CEA: A Cyclic Expansion Algorithm for Data Migration in Parallel Video Servers
Mingfu Li*, Hsun-Hao Yang
Department of Electrical Engineering, Chang Gung University
259 Wen-Hua 1st Road, Kwei-Shan, Tao-Yuan 33302, Taiwan
TEL: +886-3-2118800 Ext. 5676, FAX: +886-3-2118026

Abstract
Parallel video servers can achieve highly storage-saving and granularly load-balancing, but they suffer from a system expansion problem. As the number of users continuously increases, the system inevitably needs to expand the number of video servers. However, the expansion of a parallel video server system is not as simple as that of a replicated video server system. Hence, this work develops an efficient expansion algorithm, called the Cyclic Expansion Algorithm (CEA), for parallel video servers. The proposed CEA algorithm has several good features. First, the data layout of each video content exhibits periodicity. Consequently, the meta-data size of each video and the complexity of the CEA algorithm are reduced. Second, the number of required data movements during a system expansion is optimized. Third, the total number of required XOR recomputations for updating parity blocks during an expansion is also minimized. Additionally, the new CEA can be applied to a variety of distributed storage systems, such as the cloud-based storage systems using striping and parity check techniques.

Key words: Parallel video server; System expansion; Data migration; Load balancing; Fault-tolerant computing; Algorithm.

1. Introduction
Recently, video-on-demand (VoD) streaming services have been increasing rapidly owing to the significant growth of bandwidth in access networks, such as xDSL, FTTH, 3G/4G and WiMAX. Initially, VoD services were provided by a single server, which served the clients with limited bandwidth and capacity. To enhance service capacity and quality of service (QoS), multiple video servers with replicated multimedia contents are required. However, in such a replicated video server architecture, the required storage space grows significantly with the number of servers. To reduce the disk space consumption in the replicated video server system, several content allocation methods have been presented [1–7]. Most of them are based on the content popularity [1–3]. However, no matter which replication approach is employed, storage overhead and load balancing problems have not yet been solved completely.

To overcome the issues encountered in the replicated video server architecture, parallel video server architecture has been proposed [8,9]. In parallel video server architecture, each video is striped and distributed among all parallel servers. Hence, each streaming request is served by all parallel video servers, and load balancing is thus improved, regardless of the content popularity. Furthermore, the capacity of the system is effectively improved and the required storage space is independent of the number of parallel servers. Consequently, the storage requirement in a parallel video server system is minimized. Some related performance evaluations of parallel video servers can be found elsewhere [10–12].

Although the parallel video server architecture has been demonstrated to be a good solution for VoD systems, several implementation issues are yet to be resolved. For example, some studies [13–15] have addressed fault tolerance and data reconfiguration issues in parallel video servers. The paper [16] proposed a novel data splitting scheme and a clip striping policy for parallel video servers. Issues around load balancing based on intra-movie skewness have been investigated in [17]. The study [18] presented a data placement method for a parallel video server system to improve the space utilization by balancing the access loads. Other data placement strategies can also be found elsewhere [19–22]. The impact of topology on parallel video streaming has also been studied in [23].

* Corresponding author
Email address: mfli@mail.cgu.edu.tw (Mingfu Li).

Preprint submitted to Elsevier
When the capacity of a VoD system is insufficient, new servers must be added. In parallel video server architecture, the number of servers can be continuously increased without increasing storage consumption by redistributing data stripes. However, the system expansion in the parallel video server system is not as trivial as that in the replicated server system in which the contents are simply replicated to new servers. In the parallel video server system, some data stripes must be migrated from existing servers to new ones. Since data migrations occur at all existing video servers during the system expansion, the streaming services throughout the system may be halted or affected. To reduce the impact on services, the duration of system expansion should be as short as possible, meaning that the data migration overhead must be minimized.

However, the issue of data migration for parallel video servers or distributed storage systems has rarely been studied in literature [24, 25]. One work [24] proposed a Prime Based Hashing (PBH) for data reorganization in large distributed systems. PBH can support load balancing and efficient reorganization during the system scaling. However, the redundancy check updating issue has not been studied. Although the work [25] proposed an effective redundant data update algorithm for reducing the redundancy update overhead, it did not address the data migration overhead. In fact, the redundancy update overhead depends on the data reorganization algorithm. Thus, the design of a cost-effective data reorganization algorithm must consider both the data migration overhead and the redundancy update overhead. Consequently, this paper will propose an efficient system expansion algorithm that can simultaneously minimize the redundancy update overhead and the data migration overhead for parallel video servers.

The rest of this paper is organized as follows. Section 2 presents an overview of parallel video servers. Section 3 formulates the system expansion problem and introduces the proposed system expansion algorithm, CEA. Section 4 verifies the optimality and analyzes the complexity of the CEA. Section 5 evaluates the performance of the CEA. Finally Section 6 offers some concluding remarks.

2. Overview of Parallel Video Servers

In a parallel video server system, each video is segmented into several video blocks (stripe units), which are distributed to all parallel video servers. For example, let these segmented video blocks be denoted by \( v_0, v_1, v_2, \ldots \); a possible data layout for five parallel video servers is as displayed in Fig. 1. If the stripe unit size is much smaller than the video content, then the segmented video blocks can easily be distributed uniformly among all video servers. When a video streaming request is initiated, all video servers will cooperatively support the streaming service. Therefore, the load among all video servers can be well-balanced. Additionally, since no content is replicated in the video servers, the required storage space is minimized.

Video data striping can generally be categorized into time and space striping. When time striping is used, the playback times of all video stripes are equal: a video stream is striped in units of frames across multiple video servers. The other striping technique, which divides a video stream into several fixed-size video blocks, is called space striping. Based on the distribution of data among video servers, two data striping methods, wide and narrow, can be identified. In wide striping, a video stream is striped across all video servers. In narrow striping, a video stream is only striped into a subset of the video servers. Unless stated otherwise, space striping and wide striping are assumed in this paper.

In a parallel video server system, when a certain video server fails, a streaming service may be interrupted since it is supported by all video servers. To improve the reliability of the system, a fault-tolerant scheme in RAID-5 (Redundant Array of Inexpensive Drives) is exploited in parallel video servers. Several parity blocks \( P_0, P_1, P_2, \ldots \), are allocated uniformly among video servers, as shown in Fig. 2. The parity blocks in Fig. 2 satisfy the following equations.

\[
P_0 = v_0 \oplus v_1 \oplus v_2, \\
P_1 = v_3 \oplus v_4 \oplus v_5, \\
P_2 = v_6 \oplus v_7 \oplus v_8, \\
P_3 = v_9 \oplus v_{10} \oplus v_{11}, \\
\vdots
\]

Whenever a certain video server breaks down, such as when, for example, video server \( VS_2 \) in Fig. 2 fails, all video and parity blocks that are stored in that server can be recovered as follows.

\[
v_1 = P_0 \oplus v_0 \oplus v_2, \\
v_4 = v_3 \oplus P_1 \oplus v_5, \\
P_2 = v_6 \oplus v_7 \oplus v_8, \\
v_{11} = v_9 \oplus v_{10} \oplus P_3, \\
\vdots
\]

The above fault-tolerant design can work only if not more than one server fails. The parallel video server system with
blocks must be migrated from the existing video servers to the new video servers. Hence, the minimum number of parity block movements is \( MA/[(S + A)(S + A - 1)] \).

After the system expansion, the total number of video and parity blocks in all new video servers is equal to \( A \times M/(S + A - 1) \). Thus, the minimum number of video block movements equals \( AM/(S + A - 1) - MA/[(S + A)(S + A - 1)] \), which is equivalent to \( MA/(S + A) \).

Finally, for each parity block, the number of protected video blocks will increase from \((S - 1)\) to \((S + A - 1)\) after the system expansion. Therefore, the minimum number of XOR recomputations is \( A \times M/(S + A - 1) \).
Table: Round-robin data placement of a certain video when four video servers exist.

<table>
<thead>
<tr>
<th>Server ID</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Round 0</td>
<td>$P_0$</td>
<td>0</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>Round 1</td>
<td>3</td>
<td>$P_1$</td>
<td>4</td>
<td>5</td>
</tr>
<tr>
<td>Round 2</td>
<td>6</td>
<td>7</td>
<td>$P_2$</td>
<td>8</td>
</tr>
<tr>
<td>Round 3</td>
<td>9</td>
<td>10</td>
<td>11</td>
<td>$P_3$</td>
</tr>
<tr>
<td>Round 4</td>
<td>$P_4$</td>
<td>12</td>
<td>13</td>
<td>14</td>
</tr>
<tr>
<td>Round 5</td>
<td>15</td>
<td>$P_5$</td>
<td>16</td>
<td>17</td>
</tr>
<tr>
<td>Round 6</td>
<td>18</td>
<td>19</td>
<td>$P_6$</td>
<td>20</td>
</tr>
<tr>
<td>Round 7</td>
<td>21</td>
<td>22</td>
<td>23</td>
<td>$P_7$</td>
</tr>
<tr>
<td>Round 8</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Round 16</td>
<td>$P_{16}$</td>
<td>48</td>
<td>49</td>
<td>50</td>
</tr>
<tr>
<td>Round 17</td>
<td>51</td>
<td>$P_{17}$</td>
<td>52</td>
<td>53</td>
</tr>
<tr>
<td>Round 18</td>
<td>54</td>
<td>55</td>
<td>$P_{18}$</td>
<td>56</td>
</tr>
<tr>
<td>Round 19</td>
<td>57</td>
<td>58</td>
<td>59</td>
<td>$P_{19}$</td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Fig. 3. Round-robin data placement of a certain video when four video servers exist.

Notably, the round-robin placement scheme and the CEA algorithm proposed in this work both have the periodicity property, while the random placement scheme does not. Denote the number of video servers by $S$. Then, for the round-robin placement scheme, the placement periods $N_{VB}$ and $N_{PB}$ equal $S(S - 1)$ and $S$, respectively. For the proposed CEA, the placement periods depend on the prior placement period and the total number of video servers after expansion. When the number of video servers is increased from $S$ to $S + A$, some VBs must be migrated and some PBs must be updated. After the system expansion, each parity check round will comprise $(S + A - 1)$ VBs and one PB. Additionally, the number of PBs within a placement cycle must be a multiple of the number of video servers, $(S + A)$. Hence, to preserve the periodicity of the CEA algorithm, the new placement period of VBs after the system expansion, $N_{VB}^\prime$, must be a common multiple of $N_{VB}$, $(S + A)$ and $(S + A - 1)$. That is, $N_{VB}^\prime$ will become

$$N_{VB}^\prime = \lcm[N_{VB}, (S + A)(S + A - 1)],$$

where $[a, b]$ denotes the least common multiple (lcm) of integers $a$ and $b$. The placement period of PBs after the system expansion, $N_{PB}^\prime$, equals

$$N_{PB}^\prime = \frac{N_{VB}^\prime}{S + A - 1}.$$

Owing to the periodicity property, the same data layout can be applied to all videos in the CEA, regardless of diverse sizes of videos. Therefore, a single video is considered in our descriptions below.

3.2. Load Balancing

According to Section 2, a parallel video server system can achieve perfect load balancing if the data blocks are striped in a round-robin manner. Perfect load balancing implies that the numbers of VBs that are transmitted by different video servers in any given period are almost equal (differing by one block at most). Since each parity check round consists of $(S - 1)$ VBs and one PB, a service cycle is defined as the period during which $(S - 1)$ VBs are transmitted to the network to respond to a streaming request. Notably, to reduce the latency and buffer requirement at the client, the VBs are assumed to be transmitted in order [21]. Hence, the VBs that are transmitted in a service cycle are likely not to be in the same parity check round unless the VBs are striped in a round-robin manner. To evaluate the load balancing performance in a short time period, the short-term load balancing metric is defined. Perfect short-term load balancing requires that in each service cycle, these $(S - 1)$ VBs must be transmitted by different video servers.

To evaluate the short-term load balancing performance of a parallel video server system, the short-term load balancing indicator, $LBI_S$, is defined as follows.

$$LBI_S = \frac{1}{CS} \sum_{i=1}^C \sum_{j=0}^{S-1} (n_{i,j} - 1)^+,$$

where $n_{i,j}$ represents the number of VBs that are transmitted by video server $j$ in the $i$-th service cycle. $C$ is the total number of service cycles required to transmit the considered video content. The function $(\cdot)^+$ is defined by

$$(x)^+ = \begin{cases} x, & \text{if } x > 0, \\ 0, & \text{otherwise}. \end{cases}$$

According to Eq. (13), whenever $n_{i,j}$ is greater than 1, an overload occurs at video server $j$ during the $i$-th service cycle. Thus, a smaller $LBI_S$ corresponds to better short-term load balancing performance. When the $LBI_S$ of a data layout equals 0, perfect short-term load balancing is achieved. Consequently, perfect short-term load balancing can be achieved only when every video server transmits at most one VB during each service cycle. The $LBI_S$ can be shown to be equal 0 if the VBs are striped in a round-robin manner.

The load balancing metric involves two broad aspects—global load balancing and short-term load balancing. Global load balancing concerns the load balancing performance in the transmission of an overall video, while short-term load balancing concerns the load balancing performance in the transmission of VBs within a single service cycle. For the round-robin placement scheme, both perfect global load balancing and perfect short-term load balancing can be achieved. The proposed CEA algorithm can also yield perfect global load balancing but not necessarily perfect short-term load balancing. Nevertheless, perfect load balancing performance of the transmission of VBs within each placement cycle is also achieved using the proposed CEA algorithm.

3.3. Design Goals of CEA

The use of IPTV services has been increasing rapidly recently. The demand on streaming services will continue to
grow. However, to protect cost efficiency, service providers are usually unwilling to allocate excess video servers initially. Inevitably, a parallel video server system will finally encounter an expansion problem. Although the round-robin data placement scheme can achieve perfect load balancing, the data reorganization overhead is considerable. For example, in Fig. 3, when one new video server is added to the system, almost all VBs must be moved if they need to be placed in a round-robin manner. The migration rate of VBs in the round-robin placement scheme is much higher than the optimal migration rate \( A/(S + A) \), which is only 1/5, according to Theorem 1. Hence, to reduce the data reorganization overhead, a novel cost-effective system expansion algorithm is extremely required.

With reference to the example in Fig. 3, if \( S = 4 \) and \( A = 1 \), then \( N_{VB} = 60 \) and \( N_{PB} = 15 \) according to (11) and (12). As a result, after the system expansion, each placement cycle includes 15 parity check rounds. The purpose of a data migration algorithm is to redistribute VBs 0 to 59 in Fig. 3 to video servers 0 to 4, and to recompute the new PBs \( P_6 \) to \( P_{14} \) in the first placement cycle. Subsequent VBs and PBs can be striped by following the same rule as that used in the first placement cycle. Since the first placement cycle consists of only 15 parity check rounds after the system expansion, five of the PBs \( P_6 \) to \( P_{14} \) in the first placement cycle are eliminated, and 12 of the VBs 0 to 59 must be migrated to the new video server 4. However, a question arises: which PBs should be deleted and which VBs should be moved to minimize the numbers of data movements and XOR computations? Before this question can be answered, the system expansion problem must be clearly formulated, as follows.

**Formulation of System Expansion Problem:**
Find a data reorganization algorithm to minimize (O.1) the number of XOR recomputations that are required to update PBs and (O.2) LBI_S, subject to the following constraints; (C.1) perfect global load balancing is achieved; (C.2) the number of VB movements is optimal, and (C.3) the number of PB movements is optimal.

According to the problem formulation, the round-robin placement scheme cannot satisfy constraint (C.2). As the number of video servers increases, the number of PBs is reduced. Hence, some PBs should be removed during a system expansion. The problem now is to determine which PBs should be removed to minimize the number of XOR calculations (O.1) and LBI_S (O.2).

To solve this problem, the CEA is designed to satisfy all constraints (C.1) to (C.3) and objectives (O.1) and (O.2) during the system expansion. The algorithm Select_PB(i) is developed to identify the candidate PBs for deletion. Then, the algorithms Movable(i) and Expansion(i) are utilized to generate the data layout. Figure 4 shows the proposed CEA algorithm. The procedure Input() reads the placement matrix Allocation[ ][ ]; which records the data layout or meta-data of each video before the system expansion. The index of a VB that is allocated to the i-th parity check round and placed in video server j will be stored in Allocation[i][j]. When Allocation[i][j] = -1, it indicates that \( P_i \) is stored in video server j. The procedure Output() generates the new data layout (placement matrix) for each video after the system expansion. The proposed CEA will only generate the placement matrix for the first placement cycle, since the periodicity property can be employed to migrate the PBs and VBs outside the first placement cycle.

The parameters \( T \) and \( r \) in the CEA algorithm are illustrated as follows. First, the parameter \( r \) is the number of parity check rounds that must be reorganized before the expansion, and is defined by

\[
r = \frac{N_{VB}}{S - T}
\]

where \( N_{VB} \) is given by (11). The parameter \( T \) can be regarded as the period of a subcycle before the system expansion. That is, \( T \) is the number of parity check rounds within a subcycle, and is given by

\[
T = \frac{[S - 1, A]}{S - 1} + \frac{[S - 1, A]}{A}
\]

Equation (16) is explained as follows. Before the system expansion, each parity check round includes \((S - 1)\) VBs. If a PB has to be deleted, then the \((S - 1)\) VBs that are protected by this PB must be migrated to new video servers to reduce the number of XOR calculations (O.1). These VBs must be redistributed to other preserved parity check rounds. According to Fig. 5, every \( T \) parity check rounds can be a complete execution subcycle only if the number of VBs that are migrated to the \( A \) newly added video servers within this subcycle is a multiple of \( A \). Thus, the least number of VBs that must be migrated to the \( A \) newly added video servers within a subcycle equals \([S - 1, A]\). Consequently, the number of PBs for deletion within each subcycle equals

\[
x = \frac{[S - 1, A]}{S - 1}.
\]

Additionally, the corresponding \([S - 1, A]\) VBs must be redistributed to the other \((T - x)\) preserved parity check rounds. Hence,

\[
T - x = \frac{[S - 1, A]}{A}.
\]
The purpose of \textit{Select}_{PB}(i), \textit{Move}able(i), and \textit{Expansion}(i) are described in detail.

3.4. Algorithm \textit{Select}_{PB}(i)

The purpose of \textit{Select}_{PB}(i) is to find the proper PBs for deletion and to achieve objective (O.2). The algorithm \textit{Select}_{PB}(i) can be implemented step by step as follows.

(i) Calculate the latent net increment of \(LBI_S\) for each PB in subcycle \(i\).

(ii) Select \(x\) PBs from subcycle \(i\) based on the deletion priorities of PBs and the number of PBs allocated to each video server. The deletion priorities of PBs are determined from the results of Step (i).

(iii) All VBs that are protected by these selected PBs are candidates for migration.

The above steps will be described in detail below.

3.4.1. Computation of net increment of \(LBI_S\)

With reference to the example in Fig. 3, when \(S = 4\) and \(A = 1\), \(r = 20\), \(T = 4\) and \(x = 1\), according to Eqs. (15) to (17). Hence, one PB should be deleted from every \(T = 4\) parity check rounds, as shown in Fig. 6. Accordingly, five PBs should be deleted among \(r = 20\) PBs. Thus, the CEA will select one PB for deletion from each of the following sets (subcycles).

\[
\begin{align*}
B_0 & = \{P_0, P_1, P_2, P_3\}, \\
B_1 & = \{P_4, P_5, P_6, P_7\}, \\
B_2 & = \{P_8, P_9, P_{10}, P_{11}\}, \\
B_3 & = \{P_{12}, P_{13}, P_{14}, P_{15}\}, \\
B_4 & = \{P_{16}, P_{17}, P_{18}, P_{19}\}.
\end{align*}
\]

Notably, selecting different PBs for deletion will result in different \(LBI_S\) performance. Denote the resulting net increment of \(LBI_S\) by \(\Delta LBI_S[i][m]/[C(S + A)]\) when \(P_{T+m}\) in set \(B_i\) (subcycle \(i\)) is selected for deletion, where \(0 \leq m < T\). Then, the deletion priorities of all \(P_{T+m}\)'s in set \(B_i\) can be determined based on \(\Delta LBI_S[i][m]\).

To reduce the amount of XOR computations (O.1), when \(P_{T+m}\) in set \(B_i\) is selected for deletion, all the PBs that are protected by \(P_{T+m}\) must be migrated to newly added video servers. Therefore, certain migrated PBs that are in the same service cycle may be distributed to the same newly added video servers, so that \(LBI_S\) will be increased. However, \(LBI_S\) at the old video servers may decrease after the system expansion, since some VBs that contribute to \(LBI_S\) may be moved to new video servers.

For instance, assume that the system originally included four video servers, to which two are added: \(S = 4\) and \(A = 2\). After the system expansion, each service cycle has \(S + A - 1 = 5\) VBs. For example, VBs 0, 1, 2, 3, and 4 are in the first service cycle, while VBs 5, 6, 7, 8, and 9 are in the second service cycle. In Fig. 7, VBs 1 and 4 are both originally stored in video server 2. If VBs 1 and 4 are not moved, then video server 2 must transmit two VBs in the first service cycle. Therefore, VBs 1 and 4 will contribute one unit to \(LBI_S\). However, \(LBI_S\) at the old video servers may decrease after the system expansion, since some VBs that contribute to \(LBI_S\) may be moved to new video servers.

![Fig. 5. Derivation of values of parameters \(x\) and \(T\) based on \(S\) and \(A\).](image)

![Fig. 6. Relationships among parameters \(r\), \(T\), and \(x\): PBs selected for deletion (gray) and movable PBs (circles) for \(S = 4\) and \(A = 1\).](image)
in the second service cycle, degrading \( LBI_{S} \) performance. Hence, VBs 5, 6, 8, and 9 are marked in gray in Fig. 7. When \( P_1 \) is selected for deletion, VBs 4 and 5 (gray) will be migrated to new video servers so that \( LBI_{S} \) can be reduced by 2. Hereafter, \( LBI_{S}^{(-)}[k] \) is used to indicate the decrease in \( LBI_{S} \) if \( P_k \) is selected for deletion. In the example in Fig. 7, \( LBI_{S}^{(-)}[1] = \{1, 2, 2, 3, 2, 1, 1, 1, 3, 2, \cdots \} \). Similarly, for \( S = 4 \) and \( A = 1 \), \( LBI_{S}^{(-)}[k] = 0 \) for all \( k \), such that \( LBI_{S}^{(-)}[1] = \{0, 0, 0, 0, \cdots \} \), if the original data layout is as given by Fig. 6.

To introduce the increase in \( LBI_{S} \) that is caused by data migration, the set \( B_0 \) in Fig. 6 is considered as an example. Figure 8 shows all possible data layouts for the first subcycle after the system expansion for \( S = 4 \) and \( A = 1 \), when the differently colored VBs belong to different service cycles. According to Figs. 8(a) and 8(d), when \( P_0 \) or \( P_3 \) is selected for deletion, \( LBI_{S} \) will be increased by 2 because both \( n_{1.4} \) in Fig. 8(a) and \( n_{3.4} \) in Fig. 8(d) equal 3 (according to the definition of \( n_{ij} \) in Eq. (13)). When \( P_1 \) or \( P_5 \) is selected for deletion, \( LBI_{S} \) is increased only by 1, as shown in Figs. 8(b) and 8(c). Here, \( LBI_{S}^{(+)}[k] \) is adopted to represent the increase in \( LBI_{S} \) at all new video servers when \( P_k \) is selected for deletion. Hence, \( LBI_{S}^{(+)}[1] = \{2, 1, 1, 1, 2, \cdots \} \) in the example in Fig. 8.

As mentioned above, \( LBI_{S} \) may decrease at the original video servers and increase at new video servers when a PB is deleted. Thus, these two factors \( LBI_{S}^{(+)}[1] \) and \( LBI_{S}^{(-)}[1] \) must be combined. When \( P_{T+m} \) is selected for deletion, the net increment \( \Delta LBI_{S}[i][m] \) can be computed by

\[
\Delta LBI_{S}[i][m] = LBI_{S}^{(+)}[iT + m] - LBI_{S}^{(-)}[iT + m],
\]

where \( 0 \leq m < T \). For the example in Fig. 6, where \( S = 4 \) and \( A = 1 \), we obtain

\[
\Delta LBI_{S}[0][1] = \{2, 1, 1, 2\},
\]

since \( LBI_{S}^{(+)}[1] = \{2, 1, 1, 2, \cdots \} \) and \( LBI_{S}^{(-)}[1] = \{0, 0, 0, 0, \cdots \} \).

3.4.2. Selecting \( x \) PBs for deletion

The net increment \( \Delta LBI_{S}[i][m] \) is employed to determine the deletion priority of each PB. A smaller \( \Delta LBI_{S}[i][m] \) corresponds to a higher deletion priority of the \( P_{T+m} \). In CEA, \( PB_{Priority}[i][1] \) is used to store the indices of the PBs in set \( B_i \), in the order of non-increasing priority. For example, if \( PB_{Priority}[i][1] = m \), then \( P_m \) will have a priority of \( l \) in set \( B_i \). Here, a priority of 0 is the highest. However, since the PBs must be evenly distributed among all video servers, the cumulative numbers of PBs that are distributed to all video servers may not vary by more than one after each selection step. Hence, the cumulative number of PBs that are distributed to each video server, \( n \), is given by \( N_{PB}[n] \). Only when \( P_m \) satisfies \( N_{PB}[VS_{PB}[m]] = \max\{N_{PB}[i]\} \) can it be selected for deletion. Initially, \( N_{PB}[n] = 0 \) for all \( n \), so the selection of PBs for deletion in set \( B_0 \) is determined solely by the priorities of PBs.

According to (20), since \( \Delta LBI_{S}[0][1] \leq \Delta LBI_{S}[0][2] \leq \Delta LBI_{S}[0][3] \leq \Delta LBI_{S}[0][4] \leq \Delta LBI_{S}[0][5] \), we obtain \( PB_{Priority}[0][1] = \{1, 2, 0, 3\} \). Because \( x = 1 \) when \( S = 4 \) and \( A = 1 \), \( P_1 \) is the optimal choice for deletion. Generally, based on \( PB_{Priority}[i][1] \) and \( N_{PB}[i] \), \( x = [S - 1, A]/(S - 1) \) PBs for deletion from set \( B_i \) can be selected. These \( x \) PBs are collected in the set \( B_i = \{PB_{select}[i], PB_{select}[i+1], \cdots, PB_{select}[i+x-1]\} \). Here, the array \( PB_{select}[i] \) is exploited to record the indices of the selected PBs for deletion, hence, \( PB_{select}[0] = 1 \) for the example in Fig. 6. If \( P_m \) in set \( B_i \) is not selected for deletion, then the value of \( N_{PB}[VS_{PB}[m]] \) will be increased by 1.

The above procedure can be applied to set \( B_1 \). For the example in Fig. 6, \( \Delta LBI_{S}[0][1] \) is the same as \( \Delta LBI_{S}[0][0] \), but \( N_{PB}[1] = \{0, 0, 0, 0, 0\} \) will be updated to \( N_{PB}[1] = \{1, 0, 1, 1, 0\} \). Although the priority order of the PBs in set \( B_i \) is \( PB_{Priority}[1][1] = \{5, 6, 4, 7\} \), \( P_6 \) in Fig. 6 cannot be selected for deletion because \( N_{PB}[VS_{PB}[5]] = N_{PB}[1] \) is not the maximum. Thus, the next candidate \( P_6 \) is selected for deletion, and the result \( PB_{select}[1] = 6 \) is obtained. Notably, some preserved PBs (in circles) in Fig. 6 will be moved to the newly added video server. These movable PBs

<table>
<thead>
<tr>
<th>Server ID</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Round 0</td>
<td>( P_0 )</td>
<td>0</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>Round 1</td>
<td>( 3 )</td>
<td>( P_1 )</td>
<td>4</td>
<td>5</td>
</tr>
<tr>
<td>Round 2</td>
<td>6</td>
<td>7</td>
<td>( P_2 )</td>
<td>8</td>
</tr>
<tr>
<td>Round 3</td>
<td>9</td>
<td>10</td>
<td>11</td>
<td>( P_3 )</td>
</tr>
<tr>
<td>Round 4</td>
<td>12</td>
<td>13</td>
<td>14</td>
<td></td>
</tr>
<tr>
<td>Round 5</td>
<td>15</td>
<td>( P_5 )</td>
<td>16</td>
<td>17</td>
</tr>
<tr>
<td>Round 6</td>
<td>18</td>
<td>19</td>
<td>( P_6 )</td>
<td>20</td>
</tr>
<tr>
<td>Round 7</td>
<td>21</td>
<td>22</td>
<td>23</td>
<td>( P_7 )</td>
</tr>
<tr>
<td>Round 8</td>
<td>( P_8 )</td>
<td>24</td>
<td>25</td>
<td>26</td>
</tr>
<tr>
<td>Round 9</td>
<td>27</td>
<td>( P_9 )</td>
<td>28</td>
<td>29</td>
</tr>
</tbody>
</table>

Fig. 7. Decrease in \( LBI_{S} \) in each parity check round when \( S = 4 \) and \( A = 2 \).

<table>
<thead>
<tr>
<th>Server ID</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Round 0</td>
<td>( 0 )</td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>Round 1</td>
<td>( 1 )</td>
<td>( 3 )</td>
<td>( P_1 )</td>
<td>4</td>
</tr>
<tr>
<td>Round 2</td>
<td>6</td>
<td>7</td>
<td>( P_2 )</td>
<td>8</td>
</tr>
<tr>
<td>Round 3</td>
<td>9</td>
<td>10</td>
<td>11</td>
<td>( P_3 )</td>
</tr>
</tbody>
</table>

(c) Delete \( P_6 \), \( LBI_{S}^{(+)}[2][1] = 1 \) (d) Delete \( P_4 \), \( LBI_{S}^{(+)}[3][1] = 2 \)

<table>
<thead>
<tr>
<th>Server ID</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Round 0</td>
<td>( 0 )</td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>Round 1</td>
<td>( 0 )</td>
<td>( P_0 )</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>Round 2</td>
<td>6</td>
<td>7</td>
<td>( P_2 )</td>
<td>8</td>
</tr>
<tr>
<td>Round 3</td>
<td>9</td>
<td>10</td>
<td>11</td>
<td>( P_3 )</td>
</tr>
</tbody>
</table>

Fig. 8. \( LBI_{S}^{(+)}[i][1] \) at the new video server when each PB in \( B_0 \) is selected for deletion given \( S = 4 \) and \( A = 1 \).
are chosen by the algorithm Movable(i), which will be introduced later. From Fig. 6, the results for the other sets B_2, B_3 and B_4 are summarized as follows.

(i) set B_2: \( N_{PB}[i] = \{1, 1, 1, 2, 1\} \), \( PB\_Priority[2][i] = \{9, 10, 8, 11\} \), select[2] = 11.


(iii) set B_4: \( N_{PB}[i] = \{2, 2, 3, 2\} \), \( PB\_Priority[4][i] = \{17, 18, 16, 19\} \), select[4] = 18.

All PBs that are selected for deletion, which are marked in gray in Fig. 6, are recorded in the array select[] = \{1, 6, 11, 13, 18\}.

3.4.3. Candidate VBs for migration

All of the VBs that are protected by \( P_{select}[i] \) will be stored in the array \( PB\_VB[select[i][m]] \) in the order of increasing index. Accordingly, for the example in Fig. 6,

\[
\begin{align*}
PB\_VB[select[0][0]] &= \{3, 4, 5\}, \\
PB\_VB[select[2][2]] &= \{18, 19, 20\}, \\
PB\_VB[select[3][2]] &= \{33, 34, 35\}, \\
PB\_VB[select[3][3]] &= \{39, 40, 41\}, \\
PB\_VB[select[4][4]] &= \{54, 55, 56\}.
\end{align*}
\]

Appendix A presents the details of the algorithm Select_PB(i). Finally, the algorithms Movable(i) and Expansion(i) will generate the new data layout based on \( PB\_VB[select[i][m]] \) and the placement matrix Allocation[][]. The execution of Movable(i) and Expansion(i) will be discussed in detail below.

3.5. Algorithms Movable(i) and Expansion(i)

Once the PBs for deletion are decided, the VBs in \( PB\_VB[select[i][m]] \) will be redistributed by the algorithm Expansion(i). The VBs in \( PB\_VB[select[i][m]] \) will be sequentially distributed to new video servers in a round-robin manner. However, some preserved PBs should also be migrated to new video servers to ensure that the PBs are distributed among all video servers equally. To allocate PBs to all video servers equally and minimize the numbers of data movements (C.2 & C.3) and XOR calculations (O.1), the algorithm Movable(i) is proposed. The algorithm Movable(i) decides which preserved PBs can be migrated to new video servers and which VBs in \( PB\_VB[select[i][m]] \) need not be migrated. If the preserved PB, which is placed in the old video server \( VS\_PB[j] \), is migrated to a new video server, then the indicator Movable-PB[j] will be set to 1. Additionally, a certain VB in \( PB\_VB[select[i][m]] \) must be chosen to be moved to video server \( VS\_PB[j] \) and allocated in the original j-th parity check round (placed in the original location of \( P_j \)). To minimize the number of data movements, the VB, say \( PB\_VB[select[i][m]] \), which is placed at the same video server \( VS\_PB[j] \) with the movable \( P_j \), will be the candidate and not migrated. Such a VB is labeled with the indicator Movable-\( PB\_VB[select[i][m]] = 0 \). Such a policy can prevent unnecessary VB movement.

For example, in Fig. 6 the preserved PBs \( P_4, P_{12} \) and \( P_{19} \) are chosen to be migrated to the new video server: Movable_PB[4] = Movable_PB[12] = Movable_PB[19] = 1. As a result, the corresponding VBs 18, 39 and 56, which are circled in Fig. 6, need not be migrated: Movable[18] = Movable[39] = Movable[56] = 0. In summary, the algorithm Movable(i) is designed to select appropriate preserved PBs for migration and to identify those VBs in \( PB\_VB[select[i][m]] \) that need not be moved. To balance the number of PBs among all video servers, the selection of PBs for migration is also based on the numbers of PBs that have been distributed to all video servers.

Finally, based on the results \( select[i], PB\_VB[select[i][m]], Movable_PB[] \) and Movable[], obtained from the algorithms Select_PB(i) and Movable(i), the algorithm Expansion(i) can be executed to generate the new data layout. For example, according to Fig. 6, VBs 3, 4, 5, 19, 20, 33, 34, 35, 40, 41, 54 and 55 will be migrated to the new video server 4. PBs \( P_1, P_{12} \) and \( P_{19} \) are also migrated to the new video server 4. Of course, during the system expansion, the new PBs should be recomputed according to the following equations.

\[
\begin{align*}
P'_0 &= P_0 \oplus v_3, & P'_4 &= P_2 \oplus v_4, \\
P'_2 &= P_3 \oplus v_5, & P'_8 &= P_4 \oplus v_{18}, \\
& P'_5 &= P_3 \oplus v_{19}, & P'_9 &= P_7 \oplus v_{20}, \\
& P'_6 &= P_8 \oplus v_{33}, & P'_7 &= P_9 \oplus v_{34}, \\
& P'_8 &= P_{10} \oplus v_{35}, & P'_9 &= P_{12} \oplus v_{39}, \\
& P'_{10} &= P_{14} \oplus v_{40}, & P'_{10} &= P_{15} \oplus v_{41}, \\
& P'_{12} &= P_{16} \oplus v_{34}, & P'_{13} &= P_{17} \oplus v_{55}, \\
& P'_{14} &= P_{19} \oplus v_{56}.
\end{align*}
\]

Therefore, the number of required XOR calculations in (21) is 15 for the first placement cycle and the numbers of data movements for VBs and PBs during the first placement cycle are 12 and 3, respectively. Figure 9 shows the data layout for \( S = 4 \) and \( A = 1 \) when the CEA expansion algorithm is implemented. Obviously, the numbers of data movements and XOR recomputations are consistent with the values obtained by Theorem 1, revealing that our CEA algorithm is optimal in terms of these respects. Figure 10 shows the data layout for \( S = 4 \) and \( A = 2 \). Algorithms Movable(i) and Expansion(i) can also be found in Appendices B and C, respectively.
Clearly, after system expansion, the number of PBs in a placement cycle is
\[ r \]
(except for the last one). Before the system expansion, the number of PBs within a placement cycle is
\[ N_{PB} = \left( \sum_{i=1}^{S} \left( A + 1 \right) \right) + D, \] (23)
where
\[ D = \sum_{i=0}^{S+1} \{ N_{PB}[i] - N_{PB}[m] \}. \]
If at least one \( N_{PB}[i] \) does not equal \( N_{PB}[m] \), then the value of \( D \) will satisfy \( 0 < D < (S + A) \). Thus, the total number of PBs within a placement cycle, \( (S + A)N_{PB}[m] + D \), will not be a multiple of \( (S + A) \), which leads to a contradiction. Consequently, in our CEA, the numbers of PBs that are distributed to all video servers in a placement cycle are equal, being \( N'_{PB}/(S + A) \), or \( N'_{PB}/[(S + A)/(S + A - 1)] \). Accordinly, the total number of PB movements equals \( N'_{VB}/[(S + A)/(S + A - 1)] \). By Theorem 1, the number of PB movements in CEA is optimized (C.3).

Next, according to our CEA, the total number of VB and PB movements in a placement cycle equals
\[ r \frac{T}{(S - 1)x} = \frac{AN_{VB}}{S + A - 1}. \] (24)
Thus, subtracting the number of required PB movements \( N'_{VB}/[(S + A)/(S + A - 1)] \) from (24) yields the number of VB movements as \( AN_{VB}/(S + A) \), which equals to the optimal number of VB movements (C.2) according to Theorem 1. Additionally, in any placement cycle, the numbers of PBs that are distributed to all video servers are the same, being \( N'_{PB}/(S + A) \) to each. Therefore, the CEA provides perfect global load balancing (C.1).

The number of XOR recomputations in a placement cycle equals the sum of the number of VB movements and the number of PB movements. Based on (24), the number of XOR recomputations is \( N'_{PB}/(S + A) \). By Theorem 1, the number of required XOR recomputations is also optimized by CEA (O.1).

### 4.2. Complexity Analysis of CEA

Table 1 presents the results of a complexity analysis of the algorithms in CEA. The algorithm \( \text{Input}() \) reads the original data layout \( \text{Allocation}[i][j] \) and computes \( LB1_S^{-1}[i] \) for greater than 1 for all \( i \), where \( N_{PB}[m] = \min_i \{ N_{PB}[i] \} \).

**Table 1:** Complexity Analysis of CEA

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Complexity</th>
<th>Total Complexity</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input()</td>
<td>( O(r) )</td>
<td>( \times 1 \cdot O(r) )</td>
</tr>
<tr>
<td>Output()</td>
<td>( O(N_{PB}(S + A)) )</td>
<td>( \times 1 \cdot O(r) )</td>
</tr>
<tr>
<td>Select_PB(i)</td>
<td>( O(T^2) )</td>
<td>( \times r/T \cdot O(r) )</td>
</tr>
<tr>
<td>Moveable(i)</td>
<td>( O(xS^3) )</td>
<td>( \times r/T \cdot O(rS) )</td>
</tr>
<tr>
<td>Expansion(i)</td>
<td>( O((T - x)(S + A) + x) )</td>
<td>( \times r/T \cdot O(r) )</td>
</tr>
</tbody>
</table>
all $0 \leq i < r$ and $0 \leq j < S$. Hence, its complexity is $O(rS)$. Similarly, the algorithm $Output()$ generates the new data layout for all $0 \leq i < N'_{PB}$ and $0 \leq j < S + A$. Hence, the complexity of $Output()$ is $O(N'_{PB}(S + A))$, where $N'_{PB}$ is given by (22).

The algorithm $Select_{PB}(i)$ comprises three “for” loops. The first “for” loop implements an insertion sort to arrive at $PB_{Priority}[i]$, so its complexity is $O(T^2)$. The second “for” loop includes an insertion sort for sorting $x$ PBs and a step for updating the minimum of all $N_{PB}[t]$, so its complexity equals $O(x^2 + (T - x))$. The third “for” loop will be executed for $x$ times, and each time an insertion sort is implemented to sort $(S - 1)$ VBs. Therefore, the complexity of the third “for” loop equals $O(xS^3)$.

The algorithm $Moveable(i)$ consists of two “for” loops. The complexity of the first is $O(T)$. The complexity of the second loop can be represented by $O(xS)$. Finally, the complexity of $Expansion(i)$ can be shown to be $O((T - x)(S + A) + x)$.

Since the algorithms $Select_{PB}(i)$, $Moveable(i)$ and $Expansion(i)$ must be executed for $r/T$ times, their complexities should be multiplied by the factor $r/T$. In Table 1, the condition $x < T < r$ can be utilized to simplify the complexity results. Based on Eqs. (15) to (18) and (22), the dominant term will be $O((T - x)(S + A) + x)$. Assume that $A \in O(S)$; then the complexity function $O(rS^2x/T)$ can be simplified to $O(SN'_{VB})$. Therefore, the complexity of the proposed CEA algorithm can be given by $O(SN'_{VB})$.

4.3. Application to Cloud-Based Storage Systems

CEA can be applied not only to the parallel video server system, but also to a variety of distributed storage systems, which employ the data striping and the parity check scheme as in RAID-5. Recently, the use of cloud-based services and applications has been increasing significantly. A thin client, such as a smart phone, can access data from cloud-based storage systems. Inevitably, the number of users and the storage space requirement in a cloud system will continue to increase. Thus, system expansions may be frequent. Undoubtedly, a cost-effective system expansion algorithm like our CEA will be required for cloud-based storage systems in the near future.

5. Performance Evaluation

This section will evaluate the performance of the proposed CEA algorithm and compare it to those of other data placement schemes, such as the Random (RA) and Round-Robin (RR) placement schemes. In the RA scheme, the VBs and PBs for migration are selected at random under the perfect global load balancing constraint. Additionally, the allocation of data in new video servers is also random. In the RR scheme, the VBs and PBs are always distributed to all video servers in a round-robin manner after system expansion. The stripe unit size of each VB is assumed to be 512 kBytes. The number of VBs for the considered video content is 3780. The performance metrics, such as $LB_{1S}$, the number of VB movements, the number of PB movements, and the number of XOR computations, are investigated below.

First, the placement period $N_{VB}$ of the proposed CEA algorithm is examined. In the first instance, one-step expansion is considered: the system is expanded only once and the VBs and PBs are assumed to be striped in a round-robin manner before the system expansion is executed. Figure 11, which displays two curves, shows the resulting CEA periods. The first curve, “Increased by One Each Time”, shows the behavior of the CEA placement period when the system is expanded from $(n - 1)$ to $n$ video servers. The second curve, “Proposed Expansion Sequence”, shows the CEA placement period that results when the system is expanded from $a_{n-1}$ video servers to $a_{n}$ video servers, where $a_{n}$ is the proposed sequence $\{2, 3, 4, 6, 10, 15, 21, 28, 36, \ldots\}$. Obviously, the use of the proposed expansion sequence can reduce the placement period $N_{VB}$. Although the system expansion flexibility of the proposed sequence is limited, this limit can be overcome by appropriate advance system capacity planning.

Next, the effect of iterative expansion of the system on CEA placement period is studied. Initially, the VBs and PBs of the system are allocated to parallel video servers in a round-robin manner. Then, the system is expanded repeatedly by using the proposed CEA. Figure 12 compares two cases, “Increased by One Each Time” and “Proposed Expansion Sequence”. When the system is expanded iteratively, the CEA placement period can become extremely large if the expansion sequence is not properly designed. It is because that after the system is expanded by CEA, the data layout is no longer aligned in a round-robin manner, where the placement period for PBs equals the number of video servers, such that the placement period increases with each expansion. Hence, CEA placement periods grow larger. By contrast, the behavior of the CEA placement period for the proposed expansion sequence is the same.
as that in one-step expansion, even though the system is expanded several times. Thus, it is strongly recommended to choose the opportune moments for system expansions based on the proposed expansion sequence.

The proposed expansion sequence is selected under the constraint of the system expansion rate, $A/S$, which satisfies $0.25 < A/S < 0.75$. The sequence starts from $S = a_0 = 2$ and $N_{VB} = 2$. The new placement period is computed according to the formula $N'_{VB} = [N_{VB}, (S+A)(S+A-1)]$. Given that $S = a_n$, for all $A$’s satisfying the system expansion rate constraint, the one $A^*$ with the minimum placement period is selected. Then, we get $a_{n+1} = a_n + A^*$. Notably, service providers can set a different system expansion rate constraint according to their situations, such that the resulting expansion sequence may be slightly different. Obviously, a larger placement period corresponds to a more meta-data and higher complexity. If the intra-movie skewness is ignored, only one meta-data file is required for our CEA and the RR schemes. Hence, the overhead for storing contents is negligible in the CEA and the RR schemes. However, for the RA scheme, the placement data does not exhibit periodicity. Hence, for the RA scheme, the amount of meta-data of a content will be proportional to the number of VBs and all contents may have different meta-data files.

Subsequently, the expansion costs of the RA and RR data placement methods are compared with those of the proposed CEA. Initially, the VBs and PBs of the system are allocated to parallel video servers in a round-robin manner. Then, the parallel video server system is expanded iteratively. The proposed expansion sequence $\{2, 3, 4, 6, 10, 15, 21, 28, 36\}$ is employed here. Figure 13 indicates the required numbers of VB movements under various data placement algorithms. Figure 14 presents the required numbers of PB movements for different algorithms. In terms of the amount of data movements, RA is the best placement algorithm, since it needs only to randomly select the required numbers of VBs and PBs and migrate them to the new video servers under the perfect global load balancing constraint. However, according to Figs. 13 and 14, the numbers of VB and PB movements in our proposed CEA algorithm are exactly the same as those in the RA scheme, demonstrating that our proposed CEA algorithm is optimal for the system expansion in terms of the number of data movements. This conclusion is consistent with the verified optimality in Section 4.1. One can show that the numbers of VB and PB movements for RA and CEA conform to the values given by Theorem 1. In the RR scheme, since the data blocks must be aligned in a round-robin manner after the system expansion, extra movements of data among the old video servers are required. Therefore, the numbers of VB and PB movements in RR substantially exceed those in CEA and RA placement schemes.

Although RA is an optimal algorithm for system expansion in terms of the amount of data movements, the randomly selected PBs for migration cannot minimize the number of XOR computations. Figure 15 shows the number of required XOR computations during a system expansion. The results obtained using RA in Fig. 15 are the average values from ten independent experiments. The CEA requires many fewer XOR computations than either RA or
RR. This fact is reasonable because in the CEA, all of the VBS in a certain parity check round will be selected for migration and the corresponding PB will be deleted. Section 4.1 has also proved that the number of required XOR computations during a system expansion by using the CEA is optimal. Again, one can find that the number of XOR recomputations for the CEA also conforms to the value given by Theorem 1.

Finally, the LBI<sub>S</sub> performance for different data placement algorithms is considered. The LBI<sub>S</sub> performance of different algorithms is given in Fig. 16. Similarly, in Fig. 16, the results for RA are also average values from ten independent experiments. Undoubtedly, for the RR algorithm, LBI<sub>S</sub> always equals zero. Thus, RR is the optimal algorithm in terms of the LBI<sub>S</sub> metric. The LBI<sub>S</sub> of our CEA is worse than that of RR, but it is better than that of RA. Although our CEA has a worse LBI<sub>S</sub> performance than RR, the incurred LBI<sub>S</sub> degradation, which will result in the transmission jitter, can be eliminated by using a buffer at each video server [11, 21]. Figure 17 illustrates the induced transmission jitter versus the time duration (in service cycles) when the CEA is executed. The transmission jitter is defined as the maximum difference among the cumulative numbers of VBS that are transmitted by individual video servers in a given time duration. For the RR scheme, the maximum transmission jitter equals 1. Figure 17 shows that the transmission jitter is well-controlled and can always reduce to zero periodically in the CEA. Thus, the impact of LBI<sub>S</sub> degradation on the video transmission quality is negligible, compared to the impact of the network jitter. Based on the results presented in this section, the proposed CEA algorithm outperforms the conventional placement algorithms RR and RA.

6. Conclusions

This article introduces an expansion algorithm, CEA, for parallel video servers. The proposed CEA exhibits several features. First, the CEA algorithm has optimal amounts of VB and PB movements. Second, the total number of required XOR computations for updating PBs is also optimized. Finally, the CEA applies the LBI<sub>S</sub> factor, which is proposed and defined in this paper, to improve short-term load balancing performance. Numerical results demonstrate that the efficiency of the proposed CEA algorithm is excellent. Owing to its relatively low expansion cost, the presented CEA can shorten the service interruption time. Therefore, the QoS of a parallel video server system is enhanced significantly. Furthermore, the proposed CEA algorithm can also be applied to any distributed storage systems that use the same technology as RAID-5 or parallel video servers.

Acknowledgments

This work was supported by Chang Gung University under Grant UERPDP270141. The authors would also like to thank all reviewers for their valuable comments and suggestions.
Appendix

A. The proposed algorithm Select_PB(i)

void Select_PB (int i) //Select appropriate PB for deletion
{ int j, l, y, m, h, n, d, t;
    for (j = iT; j < (i + 1)T; j++)
        (l = mod T; 
        \Delta LB[i][l] = LB[i][l] - LB[i][l - 1];
        PB_Priority[i][l] = PB_Priority[i][l - T];
        l = l - 1;
    PB_Priority[l][l] = PB_Priority[i][l - T];
    for (l = 0; l < T; l = l + 1) // PB_Priority[i][l] select PBs based on their priorities
        { d = VS_PB[PB_Priority[i][l]]; 
          if (VS_PB[PB_Priority[i][l]] == VS_PB[PB_min] \&\& N_select[i] < x)
            { m = i + N_select[i]; // executed for x times
              while (m > ix && PB_Priority[i][m] < select[m - 1])
                select[m] = select[m - 1];
            m = m - 1;
            else \{ Sort the indice of PBs selected
                select[m] = PB_Priority[i][l];
                N_select[i] = m + 1;
            } else \{ N_PB[i] = select[i]; // executed for (T - x) times
                Update PB_min, \( PB_{min} = \min \{ N_PB[i] \} \)
            }
            for (j = ix; j < (i + 1); j++)
                y = 0;
            for (l = 0; l < T; l = l + 1)
                (l = \text{if (Allocation[select[j]][l] != -1)}
                    \{ \text{Add Allocation[select[j]][l] to PB_VB[select[j]] by Insertion Sort;}
                    y = y + 1;
                    \}
                )
        }
        for (j = iT; j < (i + 1)T; j++)
            (y = 0;
            for (l = 0; l < T; l = l + 1)
                (l = \text{if (Allocation[select[j]][l] != -1)}
                    \{ \text{Add Allocation[select[j]][l] to PB_VB[select[j]] by Insertion Sort;}
                    y = y + 1;
                    \}
                )
            )
}

B. The proposed algorithm Movable(i)

void Movable (int i)
{ l = ix; count = ix; h = 0;
    for (j = iT; j < (i + 1)T; j++)
        (l = \text{if (P_j \notin B_i,y)} \text{the set of all selected PBs for deletion}
            \{ n = VS_PB[j];
                d = select[j];
                if (M_PB[n] == M_PB_min)
                    M_PB[n]++; \}
            else \{ Movable_PB[j] = 1; \text{ PB_j move to new server k}
                Movable[Allocation[d][n]] = 0; \text{ PB_j not moved}
                Nonmovable[nonmove] = Allocation[d][n];
                nonmove++; \}
                N_PB[n] = n - 1;
                N_PB[k]++; \text{ Move P_i to new server k}
                M_PB[k]++; \text{ PB_i move to new server k}
                k = S + (k - S + 1) mod A; \text{ next new server}
                Update PB_min, \( PB_{min} = \min \{ N_PB[m] \} \)
                Update M_PB_min, \( M_PB_{min} = \min \{ M_PB[m] \} \)
                l = l + \left( \frac{\text{ix}}{2} \right);
                h = (h + A) \text{ mod (S - 1);}
            }
            count++;
            )
    for (j = ix; j < (i + 1)ix; j++)
        (for (m = 0; m < S - 1; m++)
            (if (Movable[PB_VB[select[j]][m]] == 0)
                PB_VB[select[j]][m] = -1; \text{ a PB need not be moved}
            )
        )
}

C. The proposed algorithm Expansion(i)

void Expansion (int i)
{ int i, j, h, n, y, count, k;
    y = i(T - x); \text{execute for (T - x) times}
    j = ix; h = 0; count = ix;
    for (j = iT; j < (i + 1)T; j++)
        (if (j \notin select[count])
            \{ \text{for (k = 0; k < S; k++)}
                \text{Allocation}[y][k] = Allocation[j][k];
                VS_PB[y] = VS_PB[y];
            \}
            n = VS_PB[y]; \text{ Round y after expansion}
            \text{if (Movable_PB[i] == 0) PB not moved}
            \text{for (k = S; k < S + A; k++)}
                (if (PB_VB[select[j]][k] == 0)
                    \{ j + = (h + 1) \text{ mod (S - 1)};
                    h = (h + 1) \text{ mod (S - 1)};
                \}
                else \{ Allocation[y][k] = PB_VB[select[j]][k];
                    j + = (h + 1) \text{ mod (S - 1)};
                    h = (h + 1) \text{ mod (S - 1)};
                 \}
            \}
            else \{ Allocation[y][n] = Nonmovable[stay];
                y++; \text{ executed for x times}
            \}
            \}
}

References


