International Journal of Information, Control and Computer Sciences
ISSN: 2517-9942
Vol:1, No:10, 2007

String Searching in Dispersed Files using MDS Convolutional Codes

A. S. Poornima, R. Aparna, B. B. Amberker, and Prashant Koulgi

Abstract**—**In this paper, we propose use of convolutional codes for file dispersal. The proposed method is comparable in complexity to the information Dispersal Algorithm proposed by M.Rabin and for particular choices of (non-binary) convolutional codes, is almost as efficient as that algorithm in terms of controlling expansion in the total storage. Further, our proposed dispersal method allows string search.

Keywords**—**Convolutional codes, File dispersal, File reconstruction, Information Dispersal Algorithm, String search.

I. Introduction

LARGE databases becoming increasingly ubiquitous, their storage and retrieval, and string-searching in such databases, has received much attention. In [9] Rabin formulated the following problem: Suppose that, in order to guarantee retrieval in the face of server failures, you are willing to disperse the file across several servers. How do you do this efficiently? The naive solution, of course, is to place copies of the files at each of the servers. But this is quite inefficient, in leading to a great expansion in the total storage. In [9] Rabin also proposes an efficient solution for the problem, called the Information Dispersal Algorithm.

System designers have long tried to build more reliable storage systems. Techniques such as disk mirroring [2] and RAID (Redundant Arrays of Independent Disks) [5] have been used to improve system reliability. The other techniques for file dispersal and recovery are discussed in [6, 14, 3, 13] and Alvarez,et.al [1] developed DATUM, a method that can tolerate multiple failures by spreading reconstruction accesses uniformly over disks based on information dispersal as a coding technique.

Our interest is in the question of string search when a file is stored in a dispersed manner. (We naturally wish to avoid reconstructing the entire file from data retrieved from the different servers, and searching for the string in this reconstructed file.) The obvious approach is to disperse the string itself to the various servers and to consolidate their answers to determine if the string is present in the file. But previously proposed dispersal methods do not allow string search via this approach: In these methods the file is broken up into blocks and the block of data to be sent to each server is determined. Suppose if the string is present in the file, it may be within a block or across the block boundaries. If the string is within a block, using the same dispersal algorithm to disperse the string, as for the file, we can successfully search the string. Instead, if the string is across block boundaries, it is difficult to search for the string since the string may be broken into blocks differently from how the file was dispersed.

In view of this, we propose use of convolutional codes for file dispersal. Our proposal is based on the class of error-correcting codes called convolutional codes, which have been extensively studied in coding theory (see, for example [7, 8]). The proposed method is comparable in complexity to the Information Dispersal Algorithm of [9] and, for particular choices of (non-binary) convolutional codes, is almost as efficient as that algorithm in terms of controlling expansion in the total storage. Further our dispersal technique is also compatible with string searching.

Let n denote the total number of servers and, as in [9], let a threshold m ($m \leq n$) be specified such that at any point in time at least m servers are guaranteed to respond. With our algorithm the file is dispersed into n chunks of data, each of size = (size of file)/k (here k is a number $\leq m$); each of these chunks may be stored in one of the n servers. The file can be recovered if any m out of the n servers return their chunks.[1] A given string (we impose no restriction on its minimum or maximum size) will also be passed through the algorithm, and the n chunks obtained will be passed to the servers, each chunk to its corresponding server. If at least m out of the n servers finds the string chunks in the same position in the chunks of data with them, the string is present in the file, otherwise not.

For some values on m and n (this corresponds to the class of MDS convolutional codes, which have been previously studied e.g. in [11, 10, 12]) it is possible to get $k = m - 1$. Thus in these situations the file can be recovered by retrieving chunks of data whose combined size is just more than the size

A. S. Poornima is working as an Assistant Professor in the Department of Computer Science and Engineering in Siddaganga Institute of Technology, Tumkur, Karnataka, India (e-mail: aspoornima@sit.ac.in).

R. Aparna is working as an Assistant Professor in the Department of Computer Science and Engineering in Siddaganga Institute of Technology, Tumkur, Karnataka, India (e-mail: raparna@sit.ac.in).

B. B. Amberker is working as a Professor in the Department of Computer Science and Engineering in National Institute of Technology, Warangal, Andhra Pradesh, India (e-mail: bba@nit.ac.in).

of the file itself by a factor of m/(m − 1). By comparison, if the Information Dispersal Algorithm of [9] is used, this combined size can be the same as the size of the file itself (this, of course, is the best possible); but remember that string search is not possible in this case.

In the next section, we briefly describe a version of the algorithm of [9], and interpret it in the language of error-correcting codes. This allows us to then lead up in Section 3, to our dispersal algorithm in a transparent manner. After providing a brief introduction to convolutional codes sufficient for our purpose, we describe our algorithm in some detail, illustrating it with a couple of examples. And then we analyze the complexity of our algorithm, and relate this to the implementation of the algorithm of [9].

II. Information Dispersal Algorithm: Basic Version

Throughout, the file shall be denoted as $F_1F_2F_{3.} . . F_N$, and a string as $q_1q_2q_{3}. . . q_q$. While, for a given choice of m and n, different versions of the Information Dispersal Algorithm are obtained by varying the choice of an associated matrix, we shall present here a choice corresponding to a particularly easy-to-understand version. Here the file is blocked (i.e., its contents are relabeled) as

F1,1F1,2F1,3 . . . F1,mF2,1F2,2F2,3 . . . F2,m . . . F(N/m)−1,1F(N/m)−1,2… F(N/m)-1,m

For each i =1, . . . , (N/m)−1, the block Fi1Fi2Fi3 . . . F_{im} is treated as representing the polynomial

$$F_i(x) = Fi1 + Fi2x + Fi3x^2+ . . . + F_{im}x^{m-1}$$

and the n evaluations of this polynomial $F_i(j)$ for j = 1, . . . , n are sent, one to each of the servers.

For reconstruction of the files, if m of the servers respond, for each i, i = 1, . . . (N/m) − 1, m evaluations of the (unknown) polynomial $F_i(x)$ (which, however, is known to be of degree = m − 1)are available, and these can be employed in Lagrange interpolation to reconstruct the polynomial.

Notice that the data stored at server j, j=1, . . . , n, is therefore $F_1(j)\ F_2(j) . . . F_{(N/m-1)}\ (j)$
Suppose that the given string is present in the file, in fact as

$q_1q_2q_3 . . . q_q$ = Fi1,1Fi,1,2 . . . Fi1,mFi,2,1 Fi2,2 … Fi,2,m

Then the string can be declared as present in the file by searching server j, j = 1, . . . , n, with the string $F_1(j)… F_2(j)$ and checking that at least m out of the n servers respond saying that they found the strings sent to them between the positions $i_1$and i_2, in the chunks. But suppose that the string does not thus happen to fit correctly at the block boundaries but falls within or between these boundaries. The reader may appreciate that then it is not clear how the string should be transformed in order to convert the problem of search for this string in the file to search for some related strings in the chunks of data with the servers. The same problem with searching for strings persists when other proposed methods of dispersal are used. For these too are in the same manner based on breaking the file into blocks.

Further, all these previously proposed dispersal methods may be considered as applications of block error-correcting codes. In the above algorithm the transformation Fi1,1Fi,1,2 . . . Fi1,m → $F_i(1)F_i(2) . . . F_i(n)$ is simply the Reed-Solomon encoding of the information word Fi1,1Fi,1,2 . . . Fi1,m $_{im}$as the codeword $F_i(1)F_i(2) . . . F_i(n)$. Any other error-correcting code could also be used instead.

Previously-proposed dispersal methods can be subsumed as the following approach: For a given m, n choose a block code with codeword size = n, and with every two code words differing in at least n − m + 1 positions (i.e., with minimum Hamming distance = n − m + 1). Suppose that, for this code the size of an information word is k.(The singleton bound implies that for any block code k < m. Codes for which equality holds form the special class of MDS codes. Reed-Solomon codes are an example of such codes. For more details see [4].) Then break the file into blocks of size k. Treating each such block as an information word encode the block as its corresponding codeword (of size n), and send one element to each of the n servers. At reconstruction, receiving responses from m of the servers gives values at m positions in each codeword (which, recall, is of size n). Since every two code words are known to differ in at least n − m + 1 positions, there can be only one codeword with these values at these m positions, and the information word encoding this codeword is the corresponding block in the file.

III. Our Convolutional Code Based Dispersal Algorithm

We use essentially the same approach as in the previous section; but we dispense with block codes and use convolutional codes instead. The encoder of a convolutional code is basically a Finite state machine, and can thus be implemented using shift registers, adders and multipliers (we will only need linear convolutional codes, which are linear finite state machines). The output of the state machine is a function of the input, and the contents of the shift registers. The state is given by the shift register contents; at each instant the input is sequentially shifted into the shift registers, and thus determines the next state.

In a (k, n) convolutional code the size of the encoded stream is n/k times the size of the input stream. To fix ideas we will consider the example of a particularly simple convolutional code, a (1, 4) code whose linear shift register representation is given in Fig. 1. Here D represents delay element.

The encoding process may be represented in a trellis diagram, which itself is derived from the shift register representation of Fig. 1. The trellis diagram graphically displays the transitions to new states and outputs obtained for different input streams. Thus the start and termination of a branch indicate the initial and final state respectively, the input (and output) producing this transition being labeled above the branch itself. A path through the trellis diagram identifies an output sequence, the input sequence (and initial state) which produced it, and the corresponding state sequence. See Fig. 2. Our idea is to feed the file as input to the convolutional encoder. With a (k, n) encoder each of the n values obtained at the output is sent to one of the n servers. (Notice, therefore, that the size of data received at each server is 1/ktimes the size of the file). Thus if the encoder of Fig. 1 is used in a situation where there are n = 4 servers, the stream c_i is sent to server i (i = 1, . . . , 4). (At the outset the shift registers may arbitrarily be initialized to zeros).

Consider first the question of reconstruction of the entire file. Suppose that only three out of the four servers respond with the streams with them (thus we have the situation m = 3). The received information should nevertheless unambiguously determine a single path in the trellis diagram: in this case the input sequence corresponding to this path may be declared as the reconstructed file.

Now notice this property of the trellis diagram (in Fig. 2) for the convolutional encoder we have chosen in Fig. 1: the output labels of every two branches differ in at least two positions. The case when some three servers respond corresponds to knowing three of the four positions on every branch of the path encoding the file. Therefore from the above property the responses of three servers is sufficient to determine a path in the trellis diagram, and the file can be declared to be the corresponding input sequence. (The reader can convince himself by checking, say that the responses of the servers 1, 2 and 4 are sufficient to reconstruct the file chosen as an illustration in Fig. 2.)

Let us now move on to the question of string search, of the string $q_1q_2q_3 . . . q_q$ in the file $F_1F_2F_3 . . . F_N$. The string should be encoded using the same convolutional encoder and each of the n resulting streams should be sent to its corresponding server. But how is the initial state to begin this encoding to be determined? (For remember that the location of the string in the file is also unknown, and it may therefore be a mistake to assume the initial shift register contents to say, all be zeroes.) We circumvent this problem by noticing that, if we give up the encoding of the part of the string made up of the first few values of the string, for the encoding of the subsequent part the preceding part itself determines the contents of the shift registers (and no further information is required to determine these contents. Thus in the convolutional encoder we have considered as an example, $q_1q_2$is the initial state for encoding the sequence $q_1q_2q_3 . . . q_q$). Only the streams corresponding to the encoding of this subsequent part is sent to the servers.

Given n the total number of servers, and m, the number of servers guaranteed to respond, how is the convolutional encoder to be chosen? Suppose that the string is encoded (in the manner described above), and some m of the n servers respond. (Recall that the response of a server consists of whether the stream sent to it is present in the stream with it and, if yes, the position of the match.) Consider the case when all these m servers find the streams they receive, and at the same location. We want to then be able to conclude that the string itself is present in the file.

The encoded string sequence had determined a sub-path in the trellis diagram. The string being present in the file corresponds to this sub-path being a part of the path encoding the file. Since the m servers have found the streams sent to them, and all in the same location, one possibility is that the sub-path encoding the string is a part of the path encoding the file, therefore that the string is present in the file (at the location specified by the servers). We need to be sure that the information returned by the m servers does not also permit a different sub-path to be concluded as being part of the path encoding the file.

Since the servers have found matches at a particular location, for the sequence sent to them, on each branch of every candidate sub-path (of certain length) starting at this location the values at m positions (out of the total of n) are known. We need to be sure that this information permits only the subpath encoding the string. This will indeed be the case if every two branches in the trellis diagram differ in at least n − m + 1 positions (out of the total of n).

Let us summarize: For a given m, n, if a (k, n) convolutional encoder is chosen in whose trellis diagram every two branches differ in at least n − m + 1 positions, and is used to encode and disperse the file (as outlined above), then the string is present in the file if m out of the n servers find matches (at the same location in the streams with them) for the sequences sent to them, and is not present in the file otherwise.

IV. Some Implementation Issues

If a particular (k, n) convolutional encoder, satisfy a given requirement ofn (the total number of servers) and m (the minimum number among these which can be relied upon to respond), is chosen, then queries (of any bigger length and) as small as a string of k elements can be searched for in our approach. That is to say, the question " is $q_1q_2q_3 . . . q_q$ in $F_1F_2F_3 . . . F_N$?" can be answered for strings with $Q \geq k$. $Q \leq k$ are not permitted since the encoding linear shift register takes an input of k elements.

Fix m and n. Then every possible (k, n) convolutional encoder satisfies $k \leq m$ (due to a form of the singleton bound which holds for convolutional codes; see [7]). We would like to choose the encoder with the largest possible k, since this would ensure the smallest possible size for the data sent to each server (which, we call, is (file size)/k), and the smallest possible size of data required to reconstruct the file (which, when m servers respond, is m/k*file size). The best encoder will turn out quite often not to be defined over the binary alphabet but over some extension field $GF(2^q)$ (i.e., the inputs, outputs and register contents of the linear shift registers implementing this encoder are elements of $GF(2^q)$) for some q > 1. For instance when n = 5 and m = 3 the largest possible k is 2, as shown in [11] and this (k, n) = (2, 5) encoder is defined over $GF(2^4)$.

In Table I, we summarize the maximum number of operations needed to disperse a file, $F_1F_2F_3 . . . F_N$ using the best possible convolutional encoder for a given m and n. We also compare this number with the number of operations required if the Information Dispersal Algorithm of [9] were used.

REFERENCES

[1] G.A.Alvarez, W.A.Burkhard and F.Cristian. Tolerating multiple failures in RAID architectures with optimal storage and uniform declustering. In proceedings of the 24th International Symposium on Computer Architecture,pages 62-72, Denever,CO,ACM. June 1997.
[2] D.Bitton and J.Gray. Disk Shadowing. In proceedings of the 14th conference on Very Large Databases (VLDB), pages 331-338, 1988.
[3] W.A.Burkhard and J.Menon. Disk array storage system reliability. In Proceedings of the 23rd International Symposium on Fault-Tolerant Computing (FTCS'93), Pages 432-441, June 1993.
[4] W. Cary Huffman and Vera Plees Fundamentals of Error-Correcting Codes Cambridge University Press, 2003.
[5] P.M.Chen, E.K.Lee, G.A.Gibson, R.H.Kartz, and D.A.Patterson. RAID: High-Performance, reliable secondary storage. ACM Computing Surveys,26(2), June 1994.
[6] Gui-Lang Feng, Robert H. Deng, Feng Bao, JiaChen Shen, New Efficient MDS Array Codes for RAID part II: Rabin-Like codes for Tolerating Multiple (greater than or equal to 4) Disk Failures, IEEE Transactions on Computers,V.54 n.12,p.1473-1483, December 2005.
[7] R. Johannesson and K. Sh. Zigangirov , Fundamentals of Convolutional Coding, University Press (India) Limited, 2001.
[8] R. Johannesson and K. S. Zigangirov Fundamentals of Convolutional Coding IEEE Press, Newyork, 1999.
[9] M. Rabin. Effecient Dispersal of Information for Security, Load Balancing, and Fault Tolerance. Journal of the ACM, 36(2): 335-348, 1989.
[10] J. Rosenthal and R. Smarandache. Maximum Distance Separable Convolutional codes. Appl. Algebra Engrg. Comm.Comput., 10(1): 15-32,1999.

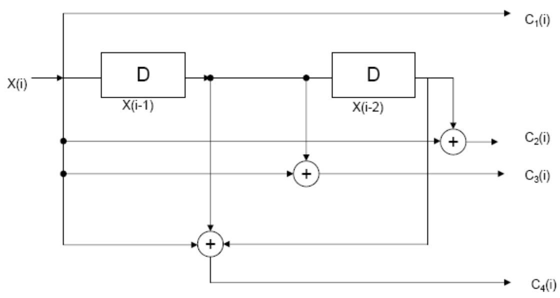

Fig. 1 The Shift Register representation of the Encoder of a (1,4) binary convolutional code. Here D represents a delay element. X(i) is the current input bit. C_1(i), C_2(i), C_3(i), and C_4(i) are the corresponding output bits

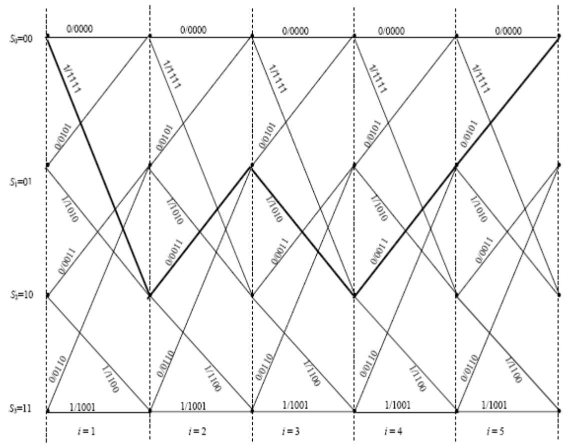

Fig. 2 Trellis diagram of the encoder of the (1, 4) binary convolutional code of Fig. 1. S_0=00, S_1=01, S_2=10, S_3=11 are the states of finite state machine of Fig. 1. Lines represent transitions from one state to the next state in the next interval. Labels on each line in interval i=1 represent the output bits c_1(i), c_2(i), c_3(i), c_4(i) respectively. Output bits at each subsequent levels remain the same. Bold lines represent the path for the input file 10100. The corresponding output sequence is 1111 0011 1010 0011 0101. Suppose at reconstruction, three servers, say 1, 2, and 4 respond with the sequence 11_1 00_110_0 00_1 01_1. It is then possible to reconstruct the correct sequence.

A. S. Poornima obtained her M. Tech. from VTU, Belgaum, Karnataka, India. She is presently working as an Assistant Professor in the Department of Computer Science and Engineering Siddaganga Institute of Technology, Tumkur, Karnataka, India and pursuing PhD in the area of Cryptography and Network Security.

R. Aparna obtained her M.S. from Birla Institute of Technology, Pilani, Rajasthan, India. She is presently working as an Assistant Professor in the Department of Computer Science and Engineering, Siddaganga Institute of Technology, Tumkur, Karnataka, India and pursuing PhD in the area of Cryptography and Network Security.

B. B. Amberker obtained his PhD from Department of Computer Science and Automation, IISc., Bangalore, India. He is presently working as Professor in the Department of Computer Science and Engineering, National Institute of Technology, Warangal, AP, India.