SCAN: An Efficient Sector Failure Recovery Algorithm for RAID-6 Codes

Jianqiang Luo, Lihao Xu
Department of Computer Science
Wayne State University
Detroit, MI 48202
jianqiang@wayne.edu, lihao@cs.wayne.edu

Abstract

Recent studies show disks fail much more often in real systems than specified in their data-sheets and RAID-5 may not be able to provide needed reliability for practical systems. It is desirable to have disk arrays and clustered storage systems with higher data redundancy, such as RAID-6. Meanwhile, latest research also indicates disk sector failures occur much more often than whole disk failures. It is hence important to study efficient disk sector failure recovery schemes for RAID-6.

This paper studies how to efficiently recover disk sector failures for RAID-6 codes. First for most well known RAID-6 codes, we provide the conditions to determine the recoverability of their erasure patterns. Then a generic erasure decoding algorithm SCAN is derived from these conditions. SCAN is able to recover both sector and whole disk failures, like Matrix Method, but SCAN is much more efficient in time complexity. Extensive simulations and analysis show SCAN outperforms Matrix Method in all the relevant decoding performance metrics. Therefore, SCAN is an attractive decoding algorithm to be integrated into these systems using RAID-6 codes, like disk arrays or clustered storage systems.

1 Introduction

Reliable storage systems are essential for most data-related applications. Yet virtually every component in a storage system can fail, from disks to various cables. Most recent studies show disks, the core component of storage systems, fail much more often in real systems than specified in their data-sheets [21, 16]. Inevitably, data redundancy needs to be introduced to ensure intact recovery of lost data caused by disk failures. A quite common practice is to use mirroring, i.e., maintaining multiple copies of a data on different disks or storage nodes. As storage systems expand and use more and more commodity components, it is not uncommon nowadays to employ triplication or more in storage systems, e.g., the Google File System [9]. It is arguable, however, if such a practice uses system resources most effectively and efficiently, since it is not just the triple number of disks that are needed, but also other expensive resources, including network components, power and physical spaces to accommodate those equipments, let alone more man power to manage and maintain them.

On the other hand, RAID-5 [7], another commonly adopted data redundancy architecture that uses one parity disk/node to tolerate one disk/node failure, may not have high enough reliability in real systems [21]. As observed in [21], disks fail in a more correlated fashion, thus theoretical reliability of RAID-5 derived on the assumption that disks fail independently may be very far from reality. Thus a natural extension to provide more data reliability is to use more redundancy, and RAID-6 which employs two parity disks/nodes is a good starting point. Here RAID-6 is used as a general data redundancy architecture, not necessary limited to disk arrays with a RAID controller: the same data placement schemes can certainly extend to clustered or distributed storage systems. In fact, this is true for any RAID-n architecture. For discussion simplicity, the term RAID will be used throughout this paper, but by no means any results hereafter are limited to disk arrays.

Most existing decoders for RAID-6 adopt a stop-and-fail error model for a disk. Under this model, a disk either functions normally, i.e., all data stored on it is accessible; or fails totally, i.e., none of its stored data is accessible. In a practical storage system, failures are certainly more complex. In general, some disk sectors or blocks can fail or corrupt due to various reasons while the rest of the disk still functions. This is the so-called disk sector failure and [19] has a very good discussion on various causes of disk sector/block failures. Very recent study shows that actual disk sector failures occur much more frequently than a whole disk failure in real storage systems [1]. As modern disks exponentially increase their storage capacity by increasing their areal density, it would be anticipated more sector fail-
ures can occur. Even though the so-called disk *scrubbing* technique that runs a background process which proactively scans disk sectors and masks failed ones, can reduce the possibility of writing data to failed sectors [22], for the data already stored on a later failed sector, the sector failure has the same effect as a whole disk failure or even worse since many systems do not replace a disk until it fails totally [16], and random sector failures make it impossible to store important data on “more” reliable sectors. So to ensure reliability of all the data in a system, recovery schemes for sector failures are very much needed.

The main contributions of this paper are both theoretical and practical. Theoretically, a necessary and sufficient condition for recoverable sector failures is derived, that can be applied to many well known codes, such as the EVEN-ODD code [3], the X-code [24], and the WEAVER code [11]. Practically, a *universal* decoder that can efficiently correct both sector failures and whole disk failures is designed. An implementation of the decoder for the EVEN-ODD code is evaluated in both computation complexity and failure recovery performance. (Though there are some published real disk sector failure data [1], they are not sufficient to derive a model characterizing sector failures. Our evaluation thus is based on reasonable simulations. We certainly encourage disk, array vendors and large scale storage system owners/operators, to provide more detailed raw disk sector failure statistics to further examine the efficiency and effectiveness of our universal decoder which can be potentially used for both commercial disk array products and clustered storage systems.)

This paper is organized as follows: Sec.2 introduces the background of RAID-6 codes and sector failure recovery. Sec.3 provides a brief description of the EVENODD code and the definition of low-message-degree (*LMD*) code. In Sec.4, a necessary and sufficient condition to determine the recoverability of any erasure pattern is derived for *LMD* code and the EVENODD code. Sec.5 presents an efficient sector failure decoding algorithm, called *SCAN*, for *LMD* code. Meantime, the patch of *SCAN* is provided for the EVENODD code. Sec.6 compares the decoding performance of *SCAN* against *Matrix Method* on the EVENODD code. Some further discussions are conducted in Sec.7 and Sec.8 concludes the paper.

## 2 Background

In a RAID architecture, the key is how to derive parity data from user data and place them properly among disks/nodes within the system. A common way is to employ *error correcting codes*, especially the so-called MDS (*Maximum Distance Separable*) codes [15]. Using a proper \((n, k)\) MDS code, an \(n\)-block coded data is computed from a \(k\)-block user data through an *encoding* process. (A block here can be any proper unit.) The MDS property ensures that any \(k\) coded blocks can recover the original \(k\)-block user data through a proper decoding operation [15]. Other non-MDS codes, such as various LDPC (*Low Density Parity Check*) codes [14], are suggested for storage applications too. But they are not optimal in storage space use [18].

The Reed-Solomon code is the most well known MDS code [20] and widely applied in communications and storage systems. It can support general \((n, k)\) data placements for storage applications. Its major drawback, which prevents it from being widely used for RAID systems, is its very expensive computation cost in both encoding (corresponding to *write*) and decoding (i.e., *read*) operations, even with various implementation improvements [5, 17], as observed by many system practitioners, e.g., in [18].

This paper focuses on how to recover disk sector failures (erasures) for RAID-6 array codes. The so-called *array codes* employ mainly only XORs (binary exclusive OR) for both encoding and decoding, thus can perform much faster (by at least one order) write and read operations than the Reed-Solomon code [18]. Existing array codes that can be applied to RAID-6 mainly include the ZZS code [26], the BCP code [2], the EVENODD code [3], the X-code [24], the B-code [25] and the RDP code [8], which are all MDS codes; and the LSI code [23] and the WEAVER code [11], which are not MDS codes.

Disk sector failures can be handled at various levels: disk controller, file system and array or clustered system. A disk controller usually has built-in error correcting code that encodes user data before writing to the medium and performs error recovery for inaccessible data whenever possible. The *IRON* file system deals with sector/block errors at file system level for a *single disk* [19]. For an array or clustered architecture such as RAIDX-6, sector failures can further be handled using the inherent data redundancy within the architecture. All the mechanisms, including the disk scrubbing technique, are important and complement with each other.

It is easy to argue that the redundant data already built in a RAID-6 system should be used to its maximum to correct more (sector) erasures in addition to just two disk failures. Figure 1 illustrates how much the intrinsic sector erasure correcting capability would be wasted if a RAID-6 is used only for correcting two whole disk failures. Figure 1 uses the well known EVENODD code for a 19-disk RAID-6 system (i.e., \(p = 17\) and this is chosen arbitrarily just for illustration purpose), and disk sectors simply fail independently with a same random probability of \(u\). (Disk sector failure modeling will be further discussed later in Sec. 6.) For a RAID-6 system, any sector erasures contained within two disks are always recoverable as designed, but sector failures on more than two disks are not recovered by a standard
A RAID-6 controller using an erasure correcting decoder that only corrects whole disk failures. Thus a lot sector erasures can be corrected if a better decoder is used. For example, when \( u = 1\% \), more than twice as many sector erases can be recovered if the decoder goes beyond just correcting whole disk failures.

![Figure 1. Erasure Pattern Composition of A 19-Disk RAID-6 Using the EVENODD code](image)

In theory, recovering sector failures (erasures) is not hard, since it is equivalent to solving a group of linear equations with linear erasure correcting codes [12]. The key, however, is the computation complexity. Solving \( m \) unknowns from \( m \) linearly independent equations in general requires \( O(m^3) \) operations, which are too high for many applications with frequent read (i.e., decoding) operations. In this study, we seek to perform sector erasure recovery with much less computation, and better yet our decoding algorithm stops without performing unnecessary computation when it sees the erasures are irrecoverable. This of course further reduces decoding computation complexity.

3 Basics

In this paper, we use the EVENODD code [3] as an example for RAID-6 in both theorems proof and performance simulation, since 1) it is a widely known MDS array code for RAID-6; and 2) our sector erasure decoder for any other RAID-6 codes is a subset of that for the EVENODD code.

3.1 The EVENODD Code

Now we briefly describe the construction of EVENODD. EVENODD uses 2 check (parity) columns together with \( p \) message (user data) columns, where \( p \) is a prime number in order for the code to be MDS. In practice, EVENODD can support any number \( n \) of disks using a simple technique called shortening, which chooses a bigger \( p \) and sets any \((p - n)\) columns to be zeros and then remove them after encoding [3]. For simplicity, we assume a RAID-6 happens to have \( p \) message (user data) disks. This assumption does not in any way affect any derived results hereafter.

The encoding process takes a \((p - 1) \times (p + 2)\) array, where the first \( p \) columns are information columns and the last 2 are check columns. Notation \( a_{i,j} \) \((0 \leq i \leq p - 2, 0 \leq j \leq p + 1)\) represents a symbol in the position of row \( i \) and column \( j \). A check symbol in column \( p \), i.e., the 1st check column, is computed as the XOR sum of all message symbols in the same row. This check column is called the horizontal check column in this paper. The computation of column \((p+1)\), i.e., the 2nd check column, takes the following steps: First, the array is augmented with an imaginary row \( p - 1 \), where all the symbols are assigned zero values. The XOR sum of all message symbols along the same diagonal (indeed a diagonal of slope 1) is then computed and assigned to their corresponding check symbol, as marked by different shapes in Figure 2 for \( p = 5 \). Symbol \( a_{p-1,p+1} \) is now non-zero in general and it is called EVENODD adjuster. To remove this symbol from the array, adjuster complement is performed, which adds (XOR addition) the adjuster to all symbols in column \( p + 1 \). We call this check column, i.e., column \( p + 1 \), the diagonal check column.

The encoding operation can be algebraically described as follows \((0 \leq i \leq p - 2)\):

\[
a_{i,p} = \bigoplus_{j=0}^{p-1} a_{i,j}
\]

\[
a_{i,p+1} = S_1 \oplus \bigoplus_{j=0}^{p-1} a_{(i-j),p+1},
\]

where \( S_1 = \bigoplus_{j=0}^{p-1} a_{(p-1-j),p+1} \).

Here, \( S_1 \) is EVENODD adjuster, \( \langle x \rangle_p \) denotes \( x \mod p \), and \( \oplus \) is the bitwise XOR operation. Again see [3] for more details.

3.2 Erasure Pattern

Using error correcting coding terminology, a coded block is called a codeword. A disk sector failure is called a codeword symbol erasure, since the physical failure location is known. For an array code, such as EVENODD, a column of symbols corresponds to a disk/node that stores all the corresponding message or check symbols. An erasure pattern for a codeword identifies the set of symbols erasure in this codeword. Then we have Lemma 1 for the recoverability of an erasure pattern.
Lemma 1. If all symbol erasures in an erasure pattern can be uniquely recovered, then this erasure pattern is recoverable.

If an erasure pattern $B$ contains all symbol erasures in erasure pattern $A$, then $B$ is called a superset of $A$. It is easy to see that an erasure pattern and any of its supersets have the following relation:

Lemma 2. If an erasure pattern is not recoverable, then any of its supersets is not recoverable.

Lemma 2 implies that if an erasure pattern has an irrecoverable subset, then it is also irrecoverable. Though simple, this lemma does give a necessary condition for a recoverable erasure pattern.

For rest of this paper, we focus on RAID-6 codes, i.e., the ones with column distance 3. If a code has a column distance of more than 3, we simply uses its derived code with column distance of exact 3. Hereafter codes simply refers to the RAID-6 codes.

3.3 Bipartite Graph Representation

Array code can be regarded as LDPC code at the symbol level and thus described by a usual (undirected) bipartite graph [4]. For a codeword, a bipartite graph for its erasure pattern contains two disjoint sets of nodes: the nodes for message symbol erasures and the nodes for known check symbols, which are connected via lines according to the check constraints. The check constraints can be simply derived from the code’s parity check matrix. Another way is from a code’s encoding rules, but not easy for some codes.

Next we define LMD codes, a class of RAID-6 code which can be decoded much more efficiently for erasures than traditional matrix based decoding algorithms.

Definition 1. Given a RAID-6 code, for an arbitrary erasure pattern, if every message node in its corresponding bipartite graph has at most degree 2, then this code is an LMD code.

It is easy to see that an LMD code has the following properties in its bipartite graph representation:

1. Any message node can connect to at most two check nodes, i.e., its degree is no more than 2.
2. For a check node, if it connects to only one message node, then it can recover that message node.
3. The recovery of message nodes is essential for the recovery of an erasure pattern.

The first property is from the definition of LMD codes. The second and third properties are from the bipartite graph’s construction. The third property indicates when recovering an erasure pattern, the focus should be on the recovery of message symbol erasures. From the three properties, a simple and efficient decoding algorithms can be derived for any LMD code, which will be further discussed in Sec. 4.1 and Sec. 5. It is worthy noting that an LMD code can be MDS or non-MDS, since its definition does not have any requirement for the MDS feature. Table 1 summarizes the LMD code property for some well known RAID-6 codes mentioned in Sec. 1, and their MDS property is also listed for comparison convenience.

<table>
<thead>
<tr>
<th>Code</th>
<th>LMD code</th>
<th>MDS</th>
</tr>
</thead>
<tbody>
<tr>
<td>B-code</td>
<td>YES</td>
<td>YES</td>
</tr>
<tr>
<td>X-code</td>
<td>YES</td>
<td>YES</td>
</tr>
<tr>
<td>BCP code</td>
<td>YES</td>
<td>YES</td>
</tr>
<tr>
<td>ZZS code</td>
<td>YES</td>
<td>YES</td>
</tr>
<tr>
<td>WEAVER code</td>
<td>YES</td>
<td>NO</td>
</tr>
<tr>
<td>LSI code</td>
<td>YES</td>
<td>NO</td>
</tr>
<tr>
<td>EVENODD code</td>
<td>NO</td>
<td>YES</td>
</tr>
<tr>
<td>RDP code</td>
<td>NO</td>
<td>YES</td>
</tr>
</tbody>
</table>

Table 1. LMD codes

3.4 Bipartite Graph for EVENODD

Notably, the EVENODD code is not LMD code, because its EVENODD adjuster causes the message nodes for the message symbol erasures along the main diagonal have degree more than 2 in the bipartite graph representation. However, the EVENODD code can be transformed to an LMD code for fast erasure decoding with the following modifications on its bipartite graph:

1. If there are message symbol erasures along main diagonal, the EVENODD adjuster will be placed in the bipartite graph as a separate check node;
2. Then for any message symbol erasure along the main diagonal, its corresponding message node connects to the EVENODD adjuster node directly.

This modification slightly changes the EVENODD code’s construction. But when the EVENODD adjuster is computable, such a modification on the bipartite graph does not
change any property of the EVENODD code, yet makes it an \textit{LMD} code. Hence, in practice, we only need to apply the above modification (or \textit{patch}) to the EVENODD code to transform it to an \textit{LMD} code when the EVENODD adjuster is computable. Figure 3 shows such a modification process.

**Figure 3. A patch (modification) for the EVENODD code**

The following example shows an erasure pattern and its bipartite graph representation for an EVENODD code with \( p = 5 \), where a cross sign (\( X \)) represents an erasure, and the last two columns are check columns. The same notations are used hereafter.

4 Erasure Pattern Recovery

In this section, we present the main theoretical result of this paper, i.e., a necessary and sufficient condition to determine the recoverability of any erasure pattern for \textit{LMD} code as defined in Sec. 3.3, as well as a patch for the EVENODD code.

4.1 Recovery Theorem

\textbf{Theorem 1.} If a code is \textit{LMD} code, a necessary and sufficient condition for its recoverable erasure pattern is that each connected component in its bipartite graph satisfies

1. There is no cycle;

2. There is at most one message node having degree 1.

\textit{Proof.} First the necessary condition part: we prove an equivalent proposition, i.e., if an erasure pattern has at least one connected component which does not satisfy both conditions, it is not recoverable. Such a connected component must have a subgraph, which belongs to one of the three types described below. For the message nodes in the subgraph, their connected check nodes in the connected component are also in the subgraph. We show that each type of subgraphs is not recoverable by Lemma 1, because there exist at least two different solutions for the message symbol erasures in the subgraph. Then the erasure pattern is not recoverable by Lemma 2.

1. A message node of degree 0: it corresponds to an erasure pattern which contains erasures on a message symbol and its all check symbols (a special case of cycle). Since this message node is not checked by any check nodes, it can be assigned any value, and thus not recoverable. An example is shown below.

2. A cycle: it indicates that all corresponding check equations are linearly dependent, and thus there exist at least two sets of solutions to the message symbol erasures in the cycle. Hence the erasure pattern is not recoverable. The following shows an example.

3. At least two message nodes having degree 1: these two nodes are connected by a path because they are in a connected component. We can then choose all the message and check nodes along this path and their associated lines to form a subgraph. It is easy to see that if \( A \) is a solution to the message nodes in this subgraph, then its bitwise complement \( \bar{A} \) is also a solution to them. Hence this erasure pattern is not recoverable. Shown below is an example.

5
Now we prove the sufficient condition by constructively showing that if an erasure pattern satisfies the necessary condition, then it can be recovered by a simple decoding algorithm working as follows:

1. Find a check node which has degree 1. Use it to recover its connected message node and remove both of them from the graph.
2. Repeat the above step until no check node having degree 1 is left.

The first step is valid because of the second property of \textit{LMD} code. When this algorithm stops for an erasure pattern which satisfies the necessary conditions, if there are still message nodes left, then there is a contradiction: after the simple algorithm is performed, randomly choose a left connected component which has message nodes, it then must belong to one of the following cases:

1. containing 1 message node having degree 0: it contradicts the first necessary condition.
2. containing 0 or 1 message node having degree 1: then all the other message nodes have degree 2 because of the first property of \textit{LMD} code. The check nodes must have degree at least 2 after the simple algorithm is stopped. If this left component does not contain any cycles, the relationship between the total number of nodes \(N\) and the total number of degree \(D\) must satisfy \(D \geq 2N - 1\). Thus there must be at least one cycle in this connected component, and this is a contradiction with the first necessary condition.
3. containing at least 2 message nodes of degree 1: it contradicts the second necessary condition.

### 4.2 Patch For \textit{EVENODD} Code

As already discussed, \textit{EVENODD} code is not \textit{LMD} code, Theorem 1 hence is not directly applicable. But if its \textit{adjuster} is known, it then becomes an \textit{LMD} code and Theorem 1 applies. We use bipartite graph to discuss the recoverability (or computability) of the \textit{adjuster}.

#### 4.2.1 Recoverability of the Adjuster

Now we discuss when the \textit{adjuster} is computable. It can be proven that:

\textbf{Theorem 2.} For an erasure pattern, a necessary and sufficient condition for its computable \textit{adjuster} is that the corresponding bipartite graph has some connected component which satisfies the following two conditions:

1. Every message node has degree 2;
2. There are odd number of diagonal check nodes.

For these erasure patterns with computable \textit{adjuster}, there are two common cases:

1. all the check symbols are known: then their sum gives the \textit{adjuster} value;
2. a diagonal has no any erasure in its message symbols and check symbol: again their sum is the \textit{adjuster} value.

These two cases provide a shortcut to both check the recoverability of the \textit{adjuster} and compute its value. They are used in our decoding algorithm in Sec. 5.

#### 4.2.2 Recoverability of \textit{EVENODD}

Whether the \textit{adjuster} is recoverable is important to the recoverability of its erasure pattern, as described by below theorem.

\textbf{Theorem 3.} For \textit{EVENODD} code, a necessary condition for a recoverable erasure pattern is that the \textit{adjuster} is computable.

\textit{Proof.} Recall that the \textit{adjuster} is the XOR sum of all the \((p - 1)\) message symbols along the main check diagonal, so when an erasure pattern is recoverable, all the \((p - 1)\) message symbols along the main check diagonal are recoverable from known symbols, the \textit{adjuster} is certainly computable then.

By combining Theorem 1, 2 and 3, the recoverability of \textit{EVENODD} code can be determined by Theorem 4.

\textbf{Theorem 4.} A necessary and sufficient condition for a recoverable erasure pattern for the \textit{EVENODD} code is:

1. some connected component of its bipartite graph satisfies the conditions given in Theorem 2; and
2. each connected component satisfies the conditions given in Theorem 1.
5 SCAN: A Sector Erasure Decoding Algorithm

Now we turn to a practical sector erasure decoding algorithm for RAID-6 codes, called SCAN. SCAN consists of two parts: a Check algorithm and a Recovery algorithm. Loosely speaking, as their names suggest, the Check algorithm examines whether an erasure pattern is recoverable; then the Recovery algorithm corrects this erasure pattern if it is recoverable. Combined together, SCAN ensures no time wasted on irrecoverable erasures and thus reduces overall decoding time. If an erasure pattern is partially recoverable, i.e., not all the erasures can be recovered, the Recovery algorithm simply stops. (Note: SCAN is able to recover all recoverable erasures for an irrecoverable erasure pattern and can be easily modified to do so. The current choice is made in order to be consistent with our definition of irrecoverable erasure pattern.)

SCAN has both graph and non-graph implementations. We introduce both implementation in Sec. 5.1 and 5.2 respectively. The graph implementation is easy to understand and mainly for illustrating how SCAN works. The non-graph implementation is derived from the graph implementation, but it only uses simple array data structure and is thus more efficient in computation and suitable for practical use. Our simulation results in Sec. 6 are from the array implementation.

5.1 Graph Implementation of SCAN

5.1.1 The Check Algorithm

A pseudocode description of the Check algorithm is shown in Algorithm 1, where Recover_MN collects the list of message symbol erasures which will be recovered in order for a recoverable erasure pattern.

Algorithm 1 The Check Algorithm
procedure CheckRecoverability()
1: for each check node CN do
2: if CN has degree 1 then
3: ZigZag_Check(CN);
4: end if
5: end for
procedure ZigZag_Check(CN)
1: Get CN's connected message node MN;
2: Add MN to Recover_MN list
3: Find another check node CN2 connecting to MN;
4: Remove nodes CN and MN with their edges;
5: if CN2 exists and has degree 1 then
6: ZigZag_Check(CN2);
7: end if

5.1.2 The Recovery Algorithm

If an erasure pattern is examined to be recoverable, the recovery algorithm inherits the list Recover_MN from the Check algorithm and corrects all message symbol erasures in this list. Here the size of this list is identical to the number of message symbol erasures in the erasure pattern. The last step is to recover all check symbol erasures, which is the same as encoding procedure. Algorithm 2 is the pseudocode description for the recovery algorithm.

Algorithm 2 The Recovery Algorithm
procedure RecoverErasures()
1: for each erasure MN in Recover_MN do
2: Recover MN;
3: end for
4: Recover all check symbol erasures;

Finally, since a disk failure is just a special case of sector failures, SCAN can readily correct whole disk failures as well.

5.1.3 Correctness of SCAN

Theorem 5. SCAN can recover all recoverable erasure patterns for any LMD code.

Proof. In the proof for the sufficient condition of Theorem 1, a simple algorithm is provided which is able to recover all recoverable erasure patterns for any LMD code. The essence of that algorithm is to use any check node having degree 1 to recover one message node until all check nodes are used. SCAN is just an equivalent form of that algorithm, but much more efficiently.

5.1.4 Complexity of SCAN

SCAN consists of Check algorithm and Recovery algorithm, and we first discuss the Check algorithm's complexity. The most frequent operation in Check algorithm is to check whether a check node has degree 1. For a code with total p check symbols, Theorem 6 shows the number of check operations needed is a small linear factor of p for worst cases.

Theorem 6. In Check algorithm, given there are p check symbols defined in a code, there are at most 2p check operations on check nodes.

Proof. In Check algorithm, if we put these check nodes which are performed check operation along a line, we get a check nodes sequence. We then prove the length of this sequence is at most 2p.

A check node can occur in the sequence for two reasons. Firstly, it can be chosen by the loop outside the ZigZag_Check procedure. Secondly, it is visited inside a
ZigZag\_Check procedure. If a check node $CB$ is visited inside a ZigZag\_Check procedure and $CA$ occurs exactly before $CB$ in the sequence, then $CA$ recovers one message node before going to check $CB$. It shows an important property of the sequence, that is, one message node is recovered before a check node is visited inside a ZigZag\_Check procedure.

Assume that the length of the sequence is at least $2p + 1$, then we get a contradiction: since there are $p$ check symbols defined in the code, at most $p$ different check nodes can be chosen by the external loop; thus at least $p + 1$ check nodes are visited inside ZigZag\_Check, which leads to at least $p + 1$ message nodes recovered. But we know the maximum recoverable erasure number is $p$, because there are only totally $p$ check symbols. Therefore, the length of the check sequence is at most $2p$.

For Recovery algorithm, its most frequent operation is to use XORs to recover erasures. Given an erasure recovery sequence, the recovery complexity is the same as the code’s encoding complexity.

So it is easy to see that SCAN algorithm has low computation complexity.

### 5.1.5 Patch for EVENODD

Since EVENODD is not an LMD code, SCAN can not be directly used. However, when its adjuster is recoverable, EVENODD becomes a LMD code, and SCAN applies then. Hence a simple patch can make SCAN work for EVENODD as well. The patch is to first check if the EVENODD adjuster is recoverable by Theorem 2. If so, SCAN is then employed for EVENODD just as for any LMD code. Otherwise the erasure pattern is declared irrecoverable.

### 5.1.6 Patched Check Algorithm for EVENODD

To achieve better performance, the patch to check the EVENODD adjuster’s recoverability does not need to be a separate part. It can piggyback the existing check operations in Check algorithm. In a practical implementation, this patch is performed right after the ZigZag\_Check procedure, as it uses the result from ZigZag\_Check. Without a global view of the bipartite graph, the patch is still able to rebuild any connected component which satisfies the conditions in Theorem 1. After a connected component is rebuilt, then the patch checks the adjuster’s recoverability by Theorem 2 immediately. We use two examples to illustrate how it works.

The first example is simple with an erasure pattern as shown below. From its bipartite graph, it is easy to see one ZigZag\_Check procedure finds all the check nodes from $a_{06}$ to $a_{26}$ and the message nodes along the path. Thus the path can easily derive a connected component from these nodes.

Since now check node $a_{05}$ has degree 3, the ZigZag\_Check procedure will be called twice, and each time one check path will be generated. So there are totally two check paths for this bipartite graph shown as following orderley:

In check path 1, the last check node $a_{05}$ is connected with a dash line, which means $a_{05}$ needs to be checked later because it still connects to two message nodes $a_{01}$ and $a_{04}$. In check path 2, after $a_{01}$ is checked, $a_{05}$ connects to only one message node $a_{04}$ and it can be used then. So this connected component can be rebuilt from these two check paths as well.

The graph implementation of SCAN for EVENODD is described by pseudocode in Algorithm 3. The ZigZag\_Check procedure is exactly the same as in Algorithm 1. Algorithm 3 differs Algorithm 1 in it has two extra places to check the recoverability of the adjuster. The first check is using the short-cut way mentioned in 4.2.1, and it can decide the recoverability for most recoverable erasure patterns of the EVENODD code, as shown in Sec. 7.1. The second check is after each ZigZag\_Check procedure as discussed above.
Algorithm 3 The Check Algorithm for EVENODD

procedure CheckRecoverability()
1: Check adjuster computability by the short-cut way
2: for each check node CN do
3: if CN has degree 1 then
4: ZigZag_Check(CN);
5: if new connected component CC is found then
6: Check adjuster computability for CC;
7: end if
8: end if
9: end for

5.1.7 Patched Recovery Algorithm for EVENODD

Recovery algorithm for EVENODD has one more step than that for LMD code, which is to compute the adjuster as the first step. The necessary information to compute the adjuster is also inherited from Check Algorithm. Its pseudocode is shown in Algorithm 4.

Algorithm 4 The Recovery Algorithm for EVENODD

procedure RecoverErasures()
1: Compute the adjuster;
2: for each erasure MN in Recover MN do
3: Recover MN;
4: end for
5: Recover all check symbol erasures;

5.2 Array Implementation of SCAN

To further improve computation efficiency, in practice, SCAN can be implemented without using any complex graph data structures at all. A key technique is to generate the following data by scanning a codeword, before starting Check Algorithm:

1. an array of known check symbols to be used in the CheckRecoverability function; and
2. an array of message symbol erasures for each known check symbol which will be used in the ZigZag_Check procedure.

With these pre-generated arrays without using graph, the correctness of Check Algorithm is still preserved with a more efficient non-graph implementation using only simple linear arrays.

5.2.1 An Example

This section briefly describes an example to show how array implementation of SCAN algorithm is achieved. Below is an erasure pattern of EVENODD with its bipartite graph. In this example, we assume the adjuster is known beforehand to simplify the discussion.

In summary, array implementation has three main steps to recover above erasure pattern, which are:

1. Generate the check constraint tables: Two check constraint tables are generated from the design of EVENODD, and each table contains the horizontal/diagonal check symbols with their checked message symbols. Table 2 and Table 3 are for EVENODD with \( p = 5 \), in which the symbol is identified by its row and column index. We also generate the reverse check constraint table, which can easily find the corresponding horizontal or diagonal check symbols for any message symbol. A portion of this reverse constraint table is shown in Table 4. These tables are independent of specific erasure pattern, so they only need to be generated once and used repeatedly. Thus the complexity involved in this step can be ignored.

<table>
<thead>
<tr>
<th>Check symbol</th>
<th>Message symbols</th>
</tr>
</thead>
<tbody>
<tr>
<td>05</td>
<td>00 01 02 03 04</td>
</tr>
<tr>
<td>15</td>
<td>10 11 12 13 14</td>
</tr>
<tr>
<td>25</td>
<td>20 21 22 23 24</td>
</tr>
<tr>
<td>35</td>
<td>30 31 32 33 34</td>
</tr>
</tbody>
</table>

Table 2. check constraints for horizontal check symbols

<table>
<thead>
<tr>
<th>Check symbol</th>
<th>Message symbols</th>
</tr>
</thead>
<tbody>
<tr>
<td>06</td>
<td>00 32 23 14</td>
</tr>
<tr>
<td>16</td>
<td>10 01 33 24</td>
</tr>
<tr>
<td>26</td>
<td>20 11 02 34</td>
</tr>
<tr>
<td>36</td>
<td>30 21 12 03</td>
</tr>
<tr>
<td>adjuster</td>
<td>31 22 13 04</td>
</tr>
</tbody>
</table>

Table 3. check constraints for diagonal check symbols
2. Generate the message symbol erasures tables: For a given erasure pattern, two actions are taken to each check symbol by scanning the codeword: 1) to know whether it is known or not; 2) to collect the message symbol erasures covered by its check constraint. Regarding to the the example erasure pattern, their message symbol erasures tables are shown in Table 5 and Table 6. It can be easily examined that these two tables together contain the same information as the bipartite graph. Particularly, for any check symbol, the number of message symbol erasures in its corresponding entry of the table is equal to the degree of its check node in the bipartite graph.

<table>
<thead>
<tr>
<th>Check symbol</th>
<th>Flag</th>
<th>Message symbol erasures</th>
</tr>
</thead>
<tbody>
<tr>
<td>05</td>
<td>Known</td>
<td>00 02</td>
</tr>
<tr>
<td>15</td>
<td>Known</td>
<td></td>
</tr>
<tr>
<td>25</td>
<td>Known</td>
<td>20 22</td>
</tr>
<tr>
<td>35</td>
<td>Known</td>
<td></td>
</tr>
</tbody>
</table>

Table 5. message symbol erasures for horizontal check symbols

<table>
<thead>
<tr>
<th>Check symbol</th>
<th>Flag</th>
<th>Message symbol erasures</th>
</tr>
</thead>
<tbody>
<tr>
<td>06</td>
<td>Unknown</td>
<td></td>
</tr>
<tr>
<td>16</td>
<td>Known</td>
<td></td>
</tr>
<tr>
<td>26</td>
<td>Known</td>
<td>02 20</td>
</tr>
<tr>
<td>36</td>
<td>Known</td>
<td></td>
</tr>
<tr>
<td>adjuster</td>
<td>Known</td>
<td>22</td>
</tr>
</tbody>
</table>

Table 6. message symbol erasures for diagonal check symbols

3. Run Check and Recover algorithm: Because Recover algorithm is very straightforward, here we only describe Check algorithm. At the beginning, the Check algorithm, i.e., Algorithm 1, knows only the check symbol adjuster has degree 1 from Table 6, and then it enters ZigZag_CHECK procedure and uses adjuster as the first check symbol. In ZigZag_CHECK procedure, the message symbol erasure a_{22} can be recovered by using adjuster symbol. Then by consulting the inverse constrain table, the next check symbol a_{25} can be found. By Table 5, it can be known a_{25} has degree 1 now and a_{25} enters the ZigZag_CHECK, in which the same operation will be performed for a_{25} as for adjuster. The final result is this erasure pattern is recoverable, and the erasure list to be recovered has in-order elements: a_{22}, a_{20}, a_{02}, a_{00}.

6 Decoding Performance Evaluation

A generic Matrix Method is proposed in [12] for data reconstruction for any erasure codes, by essentially solving a group of linear equations. This method can certainly correct erasures for any RAID-6 codes. To our best knowledge, the SCAN algorithm is the only erasure recovery algorithm that is functionally equivalent to the Matrix Method for all aforementioned RAID-6 codes except the RDP code, i.e., it can correct all sector erasures recoverable by the Matrix Method and vice versa. Moreover, both algorithms consist of the Check and Recovery components as discussed in the previous section. Thus in this section, we compare in detail the decoding performance of the two algorithms, including their overall output bandwidth, Check and Recovery time complexity(i.e., time spent in the Check/Recovery algorithm) for various erasure patterns, as well as their memory space usage. All decoding performance metrics except the memory space usage are measured through simulations rather than on real sector failure data traces. For the reasons mentioned in Sec. 3, the EVENODD code is used for all comparisons.

6.1 Matrix Method

6.1.1 Analysis of Matrix Method

Matrix Method [12] has two main algorithms, the basic algorithm: Column-Incremental Construction Algorithm, and the optional algorithm: Reversing The Column Incremental Construction. The optional algorithm aims to improve the performance of the basic one, but when to use it is a major problem, which is not clearly addressed in [12]. The example below illustrates this problem.

The first erasure pattern shown below is simple, and the basic algorithm of Matrix Method can be applied to get the best decoding performance.
and use the following two techniques: we make our best efforts to optimize the implementation, for fair comparison, since there is no available source code from [12], we implement the basic algorithm of the Incremental Construction Algorithm, in our simulations.

For the reason discussed above in Sec. 6.1.1, we use the Matrix Method, namely the Column- Incremental Construction Algorithm, in our simulations. Since there is no available source code from [12], we implement the basic algorithm by ourselves. For fair comparison, we make our best efforts to optimize the implementation, and use the following two techniques:

1. All matrices are stored in two-dimensional arrays, and one bit represents one cell unit. Because the matrix operations in Matrix Method are only applied to columns, one int value in C language contains 32 cell units in column dimension, this makes inter-column operations efficient.

2. The workspace matrix is generated only once, and its content is copied to another working matrix for decoding each codeword. Additionally, they are both allocated with continuous memory space, so the copy operation can be efficiently executed by one memcpy.

6.2 Simulation Configurations

6.2.1 Gilbert Model for Sector Failures

The Gilbert model [10] is a common finite-state Markov model for bursty failures, as depicted in Fig. 4.

For a real disk, the same factors causing one sector failure may very likely affect a few neighboring sectors and thus make those sectors to fail as well, as observed in [1]. Thus the Gilbert model closely captures the characteristics of disk sector failures, and hence adopted in our simulations as failure model.

Using this model, a disk sector is either in the G (i.e., good or normal) state or the B (i.e., bad or failure) state, as shown in Figure 4. If s1 and s2 are two consecutive sectors and s2 follows s1, then the state transition probability represents the state probability of s2 for the given state s1 is in. For example, if s1 is in state G, then s2 is in state B with a probability of u and in state G with a probability of 1 − u. It is well known that with this model, the stationary failure rate, i.e., the overall sector failure rate or probability, is u/(u + v) when the disk becomes stable. Also when u + v = 1, it reduces to an i.i.d (independent identically-distributed) random failure model with sector failure probability u [10], i.e., sectors fail independently with the same probability u. Thus by choosing various (u, v) pairs, different sector failure patterns, including independent random failures, can be simulated.

For a group of disks, each disk independently follows the same Gilbert model with the same (u, v). This is a relatively simple but yet reasonable model.

6.2.2 Simulation Parameters

There are three parameters in our simulations, p, u and v. p is the number of message (user data) disks in RAID-6. We use p = 5, 7, 11, 13, 17, 19, 23, 29 in our simulations to evaluate the decoding algorithms performance, but only the test results for p = 11, 17 are shown here due to page limitation. Note a RAID-6 disk array consists of a total of p + 2 disks with 2 check disks. u and v represent the sector failure parameters of one disk, in which u is the transition probability from G to B and v is the converse. All disks share the same values of u and v based on the observations in [21]. Six values of u are tested in our simulation: 1/1000, 1/100, 1/10, 1/5, 1/3, 1/2.
1/500, 1/200, 1/100, 1/50, 1/20, and 2 values for \( v \): \( 1 - u \) for i.i.d random failures/erasures and 1/2 for correlated erasures. The sample size, i.e., the number of simulations for each triple \((p, u, v)\), is 100,000 with different erasure patterns. All experiments are repeated three times to measure the simulation error range of performance metrics. The test results show the maximum simulation error range is no more than 5%, and normally it is less than 1% and thus simulation error bars are not plotted in the following figures.

The values of \( u \) and \( v \) are the synthetic data for demonstration purpose, and they are relatively high. There are two reasons. First, the latest research in [1] and [21] shows the disk sector failure rate in practical systems is much higher than specified in the disk data sheets. Second, RAID-6 codes can be applied to both disk array and other fault tolerance systems, the wide coverage of \( u \) values are useful to evaluate the SCAN algorithm’s performance in different situations.

In each of the following figures, the Y-axis is the performance metric and the X-axis is always the value of \( u \). Because \( v \) is relatively stable in practice, it is more useful to examine how the change of \( u \) will affect the performance. Again the stationary sector failure rate can be easily computed by \( u/(u+v) \).

### 6.2.3 Simulation Environment

We implement both SCAN and Matrix Method in C language. All experiments are conducted on a HP de7600 workstation with following relevant hardware and software configurations: a) CPU: Intel Pentium D 2.8GHz; b) Memory: DDR2 533M, 1GB; c) OS: openSUSE 10.2 (X86-64) and d) OS kernel: Linux 2.6.18.2-34-default X86-64.

### 6.3 Recoverability of EVENODD

First we measure the erasure recoverability of EVENODD, i.e., the percentage of recoverable erasure patterns out of all generated erasure patterns for a given \((p, u, v)\). This is helpful to understand performance comparisons later.

Fig. 5(a) is for random erasures and Fig. 5(b) is for correlated erasures. These two figures show that:

1. when \( u \leq 0.02 \), the erasure recoverability for both random and correlated sector failure patterns is good.

2. when \( u > 0.02 \), the erasure recoverability drops quickly, for correlated failure patterns \((v = 1/2)\). In practice, of course, this corresponds to very high sector failure rates and should happen very rarely.

![Figure 5. EVENODD’s Erasure Recoverability](image)

### 6.4 Output Bandwidth

The output bandwidth is measured by the average number of message bits output from a decoding algorithm per time unit, and it is normalized for different values of \( p \).

#### 6.4.1 Random Failures

From Fig. 6(a) and Fig. 6(b), it can be observed that

1. When \( u \) and \( v \) are fixed, SCAN and Matrix Method both have higher output bandwidth for \( p = 17 \) than that for \( p = 11 \). This is because the basic overhead in SCAN for different \( p \) is similar, by amortizing over more message bits, larger \( p \) yields higher bandwidth.

2. When \( p \) and \( v \) are fixed, as \( u \) increases, the bandwidth decreases for both SCAN and Method, but SCAN’s bandwidth drops slower than Matrix Method’s. It shows SCAN is more robust in output bandwidth than Matrix Method as the number of sector failures increases.

3. When \( u \) and \( v \) are fixed, the gap between SCAN and Matrix Method for \( p = 17 \) is larger than that for \( p = 11 \). It shows SCAN scales better than Matrix Method in terms of output bandwidth, as the size of storage system/array \((p)\) increases.
6.4.2 Correlated Failures

Fig. 7(a) and Fig. 7(b) show the observations for random failures exhibited in Sec. 6.4.1 remain valid for correlated failures as well. Note that for a given \( u \), a lower \( v \) corresponds to higher stationary sector failure rate.

In summary, for both random and correlated failures, SCAN outperforms Matrix Method in output bandwidth, as it is more robust when the sector failure rate increases, and scales better when RAID-6 system increases in size.

6.5 Check Time

To further compare SCAN and Matrix, we examine both the Check time and Recovery time, i.e., average time consumed by the Check and Recovery algorithms respectively. Recall their sum is the overall decoding time. Since the Check algorithm also detects irrecoverable erasure patterns, the Check time includes time spent on irrecoverable erasure patterns as well.

6.5.1 Random Failures

Fig. 8(a) shows that the check time of Matrix Method is close to SCAN algorithm when \( p = 11 \), but much larger when \( p = 17 \) as shown in Fig. 8(b). A possible reason is the inter-column operations in Matrix Method become less efficient as \( p \) increases.

6.5.2 Correlated Failures

Again correlated failures in Fig. 9 show the similar results as random failures in Fig. 8.

In summary, we observe that for both random and correlated sector failures

1. For small \( p \)'s, such as \( p = 11 \), Matrix Method is comparable to SCAN algorithm in the check time;

2. When \( p \) is large, like \( p = 17 \), the check time of SCAN remains more stable than Matrix Method as sector failure rate increases.

6.6 Recovery Time

Obviously, the Recovery time is only applicable to recoverable erasure patterns. For both Matrix Method and SCAN, their Recovery algorithms inherit information from the corresponding Check algorithms to recover all erasures, thus their operations are quite simple XORs based on inherited recovery sequences.
6.6.1 Random Failures

From Fig. 10(a) and 10(b), it is surprising (or maybe not) to see that Matrix Method needs much more recovery time than SCAN, given their close check times shown in Sec. 6.5. Particularly for $p = 17$ and $u = 0.05$ in Fig. 10(b), the recovery time of Matrix Method is almost 20 times of that of SCAN.

Figure 8. Average Check Time for i.i.d Random Sector Failures ($u + v = 1$)

Figure 10. Average Recovery Time for i.i.d Random Sector Failures ($u + v = 1$)

6.6.2 Correlated Failures

The big gap of recovery time between SCAN and Matrix Method also exists for correlated failures, as shown in Fig. 11.

In summary, we observe that:

1. The Recovery algorithm of SCAN is much more efficient than that of Matrix Method even when $p$ is small.

2. One reason for Matrix Method to be so inefficient is when recovering erasures, the Matrix Method does not use recovered erasures as available data to help recover subsequent erasures. For example, for the EVEN-ODD code, after the adjuster is recovered, its value is not stored and it has to be computed again when it is used. So there are a lot of unnecessary computations.
6.7 Memory Space Usage

For a \((n, k)\) RAID-6 code with \(r\) rows, the array implementation of SCAN requires at most \(O(k^2r^2)\) memory space to store known check symbols and message symbol erasures, as discussed in Sec. 5.2. However, Matrix Method needs \(O((n + k)^2r^2)\) memory space to store all the cell data for its workspace matrix, as shown in Sec.6.1.2. Therefore, SCAN algorithm is much more efficient than Matrix Method in memory space usage. In [6], there is a new space efficient implementation of Matrix Method, in which the space requirement is decreased to at least \(O(k^2r^2)\). Hence the Matrix Method’s best memory complexity is only comparable to the worst case of the SCAN algorithm.

Finally, though this section shows the comparison of SCAN and Matrix Method on the EVENODD code, SCAN also outperforms Matrix Method for any other applicable RAID-6 codes in our extra tests.

7 Further Discussions

7.1 More on EVENODD

The SCAN algorithm can get better performance on EVENODD than our simulation results at the cost of negligible recovery power. The reason is below. If SCAN cannot recover the EVENODD adjuster by the short-cut way (in Sec. 4.2.1), it needs to perform a further check on reconstructed connected components. The complexity caused by such a check is not small. However, Our experiments show this check can be simply skipped. Table 7 gives the percentage of this check’s contribution to the recoverable erase patterns. From Table 7, it can be seen that the contribution of this further check is extremely small for decoding recoverable erase patterns. Thus in most applications, this step can be ignored safely to further improve the decoding output bandwidth without sacrificing EVENODD’s erasure recoverability.

<table>
<thead>
<tr>
<th>(u)</th>
<th>(p = 11)</th>
<th>(p = 17)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1/1000</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1/500</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1/200</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1/100</td>
<td>0.001%</td>
<td>0</td>
</tr>
<tr>
<td>1/50</td>
<td>0.011%</td>
<td>0.002%</td>
</tr>
<tr>
<td>1/20</td>
<td>0.081%</td>
<td>0.016%</td>
</tr>
</tbody>
</table>

Table 7. Percentage of Contribution by The Extra Check for \(v = 1/2\)

7.2 The RDP Code

The RDP code is a RAID-6 code derived from the EVENODD code[8]. Its encoding performance is better than the EVENODD code, at the cost of its worse decoding performance. Different from other popular codes, such as the X-code or the EVENODD code, its two parity columns are highly dependent in its encoding to achieve lower computation complexity, i.e., its parity check matrix has very high density. It hence does not have a single parity symbol that can be separated to transform to an LMD code, as applied to the EVENODD code. In coding terminology, it cannot be easily converted to an LDPC code. Thus it remains a meaningful research problem to design efficient erasure decoding algorithm for the RDP code.
7.3 More on the Matrix Method

A main difference between the basic algorithm of Matrix Method, i.e., Column-Incremental Construction Algorithm, and SCAN is whether the algorithm is sensitive to erasure recovery order. The basic algorithm of Matrix Method is insensitive to erasure recovery order, i.e., any erasure can be recovered at any step without prior conditions.

The SCAN algorithm, on the other hand, is sensitive to the erasure recovery order. The benefit of a sensitive algorithm is all previously recovered erasures can be used to recover other ones, thus avoiding repetitive computations in erasure recovery. Such a difference affects the decoding performance greatly, as shown in Sec. 6. As already discussed in Sec. 6.1.1, the Matrix Method has an optional algorithm, called Reversing The Column Incremental Construction. Like SCAN, the optional algorithm is also sensitive to the erasure recovery order. For best cases, the optional algorithm can perform as well as SCAN algorithm. However, it is hard to decide when to apply the optional algorithm, as already shown in 6.1.1. Hence an efficient erasure decoding algorithm should be designed to be sensitive to erasure recovery order.

7.4 The REO

A generic RAID engine called REO is proposed in [13], which is designed to systematically deduce the least cost strategy to recover the lost data for a read request or update strategy for a write request. It contains the function to recover both disk and sector failures for any XOR-based erasure codes, including the RAID-6 codes discussed in this paper.

REO has two engines, the RAID Engine and the Execution engine, and they both work for the reo_read request. When a reo_read request is received, if there is any sector or disk failure, the RAID Engine needs to use some decoding algorithm to deduce a reconstruction strategy, and then the Execution engine proceeds to execute the plan. In [13], the RAID Engine employs Matrix Method as its decoding algorithm, but it does not imply it can only use the Matrix Method, since the RAID Engine treats the decoding algorithm as a black box. It inputs the erasure pattern to the decoding algorithm and gets the recovery plan back, so to use which decoding algorithm is totally transparent to the REO. Therefore, SCAN can be readily embedded into REO as a replacement of the Matrix Method for RAID-6 code to achieve better performance in both output bandwidth and memory space usage.

8 Conclusions

This paper studies efficient disk sector failure recovery algorithm for RAID-6 codes, with contributions in both theory and practice. Theoretically, the concept of LMD code is introduced, which includes well known RAID-6 erasure codes. Then a necessary and sufficient condition is derived to determine the erasure patterns recoverability for LMD code and the EVENODD code. Practically, a novel universal decoding algorithm, called SCAN, is designed and implemented for correcting both disk sector and whole disk failures for RAID-6 systems using any LMD code or the EVENODD code.

SCAN consists of two components, the Check algorithm and the Recovery algorithm. The Check algorithm efficiently decides the recoverability of an erasure pattern without performing expensive full decoding operations. Inheriting erasure recovery consequence information from the Check algorithm, the Recovery algorithm then corrects all erasures in any recoverable erasure pattern using only XOR operations. Thus SCAN does not perform any unnecessary operations for erasure recovery.

The decoding performance of SCAN is evaluated by comparisons with another generic erasure recovery algorithm Matrix Method, using the EVENODD code as an example. Extensive simulation results and analysis show SCAN outperforms Matrix Method in all the relevant decoding metrics, including output bandwidth, memory space usage and algorithm scalability as RAID-6’s size increases, as well as algorithm robustness as sector failure rate increases. Thus SCAN has a great potential to be integrated into practical RAID-6 arrays/clusters.

A possible future research direction is to extend these results for RAID-6 to RAID-n (n ≥ 7) systems.

References


