Postprint: A Distributed Network Storage Coded Caching Scheme Based on Balanced Data Placement Strategy

Authors: Xue Chen, Hu Yuping

Date: 2019-04-01T00:00:00+00:00

Abstract

To ensure load balancing in network storage and avoid irrecoverable losses in the event of node or disk failures, this paper proposes a distributed network storage coded caching scheme based on a balanced data placement strategy, offering distinct solutions for large-scale and small-scale caches. First, the Maddah scheme is extended to multi-server systems; combined with the balanced data placement strategy, each file is stored as a unit in data servers, thereby addressing the large-scale cache problem. Then, the interference cancellation scheme is extended to multi-server systems; leveraging the interference cancellation scheme to reduce the peak rate of caching and combined with the balanced data placement strategy, a linear combination of cache segments is proposed to solve the small-scale cache problem. Finally, simulation experiments are conducted using Linux-based NS2 simulation software in systems with one and two parity servers, respectively. The simulation results demonstrate that the proposed scheme can effectively reduce the peak transmission rate and achieves superior performance compared to two other relatively recent caching schemes. Additionally, although the adoption of distributed storage limits the ability to combine content from different servers into a single message, resulting in performance degradation for coded caching schemes, it enables full exploitation of the inherent redundancy present in distributed storage systems, thereby enhancing overall storage system performance.

Full Text

Distributed Network Storage and Coding Scheme Based on Balanced Data Placement Strategy

Chen Xue[1], Hu Yuping[2] [1]Dept. of Computer Science & Engineering, Guangzhou College of Technology & Business, Guangzhou 510850, China

[2]School of Information Science, Guangdong University of Finance & Economics, Guangzhou 510320, China

Abstract: To ensure load balancing in network storage and avoid unrecoverable losses in case of node or disk failures, this paper proposes a distributed network storage and coding cache scheme based on a balanced data placement strategy, offering different solutions for large caches and small caches. First, the Maddah scheme is extended to multi-server systems, where each file is stored as a unit in data servers by combining with a balanced data placement strategy, thereby solving the large-scale cache problem. Then, the interference cancellation scheme is extended to multi-server systems, utilizing interference cancellation to reduce the peak rate of the cache, and proposing linear combinations of cache segments combined with the balanced data placement strategy to address the small cache problem. Finally, simulation experiments are conducted in systems with one and two parity servers using Linux-based NS2 simulation software. The results demonstrate that the proposed scheme can effectively reduce the peak transmission rate and achieves better performance compared to two other recent caching schemes. Although distributed storage limits the ability to combine content from different servers into a single message, causing performance loss in coding cache schemes, it can fully exploit the inherent redundancy in distributed storage systems, thereby improving overall storage system performance.

Key words: balanced data placement strategy; distributed network storage; coded cache; file striping; parity check server; interference cancellation

0 Introduction

In recent years, Redundant Array of Independent Disks (RAID) [**?**, **?**] has been increasingly applied in coded caching. RAID is a distributed storage technology based on erasure codes that combines multiple storage nodes (disks, servers, etc.) into a logical unit with data redundancy. The two most common implementations are RAID-4 and RAID-6, which consist of block-level striping with one and two dedicated parity nodes, respectively [**?**, **?**]. Most large-scale systems use some form of RAID to stripe multiple storage drives, but store or replicate entire files as a single unit across network nodes [**?**]. This approach improves peak speed while simplifying logging and deduplication, enhancing security, and increasing network flexibility.

The study in [**?**] focused on how a group of users with local memory can efficiently receive data from a server via a common link, proposing a caching and delivery scheme that provides worst-case performance within a constant factor of the information-theoretic optimum, along with upper and lower bounds for this optimum. Reference [**?**] refined the lower bound and designed new schemes to account for non-uniform file sizes. Reference [**?**] developed a novel algorithm by encoding both cache and transmission segments to achieve better performance when cache capacity is small or the number of users exceeds the number

of files. However, this scheme focuses on single-server systems.

In distributed storage, researchers have investigated how a single user can efficiently recover data distributed across a set of nodes, and how a group of users with local memory can effectively receive data from a single node. To ensure load balancing in network storage and avoid unrecoverable losses during node or disk failures, this paper proposes a distributed network storage and coding cache scheme based on a balanced data placement strategy, offering different solutions for large caches and small caches. The proposed scheme aims to design a joint storage and transmission protocol for multi-server, multi-user systems that combines distributed storage with coded caching to exploit parallelism and redundancy, thereby reducing peak traffic rates.

1.1 System Model

This paper considers a network with K users and N files stored across L data servers. When data storage uses different linear combinations, additional parity servers are employed, each storing the same number and size of files. Each user has a cache capacity of M files, and all files are assumed to have equal length and popularity.

Servers operate over independent error-free channels, enabling two or more servers to transmit messages simultaneously to the same or different users without interference [**?**]. Servers can broadcast identical messages to multiple users without additional bandwidth cost, but users cannot share their cache contents. Similarly, each server can only access files it stores, not files stored on other servers. Servers can read multiple segments from their own files and combine them into a single message, but two files stored on different servers cannot be combined into one message. However, servers are assumed to know the contents of each user' s cache and what is stored on other servers to coordinate their messages [**?**].

Subscripts denote file indices, while superscripts denote segment indices, so F_{ji} represents the j-th segment of file i. Different letters denote files from different servers. For example, A_i represents the i-th file from server A, and A_{ji} represents the j-th segment of file A_i.

1.2 Maddah Scheme

Maddah-Ali and Niesen proposed a coded caching scheme in [**?**] with a single server storing all files $F_1, F_2, \ldots, F_N$. Users connect to this server via a shared broadcast link, with the goal of designing cache placement and delivery schemes to reduce peak link load. This scheme divides each file F_i into $\binom{K}{t}$ equal-sized non-overlapping segments $F_{i,\mathcal{T}}$, where $\mathcal{T} \subseteq \{1, 2, \ldots, K\}$ and $|\mathcal{T}| = t$. Each segment is cached by a different group of t users. During the delivery phase, the server transmits one message to each subset of $t+1$ users, totaling $\binom{K}{t+1}$ messages. A group of users $\mathcal{S}$ requesting file F_i will receive the message:

$$\bigoplus_{k\in\mathcal{S}} F_{i,\mathcal{S}\setminus\{k\}}$$

where $\mathcal{S}\setminus\{k\}$ indexes segments cached by all users in $\mathcal{S}$ except the requesting user k. Each user can then decode its desired segment using the segments already cached. In the worst case, when all users request different files, this scheme achieves a peak rate (normalized by file size) of:

$$R_{\text{Maddah}}(K,t) = \frac{\binom{K}{t+1}}{\binom{K}{t}} = \frac{K-t}{t+1}$$

For some parameter combinations, broadcasting all uncoded missing segments may require a lower rate, so the generalized peak rate is:

$$R(K,t) = \min\left\{\frac{K-t}{t+1}, K\left(1-\frac{M}{N}\right)\right\}$$

Assuming the relationship between K, t, and M satisfies the inequality $M/N = t/K$, we ignore pathological cases. Research shows this peak rate is optimal for some parameter combinations.

1.3 Interference Cancellation

When $M < N/K$, the Maddah scheme performs poorly. Therefore, [**?**] proposed a new interference cancellation-based coded caching scheme. Instead of caching file segments in simple form, it suggests that users cache linear combinations of multiple segments. Transmitted messages are designed to use Maximum Distance Separable (MDS) codes [**?**].

During the placement phase, this scheme also divides each file into K non-overlapping segments of equal size. Let S_i denote the set of users that cache segments from file F_i. User k collects file segments:

$$\bigcup_{i:k\in S_i} F_{i,k}$$

using an MDS code of length K and stores t parity symbols in its cache.

The delivery phase proceeds as if all files were requested. When only a subset of files is requested, the scheme replaces some user requests with "unrequested files" and continues as if all files were requested. Regardless of the request, each file F_i transmits K (uncoded or coded) messages. Uncoded messages provide segments not cached by users requesting F_i, while coded messages combining multiple segments from F_i eliminate interference in their cached segments. Each

user collects K useful messages, which together with the t components stored in its cache are used to recover all K components of the MDS code.

Thus, the total number of messages sent from the server is $K\binom{K-1}{t}$. In this interference cancellation scheme, the normalized peak rate is:

$$R_{\mathrm{IC}}(K,t) = \frac{K\binom{K-1}{t}}{\binom{K}{t}} = \frac{K(K-t)}{t+1}$$

1.4 File Striping

The simplest method to extend single-server coded caching algorithms to multi-server systems is to distribute each file across all servers. This way, each user requests the same amount of information from each server, balancing the load. This is called data striping [**?**, **?**], a common practice in data centers and solid-state drives that enables parallel writing or reading to multiple drives or memory blocks. Users allocate equal portions of their cache to each server, and delivery is constructed as L independent single-server demands. We now describe in detail how striping reduces the peak rate of the Maddah scheme, though the same idea applies to any other scheme.

Each file F_i is split into L blocks stored across different servers, with each block divided into $\binom{K}{t}$ segments. These segments are denoted by $F_{ji}^{(m)}$, where j represents the segment number, i represents the file number, and m represents the block number. Server m stores the m-th fragment of each file, i.e., $F_{ji}^{(m)}$ for all i and j. This configuration is identical to Maddah' s scheme, where each segment is cached by t users, and the same segment is cached by the same user across all servers. The Maddah scheme can be decomposed into L components:

$$\bigoplus_{k\in\mathcal{S}} F_{i,\mathcal{S}\setminus\{k\}}^{(m)}$$

sent by different servers. The problem is then decomposed into L independent single-server subproblems, reducing the file size by a factor of L. The subproblem has the same number of users, files, and cache capacity as the global problem.

If an additional parity server P is available, it stores the bitwise XOR of blocks for each file, i.e., $P_{ji} = \bigoplus_{m=1}^{L} F_{ji}^{(m)}$ for all j and i. Then server P can take over some transmissions, reducing the peak load to $1/L$ of the Maddah scheme. In Equation (5), server P can transmit the XOR of all components to relieve the transmission burden of one data server. If two additional parity servers P and Q are available, any L of the $L+2$ servers can handle each group of messages in Equation (5), reducing the Maddah scheme' s peak to $1/(L+2)$.

For the interference cancellation scheme, a similar file distribution process can be followed to achieve the same peak rate scaling: $1/L$ with no parity servers,

$1/(L+1)$ with a single parity server, and $1/(L+2)$ with two parity servers.

1.5 Balanced Data Placement Strategy

In distributed file systems, when client request flows are relatively balanced, if data placement from clients to storage servers is also balanced, the overall system data access is considered relatively balanced. Balanced data placement can be viewed as a mapping relationship that maintains balanced data access between storage servers and clients.

During coded storage, data is typically stored in stripes as the minimum unit. A large file is divided into multiple stripes, where each stripe contains n data blocks. The first m blocks store original data, while the remaining $n-m$ blocks store coded data. These blocks can be treated as file storage objects. In a distributed file system, an object file' s content consists of: the large file' s inode number; each stripe' s stripeid; and the blockid within the stripe. For example, consider a 100 GB file with inode number 2018, where each stripe contains 8 data blocks (4 original and 4 coded), and blocksize is set to 64 MB. Through calculation, the data at position 9001 MB is located at stripeid = 9001/(64×4) = 36, and its blockid = 9001%(64×4)/64 = 0. The object file is then represented by the triple (2018,36,0). The server node storing the object is determined by the client' s clientid. Let *num_osdnodes* denote the total number of object file nodes in the file system, then each object file' s node is determined by a triple array. The mapping function is:

$$func : (\text{inode}, \text{stripeid}, \text{blockid}, \text{clientid}) \rightarrow \text{osdid}$$

where $func$ satisfies: $\forall x \in X, \exists y \in Y$ and $\forall y \in Y, \exists x \in X$.

Establishing a two-level mapping ensures balanced data placement across the entire file system when storing large files, achieving the goal of reducing disk contention.

2.1 No Parity Server

In systems without redundancy, servers cannot cooperate. During the delivery phase, each user is assigned to the server storing its requested file, and each data server transmits messages to fulfill its requests. Following Maddah' s scheme, a server receiving m requests will need to transmit $\binom{m}{t+1}$ messages. The normalized peak rate for this server can be expressed as:

$$R_{\text{no-parity}} = \frac{1}{L} \cdot \frac{K-t}{t+1}$$

The worst case occurs when all users request files from the same server, making the peak transmission rate identical to a single-server system.

2.2 One Parity and Two Data Servers

This section presents a simple storage system with two data servers and a third server storing the bitwise XOR of the two data servers, as shown in Table 2 . Although each server can only access its own files, the configuration in Table 2 allows messages to be constructed by combining messages from any two servers. That is, if server A (or B) completes its transmission task before the other server, it can work with the parity server to help server B (or A). This cooperative scheme enables simultaneous processing of requests for two files stored on the same server, balancing the load and reducing the worst-case peak rate below that achieved without a parity server.

Table 2 Files stored on each server

The Maddah scheme is extended to multi-server systems by combining it with a balanced data placement strategy, storing each file as a unit in data servers, as shown in Table 1 .

Table 1 Files stored on each server

The Maddah scheme' s advantage lies in storing file segments in simple form rather than as linear combinations, requiring larger cache capacity to achieve coded caching gains. Therefore, when cache capacity is large, the Maddah scheme is suitable. The placement phase of our algorithm is identical to traditional schemes. For example, in a system with $K = 6$ users, $M = 4$ files per server, and $N = 3$ servers, each file is divided into 20 segments, with each segment stored by $t = 2$ users. Table 3 shows the indices of 10 file fragments stored by each user, assuming all files are equally popular.

Table 3 File segment mapping to user cache

Considering the peak rate of the storage system, we assume all users request different files, so each user can be represented by its requested file. Let S denote the user set, S_A denote the set of all users receiving messages from server A, and S_B denote the set of all users receiving messages from server B. Let α represent the vector of segment indices, γ represent the vector of file indices, and m represent the message, where A_{ij} represents the j-th segment of the i-th file in server A, and B_{ij} represents the j-th segment of the i-th file in server B. The message is:

$$m = \bigoplus_{i=1}^{|\alpha|} A_{\gamma_i,\alpha_i} \oplus \bigoplus_{i=1}^{|\beta|} B_{\eta_i,\beta_i}$$

Lemma 1 Let the recipients of servers A and B be S_1 and S_2, respectively, where S_1 and S_2 represent the displayed sets, and α^* represents any set. If $S_1 \neq S_2$, then the corresponding message is:

$$m = \bigoplus_{i \in S_1 \cap S_2} A_{\gamma_i, \alpha_i^*} \oplus \bigoplus_{i \in S_1 \cap S_2} B_{\eta_i, \beta_i^*}$$

where α^* and β^* are defined in Equation (2).

Proof: When $S_1 \cap S_2$ is empty (which can be empty or non-empty), this is a special case of Lemma 1.

Lemma 2 The peak rate for the server system in Table 2 is:

$$R_{\text{peak}} = \frac{1}{2} \cdot \frac{K-t}{t+1} + \Delta$$

where Δ represents the ratio of unpaired messages, and $\Delta \leq 1/3$.

Proof: For each valid pair, we can use a single transmission from each server to provide the same information as two transmissions in Maddah' s single-server scheme, achieving a rate of $\frac{1}{2} \cdot \frac{K-t}{t+1}$. Unpaired messages are transmitted as described above, combining messages from any two of the three servers. Assuming balanced load across the three servers, the rate is $\frac{1}{3} \cdot \frac{K-t}{t+1}$.

Lemma 3 If requests are symmetric, then when K is even, the ratio of unpaired user subsets is $\Delta = 0$; when K is odd, $\Delta \leq 1/3$.

Proof: Messages are grouped by the number of segments they have from servers A or B. Within each group, messages are paired. At least $1 - \frac{1}{3}$ of messages in each group can be paired. Messages without segments from A or B require no transmission from servers A, B, or P.

Theorem 1 If user subsets S_1 and S_2 satisfy the conditions in Lemma 1, then (S_1, S_2) is a valid pair.

Proof: All users in S_1 and S_2 receive at least one desired segment from the server storing their requested file. Those in S_1 also receive an unrequested segment from server A or B. Users in S_1 can use this unrequested segment to obtain its complement from server P. We only need to prove that the unrequested segment from S_1 is different from the segment obtained from S_2.

Without loss of generality, consider user $i \in S_1$. Select a set of segment indices from S_1 such that user i caches all segments except the j-th one. Similarly, select indices from S_2 such that user i can cache all files. Thus, user i can obtain $B_{i,\beta}$ from S_1. Combining these yields the desired segment $A_{i,\alpha}$. As long as $S_1 \neq S_2$, this segment will differ from the one obtained from S_2 due to the one-to-one relationship between segment indices and user subsets.

Corollary 1 Assuming $S_1 = \{S_{AB}\}$ (containing only requests for server B), the normalized peak rate of the pairing algorithm is 2/5, significantly lower than the 3/5 rate of Maddah' s single-server scheme.

2.3 One Parity and L Data Servers

The previous discussion of two data servers and one parity server can be extended to systems with more than two data servers. With L data servers and one parity server, messages can be constructed by combining messages from L servers. Let S_1 and S_2 be two user subsets, where Y and Y' represent arbitrary index sets for server C, and Y and Y' can be paired so that servers A, B, and P need one transmission to provide the same information as messages m_{S_1} and m_{S_2} in Maddah' s single-server scheme. Other data servers C to L require at most two transmissions, as shown in the paired transmission in Figure 1 [Figure 1: see original paper], where A, B, C, and D are data servers and P represents the parity server (X indicates messages sent by corresponding servers).

The entire transmission process is as follows: a) Server L sends two messages to S_1 and S_2. For example, server L wants to send m_{S_1} and m_{S_2}, sending request information to corresponding resource users. b) Server A sends m_{S_1}, providing the required segment to users requesting A and the corresponding unrequested A segment to users requesting B. c) Server B sends m_{S_2}, providing the required segment to users requesting B and the corresponding unrequested B segment to users requesting A. d) Server P sends $m_{S_1 \cap S_2}$ to requesting users. Using previously received unrequested segments, users in $S_1 \cap S_2$ can solve for the required A and B segments.

A simple comparison of requested and received segments shows these transmissions deliver the same information as messages m_{S_1} and m_{S_2} in Maddah' s single-server scheme.

Theorem 2 A system with L data servers and one parity server can achieve the following normalized peak rate:

$$R_{\text{peak}} = \frac{1}{L} \cdot \frac{K-t}{t+1} + \Delta$$

where Δ is the pairing loss and $\Delta \leq \frac{1}{3L}$.

Proof: In Maddah' s scheme, $\binom{K}{t+1}$ messages can be sent from servers A, B, and P. These messages can be transmitted as follows: group messages by the number of segments they have from servers A or B, pair messages within each group, and at least $1 - \frac{1}{3L}$ of messages in each group can be paired. Messages without segments from A or B require no transmission from servers A, B, or P. $\binom{K}{t+1}$ messages can be transmitted through server L.

2.4 Two Parity and L Data Servers

This section extends the algorithm to systems with L data servers and two linear parity servers. The parity servers store different linear combinations of files by row, as shown in Table 4 . The servers are assumed to form an MDS code.

Our delivery strategy demonstrates that the peak rate can be reduced to half of Maddah' s single-server scheme.

Table 4 Files stored in RAID-6 Check Server

Lemma 4 Let $S_1 = \{S_{AY}\}$ and $S_2 = \{S_{BY}\}$, where Y represents a set of common requests from any server. S_1 and S_2 can be paired so that a single transmission from each server fulfills the same requests as messages m_{S_1} and m_{S_2} in Equation (1).

Proof: The transmission scheme follows the same pairing idea as the algorithm in Section 2.2. The transmission process is: a) Server A sends m_{S_1}, providing the desired segment to users requesting its file and the corresponding undesired A segment to other users. b) Server B sends m_{S_2}, providing the required segment to users requesting its file and the corresponding undesired B segment. c) Server L sends a single message to each user in $S_1 \cap S_2$ with the following content: (a) users requesting files from server B receive some undesired segments from server A; (b) remaining users in Y receive the corresponding segment if possible, otherwise the undesired segment corresponding to S_1. d) Parity servers P and Q send messages to $S_1 \cap S_2$ using combinations of each user' s segments. Those requesting files from server B receive a combination of segments corresponding to S_1, while others receive a combination corresponding to S_2. Since each user now has t independent segments and two independent linear combinations of all K segments, it can isolate the requested segment.

Comparing requested and received segments shows this transmission has the same messages m_{S_1} and m_{S_2} as Maddah' s single-server scheme.

Theorem 3 For a system with L data servers and two parity servers, the following normalized peak rate can be achieved:

$$R_{\text{peak}} = \frac{1}{L} \cdot \frac{K-t}{t+1} + \Delta$$

where $\Delta \leq \frac{1}{3L}$.

Proof: Messages are grouped by the number of segments they have from servers A or B. Within each group, messages are paired. For messages without segments from A or B, we repeat the same process with the other two servers, obtaining the same result: at most 1/3 remain unpaired. As shown above, each pair of messages can be delivered using a single transmission from each server, so the paired message rate is $\frac{1}{L} \cdot \frac{K-t}{t+1}$, where Δ represents the ratio of unpaired messages. Balancing load across all servers yields the result.

Extending the interference cancellation scheme from Section 1.3 to multi-server systems and combining it with a balanced data placement strategy, we propose linear combinations of cached segments. The interference cancellation scheme is specifically designed to reduce peak rates when cache size is small. Table 1 shows that transmission rate decreases as the number of servers L decreases.

Similar analysis for parity servers can be interpreted as an expansion of user caching.

Theorem 4 In a system with L data servers and parallel channels, the peak rate of the interference cancellation scheme can be reduced to $1/L$ of the single-server system, achieving:

$$R_{\mathrm{IC}}(K,t) = \frac{1}{L} \cdot \frac{K(K-t)}{t+1}$$

This holds whether each file is striped across servers or stored as a single block in one server.

Proof: Compared to a single-server system, file striping across L servers reduces the interference cancellation scheme' s peak rate by a factor of L.

Unlike Maddah' s scheme, the interference cancellation scheme sends the same number of segments from each file regardless of user requests, and each message consists of segments from one file. Therefore, even when different files are stored on different servers, the same messages can be transmitted. Each server will need to transmit only a small fraction of messages because it stores the same proportion of files. The peak load can then be reduced to $1/L$ of that in Equation (4).

With parity servers, we can further reduce transmission rates by treating them as an extension of user caches. Section 1.3 explains that in the interference cancellation algorithm, each user cache stores parity symbols generated by a systematic MDS code. The code can be chosen such that some of these parity symbols can be found as combinations of information stored in servers P and Q.

For example, parity server P stores the horizontal sum of files, so it can transmit messages in the format:

$$\bigoplus_{i=1}^{N} \lambda_{ij} F_{i,j}$$

for any user set $\mathcal{S}$ with arbitrary coefficients λ_{ij}. This corresponds to linear combinations of all segments in Equation (3). Similarly, parity server Q can transmit other linear combinations of segments that serve as components of the MDS code. This effectively increases the cache memory size by M' file units, corresponding to the amount of information parity servers can send to each user during the delivery phase.

Theorem 5 If there are η parity servers and $M' = \eta \cdot \frac{K-t}{t+1}$, the following can be achieved:

$$R_{\mathrm{IC}}^{\mathrm{parity}}(K,t) = \frac{1}{L} \cdot \frac{K(K-t)}{t+1} - \frac{M'}{N}$$

4 Simulation Experiments and Analysis

We conduct simulation experiments on the two proposed schemes using Linux-based NS2 simulation software. Two sets of experiments are performed, both in systems with one parity server and two parity servers. The proposed Scheme 1 and Scheme 2 are compared with Maddah scheme striping [**?**] and interference cancellation striping [**?**]. We briefly introduce the experimental parameter settings.

4.1 Experimental Parameter Settings

Two sets of experiments use different K and N values: Set 1 uses $K = 15, N = 20$, and Set 2 uses $K = 20, N = 15$. For a system with $L = 4$ data servers, each server has 5 files, and the user group is set to 5. The parameter calculation formulas for Scheme 1 and Scheme 2 are shown in Table 5 and Table 6 , respectively, allowing calculation of corresponding server parameter values for different K, N, and t values.

Table 5 Normalized peak rate of Scheme 1

Table 6 Normalized pairs of Scheme 2

4.2 Experimental Results and Analysis

Figures 3 [Figure 3: see original paper] and 4 [Figure 4: see original paper] show the results for one and two parity server cases, respectively, with $K = 15$ users and $N = 20$ files. The figures demonstrate that striping provides lower peak rates. Additionally, since $N > K$, the interference cancellation scheme always performs worse than the Maddah scheme when using striping. Without striping, Scheme 2 achieves lower peak rates than Scheme 1 when cache capacity is small.

Figure 3 Performance of four schemes in a parity-check server system

Figure 4 Performance of four schemes in two parity-check server systems

The interference cancellation scheme is designed for single-server systems where the number of files is less than the number of users ($N \leq K$). However, in multi-server systems, peak load is reduced by $1/L$, so when $N > K$, the interference cancellation scheme may also perform well. To apply this algorithm, we can add $K - N$ virtual users with arbitrary requests. This leads to the conclusion: if there are η parity servers and $M' = \eta \cdot \frac{K-t}{t+1}$, the following can be achieved:

$$R_{\mathrm{IC}}^{\mathrm{multi}}(K,t) = \frac{1}{L} \cdot \frac{K(K-t)}{t+1} - \frac{M'}{N}$$

Figure 5 [Figure 5: see original paper] and Figure 6 [Figure 6: see original paper] compare the performance of Schemes 1 and 2 with varying numbers of users for one or two parity server cases. Again, striping provides lower peak rates, and when cache capacity is small, striping with interference cancellation performs better than striping with Maddah' s scheme. For Schemes 1 and 2, Scheme 2 provides lower peak rates when cache capacity is small, while Scheme 1 performs better as cache capacity increases. The crossover point in the curves occurs at a larger M value than in Figures 3 and 4, indicating that when the number of users exceeds the number of files, Scheme 2 is preferred for caching.

5 Conclusion

This paper proposes a coded caching scheme for reducing peak data rates in multi-server systems with distributed storage and different redundancy levels. By splitting each file across multiple servers, the peak rate can be reduced proportionally to the number of servers. When each file is stored as a single block in one server, the algorithm provides different caching and delivery schemes based on cache memory size. The proposed coded caching scheme creatively exploits this redundancy to reduce achievable traffic peak rates. Future work will design erasure codes with different file protection levels [**?**, **?**] to generalize our scheme to file systems with varying popularity.

References

[1] Bian Jianchao, Chaya Hang, Luo Shoushan, et al. A hybrid coding scheme based on intra-disk and inter-disk redundancy [J]. Computer Research and Development, 2016, 53(9): 1906-1917.

[2] Luo Ping, Zhang Tong. Research on spaceborne storage data management based on flash memory [J]. Application Research of Computers, 2018, 35(2): 479-482.

[3] Zhang Guangyan, Wang Jigang, Li Keqing, et al. Redistribute data to regain load balance during RAID-4 scaling [J]. IEEE Trans on Parallel & Distributed Systems, 2014, 26(1): 219-229.

[4] Wang Yan, Yin Xunrui, Wang Xin. MDR Codes: A new class of RAID-6 codes with optimal rebuilding and encoding [J]. IEEE Journal on Selected Areas in Communications, 2014, 32(5): 1008-1018.

[5] Balasubramanian B, Lan T, Chiang M. SAP: similarity-aware partitioning for efficient cloud storage [C]// INFOCOM. Piscataway, NJ: IEEE Press, 2014: 592-600.

[6] Maddah-Ali M A, Niesen U. Fundamental limits of caching [J]. IEEE Trans on Information Theory, 2012, 60(5): 2856-2867.

[7] Ghasemi H, Ramamoorthy A. Improved lower bounds for coded caching [J]. IEEE Trans on Information Theory, 2017, 63(7): 4388-4413.

[8] Tian C, Chen J. Caching and delivery via interference elimination [C]// IEEE International Symposium on Information Theory. Piscataway, NJ: IEEE Press, 2016.

[9] Deng Yan, Zhang Xinyou, Xing Huanlai. A DTN dynamic segment coding routing algorithm based on transmission capacity control [J]. Application Research of Computers, 2017, 34(9): 2753-2757.

[10] Zhu Hong, Li Li, Huang Puming. Low-cache and high-speed transmission of massive satellite-borne remote sensing data [J]. Journal of Electronics, 2013, 41(10): 2016-2020.

[11] Suh C, Ramchandran K. Exact-repair MDS code construction using interference alignment [J]. IEEE Trans on Information Theory, 2011, 57(3): 1425-1442.

[12] Dong Qiuxiang, Guan Zhi, Chen Zhong. Review of computational cryptography on encrypted data [J]. Application Research of Computers, 2016, 33(9): 2561-2572.

[13] Shen Xiaohui, Choudhary A. A high-performance distributed parallel file system for data-intensive computations [J]. Journal of Parallel & Distributed Computing, 2004, 64(10): 1157-1167.

[14] Yu Quan, Sung C W, Chan T H. Irregular fractional repetition code optimization for heterogeneous cloud storage [J]. IEEE Journal on Selected Areas in Communications, 2014, 32(5): 1048-1060.

[15] Yin Chao, Wang Jianzong, Lv Haitao, et al. BDCode: an erasure code algorithm for big data storage systems [J]. Journal of University of Science & Technology of China, 2016, 46(3): 188-199.

Note: Figure translations are in progress. See original paper for figures.