International Journal of Computer and Communication Technology

Volume 5 | Issue 1 Article 8

January 2014

Coding Schemes for Distributed Storage Systems: Implementation and Improvements

Saman Tabatabaeian
Department of Computer Science and Information Sciences, University of Hyderabad, Hyderabad, AP, India, saman_tabatabaeian@yahoo.com

Rajendra P. Lal
Department of Computer Science and Information Sciences, University of Hyderabad, Hyderabad, AP, India, rplcs@uohyd.ernet.in

Wilson Naik
Department of Computer Science and Information Sciences, University of Hyderabad, Hyderabad, AP, India, naikcs@uohyd.ernet.in

Recommended Citation
Tabatabaeian, Saman; Lal, Rajendra P.; and Naik, Wilson (2014) "Coding Schemes for Distributed Storage Systems: Implementation and Improvements," *International Journal of Computer and Communication Technology*: Vol. 5 : Iss. 1 , Article 8.
DOI: 10.47893/IJCCT.2014.1218
Available at: https://www.interscience.in/ijcct/vol5/iss1/8

Coding Schemes for Distributed Storage Systems: Implementation and Improvements

Saman Tabatabaeian, Rajendra P Lal, Wilson Naik
Department of Computer Science and Information Sciences,
University of Hyderabad, Hyderabad, AP, India
saman_tabatabaeian@yahoo.com, rplcs@uohyd.ernet.in, naikcs@uohyd.ernet.in

Abstract—Distributed data storage systems are used to store data reliably over a distributed collection of storage locations, called peers. Coding schemes are used to store a portion of the data in the peers ensuring the complete retrieval of data, during peer failures. This has applications in various areas like Wireless Networks, Sensor Networks etc. In this framework we consider a large file to be stored in a distributed manner over few peers of limited capacity. Each peer stores a portion of the coded data, without the knowledge of the contents of other peers. Random Coding is one of the coding schemes used for this. In [1] coding coefficients are chosen randomly from a finite field to encode the data. The encoding is basically a linear combination of file pieces (pieces are elements of finite fields). The data downloader downloads these coded data from several peers and decodes to get the original data. The decoding is basically solving a system of linear equations over a finite field, which is the most time consuming step in the whole process. We give a simple C++ implementation of the schemes in [1] and plot the results. We are trying to find a scheme where coding vectors can be chosen such that the decoding complexity is reduced significantly. Also in a dynamic setting where nodes enter and leave system intermittently, are discussed.

I. INTRODUCTION

The research upon which distributed systems field is growing mainly deals with rapid dissemination as well as an efficient storage of information. However, these two aspects are involved with constraints. For rapid dissemination we should take into that the bandwidth available is limited, and further, each node only has knowledge about its own contents. On the other hand, for efficient storage, the memory at any particular node is considerable. This means although the total memory of all nodes may be sufficient, the memory available at every node may be limited. This problem is well studied [1][2] [6].

In this section we will discuss the efficient distributed storage of a large file as explained in [1]. We explain the utility of coding based approaches in distributed storage. Two different types of schemes are discussed: Random Uncoded Storage (RUS) and Random Linear Coding based storage (RLC). The model used for distributed storage is very simple. We consider m messages, which are all fragments of a large file. These fragments are randomly distributed on n neighboring clients, called peers, with size k(messages or fragments). We assume that data is distributed randomly over the peers such that no peer has any knowledge about what the other peers have stored. A downloader randomly connects to $r(\leq n)$ out of n of peers to download the file fragments and constitute the entire file . The question here is: what the percentage that the downloader retrieves the original file would be? In RUS, for every peer, k fragments are chosen randomly and placed in the peer. The algorithm and implementation details are discussed in the next section.

In section III, we give a detailed interpretation of the RLC scheme for storing a large file in a distributed manner. In this scheme data is encoded using coding vectors over Galois field. Each encoded message is sent along with its coding vectors. The key difference that differentiates random uncoded storage and RLC is that in RLC, a linear combination of all file fragments is used which is a one time process. Whereas in random uncoded storage, at every communication instant, one message is randomly chosen by any transmitting node, in RLC, after the completion of download, an additional computation is also performed by the downloader to decode the encoded message using coding vectors which are sent along with it. This computation involves solving the Gaussian elimination of the unknowns over finite field.

II. RANDOM UNCODED STORAGE

A direct connection between client and server is a very usual scenario that systems encounter. However, when multiple clients simultaneously establish a connection to the server to download a large file, direct connections between clients and the server would increase the server traffic, especially when the number of users is significant.

In random uncoded approach smaller chunks of a large file are distributed randomly among the neighboring peers by the server. This is a simple scheme where k fragments out of m fragments are stored in each peer. Thus there are C(n, k) ways of storing data elements in a peer. Each client (downloader) connects to their neighboring peers and not directly to the server.

A. Implementation

In this program a large file is split into fragments of smaller size. Once the file is split, the file fragments are randomly distributed among r number of peers of size k. The distribution is random as no peer has knowledge of what is stored on the other peers.

```
for i = 1 to #peers_num
    do choose peer p randomly out of r total peers
```

Next, the downloader runs the client background process for each of the randomly chosen peers. The downloader program then will be waiting for each of the client programs to send file fragments by writing to its FIFO.

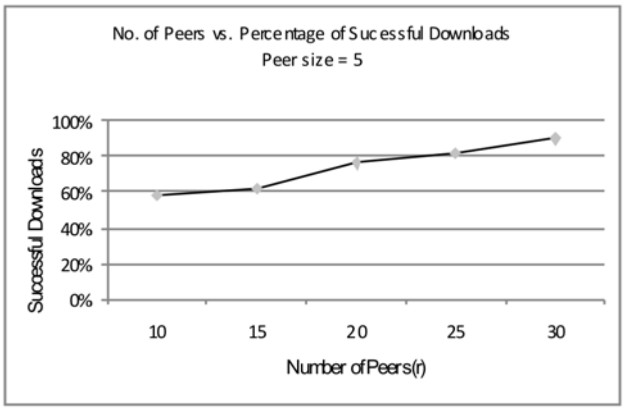
```
for peer_index = 1 to  #peers_to_connect
  do
      call the background client program for
      peer_index.
      open the Downloader FIFO for reading
      and writing
      for message_count = 1 to  #frags_to_send
         do
         extract the file fragment index i from
           buffer;
           create a file using fragment i and copy
           the buffer contents to it
         message_count++

  peer_index++
```

After the completion of the download, the downloader program calls a shell script to merge the file fragments and count if it already acquired all fragments.

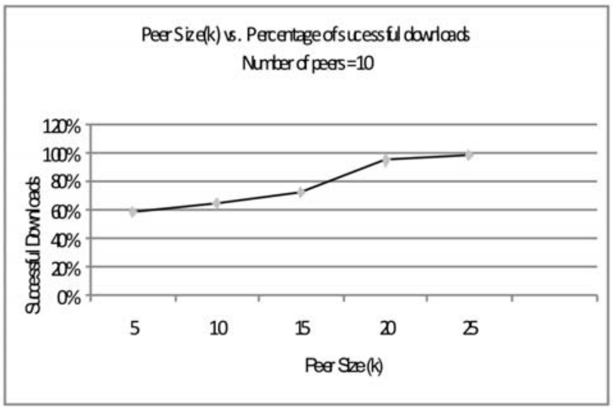
The Neighboring peers Client programs:

In each client program which acts the roll of each peer which keep some number of file fragments according to its size. The client program get the peer ID from the download program by whom it had been called as a background process. Next, it generates random fragment numbers out of the total number of fragments as each peer cannot store repetitive file parts.

```
for each random file fragment f generated by
        the client program
  do
   open the fragment and copy the contetns
   open the individual FIFO considered for that
  file.
   Add the fragment ID at the beginning of the
   buffer.
   write the fragment contents to the buffer
```



[Figure-1]



[Figure-2]

III. RANDOM LINEAR CODING APPROACH

A. Random Linear Coding

In this scheme, each packet sent to peers is a linear combination of file fragments encoded with an associated coding vector over the finite field. For every encoded message vector, the associated coding vector is also sent to peers. If we have m pieces, this will take an additional storage space of, where *k* is the peer size. This is typically a small number compared to each piece of the broken file. After the packets are downloaded, some computation needs to be performed on the downloader system. That is, it should solve a Gaussian elimination equation with n unknowns, where n is the size of file fragments. The arithmetic operations performed is all over the finite field. Once the computation completes, the downloader merges all decoded pieces to retrieve the original file.

It is shown in [1] that a successful download will be achieved by connecting to a number of peers close to “total pieces of the file” divided by “number of pieces each peer can store”. It is also noted that this approach takes advantage of the random uncoded storage in terms of rapid dissemination and efficient storage.

B. Encoding and Decoding Implementation Over Finite Field

In this scheme, the fragments of the files are vectors of size s in a field of size q i.e. ÎF(q). If the fragments are denoted by f_i then a typical element in a peer can be written as $c_i = \sum_{j=1}^{m} \beta_i f_i$ where $\beta_i \in F(q)$ with equal probability of being selected from the field. The vector $(\beta_1, \beta_2, .. \beta_m)$ is called the coding vector. These coding vectors would be stored in the peers along with the coded fragments. It's easy to check that the overhead of storing takes only 0.02% extra space. The downloader solves a system of linear equation in the finite field F(q). We implemented Gaussian elimination over finite field using logarithm and inversed logarithm tables[5] . These tables make arithmetic operations on finite field faster and supports computations for higher values of q(=32).

Multiplication = inversed log ((log a + log b)),
Division = inversed log ((log a – log b))
noting that addition and subtraction are simply the xor operation.

Pseudocode of Encoding:

```
range = 2 ^ W
for combination_count = 1 to peer_size
do
 for message_count = 1 to message_size
   do
    generate a unique value in range 2 ^ W
     coding_vector[message_count] = rand_val
   sum = sum ^
     (galois_ilog(galois_log(msg_vector[msg_cou
   nt], _W) +
    galois_log(coding_vector[msg_count], _W),
   _W))
   message_count++;

 encoded_msg_vector[combination_count] = sum
combination_count++
```

Psuedocode of Decoding:

```
for combination_count = 1 to peer_size
do
    download every encoded message along with
    its coding vector
    solve n equations with Gaussian Elimination
    over the Finite Field(2 ^ W)
   store decoded file fragments

merge file fragments to retrieve the original file
```

IV. CONCLUSION AND FUTURE WORK

The plots in fig-1 and 2, show that it requires a lot of fragments to download the whole file. This is due to the unequal distribution of fragments among the peers. Several ideas are explained in [2] to have equal distribution of fragments among the peers. Otherwise the download time will be more for the file. RLC coding scheme has very high probability of getting the whole file with no extra fragments being sent. But this is achived at the cost of encoding and decoding done at source and receiver end respectively. The implementation of the RLC is not complete as of now. We provided the pseudocode for the whole process. We are trying to apply simple matrix
theory techniques to reduce the decoding complexity. Also in a dynamic setting where peers leave and enter the system intermittently, [6] describes the idea for efficient coded fragment regeneration and also minimization of repair bandwidth. Implementation of the ideas in [6] would be incorporated in the future stages of this work

REFERENCES

[1] Supratim Deb, Clifford Choute, Murie Medard, and Ralf Koetter- *"Data Harvesting: A Random Coding Approach to Rapid Dissemination and Efficient Storage of Data"*INFOCOM 2005.

[2] Bit torrent file sharing protocol. http://bitconjurer.org/bittorrent

[3] Christina Fragouli, Jean-Yves Le Boudec, Jorg Widmer-*Network Coding*: *An instant Primer*.

[4] D. Qiu and R. Srikant. "*Modeling and performance analysis of bittorrent-like peer-to-peer networks*". Aug 2004.

[5] James S. Plank "*A Tutorial on Reed-Solomon Coding for Fault-Tolerance in RAID-like Systems*". Software — Practice & Experience, 27(9), 1997.

[6] Nihar B. Shah et al "*Regenerating Codes for Distributed Storage Networks*" LNCS, 2010, Volume 6087/2010, 215-223.