

A Row-Permutated Data Reorganization Algorithm for Growing Server-less Video-on-Demand Systems

T. K. Ho and Jack Y. B. Lee

Department of Information Engineering

The Chinese University of Hong Kong

Shatin, N.T., Hong Kong

Email: {tkho2@ie.cuhk.edu.hk, jacklee@computer.org}

Abstract

Recently, a new server-less architecture is proposed for building low-cost yet scalable video streaming systems. Compare to conventional client-server-based video streaming systems, this server-less architecture does not need any dedicated video server and yet is highly scalable. Video data are distributed among user hosts and these hosts cooperate to stream video data to one another. Thus as new hosts join the system, they also add streaming and storage capacity to absorb the added streaming load. This study investigates the data reorganization problem when growing a server-less video streaming system. Specifically, as video data are distributed among user hosts, these data will need to be redistributed to newly joined hosts to utilize their storage and streaming capacity. This study presents a new data reorganization algorithm that allows controllable tradeoff between data reorganization overhead and streaming load balance.

1. Introduction

Peer-to-peer and grid computing have shown great potentials in high-performance computing applications. Apart from computational problems, data and I/O-intensive applications can also benefit from the inherent scalability offered by grid-type architectures. One such architecture, called server-less video-on-demand architecture, recently proposed by Lee and Leung [1] adopted this completely decentralized approach to eliminate the need for costly high-capacity video servers.

Unlike conventional video-on-demand (VoD) systems built around the well-understood client-server model, a server-less VoD system is built entirely from user hosts. Video data are distributed among these user hosts which then cooperate to stream video data to one another for playback. Lee and Leung [1] showed that this server-less architecture is easily scalable to hundreds of user hosts

using off-the-shelf computers and network switches. Moreover, by incorporating data and capacity redundancy into the system, one can even achieve system-level reliability comparable to or even exceeding those of dedicated video servers [2].

The study by Lee and Leung [1] is focused on the scalability and feasibility of the server-less architecture. They did not, however, address the practical problem of system growth when new user hosts join the system. Specifically, as video data are distributed among user hosts, these data will need to be redistributed to newly joined hosts to utilize their storage and streaming capacity. We tackle this problem in this study by presenting a new data reorganization algorithm that allows controllable tradeoff between data reorganization overhead and streaming load balance.

We first review the server-less VoD architecture and two previous works on data reorganization in Section 2, and then present the new data reorganization algorithms in Section 3 and 4; Section 5 compares the performance of the studied algorithms and Section 6 concludes the paper.

2. Background

In this section, we first give a brief overview of the server-less VoD architecture proposed by Lee and Leung [1] and then review two existing works on data reorganization.

2.1 The server-less architecture

A server-less VoD system comprises a pool of user hosts, henceforth called nodes, connected by a network as shown in Fig. 1. Each node has its own CPU, memory and disk storage. Inside each node is a mini video server software that serves a portion of each video title to other nodes in the system. Unlike conventional video server, this mini server software serves a much lower aggregate bandwidth and therefore can readily be implemented in today's STBs and PCs. For large systems, the nodes can be

further divided into clusters where each cluster forms an autonomous system that is independent from other clusters.

For data placement, a video title is first divided into fixed-size striping units (or called blocks) and then distributed to all nodes in the cluster in a round-robin manner. This node-level striping scheme avoids data replication while at the same time divides the storage requirement equally among all nodes in the cluster.

To initiate a video streaming session, a receiver node will first locate the set of sender nodes carrying blocks of the desired video title, the striping policy and other parameters (format, bitrate, etc.) through the directory service. These sender nodes will then be notified to start transmitting the video blocks to the receiver node.

Let N be the number of nodes in the cluster and assume all video titles are constant-bit-rate (CBR) encoded at the same bitrate R_v . For a sender node in a cluster, it may have to retrieve video data for up to N video streams, of which $N-1$ of them are transmitted while the remaining one played back locally. Note that as a video stream is served by N nodes concurrently, each node only needs to serve a bitrate of R_v/N for each video stream. With a round-based transmission scheduler, a sender node simply transmits one block to each receiver node in each round. Interested readers are referred to the study by Lee and Leung [1] for more details.

2.2 Previous works on data reorganization

The problem of data reorganization has been studied in the context of disk arrays [3-4]. The study by Ghandeharizadeh and Kim [3] is the earliest study on data reorganization known to the authors. They investigated the data reorganization problem in the context of adding disks to a continuous media server. They employed round-robin data striping common in disk arrays and investigated and analyzed techniques to perform data reorganization online, i.e., without disrupting on-going video streams.

Due to the round-robin placement requirement, a large portion of the data blocks will need to be redistributed to maintain the data placement order when a new disk is added, thus incurring significant data reorganization overhead. Nevertheless, this approach has the distinct advantage of achieving perfect streaming load balance. When applied to the server-less VoD system, the round-robin placement policy enables nodes in the system to simply all transmit one block in each round to achieve streaming load balance.

Given that the need to add new disks to a video server occurs only sparingly, the tradeoff in reorganization overhead to achieve perfect load balance is well justified. By contrast, in a server-less VoD system with hundreds of

nodes, the frequency at which new nodes joining the system will be significantly higher. Thus the same data reorganization algorithm may incur too much overhead for use in a server-less VoD system.

In a more recent study by Goel et al. [4], a pseudo-random algorithm called SCADDAR for data placement and data reorganization is proposed for use in disk arrays. In this algorithm, each data block is initially randomly distributed to the disks with equal probabilities. When a new disk is added to the disk array, each block will obtain a new sequence number according to their randomized SCADDAR algorithm. If the remainder of this number is equal to the disk number of the newly added disk, the corresponding block will be moved to this new disk. Otherwise, the block will reside at the original disk.

As SCADDAR no longer needs to maintain a fixed round-robin placement order, it can reduce the reorganization overhead significantly to approach the theoretical lower bound. However, the authors did not consider streaming load balance. If we apply SCADDAR to the server-less VoD system that can grow to hundreds to thousands of nodes, our results reveal that it can result in significant streaming load imbalance, especially after a large number of nodes are added to the system. For example, in the same service round some nodes may need to transmit more than one block while some other nodes are idle. This load imbalance makes data transmission scheduling more difficult and may reduce the streaming capacity and/or the response time of the system.

The previous two pioneering studies can be considered as two extremes of the tradeoff between data reorganization overhead and load balance. In particular, Ghandeharizadeh and Kim's algorithm achieves perfect load balance at the expense of substantial data reorganization overhead; while the SCADDAR algorithm achieves near-minimal data reorganization overhead at the expense of load imbalance. In the next section, we present a new data reorganization algorithm that can achieve perfect streaming load balance and yet incurs significantly lower reorganization overhead than the round-robin algorithm, and in Section 4 we further generalize this algorithm to allow one to strike a balance between data reorganization overhead and streaming load balance.

3. Row-permuted data reorganization

We present a new row-permuted data reorganization (RPDR) algorithm in this section. Section 3.1 describes the placement policy while Section 3.2 explains the algorithm.

3.1 Placement policy

As Ghandeharizadeh and Kim's study [3] showed, the

data reorganization overhead incurred in maintaining the round-robin data placement order is very high. Therefore we replace the round-robin placement policy in the original server-less VoD architecture by a row-permuted placement policy.

Specifically, a video title is again divided into fixed-size blocks. With a N -node cluster, the first N video blocks will be distributed to all N nodes in random order, with each node storing exactly one of the N video blocks as shown in Fig. 2a. This process repeats for the next N video blocks and so on until all video blocks are distributed.

It is easy to see that this row-permuted placement policy achieves perfect streaming load balance same as the original round-robin placement policy. Note that the receiver node does not need to know the exact placement order in each row as the sender nodes all stream video blocks to the receiver node concurrently [1]. As long as each video packet carries a sequence number relative to the video stream, the receiver node can then re-sequence the incoming packets for playback.

3.2 Data reorganization

Assuming the system has one cluster. Let B be the total number of fixed-size blocks of a video title, and N be the number of nodes in the system before the addition of a new node. We use n to denote the current system size and thus initially $n=N$. We denote block i of row j by $v_{i,j}$, where $i=0,1,\dots,(B/n)-1$, and $j=0,1,\dots,n-1$, or simply the $(in+j)^{\text{th}}$ block of the video title.

After adding a new node to the system, n is increased to $N+1$, and data reorganization is needed to redistribute video data and streaming load to the newly added node. We first re-index the video blocks $v_{i,j}$, using $n=N+1$. For example, $v_{1,0}$ and $v_{1,1}$ will become $v_{0,N}$ and $v_{1,0}$ respectively after re-indexing (see Fig. 2b for an example).

Next, we consider the re-indexed blocks in a row-by-row manner to identify block overflows and block underflows. Block overflows occur in a node when more than one block from the same row resides in the node; and block underflows occur in a node when none of the blocks from the row resides in the node.

When a block overflow is detected (e.g. block $v_{0,4}$, for row 0 and blocks $v_{1,3}$, $v_{1,4}$ for row 1), then the overflow blocks will be redistributed to nodes experiencing block underflows (e.g. moving block $v_{0,4}$ to node n_4). Note that the choice of target nodes to receive an overflow block can be arbitrary as we no longer need to maintain a round-robin data placement order. It is easy to see that the number of overflows equal to the number of underflows and thus after reorganization each node will store exactly one block from a row as shown in Fig. 2d. As a result, this row-permuted reorganization algorithm can achieve

perfect storage balanced and streaming load balance.

Compared to the round-robin placement policy, this algorithm has significantly lower reorganization overhead and at the same time, can still achieve perfect load balance. In the next section, we relax the perfect load balance constraint to further reduce the reorganization overhead.

4. Multi-row-permuted data reorganization

While perfect streaming load balance is desirable, the cost of data reorganization, which itself consumes system resources, can still be substantial. Depending on the particular system configuration (e.g. disk throughput, network bandwidth, system utilization, etc.), it may be desirable to tradeoff some streaming load balance to reduce the data reorganization overhead.

To tackle this challenge, we generalize the row-permuted data reorganization algorithm into a multi-row-permuted data reorganization (mRPDR) algorithm. Specifically, a window size, denoted by w , is used to configure the number of rows to consider when identifying block overflows and underflows. Block overflows are redefined to occur only if more than w blocks from the w rows under consideration reside in the same node; and block underflows are redefined to occur only if fewer than w blocks from the w rows under consideration reside in the same node.

Similarly, the overflow blocks in an overflow node will be moved to the underflow nodes. However, unlike the RPDR algorithm, the choice of which overflow blocks to redistribute and which underflow nodes to receive the overflow blocks will affect the streaming load balance of the system. Fig. 3 illustrates this issue by comparing different choices of overflow blocks and underflow nodes with $w=2$.

Fig. 3a shows the placement of blocks after re-indexing. The overflow nodes are n_0 and n_2 while the underflow nodes are n_1 , n_3 and n_4 . Fig. 3b shows one example of poor selection of underflow nodes. Note that blocks $v_{i,4}$ and $v_{i,1}$ belonging to the same row are stored to the same node n_1 , thus forcing this node to stream out both these two blocks in a round. Fig. 3c shows another example of poor selection of overflow blocks. Note that blocks $v_{i,0}$ and $v_{i,1}$ belonging to the same row are again stored to the same node n_0 . By contrast, the case in Fig. 3d can achieve better streaming load balance as blocks belonging to the same row are now evenly distributed to all the nodes. Note that the order of the blocks within a node is not important as the node will independently retrieve a block from the disk for transmission in each service round (c.f. Section 2.1).

Fig. 4 lists an algorithm to perform this load balanced data reorganization. The algorithm proceeds in iterations,

each time considering one excess block. For each excess block in each overflow node, the algorithm will first find the row, denoted by $A_{i,j}$, containing the largest number of blocks inside this node (Step 12). Then it locates an underflow node (Step 13) and move one overflow block to the underflow node (Step 14). If there are more than one underflow nodes satisfying the criteria, it will be chosen randomly. This process then repeats for each overflow block in each overflow node.

Clearly increasing the window size w will decrease the reorganization overhead at the expense of streaming load imbalance. The RPDR algorithm presented in Section 3 is a special case of the mRPDR algorithm with a window size of $w=1$. At the other extreme, setting w to B/n will result in minimal reorganization overhead but with significant streaming load imbalance. We investigate in the next section tradeoff between reorganization overhead and streaming load balance.

5. Performance comparisons

In this section, we evaluate and compare the proposed multi-row-permuted data reorganization algorithm with the round-robin [3] and the SCADDAR [4] algorithms originally proposed for disk arrays. The primary performance metrics used for comparison are data reorganization overhead and streaming load balance.

The results are computed numerically for a video title with $B=4,000$ blocks. Unless stated otherwise the system begins with a single node and then incrementally grows to 200 nodes by adding new nodes one by one. Each data point is averaged over 50 results obtained from using different random seeds for the random number generator (used in SCADDAR and the mRPDR algorithms).

5.1 Data reorganization overhead

Data reorganization overhead is defined as the number of data blocks that need to be redistributed during data reorganization. This metric can reflect the time, memory and bandwidth requirement incurred by the reorganization process.

First, for a system with B blocks already evenly distributed to $n-1$ nodes, the minimum number of blocks that need to be redistributed when a new node is added is simply equal to B/n , provided that perfect storage balance is to be maintained. It is possible to achieve even lower reorganization overhead (e.g. in SCADDAR) but then some of the nodes will store and serve more data blocks than others.

Next we derive the reorganization overhead for round-robin placement. We observe that the i^{th} block and the $(i+\text{LCM}(n-1,n))^{\text{th}}$ block, where the function

$\text{LCM}(n-1,n)$ computes the least common multiple of $n-1$ (original cluster size) and n (new cluster size), must reside on the same node for all i so these blocks do not need to be redistributed. As only one node is added each time, we have $\text{LCM}(n-1,n)=n(n-1)$. The overhead of round-robin reorganization is thus approximately equal to $B(1-(1/\text{LCM}(n-1,n)))=B(n^2-n-1)/(n^2-n)$.

For the SCADDAR algorithm the reorganization overhead must be measured but it has been shown to approach the theoretical lower bound of B/n [4]. Similarly the reorganization overheads of the proposed algorithms are also measured and the results are plotted in Fig. 5.

There are three key observations. First, the round-robin algorithm and the SCADDAR algorithm have the highest and lowest reorganization overhead respectively. Moreover, the differences increase when the system grows larger. Second, the reorganization overhead of the mRPDR algorithm varies within the top (round-robin) and the bottom (SCADDAR) curves depending on the parameter w , showing the tunable range of the mRPDR algorithm. Third, for mRPDR with $w=1$, which achieves perfect streaming load balance, the reorganization overhead is still significantly lower than the round-robin algorithm. As maintaining the round-robin placement order offers no advantage in a server-less VoD system (c.f. Section 2.1), the RPDR algorithm should be used in place of round-robin when perfect streaming load balance is required.

Another shortcoming of SCADDAR is the unpredictable streaming imbalance. In particular, we can determine the worst case streaming load imbalance for a video stored using SCADDAR only after the reorganization is completed. This may incur additional complexity in scheduling as the amount of load imbalance is unpredictable. By contrast, the mRPDR algorithm simply guarantees that the load imbalance will always be bounded from the above by w . Thus enabling a system designer to incorporate this streaming load imbalance into the system's admission policy and scheduling algorithm.

So far we have assumed that the system is reorganized whenever a new node joins the system. Clearly this is inefficient for systems where new nodes are frequently added. Instead, we can wait until there are a number of nodes, say k , added before performing reorganization. Fig. 6 shows the per-node reorganization overhead for $w=1$ and k ranging from 1 to 5. We observe that the per-node reorganization overhead does decrease significantly for larger value of k . However, this is done at the expense of resource utilization as resources in the newly added nodes cannot be utilized until data reorganization is completed. Thus there is a tradeoff between reorganization overhead and resource utilization, and further investigation is warranted to quantify the tradeoff in terms of the other

system parameters.

5.2 Streaming load balance

To evaluate streaming load balance, recall that the sender nodes transmit one video block per round per video stream (c.f. Section 2.1). Thus if there are more than one block of a row residing in a node, this node may experience scheduling overflow during transmission, depending on the utilization of the system. Therefore counting the number of such overflow blocks will give an indicator on the degree of load imbalance.

Fig. 7 plots the proportion of blocks that are overflow blocks for the various data reorganization algorithms. As expected, both round-robin and RPDR achieve zero overflow, i.e., perfect streaming load balance. Surprisingly, the SCADDAR algorithm results in over 35% overflow blocks. By contrast, the mRPDR algorithm can achieve different levels of block overflow using different window size. This enables the system designer, knowing the system configurations, to choose the best tradeoff between reorganization overhead and streaming load balance.

6. Conclusion and future works

In this study, we investigated the problem of data reorganization when growing a server-less VoD system. We found that the round-robin and the SCADDAR algorithms are two extremes in the tradeoff between reorganization overhead and streaming load balance. We presented a new RPDR algorithm that can achieve perfect streaming load balance as the round-robin algorithm and yet required significantly less reorganization overhead. We then generalize this to a multi-row-permuted data reorganization (mRPDR) algorithm that can further allow

the system designer to control the tradeoff between reorganization overhead and streaming load balance.

This study is a small step in studying the larger problem of data reorganization in grid-based storage systems in general, and server-less VoD system in particular. Some of the open problems include how to integrate the cost of data reorganization and the cost of streaming load imbalance into a unified optimization framework to determine the optimal configuration of the mRPDR algorithm; how to transparently perform data reorganization without disrupting on-going video streams; how to perform data reorganization with heterogeneous nodes with varying storage and streaming capacity; how to support node-level fault tolerance to improve system reliability; and how to shrink a system when nodes leave the system.

References

- [1] Jack Y. B. Lee and W. T. Leung, "Study of a Server-less Architecture for Video-on-Demand Applications," *Proc. IEEE International Conference on Multimedia and Expo.*, August 2002.
- [2] Jack Y. B. Lee and W. T. Leung, "Design and Analysis of a Fault-Tolerant Mechanism for a Server-Less Video-On-Demand System," *Proc. 2002 International Conference on Parallel and Distributed Systems*, Taiwan, Dec 17-20, 2002.
- [3] S. Ghandeharizadeh and D. Kim, "On-line reorganization of data in scalable continuous media servers," *Proc. 7th International Conference on Database and Expert Systems Applications*, September 1996.
- [4] A. Goel, C. Shahabi, S.-Y. Yao, and R. Zimmerman, "SCADDAR: An efficient randomized technique to reorganize continuous media blocks," *Proc. International Conference on Data Engineering*, 2002.

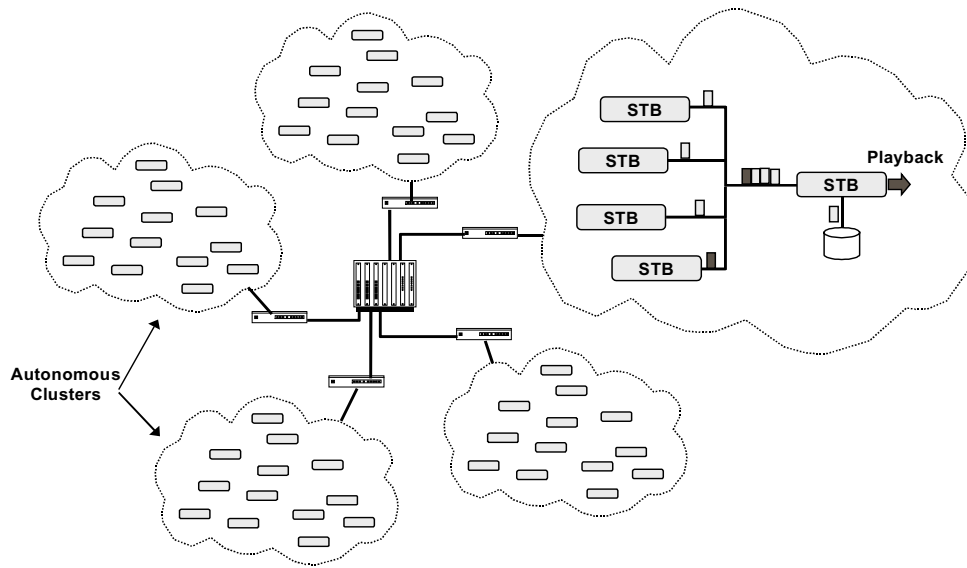


Figure 1. A server-less architecture for video streaming.

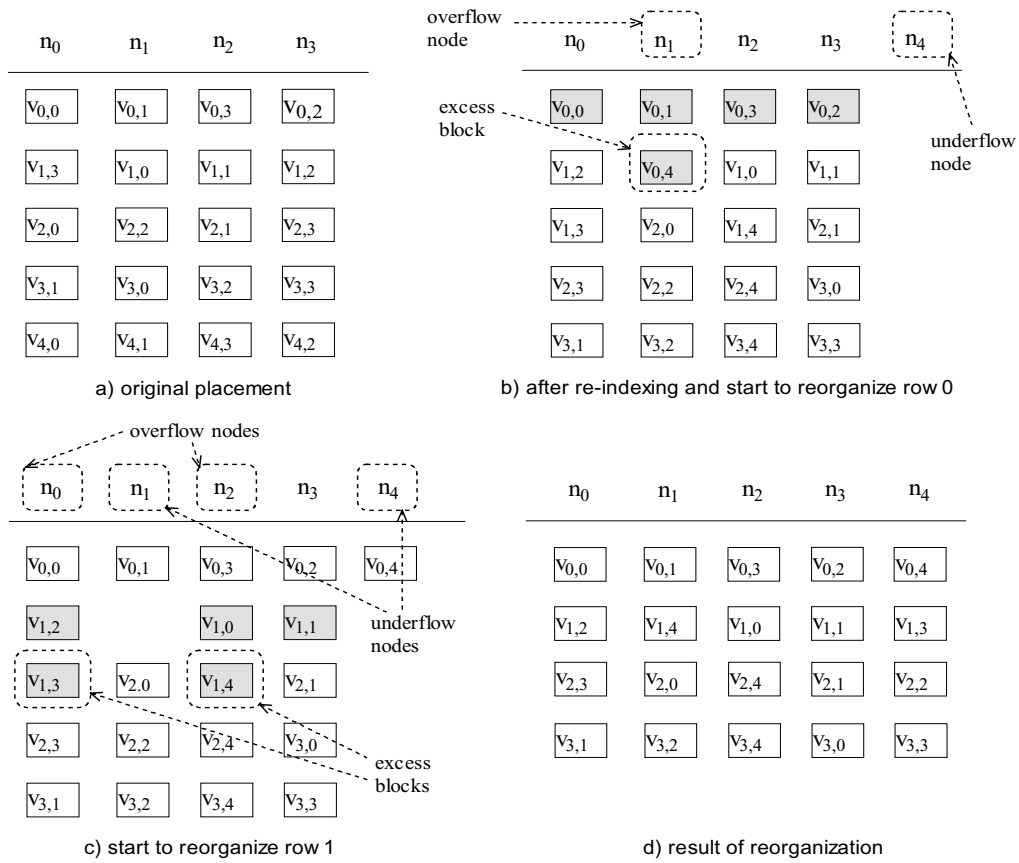


Figure 2. The row-permuted data reorganization algorithm.

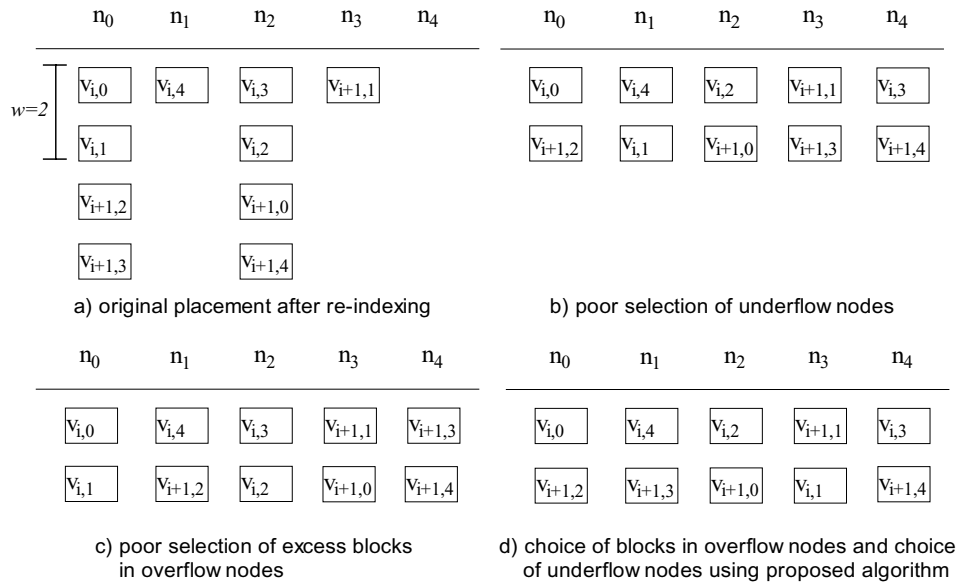


Figure 3. The multi-row-permuted data reorganization algorithm.

```

Step 01:  $R_0, R_1, \dots, R_{w-1}$  is the rows under consideration
Step 02:  $n_0, n_1, \dots, n_{n-1}$  is the nodes in the system
Step 03:  $A_{x,y}$  is the number of blocks of row  $R_x$  in node  $n_y$ 
Step 04:  $Y$  is the set of overflow nodes
Step 05:  $Y \leftarrow \emptyset$ 
Step 06: for (int  $y=0$  to  $n-1$ ) {
Step 07:   if  $\sum_{x=0}^{w-1} A_{x,y} > w$ 
Step 08:     then add node  $n_y$  to the set  $Y$ 
Step 09: }
Step 10: for each  $n_y \in Y$  {
Step 11:   for each overflow block in  $n_y$  {
Step 12:      $A_{i,y} \leftarrow \max(A_{0,y}, A_{1,y}, \dots, A_{w-1,y})$ 
Step 13:     find  $y'$  such that  $(A_{i,y'} < A_{i,y})$  AND  $(\sum_x A_{x,y'} < w)$ 
                    AND  $(A_{i,y'}$  is minimum within underflow nodes)
Step 14:     Move one block of  $R_i$  in  $n_y$  to  $n_{y'}$ 
Step 15:      $A_{i,y} \leftarrow A_{i,y} - 1$ 
Step 16:      $A_{i,y'} \leftarrow A_{i,y'} + 1$ 
Step 17:   }
Step 18: }

```

Figure 4. Pseudo-code for the multi-row-permuted data reorganization algorithm.

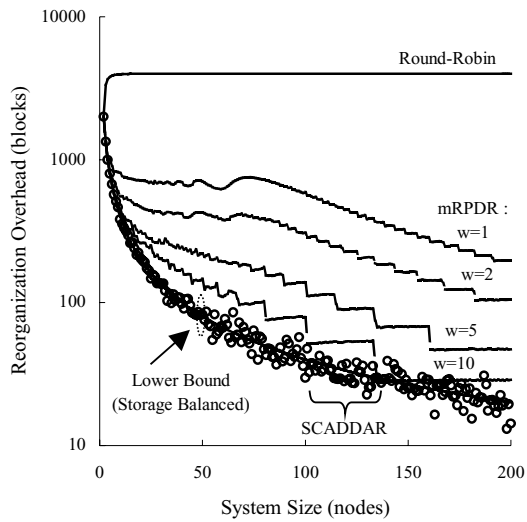


Figure 5. Comparison of reorganization overhead versus system size.

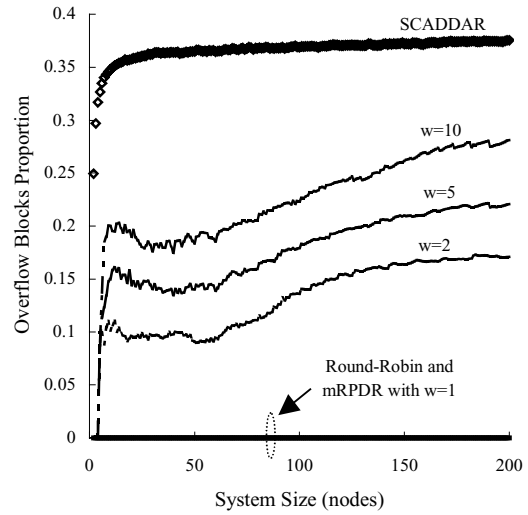


Figure 7. Comparison of overflow blocks proportion versus system size.

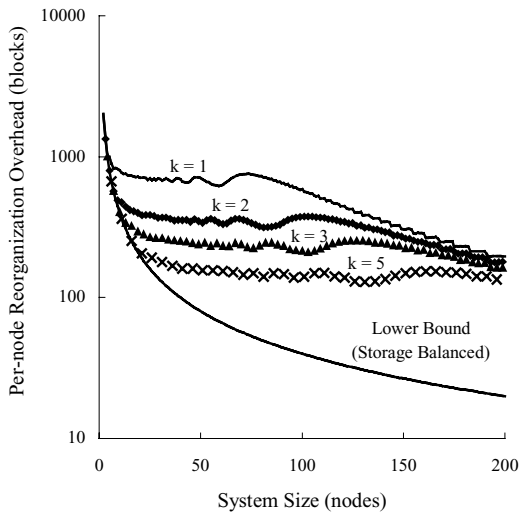


Figure 6. Comparison of per-node reorganization overhead versus system size.