledo, "Secure network coding for multi-resolution wireless video streaming," *IEEE J.Sel. A. Commun.*, vol. 28, no. 3, pp. 377–388, Apr. 2010.

[16] D. Vukobratovi, C. Khirallah, V. Stankovi, and J. S. Thompson, "Random network coding for multimedia delivery services in lte/lte-advanced," *IEEE Transactions on Multimedia*, vol. 16, no. 1, pp. 277–282, Jan 2014.

[17] Z. Liu, C. Wu, B. Li, and S. Zhao, "Uusee: Large-scale operational on-demand streaming with random network coding," in *2010 Proceedings IEEE INFOCOM*, March 2010, pp. 1–9.

[18] James S. Plank, Kevin M. Greenan, and Ethan L. Miller, "Screaming fast galois field arithmetic using intel simd instructions," in *11th USENIX Conference on File and Storage Technologies (FAST 13)*, San Jose, CA, Feb. 2013, pp. 298–306, USENIX Association.

[19] James S. Plank and et al., "Gf-complete: A comprehensive open source library for galois field arithmetic, version 1.0," http://web.eecs.utk.edu/~plank/plank/papers/CS-13-716.html.

[20] Hiroshi Nishida, "gf-nishida-16 web site," https://github.com/scopedog/gf-nishida-16/.

[21] Hiroshi Nishida, "gf-nishida-16: Simple and efficient $gf(2^{16})$ library," https://github.com/scopedog/gf-nishida-16/blob/master/gf-nishida-16.pdf.

[22] "Basic library for calculation on finite field," http://www.codeforge.com/article/242688/.

[23] James S. Plank., "Fast galois field arithmetic library in c/c++," http://web.eecs.utk.edu/~plank/plank/papers/CS-07-593/.

[24] Antonio Bellezza, "Binary finite field library," http://www.beautylabs.net/software/finitefields.html.

[25] David Karger, Eric Lehman, Tom Leighton, Rina Panigrahy, Matthew Levine, and Daniel Lewin, "Consistent hashing and random trees: distributed caching protocols for relieving hot spots on the world wide web," in *Proceedings of the twenty-ninth annual ACM symposium on Theory of computing*, New York, NY, USA, 1997, STOC '97, pp. 654–663, ACM.

[26] "Fuse (filesystem in userspace)," https://github.com/libfuse/.

[27] "Dealing with hadoop's small files problem," http://snowplowanalytics.com/blog/2013/05/30/dealing-with-hadoops-small-files-problem/.

[28] "The small files problem," http://blog.cloudera.com/blog/2009/02/the-small-files-problem/.