A Cost Effective Fault-Tolerant Scheme for RAIDs

FANG Liang (方 粮) and LU XiCheng (卢锡城)

School of Computer, National University of Defense Technology, Changsha 410073, P.R. China
E-mail: lfang@nudt.edu.cn
Received June 8, 2001; revised June 8, 2002.

Abstract The rapid progress in mass storage technology has made it possible for designers to implement large data storage systems for a variety of applications. One of the efficient ways to build large storage systems is to use RAIDs as basic storage modules. In general, the data can be recovered in RAIDs only when one error occurs. But in large RAIDs systems, the fault probability will increase when the number of disks increases, and the use of disks with big storage capacity will cause the recovering time to prolong, thus the probability of the second disk’s fault will increase. Therefore, it is necessary to develop methods to recover data when two or more errors have occurred. In this paper, a fault tolerant scheme is proposed based on extended Reed-Solomon code, a recovery procedure is designed to correct up to two errors which is implemented by software and hardware together, and the scheme is verified by computer simulation. In this scheme, only two redundant disks are used to recover up to two disks’ fault. The encoding and decoding methods, and the implementation based on software and hardware are described. The application of the scheme in software RAIDs that are built in cluster computers are also described. Compared with the existing methods such as EVENODD and DH, the proposed scheme has distinct improvement in implementation and redundancy.

Keywords RAID, fault-tolerant technology, data availability

1 Introduction

Although performance of disk has been improved rapidly these years, it cannot match the processing speeds of processors because disks are mechanical devices. Nevertheless, when we organize disks into RAIDs (Redundant Arrays of Independent Disks), the total performance can be increased through concurrent accesses to data on different disks[1]. RAIDs provide the data throughput that is required to balance the increasing performance of processors[2]. Therefore, RAIDs have been used more and more widely. RAIDs provide an efficient stable storage system for parallel access and fault tolerance[3].

An optimal storage system should provide both high performance and high reliability[4]. A RAID with a group of disks is able to improve I/O performance by accessing several disks in parallel. To achieve more performance improvement in disk arrays, many researches have been done such as the optimal stripe unit size, data striping and parity placement, and data prefetching and catching[5]. However, the reliability problem should be more seriously taken into account than I/O rate performance, because even a single disk failure may cause a catastrophic result to the overall disk array system. In order to increase the reliability of the system, some redundant information, called parities, on the data in a disk array should be encoded. And the parities should be placed in the disk array along with the data for recoveries from disk failures.

But in large RAIDs systems, because: 1) the fault probability will increase when the number of disks increases; and 2) the use of disks with big storage capacity will cause the recovering time to prolong, thus the probability of the second disk’s fault will increase; therefore, it is necessary to develop methods to recover data when two or more errors have occurred[5,6,7]. In other words, it is required to provide higher data availability than usual RAIDs provide. Generally, data on RAIDs can be recovered if no more than one error has occurred. If it is required to correct two errors, the cost penalty is very high by usual methods. As a result, a lot of interest has arisen in attempting to design effective fault tolerant schemes that will not lose data even when multiple disks fail simultaneously. So, the use of erasure-correcting codes with higher correcting capability than simple parity is suggested (in coding theory terminology, an erasure is an error whose location is known)[8,9].

This research is supported by the National Natural Science Foundation of China (No.69933030).
In this paper, we propose a fault tolerant scheme based on extended Reed-Solomon code, design a recovery procedure to correct up to two errors, which is implemented by software and hardware together, and verify the scheme by computer simulation. In this scheme, only two redundant disks are used to recover up to two disks’ fault. We describe the encoding and decoding methods, and the implementation based on software and hardware. We also describe the application of the scheme in software RAIDs that are built in cluster computers. Compared with the existing methods, such as EVENODD\cite{10} and DH\cite{7}, our scheme has distinct improvement in implementation and redundancy.

## 2 Cost Effective Fault-Tolerant Scheme for RAIDs Based on Extended Reed-Solomon Code

Suppose there are \( k + 2 \) disks in the RAID system, in which \( k \) disks are used to store data and 2 disks are used to store redundant information. We note the data elements on data disks as \( X_0, X_1, \ldots, X_{k-1} \), and check elements on checker disks as \( C_0, C_1 \). When data are written to disks, the operational results of \( X_0, X_1, \ldots, X_{k-1} \) on the encoding/decoding logic form checker elements \( C_0, C_1 \). \( C_0 \) and \( C_1 \) are written on disks at the same time with data elements. When data are read out from disks, the encoding/decoding logic generates Syndromes \( S_0, S_1 \). Single error can be corrected by the error-correcting logic. Double errors can be corrected by the microprocessor with the support of the hardware logic in the error-correcting logic.

Because the disks have their own ECC (Error Correcting Code), so at the level of the RAID controller, the main task is to recover the data when errors cannot be corrected by the disk ECC or when a whole disk cannot be correctly read out.

### 2.1 Coding Equations

Assume code \( C \) has \( m \) elements, and its format looks like

\[
C = (X_0 X_1 \ldots X_{k-1} C_0 C_1)
\]

where \( X_i \) is a data element and \( X_i \in GF(2^m) \). Each element can be expressed as a column vector of \( m \) bits of binary. \( C_0 \) and \( C_1 \) are check element, \( C_0, C_1 \in GF(2^m) \). The check equation is

\[
C_0 = \sum_{i=0}^{k-1} X_i \tag{1}
\]

\[
C_1 = \sum_{i=0}^{k-1} \alpha^i X_i \tag{2}
\]

where \( \alpha \) is the primary element over \( GF(2^m) \), the factor of \( \alpha \) is \( 2^m - 1 \), the number of information elements can be variable, and the maximum is \( 2^m - 1 \). The symbol \( \leftrightarrow \) means XOR operation. The check matrix is as follows.

\[
H = \begin{bmatrix}
1 & 1 & 1 & \cdots & 1 & 1 & 0 \\
1 & \alpha & \alpha^2 & \cdots & \alpha^{k-1} & 0 & 1
\end{bmatrix}
\]

Obviously, the matrix \( H \) is nonsingular, its rank \( r = 2 \), the minimum Hamming distance \( d = 3 \), so it can correct an element of burst error or two elements of erasures.

### 2.2 Encoding Method

According to the encoding equation \( HC^T = 0 \), we can generate checkers \( C_0, C_1 \). Select a primary polynomial over \( GF(2) \),

\[
p(x) = x^m + g_{m-1}x^{m-1} + g_{m-2}x^{m-2} + \cdots + g_1x + 1
\]

The companion matrix of \( p(x) \) is

\[
T = \begin{bmatrix}
0 & 0 & 0 & \cdots & 0 & 1 \\
1 & 0 & 0 & \cdots & 0 & g_1 \\
0 & 1 & 0 & \cdots & 0 & g_2 \\
\vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\
0 & 0 & 0 & \cdots & 1 & g_{m-1}
\end{bmatrix}
\]

Because the elements over \( GF(2^m) \) can be expressed as \( m \) bits vectors or \( m \times m \) matrix \( T \), so \( T^i \) and \( \alpha^i \) are one-to-one correspondence, (1) and (2) can be transformed into

\[
C_0 = \sum_{i=0}^{k-1} X_i \tag{3}
\]

\[
C_1 = \sum_{i=0}^{k-1} T^i X_i \tag{4}
\]

According to the equations above, we can design the encoding logic circuits.

### 2.3 Decoding Method

#### 2.3.1 Calculating the Syndrome

Assume the code vector received is \( \hat{C} = [X_0^\wedge X_1^\wedge \cdots X_{k-1}^\wedge C_0^\wedge C_1^\wedge] \), then we have \( \hat{C} = C + C^+ \)