A NOVEL REDUNDANT DATA UPDATE ALGORITHM FOR FAULT-TOLERANT 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. In this architecture, video data are distributed among user hosts and these hosts cooperate to stream video data to one another. To improve reliability, data and capacity redundancy are introduced to sustain node failures. However, the data placement as well as the redundant data in the system will need to be updated whenever new nodes join the system. This study is a first step in investigating the problem of updating redundant data when growing such a server-less system by assimilating new nodes. Results show that the redundancy update overhead is very significant and even exceeds that in data reorganization. To tackle this problem, this study presents a novel Sequential Redundant Data Update (SRDU) algorithm that takes advantage of the structure of Reed-Solomon Erasure Correction codes to reduce the redundancy update overhead by as much as 75%. Numerical results show that by further delaying the update of redundant data until adding multiple nodes, say 10, we can further reduce the redundancy update overhead by as much as 97%.

Keywords: Server-less, video-on-demand, redundant, update, reliable.

1. INTRODUCTION

Peer-to-peer and distributed computing has 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 distributed architectures. One such architecture, called server-less video-on-demand architecture, recently proposed by Lee and Leung [Lee and Leung, 2002a] 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 [Lee and Leung, 2002a] 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 [Lee and Leung, 2002b].

The study by Lee and Leung [Lee and Leung, 2002a] 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. This problem has been investigated by Ghandeharizadeh and Kim [Ghandeharizadeh and Kim, 1996], Goel et al. [Goel et al, 2002], and Ho and Lee [Ho and Lee, 2003] respectively. Nevertheless, all three studies are focused on the reorganization of the video data. The problem of updating redundant data that are themselves computed from the video data has not been addressed.

This work was supported in part by the Hong Kong Special Administrative Region Research Grant Council under a Direct Grant, Grant CUHK4211/03E, and the Area-of-Excellence in Information Technology.

 ISBN 3-936150-25-7

In this study, we investigate the problem of efficient update of redundant data when video data are reorganized during the growth of a server-less VoD system. We found that updating redundant data can incur significantly more overhead than data reorganization. To tackle this problem, we are going to present a new redundant data update algorithm called Sequential Redundant Data Update that takes advantage of the structure of Reed-Solomon erasure codes [Plank, 1997] to reduce the redundant data update overhead by as much as 75%. Numerical results show that by further delaying the update of redundant data until adding multiple nodes, say 10, we can further reduce the redundancy update overhead by as much as 97%.

In the next section, we first briefly review the server-less VoD architecture and the previous works on data reorganization. We formulate the data reorganization problem in Section 3. The redundant data regeneration and proposed update algorithm are presented in Section 4 and 5 respectively; Section 6 gives the performance evaluation and Section 7 concludes the paper.

2. BACKGROUND

In this section, we first give a brief overview of the server-less VoD architecture [Lee and Leung, 2002a] and then review the existing works on data reorganization.

2.1 Server-less VoD Architecture

A server-less VoD system comprises a pool of fully connected user hosts, or called nodes in this paper. Inside each node is a system software that can stream a portion of each video title to as well as playback video received from other nodes in the system. Unlike conventional video server, this system software serves a much lower aggregate bandwidth and thus can readily be implemented in today's set-top boxes (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 blocks and then equally distributed to all nodes in the cluster. This node-level striping scheme avoids data replication while at the same time share the storage and streaming 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 placement of the data blocks and other parameters (format, bitrate, etc.) through the directory service. These sender nodes will then be notified to start streaming the video blocks to the receiver node for playback.

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. A sender node in a cluster 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 of video data to each receiver node in each round. Interested readers are referred to the study by Lee and Leung [Lee and Leung, 2002a; Lee and Leung, 2002a] for more details.

2.2 Related Works

The problem of data reorganization has been studied in the context of disk arrays [Ghandeharizadeh and Kim, 1996; Goel et al, 2002]. The study by Ghandeharizadeh and Kim [Ghandeharizadeh and Kim, 1996] 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. However, their study assumed there is no data redundancy in the disk array and thus did not address the redundancy update problem.

In another study by Goel et al. [Goel et al, 2002], a pseudo-random algorithm called SCADDAR for data placement and data reorganization was 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 reminder 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.

In a recent study [Ho and Lee, 2003], Ho and Lee proposed a more efficient data reorganization

algorithm called Row-Permutated Data Reorganization that can achieve lower data reorganization overhead and also allow controllable tradeoff between streaming load balance and data reorganization overhead.

While the previous pioneering studies have been successful in reducing the data reorganization overhead substantially, they did not yet address the issue of redundant data update. Given that a server-less VoD system is built from user hosts that are inherently less reliable than dedicated video servers, fault tolerant capability clearly becomes a necessity. To this end, one will need to incorporate data and capacity redundancies into the system and these redundant data will need to be updated whenever new nodes are added. To our knowledge this study is the first attempt at tackling this redundant data update challenge. Our study reveals that the overhead incurred in updating these redundant data far exceeds even the overhead in data reorganization.

3. OVERHEADS IN DATA REORGANIZATION

Based on the server-less VoD architecture presented in Section 2.1, we formulate the system model in this section and present the three types of overhead in reorganizing data to accommodate newly added nodes. Let B be the total number of fixed-size video data blocks in the system and v_j be the j^{th} block of the video title. For simplicity we consider only one video title although the results can be readily extended to multiple video titles.

Fig. 1 illustrates one possible placement of video data in a server-less VoD system. Each block in the figure represents either a Q-byte video data or a Q-byte redundant data block. Blocks under the same column are stored in the same node. The j^{th} redundant data block, denoted by $c_{i,j}$, are computed from video data stripe i, comprising blocks $\{v_k, k=i(N-h), i(N-h)+1, \ldots, (i+1)(N-h)-1\}$, using a systematic erasure-correction code such as the Reed-Solomon Erasure Correction (RSE) code [Plank, 1997]. Briefly speaking, with h redundant data blocks in a data stripe, the system will be able to sustain the failure of up to h nodes without loosing any data. A previous study [Lee and Leung, 2002b] had shown that one can achieve system-level reliability comparable to high-end dedicated video server with redundancies of $h/(N-h) \approx 0.2$.

When one or more new nodes join the system, they will add both streaming load as well as capacity to the system. There are three types of overhead in assimilating these new nodes into the system. First, to utilize their streaming and storage capacity, the system will need to redistribute portion of the video data to these new nodes. The system may also need to reorganize video data in the existing nodes to maintain streaming load balance [Ho and Lee, 2003]. This *data reorganization process* incurs overhead in the form of relocating data blocks within nodes in the system.

Second, as these redundant data are computed from the data stripe, relocation of the data blocks will require corresponding update to the redundant data blocks. This *redundant data update process* incurs overhead in transmitting data blocks to the nodes for regenerating the redundant data blocks.

Third, as the system grows larger with more nodes, the system reliability will decrease if the number of redundancies h is kept constant. To improve reliability, we will need to introduce new redundancies to the system (i.e., increasing h). This *redundant data addition process* incurs overhead in transmitting data blocks to the nodes for generating the new redundant data blocks.

To our knowledge, only the data reorganization process has been investigated [Ho and Lee, 2003; Ghandeharizadeh and Kim, 1996; Goel et al, 2002]. In this study, we investigate the redundant data update process and leave the redundant data addition process for future work. Common to all three processes, the goal is to minimize the overhead incurred when new nodes are assimilated into the system.

4. REDUNDANT DATA REGENERATION

For a general systematic erasure-correction code in a system with N nodes and h redundancies, we will need all $(N-h)$ data blocks in a stripe to compute the corresponding h redundant data blocks. As individual data and redundant blocks of a stripe are all stored in different nodes, the data blocks will all need to be transmitted to the redundant nodes (i.e., nodes storing the redundant data blocks) for regenerating the new redundant data blocks.

Therefore for a system with B data blocks, a total of B blocks will need to be transmitted to and received by the redundant node to support redundant data regeneration. Clearly this overhead is very significant and worst, increases with the system scale and level of redundancies.

On the other hand, if a central archive server storing all

video data is available in the system, then it can simply regenerate the new redundant data blocks locally and send them to the redundant nodes to replace the old redundant data blocks. In this case, the number of blocks sent will be reduced by a factor of (N–h) to (B/(N–h)). Nevertheless maintaining this central archive server will incur its own costs, and depending on applications, may not be desirable or even feasible.

Reconsidering the generation of a redundant data block from a data stripe, we can observe that in most cases, the reorganized data stripe still comprises many data blocks from the old data stripe before reorganization. For example, in growing a system from N nodes to N+1 nodes, the first data stripe will be reorganized from the composition of $\{v_0, v_1, \ldots, v_{N-h-1}\}$ to $\{v_0, v_1, \ldots, v_{N-h-1}, v_{N-h}\}$, which differs by only one data block v_{N-h}. This motivates us to investigate techniques to reuse the old redundant block to compute the new redundant block such that only a portion of the data stripe will be needed.

5. SEQUENTIAL REDUNDANT DATA UPDATE

Among different erasure correction codes there is a class of codes called linear systematic block erasure correction codes, with the Reed-Solomon Erasure Correction code being one well-known example. One key property of linear systematic block codes is the use of strictly linear matrix multiplications in computing the redundant data, and this very property enables us to reuse original redundant data to compute the updated redundant data.

Specifically, let (N-h) and h be the number of data nodes and redundant nodes in the system respectively. Assuming the number of redundant nodes in the system is fixed, then we can apply the (N, h)-RSE code to compute the h redundant data blocks from each stripe of (N-h) data blocks using

$$F \cdot D = \begin{bmatrix} f_{1,1} & f_{1,2} & f_{1,3} & \cdots & f_{1,N-h} \\ f_{2,1} & f_{2,2} & f_{2,3} & \cdots & f_{2,N-h} \\ \vdots & \vdots & \vdots & & \vdots \\ f_{h,1} & f_{h,2} & f_{h,3} & \cdots & f_{h,N-h} \end{bmatrix} \begin{bmatrix} d_{i,0} \\ d_{i,1} \\ \vdots \\ d_{i,N-h-1} \end{bmatrix}$$
$$= \begin{bmatrix} 1 & 1 & 1 & \cdots & 1 \\ 1 & 2 & 3 & \cdots & N-h \\ \vdots & \vdots & \vdots & & \vdots \\ 1 & 2^{h-1} & 3^{h-1} & \cdots & (N-h)^{h-1} \end{bmatrix} \begin{bmatrix} d_{i,0} \\ d_{i,1} \\ \vdots \\ d_{i,N-h-1} \end{bmatrix}$$
$$= \begin{bmatrix} c_{i,0} \\ c_{i,1} \\ \vdots \\ c_{i,h-1} \end{bmatrix} = C \qquad (1)$$

where the F, D, and C are the Vandermonde matrix [Plank, 1997], the video data vector, and the redundant data vector respectively; and $d_{i,j}$, $c_{i,k}$ represent data block j (j=0,1,…,N–h–1) and redundant block k (k=0,1,…,h–1) of stripe i respectively. Elements in F is computed from $f_{i,j} = j^{i-1}$ and are constants. Note that the matrix multiplication in (1) is computed over Galois Fields of 2^w where $N < 2^w$. For example, by setting w=16 then the code can support up to 65,535 nodes.

In the following sections, we present a novel Sequential Redundant Data Update (SRDU) algorithm comprising three techniques, namely Reuse of Original Redundant Data, Parity Group Reshuffling, and Reuse of Transmitted Data, to substantially reduce the redundancy update overhead.

5.1 Reuse of Original Redundant Data

To illustrate how original redundant data can be reused, consider the examples in Fig. 1 and 2, which represent respectively the system configuration before and after the addition of one new node. In the original configuration in Fig. 1, there are 4 data nodes and 2 redundant nodes. Now the first two original redundant data in redundant node r_1, denoted by $c_{0,1}$ and $c_{1,1}$, are computed from

$$c_{0,1} = \sum_{j=0}^{3} f_{2,j+1} v_j \qquad (2)$$

and

$$c_{1,1} = \sum_{j=4}^{7} f_{2,j-4+1} v_j \qquad (3)$$

according to (1).

After a new node is added, the system configuration will be changed to that in Fig. 2. Now the two new redundant data block, denoted by $c'_{0,1}$ and $c'_{1,1}$, are computed from

$$c'_{0,1} = \sum_{j=0}^{4} f_{2,j+1} v_j \tag{4}$$

and

$$c'_{1,1} = \sum_{j=5}^{9} f_{2,j-5+1} v_j \tag{5}$$

Comparing (4) with (2) we can observe that they share four common terms in $v_j - v_0$, v_1, v_2, v_3. Thus we can rewrite (4) as follows:

$$\begin{aligned} c'_{0,1} &= \sum_{j=0}^{3} f_{2,j+1} v_j + f_{2,5} v_4 \\ &= c_{0,1} + f_{2,5} v_4 \end{aligned} \tag{6}$$

In other words, we can compute $c'_{0,1}$ using the original redundant data $c_{0,1}$ plus data block v_4. Therefore instead of sending all five data blocks to redundant node r_1, we now only need to send one data block, i.e., v_4, thereby dramatically reducing the overheads in updating the redundant data $c'_{0,1}$.

5.2 Parity Group Reshuffling

In some cases, the previous straightforward reuse technique cannot be applied due to differences in the coefficients $f_{i,j}$. For example, $c'_{1,1}$ is computed from v_5 to v_9 and share common terms in v_5, v_6, and v_7 with $c_{1,1}$. Thus it may appears that we can reuse the common terms and send only v_8 and v_9 to r_1 to compute $c'_{1,1}$. However, analyzing the equation for $c'_{1,1}$ –

$$\begin{aligned} c'_{1,1} &= \sum_{j=5}^{9} f_{2,j-5+1} v_j \\ &= \left(f_{2,1} v_5 + f_{2,2} v_6 + f_{2,3} v_7 \right) + f_{2,4} v_8 + f_{2,5} v_9 \end{aligned} \tag{7}$$

and for $c_{1,1}$ –

$$\begin{aligned} c_{1,1} &= \sum_{j=4}^{7} f_{2,j-4+1} v_j \\ &= f_{2,1} v_4 + \left(f_{2,2} v_5 + f_{2,3} v_6 + f_{2,4} v_7 \right) \end{aligned} \tag{8}$$

we found that the common terms v_5, v_6, and v_7 now have different coefficients $f_{i,j}$ (e.g., $f_{2,1} v_5$ versus $f_{2,2} v_5$). As a result, we cannot reuse $c_{1,1}$ in computing $c'_{1,1}$.

To tackle this problem, we propose to reshuffle the order of computations for $c'_{1,1}$ to

$$c'_{1,1} = f_{2,1} v_8 + \left(f_{2,2} v_5 + f_{2,3} v_6 + f_{2,4} v_7 \right) + f_{2,5} v_9 \tag{9}$$

thus enabling us to reuse $c_{1,1}$ in the computation:

$$\begin{aligned} c'_{1,1} &= f_{2,1} v_8 + \left(\sum_{j=4}^{7} f_{2,j-4+1} v_j - f_{2,1} v_4 \right) + f_{2,5} v_9 \\ &= f_{2,1} v_8 + \left(c_{1,1} - f_{2,1} v_4 \right) + f_{2,5} v_9 \end{aligned} \tag{10}$$

This reduces the number of data block transmissions from 5 to 3. Note that the receiver node will also need to use the reshuffled order to correctly decode the parity group. This parity group order information can either be generated dynamically, or simply be sent along the video data blocks.

Interestingly, there may be more than one way to reuse redundant block in updating the redundant data, and possibly with different redundant update overhead. For example, consider the redundant generation function for $c_{2,1}$:

$$\begin{aligned} c_{2,1} &= \sum_{j=8}^{11} f_{2,j-8+1} v_j \\ &= (f_{2,1} v_8 + f_{2,2} v_9) + f_{2,3} v_{10} + f_{2,4} v_{11} \end{aligned} \tag{11}$$

If we reshuffle the order of computations for $c'_{1,1}$ to

$$c'_{1,1} = (f_{2,1} v_8 + f_{2,2} v_9) + f_{2,3} v_5 + f_{2,4} v_6 + f_{2,5} v_7 \tag{12}$$

then we can reuse $c_{2,1}$ in the computation:

$$\begin{aligned} c'_{1,1} &= (\sum_{j=8}^{11} f_{2,j-8+1} v_j - f_{2,3} v_{10} + f_{2,4} v_{11}) \\ &\qquad + f_{2,3} v_5 + f_{2,4} v_6 + f_{2,5} v_7 \\ &= (c_{2,1} - f_{2,3} v_{10} + f_{2,4} v_{11}) \\ &\qquad + f_{2,3} v_5 + f_{2,4} v_6 + f_{2,5} v_7 \end{aligned} \tag{13}$$

However, in this case the number of data block transmissions is 5, which is two blocks more than that of reusing $c_{1,1}$. Thus in the SRDU algorithm, the system will first compute the redundancy update overhead for all reusable redundant blocks and select the one with the lowest overhead for reuse.

5.3 Reuse of Transmitted Data

A third way to reduce overhead is to reuse data blocks already transmitted to a redundant node in computing another redundant data. Reconsidering the previous example in computing $c'_{1,1}$, the data blocks needed are v_4, v_8, and v_9. However, v_4 has already been sent to the redundant node when computing $c'_{0,1}$ (c.f. Equation (6)) and thus can simply be reused. As a redundant

block is computed from a stripe of (N-h) data blocks, we need to cache at most (N-h) data blocks from previous updates at the redundant node.

6. PERFORMANCE EVALUATION

In this section, we evaluate the Sequential Redundant Data Update algorithm using simulation. Beginning with a small system, we add new nodes to the system and then apply the SRDU algorithm to update the redundant data blocks. Performance is measured by the number of data blocks that need to be sent to the redundant nodes – or simply called redundancy update overhead. The total number of data blocks is 40,000 and is fixed throughout the simulation. For simplicity the redundancy update overhead for updating one redundant node is presented. For systems with more than one redundant node, the total overhead is simply multiplied by the number of redundant nodes.

6.1 Redundancy Update Overhead in Continuous System Growth

In the first experiment, we begin with a system of five data nodes and one redundant node. Then we add a new node to the system one by one, each time the redundant data blocks are completely updated using the SRDU algorithm. This continues until the system grows to 400 data nodes.

Fig. 3 plots the redundancy update overhead versus system size from 6 to 400. As expected, Redundant Data Regeneration performs the worst, essentially requiring all data blocks to be sent to the redundant node for regenerating the redundant data. On the other hand, regenerating redundant data using a centralized archive server incurs the least overhead, albeit at the expense of extra centralized facility.

Surprisingly, direct reuse of the original redundant data also performs very poorly. This is because the algorithm maintains the same data order within the parity group in computing the redundant data, and thus severely restricts the redundant data that can be reused. Once this restriction is relaxed by reshuffling the parity group, the overhead is reduced by half to around 20,000 blocks. Caching already transmitted data blocks further reduces the overhead by half to around 10,000 blocks. Thus with all three techniques combined, the SRDU algorithm can reduce the redundancy update overhead by as much as 75%.

6.2 Batched Redundancy Update

In the previous experiment, we always completely update all redundant data blocks before adding another new node. Clearly this is inefficient if new nodes are added frequently or added to the system in a batch. To address this issue, we conduct a second experiment where redundant data blocks are not updated until a fixed number of nodes, say W, are added – *batched redundancy update*. During this time, storage and streaming capacity in the new nodes are not utilized and thus this approach represents tradeoffs between redundancy update overhead and resource utilization. Fig. 4 plots the redundancy update overhead versus the batch size W for initial system size of 80 nodes. The key observation is that the normalized per-node redundancy update overhead decreases significantly with the batch size. A second observation is that the gain in caching transmitted data blocks reduces when the batch size increases. This is because the number of common term in reusable redundant blocks increases with larger batch size, and thus reducing the need for raw data blocks to update the new redundant data.

7. CONCLUSION AND FUTURE WORKS

This study is a first step in tackling the problem of redundant data update. As the results clearly showed, the redundancy update overhead is even more significant than data reorganization overhead and thus cannot be ignored. By taking advantage of the structure of RSE codes, we were able to substantially reduce the overhead by as much as 75%, and by performing update in a batch, we manage to reduce the overhead further by 97% for a batch size of 10. Nevertheless, batched redundancy update is not without tradeoffs. In particular, the storage and streaming capacity of the new nodes cannot be utilized until the system is reconfigured. Thus further investigations are warranted to quantify the tradeoffs to determine the optimal batch size that can balance between redundancy update overhead and resource utilization.

REFERENCES

[Lee and Leung, 2002a] Jack Y. B. Lee and W. T. Leung, “Study of a Server-less Architecture for Video-on-Demand Applications”. In *Proc. IEEE International Conference on Multimedia and Expo*., August 2002.

[Lee and Leung, 2002b] Jack Y. B. Lee and W. T. Leung, “Design and Analysis of a Fault-Tolerant

Mechanism for a Server-Less Video-On-Demand System". In *Proc. 2002 International Conference on Parallel and Distributed Systems*, Taiwan, Dec 17-20, 2002.

[Ho and Lee, 2003] T. K. Ho and Jack Y. B. Lee, "A Row-Permutated Data Reorganization Algorithm for Growing Server-less Video-on-Demand Systems". In *Proc. International Symposium on Cluster Computing and the Grid 2003*, Tokyo, Japan.

[Ghandeharizadeh and Kim, 1996] S. Ghandeharizadeh and D. Kim, "On-line Reorganization of Data in Scalable Continuous Media Servers". In *Proc. 7th International Conference on Database and Expert Systems Applications*, September 1996.

[Goel et al, 2002] A. Goel, C. Shahabi, S.-Y. Yao, and R. Zimmerman, "SCADDAR: An Efficient Randomized Technique to Reorganize Continuous Media Blocks". In *Proc. International Conference on Data Engineering*, 2002.

[Plank, 1997] J. S. Plank, "A Tutorial on Reed-Solomon Coding for Fault-Tolerance in RAID-like Systems". *Software -- Practice and Experience*, vol. 27, no. 9, pp. 995-1012, September, 1997.

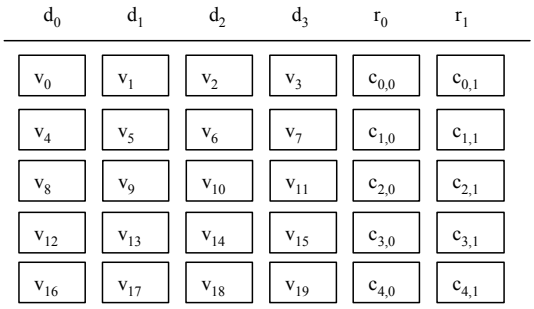

Fig. 1. Data placement before addition of nodes.

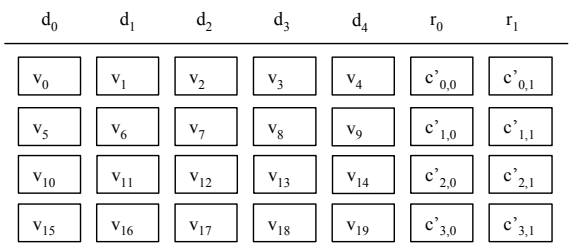

Fig. 2. Data placement after adding one data node.

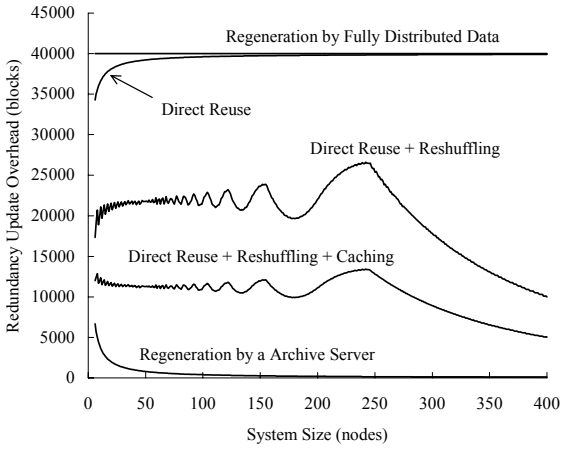

Fig. 3. Redundancy update overhead versus system size.

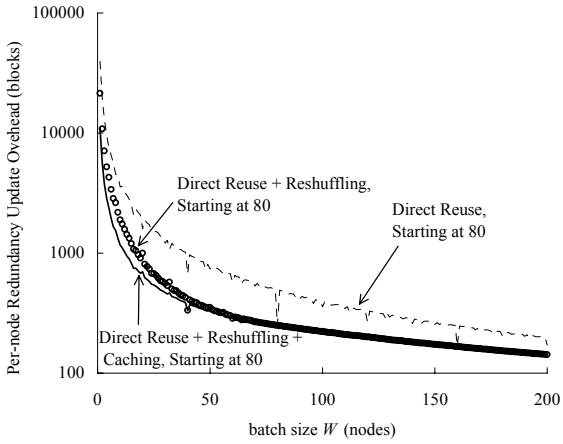

Fig. 4. Per-node redundancy update overhead versus batch size.

T.K. Ho received his BEng degree in Information Engineering from The Chinese University of Hong Kong in 2002. He is currently studying for his MPhil degree at the same department and participating in the Server-less Video Streaming Systems Project in the Multimedia Communications Laboratory (http://www.mcl.ie.cuhk.edu.hk).