Reducing Access Latency in Erasure Coded Cloud Storage with Local Block Migration

Yaochen Hu, Di Niu
Department of Electrical and Computer Engineering
University of Alberta
{yaochen, dniu}@ualberta.ca

***Abstract*—Erasure coding has been applied in many cloud storage systems to enhance reliability at a lower storage cost than replication. While a large amount of prior work aims to enhance recovery performance and reliability, the overall access delay in coded storage still needs to be optimized. As most production systems adopt a systematic code and place the original copy of each block on only one server to be read normally, it is harder to balance server loads and more likely to incur latency tails in coded storage than in three-way replication, where a block can be read from any of the 3 servers storing the block. In this paper, we propose to reduce the access latency in coded storage systems by moving blocks with anti-correlated demands onto same servers for statistical load balancing. We formulate the optimal block placement as a problem similar to Min-k-Partition, propose a local block migration scheme, and derive an approximation ratio as a function of demand variation across blocks. Based on request traces from Windows Azure Storage, we demonstrate that our scheme can significantly reduce the access latency with only a few block moves, especially when the request demand is skewed.**

I. Introduction

Cloud storage systems, such as Hadoop Distributed File System (HDFS) [1], Google File System (GFS) [2] and Windows Azure Storage (WAS) [3], store large amounts of enterprise-level and personal data. Since these systems rely on commodity servers in datacenters, content must be replicated (e.g., for 3 times in HDFS) for fault-tolerance. Erasure coding, e.g., an (k, r) Reed-Solomon (RS) code, is further adopted in many production systems, e.g., Windows Azure Storage [4], Googles ColossusFS, Facebook's HDFS, to offer significantly higher reliability than data replication at a much lower storage cost [5], [6]. However, as a tradeoff, when a data block is unavailable due to disk failures or degraded reads (e.g., when the server is temporarily congested or offline), multiple (coded or uncoded) blocks in the same coded group must be read from other servers to recover the unavailable block, leading to higher recovery traffic. As a result, many techniques have been proposed to reduce the amount of recovery traffic [6]–[8] or recovery latency [9], [10] in coded storage systems.

However, an equally important performance metric that is yet to be optimized in coded storage systems is the general access latency per request. Google, Microsoft and Amazon all have observed that a slight increase in service delay (e.g., by as small as 400 ms) may lead to observable fewer accesses from users and potential revenue loss [11]. Although a few recent studies [9], [10] attempt to optimize the delay performance of coded storage, they rely on parallel downloads to exploit a queueing-theoretical gain, where each request must access k or more servers, and abort the remaining download threads when k blocks are obtained. However, such a scheme mainly benefits systems that are based on *non-systematic* codes, since for *systematic* codes adopted by most production systems today, normal reads are served by the single uncoded original block, while parallel downloads may unnecessarily increase traffic.

We study the access latency in coded storage systems from a new perspective of load balancing. Without surprise, unbalanced server loads and long server queues are more likely to happen in coded storage. The reason is that in systems based on systematic erasure codes, like Google's ColossusFS and WAS, each original uncoded block is placed on *only one* server [6]. As a result, unlike three-way replication (where a request can be served by any of the 3 servers storing the block), the request has to be directed to the only server hosting the original block, with little opportunity of load balancing. When the server load is heavy and the request is not responded by a timeout deadline, degraded reads may be performed which further increase the queues at multiple other servers and exacerbate the overall response latency.

In this paper, we propose to reduce the access latency in coded storage systems through a new approach of block placement adjustment and controlled block migration. Although there is little chance to choose servers during normal reads, we may place blocks with *anti-correlated* demands on a same server to benefit from statistical multiplexing and prevent certain hot blocks from congesting a specific server. We formulate the content placement optimization to minimize the expected average waiting time of all incoming requests, based on the mean and covariance of demands for different blocks, which can be readily measured according to recent request history. To avoid globally shuffling blocks across the system, we require all block migration to be *local*, and move as few blocks as possible with respect to the current placement to reduce the migration overhead.

Our statistical content placement problem is similar to the Min-k-Partition problem, a well-known NP-complete problem [12], [13], which aims to divide a graph into k partitions to minimize the sum of all intra-partition edge weights. Yet, our problem turns out to be even more challenging, since we also need to handle an additional constraint that no blocks from the same coded group can be placed on the same server, which

is needed to maintain the promised reliability of an (k, r) RS code. We introduce a novel technique to convert this constraint into carefully designed weights in the objective function and propose a time-efficient local search algorithm which only moves the block that reduces the latency objective the most at a time. We prove that our algorithm always produces a feasible solution that satisfies the special constraint and theoretically derive the worst-case approximation ratio of our algorithm with respect to the global optimality. We characterize such a ratio as a function of demand statistics; the larger the demand variation among blocks, the better the approximation.

Through simulations based on real request traces collected from the Windows Azure Storage system, we show that our local block migration scheme can greatly reduce the overall access latency by only moving a small portion of all the blocks, and outperforms a best randomized content placement that requires global shuffling. In the meantime, our scheme does not affect storage overhead, reliability or repair cost. It turns out that the real request pattern exhibits high skewness and variation, which can significantly benefit from our local block migration with only a few necessary moves.

II. System Model and Problem Formulation

The content in a typical cloud storage system is stored in *data blocks*. When an erasure code is used, e.g., a systematic (k, r) RS code, every k original uncoded data blocks are grouped in a *coded group* and another r parity blocks are generated. In order to maintain a high availability, all these $k+r$ blocks are placed on *different* server nodes. We will call them "coded blocks" in general with respect to the k original blocks. In a *normal read*, any access request will be directed to the server containing the original block. If the server is unavailable, a *degraded read* is performed by reading any other k blocks in the same coded group, requiring k server accesses. Suppose the system has a total number of n coded blocks placed on m servers.

In a small unit of time, which we call *time slot* (e.g., a second), we denote the number of requests for each *coded block* i by a random variable D_i. Let $\vec{D} := \{D_1, D_2, \ldots, D_n\}$. With request rates represented by random variables, we can model demand fluctuation in different time slots. We use $\vec{\mu} := \mathbb{E}(\vec{D})$ to denote the mean of $\vec{D}$, and $\mathbf{\Sigma} := \mathbb{COV}(\vec{D})$ the covariance matrix of $\vec{D}$. We can assume that within a certain *measurement period*, $\vec{\mu}$ and $\mathbf{\Sigma}$ remain unchanged.

Note that the mean and covariance of $\vec{D}$ can be readily measured or estimated from system traces. For example, the system can keep track of the number of requests per second or per minute for each *original* content block at a frontend gateway [3] to calculate the empirical mean and covariances of request rates for original content in the measurement period. It can also easily record the rate of degraded reads (due to node failures or temporary unavailability) and convert the request rate statistics for original content blocks to those for all the coded blocks, assuming degraded reads are randomly directed to k other coded blocks. Alternatively, the access statistics for all the coded blocks can even be measured directly in the backend storage system. This way, $\vec{\mu}$ and $\mathbf{\Sigma}$ for all coded blocks are directly computed.

We use an integer variable y_i, $i = 1, \ldots, n$ to represent the index of the server on which the i^{th} coded block is placed. Denote $\{L_1, \ldots, L_m\}$ the server loads, where $L_i = \sum_{j:y_j=i} D_j$ represents the amount of requests directed to server i in the time slot of interest. Let $\alpha := k+r$ denote the total number of coded blocks in each coded group. Furthermore, we use G_i, $i = 1, \ldots, n$, to denote the index of the coded group to which the i^{th} coded block belongs; two blocks are in the same coded group if and only if they have the same group index.

Considering a specific time slot, we formulate the optimal content placement (CP) problem as

$$\text{(CP)} \underset{y_1, y_2, \ldots, y_n}{\text{minimize}} \quad \mathbb{E}\left(\sum_{i=1}^{m} \frac{1}{2} L_i^2\right) \tag{1}$$

$$\text{subject to} \quad L_i = \sum_{j:y_j=i} D_j, \quad \forall i, \tag{2}$$

$$y_i \neq y_j, \text{ if } G_i = G_j, \quad \forall i \neq j, \tag{3}$$

$$y_i = \{1, 2, \ldots, m\}, \quad \forall i, \tag{4}$$

Problem (CP) minimizes the expected squared l^2-norm of server loads, which represents the expected sum of waiting times of all the requests in this time slot. We assume that the request processing speed of servers are homogeneous [14], which is common for storage servers in the same rack in a datacenter. The purpose of (1) is to distribute random loads $\vec{D}$ across different servers in a statistically balanced manner. Constraint (2) is the mapping from request rates to server loads according to content placement $y_1, \ldots, y_n$. Constraint (3) requires that the coded blocks from the same coded group must be placed on different servers to guarantee the promised reliability of an RS (k, r) code. We do not consider queue accumulation along multiple time slots in our model and focus on solving the single period problem (CP). Note that server processing capacity is usually over-provisioned in production systems, and once server loads are balanced, queues will vanish fast in a stable system.

We now convert Problem (CP) to an equivalent form similar to the well-known Min-k-Partition Problem in graph theory yet with one additional constraint. We define a weight matrix $\mathbf{W}$ by

$$\mathbf{W} := \mathbb{E}(\vec{D} \cdot \vec{D}^T) = \vec{\mu} \cdot \vec{\mu}^T + \mathbf{\Sigma}, \tag{5}$$

Clearly, all the elements in $\mathbf{W}$ are **nonnegative**. Consider the following problem, which we call Constrained Min-k-Partition Problem (CMKP+):

$$\text{(CMKP+)} \underset{y_1, y_2, \ldots, y_n}{\text{minimize}} \quad \sum_{i<j} \mathbf{W}_{ij} \delta(y_i - y_j) + \frac{1}{2} \sum_i \mathbf{W}_{ii}, \tag{6}$$

$$\text{subject to} \quad y_i \neq y_j, \text{ if } G_i = G_j, \forall i \neq j, \tag{7}$$

$$y_i = \{1, 2, \ldots, m\}, \forall i, \tag{8}$$

where $\delta(\cdot)$ is an indicator function, i.e.,

$$\delta(x) := \begin{cases} 1, & \text{if } x = 0, \\ 0, & \text{otherwise.} \end{cases}$$

Note that we use CMKP to represent the problem with the constant term $\frac{1}{2}\sum_i \mathbf{W}_{ii}$ removed from (6).

Proposition 1. *Problem (CP) is equivalent to Problem (CMKP+).*

Proof. Please refer to the Appendix for the proof. □

Therefore, we can consider the (CMKP+) problem instead of the original (CP) problem. In fact, (CMKP+) is a partition problem in graph, where all the coded blocks can be deemed as nodes, with $\mathbf{W}$ representing edge weights between every pair of nodes. The objective is to divide nodes into k partitions to minimize the sum of intra-partition edge weights, subject to constraint (7), that is, no coded blocks from the same coded group appear in the same partition. Without constraint (7), the (CMKP+) problem can be converted to Min-k-Partition (MKP) and Max-k-Cut (MKC), which are well-known NP-complete problems [12], [13]. However, our problem (CMKP+) is even more challenging due to the additional constraint (7) to maintain the promised reliability offered by erasure coding.

III. Local Block Migration Algorithm

We present the local block migration (LBM) algorithm to solve (CMKP+) with theoretical worst-case approximation guarantees, which equivalently solves the optimal content placement problem (CP). We first present our technique to handle the special constraint (7) before presenting the algorithm.

A. Problem Reduction

First, we reduce (CMKP+) to a form without constraint (7). Our idea is to solve the problem with constraint (7) removed, while setting a *sufficiently large* weight for each pair of coded blocks in the same coded group to prevent them from being placed on the same server. Define $\mathbf{W}'$ as

$$\mathbf{W}'_{ij} = \begin{cases} f_{ij}(\mathbf{W}) & \text{, if } G_i = G_j, i \neq j, \\ \mathbf{W}_{ij}, & \text{otherwise,} \end{cases} \tag{9}$$

where $f_{ij}(\mathbf{W})$ is a penalty function to be defined later. Replacing $\mathbf{W}$ by $\mathbf{W}'$ and removing constraint (7) in (CMKP+), we obtain

$$\text{(MKP+)} \underset{y_1,y_2,\ldots,y_n}{\text{minimize}} \quad \sum_{i<j} \mathbf{W}'_{ij}\delta(y_i - y_j) + \frac{1}{2}\sum_i \mathbf{W}'_{ii}, \tag{10}$$

$$\text{subject to} \quad y_i = \{1, 2, \ldots, m\}, \text{ for } \forall i. \tag{11}$$

Note that the term $\frac{1}{2}\sum_i \mathbf{W}'_{ii}$ in (10) is a constant. Hence, (MKP+) is equivalent to Min-k-Partition Problem (MKP):

$$\text{(MKP)} \underset{y_1,y_2,\ldots,y_n}{\text{minimize}} \quad \sum_{i<j} \mathbf{W}'_{ij}\delta(y_i - y_j), \tag{12}$$

$$\text{subject to} \quad y_i = \{1, 2, \ldots, m\}, \text{ for } \forall i, \tag{13}$$

whose dual problem is the famous Max-k-Cut (MKC) problem:

$$\text{(MKC)} \underset{y_1,y_2,\ldots,y_n}{\text{maximize}} \quad \sum_{i<j} \mathbf{W}'_{ij}(1 - \delta(y_i - y_j)), \tag{14}$$

$$\text{subject to} \quad y_i = \{1, 2, \ldots, m\}, \text{ for } \forall i. \tag{15}$$

Furthermore, (MKC) and (MKP) also have the same optimal solution(s). The reason is that the sum of the objective values of the two problems is

$$\sum_{i<j} \mathbf{W}'_{ij}\delta(y_i - y_j) + \sum_{i<j} \mathbf{W}'_{ij}(1 - \delta(y_i - y_j)) = \sum_{i<j} \mathbf{W}'_{ij}, \tag{16}$$

which is a constant. Since they are minimization and maximization problems, respectively, they will have the same optimal solution(s).

In the following, to solve (CMKP+), we carefully design a penalty function f, such that we always get a feasible solution to the original problem (CMKP+) by solving (MKC) with a new $\mathbf{W}'$ yet without constraint (7). We propose an algorithm to solve (CMKP+) and thus the original (CP) problem, by approximately solving (MKC) using a classical *local search heuristic* [15]–[17]. We are able to theoretically derive a worst-case approximation ratio of the proposed solution to our problem (CMKP+), which did not appear in prior literature [15]–[17], by leveraging a unique problem structure in our objective function.

B. Local Block Migration Algorithm

Considering the influence of every single move on the objective, we define the *gain* of moving block i to server j as

$$g_j(i) := \sum_{k:y_k=y_i;k\neq i} \mathbf{W}'_{ik} - \sum_{k:y_k=j;k\neq i} \mathbf{W}'_{ik}, \forall i, j : j \neq y_i, \tag{17}$$

which is the reduction of the objective of (MKP) if this move is carried out. For consistency, let $g_j(i) = -\infty$ for $j = y_i$. Define $f^l_{ij}(\mathbf{W})$ as

$$f^l_{ij}(\mathbf{W}) := \epsilon + \frac{1}{m - \alpha + 1}\min\Bigg\{ \sum_{k:k\neq i;G_k\neq G_i} \mathbf{W}_{ik}, \sum_{k:k\neq j;G_k\neq G_j} \mathbf{W}_{kj} \Bigg\}, \tag{18}$$

where ϵ is an *arbitrary* positive constant.

Algorithm 1 describes our Local Block Migration (LBM) algorithm to solve (CMKP+). In every iteration, we execute the move of block i to server j who achieves the largest *gain* $g_j(i)$, until no move can reduce the objective any more, as shown in Lines 4-8.

To pick the best move $\text{argmax}_{\{i,j\}} g_j(i)$, in Line 6 of Algorithm 1, we do not need to recalculate all the $m \times n$ $g_j(i)$ by (17) in every iteration. Instead, since there is only one move in each iteration, we only need to update the gains $g_j(i)$ affected by the move. Moreover, even the affected gains $g_j(i)$ do not need to be recalculated by (17), and can be updated by incrementally. The details of our efficient procedure to update the gains $g_j(i)$ is described in Algorithm 2.

Note that LBM is a "local" algorithm that performs one best move at a time to improve the latency performance. In real systems, since it is impractical to globally shuffle all the block placement to optimize load balancing, we can use the proposed LBM to locally improve an existing arbitrary placement at some frequency, e.g., every hour or every day. Moreover, we

Algorithm 1 Local Block Migration

Input: initial placement $\{y_1, y_2, \ldots, y_n\}$, $\mathbf{W}$.
Output: migrated placement $\{y_1, y_2, \ldots, y_n\}$.

Calculate $\mathbf{W}'$ by (9) and (18)
Calculate $g_j(i)$ for $\forall i, j$ by (17)
procedure LOCAL BLOCK MIGRATION
  while $\max_{\{i,j\}} g_j(i) > 0$ **do**
    Find the best move: $i_m, j_m \leftarrow \text{argmax}_{\{i,j\}} g_j(i)$
    Update all the affected $g_j(i)$ entries by Algorithm 2
    Execute the move: $y_{i_m} \leftarrow j_m$
  end while
end procedure

do not actually carry out all the moves $y_{i_m} \leftarrow j_m$ computed by Algorithm 1. Instead, we only make the moves to change the initial placement to the LBM outcome. In Sec. IV, we show that only a few moves will achieve the most latency reduction.

C. Feasibility and Worst-Case Approximation Ratio

We first show Algorithm 1 yields a feasible solution to our problem with the special constraint (7).

Theorem 2. *If $\mathbf{W}'$ in (9) is defined with $f_{ij}(\mathbf{W})$ given by $f_{ij}(\mathbf{W}) = f^l_{ij}(\mathbf{W})$ in (18), any solution given by Algorithm 1 will satisfy (7), and thus will be a feasible solution to (CMKP+) and (CP).*

Proof. Please refer to the Appendix for the proof. □

Note that we do not necessarily have $f_{ij}(\mathbf{W}) > \mathbf{W}_{ij}$. However, Theorem 2 guarantees that if $f_{ij}(\mathbf{W}) = f^l_{ij}(\mathbf{W})$, LBM always produces a feasible solution that no two blocks from the same coded group are placed on the same server. Theorem 3 provides a worst case approximation for our Local Block Migration with respect to the globally optimal solution to (CP).

Theorem 3. *Suppose $f_{ij}(\mathbf{W})$ in (9) is given by $f^l_{ij}(\mathbf{W})$ in (18). Then, the worst-case approximation ratio of Algorithm 1 with respect to the optimal solution of (CMKP+) and (CP) is given by*

$$1 + \frac{1}{m - \alpha + 1}\left(\frac{\mathbb{E}\big(\big(\sum_i D_i\big)^2\big)}{\sum_i \mathbb{E}(D_i^2)} - 1\right). \tag{19}$$

Proof. Please refer to the Appendix for the proof. □

Furthermore, since we have

$$\frac{\mathbb{E}\big(\big(\sum_i D_i\big)^2\big)}{\sum_i \mathbb{E}(D_i^2)} \leq \max_{\vec{D}} \Big\{\big(\sum_i D_i\big)^2 \Big/ \sum_i D_i^2\Big\} \leq n,$$

where the equality holds if and only if $D_1 = D_2 = \ldots = D_n$, we can derive the worst-case ratio among all the distributions of $\vec{D}$.

Corollary 4. *For any $\vec{D}$, the approximation ratio given by Theorem 3 is at most $1 + \frac{n-1}{m-\alpha+1}$.*

Algorithm 2 Gain Update Algorithm

Input: the current $g_j(i)$, $\forall i, j$, the current moving block index i_m and its destination server j_m.
Output: the updated $g_j(i)$, $\forall i, j$.

procedure g UPDATE
  for $\forall i \neq i_m$, such that $y_i = y_{i_m}$ **do**
    for $\forall j \neq y_{i_m}$ **do**
      $g_j(i) \leftarrow g_j(i) - \mathbf{W}'_{ii_m}$
    end for
  end for
  for $\forall i \neq i_m$, such that $y_i = j_m$ **do**
    for $\forall j \neq j_m$ **do**
      $g_j(i) \leftarrow g_j(i) + \mathbf{W}'_{ii_m}$
    end for
  end for
  for $\forall i \neq i_m$, $y_i \neq y_{i_m}$ **do**
    $g_{y_{i_m}}(i) \leftarrow g_{y_{i_m}}(i) + \mathbf{W}'_{ii_m}$
  end for
  for $\forall i \neq i_m$, $y_i \neq j_m$ **do**
    $g_{j_m}(i) \leftarrow g_{j_m}(i) - \mathbf{W}'_{ii_m}$
  end for
  for $\forall j, j \neq y_{i_m}$ and $j \neq j_m$ **do**
    $g_j(i_m) \leftarrow g_j(i_m) + \sum_{i:i\neq i_m; y_i = j_m} \mathbf{W}'_{i_m i} - \sum_{i:i\neq i_m; y_i = y_{i_m}} \mathbf{W}'_{i_m i}$
  end for
  Calculate $g_{y_{i_m}}(i_m)$ by (17)
  $g_{j_m}(i_m) \leftarrow -\infty$.
end procedure

Remarks: the approximation ratio provided by Theorem 3 is dependent on the distribution of the requests $\vec{D}$. In the extreme case when requests for different coded blocks are identical, i.e., $D_1 = D_2 = \ldots = D_n$, the offered approximation ratio (19) is large as shown in Corollary 4. In fact, the ratio of $\mathbb{E}(||\vec{D}||_1^2)/\sum_i \mathbb{E}(D_i^2)$ characterizes the demand variation among different blocks. When this variation is large, the approximation ratio (19) is small and our algorithm is guaranteed to yield a good result even in the worst case.

On the other hand, when the demand variation is small, although the offered theoretical *worst-case* performance bound (19) is large, LBM can actually still generate a load balanced solution. In fact, in this case, D_i behaves uniformly cross different blocks and simple randomized or round robin placement can already achieve load balancing, so can LBM. In a nutshell, LBM provides good solutions for most situations and is especially beneficial when the requests for different blocks have a large variation and are highly skewed. In Sec. IV, we show that our request traces in the real world usually have a small $\frac{\mathbb{E}(||\vec{D}||_1^2)}{\sum_i \mathbb{E}(D_i^2)}$, in which case LBM will have a large benefit.

D. Further Reducing Migration Overhead

Although Theorem 2 guarantees the feasibility of the final converged solution from Algorithm 1, in reality, we may want to stop looping after a fixed number of iterations to limit the number of moves produced by LBM. In this case, the

solution may not be feasible to (CP) in theory with the $f_{ij}(\mathbf{W})$ definition in (18). In order to propose an alternative scheme, we let $f_{ij}(\mathbf{W})$ in (9) be given by

$$f_{ij}^r(\mathbf{W}) := \epsilon + \frac{1}{m-\alpha}\max\Bigg\{\sum_{k:k\neq i;G_k\neq G_i}\mathbf{W}_{ik}, \sum_{k:k\neq j;G_k\neq G_j}\mathbf{W}_{kj}\Bigg\}, \tag{20}$$

where ϵ is an *arbitrary* positive constant.

Theorem 5. *If the $\mathbf{W}'$ in (9) is defined with $f_{ij}(\mathbf{W})$ given by $f_{ij}(\mathbf{W}) = f_{ij}^r(\mathbf{W})$ in (20), and the initial content placement satisfies (7), the solution after any iteration in Algorithm 1 will always satisfy (7).*

Proof. Please refer to the Appendix for the proof. □

Remarks: Theorem 5 implies that as long as we start from a valid placement, we can put a maximum iteration number in LBM and can always get feasible solutions in any iteration. This way, we can stop the loop in Algorithm 1 when the maximum iteration number is reached and still get a feasible solution that satisfies constraint (7). In other words, $f_{ij}^r(\mathbf{W})$ allows us to trade the latency reduction off for fewer block moves, according to a budget on migration overhead.

E. Time Complexity and the Breakout Method

Algorithm 1 runs in linear time with respect to the number of coded blocks in each iteration and is very efficient. In the main loop from Line 4 to Line 8 in Algorithm 1, it only contains a finding max operation and an updating operation. The finding max runs in linear time with respect to the searching space and it is $O(mn)$ since we have mn gain entries. For the gain updating procedure described in Algorithm 2, Line 2 to Line 11 has only $O(\frac{2n}{m}\cdot m) = O(2n)$ additions or subtractions. Line 12 to Line 17 also has $O(2n)$ additions or subtractions. Line 18 to Line 22 has two updating entries with $O(\frac{2n}{m})$ basic calculations. Therefore, our Local Block Migration can finish each iteration with $O(mn)$ basic calculations.

The LBM is a local heuristic search and may get trapped into some local optimum. In order to reach the global optimum, some breakout method [18] can be engaged. Similar techniques in [15], [16] can be used to even improve over the local optimum. The idea is that when a local optimum is reached, we may keep looping in Algorithm 1 even if the max value of $g_j(i)$ is negative. To avoid infinite loops, the blocks that have been moved are locked. When all the blocks are moved for once, the history of all the moves are inspected and the placement in the history with best performance is picked. If it is better than the former converged local solution, a better solution is produced and a new round of the local search in Algorithm 1 is started from the new solution.

The time complexity of the escaping method is $O(mn^2)$. Although it will usually come to a better solution, it needs lots of block moves, which results in high system overhead. Moreover, as we will show in Sec. IV, the improvement of the breakout extension is limited. Therefore, our proposed LBM is enough to produce good solutions without the breakout method.

IV. Simulation

We conduct simulations driven by real workload traces collected from a production cluster in the Windows Azure Storage (WAS) system. It contains the request traces of 252 equal-sized original data blocks *in every second* for a 22-hour period. We adopt a systematic $(6, 3)$ RS code, and the blocks will be placed on 20 server nodes. We assume that 5% of all requests may encounter block unavailability events, which happen randomly. During degraded read when the original block is unavailable, the requests are randomly directed to the combinations of nodes that can reconstruct the original block.

The properties of the data are demonstrated in Fig. 1. Fig. 1(a) is the average request per block at different times. Fig. 1(b) shows the distribution of the total number of requests for each block. Fig. 1(c) is the statistical value of $\mathbb{E}((\sum_i D_i)^2)/\sum_i \mathbb{E}(D_i^2)$ for each 2-hour measurement period, which will influence the worst-case performance of our algorithm according to Theorem 3. We can see that it is no greater than 50 and leads to an approximation ratio of 5.08.

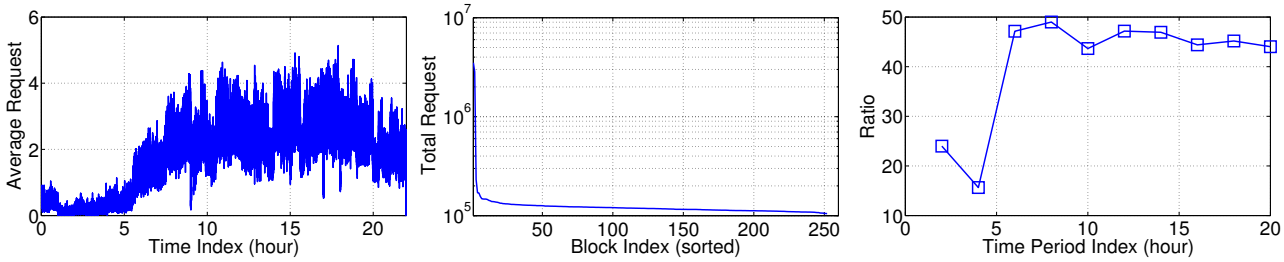
We first evaluate our Local Block Migration algorithm in terms of the reduction on the objective function in different measurement periods. To mimic the behavior in real systems, we have also conducted a round-based simulation to replay the real request traces, considering queue accumulation over different time periods.

A. Performance of the LBM Optimization Algorithm

We test our algorithm in each 2-hour measurement period and present results from 3 typical sample periods in Fig. 2. Assuming a 5% block unavailability rate and random load direction for degraded reads, we can get the empirical mean and covariance matrix of all the 378 *coded blocks* in each 2-hour period. We perform Local Block Migration (LBM) to find the optimal block placement in each sample period, and also adopt the breakout method after the LBM algorithm converges. We compare our algorithm to choosing the best out of 1000 random placements (which requires shuffling and thus lots of block moves).

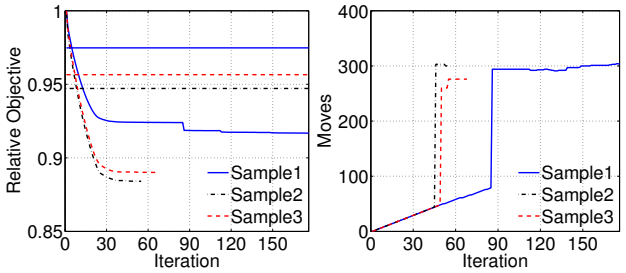
Fig. 2 shows how the LBM reduces the objective function in (1) from a random initial placement as well as the number of block moves incurred. In Fig. 2(a), the decreasing curves indicate the performance of the LBM while the horizontal lines are the reference performance of the best out of 1000 random placement. For each 2-hour sample period, the breakout takes place at the iteration where there is a sudden increase of moves in Fig. 2(b).

We can see that LBM can effectively reduce the optimization objective and clearly outperforms the global optimal approximation provided by the best out of 1000 random placements. More importantly, in LBM, most of the reduction on the objective is achieved within 30 iterations (thus 30 block moves) while choosing the best out of 1000 random



(a) Average requests per block (b) Distribution of total requests per block (c) $\mathbb{E}\big(\big(\sum_i D_i\big)^2\big)/\sum_i \mathbb{E}(D_i^2)$ over time

Fig. 1. The properties of the trace data collected from Windows Azure Storage (WAS).



(a) Objective v.s. Iteration (b) Necessary Moves v.s. Iteration

Fig. 2. The performance of the Local Block Migration Algorithm with breakout extension. The horizontal lines in Fig. 2(a) indicate the reference performance of the best of 1000 random placement and the decreasing curves indicate the performance from the LBM. For each sample, the performance without breakout extension can be spot on iterations before the "sudden" rise of the moves in Fig. 2(b).

placements will typically incur a large overhead of more than 350 moves. Furthermore, although the breakout can further reduce the objective, such further reduction is limited. Given that the breakout will incur a large number of additional moves, we can gain most of the benefit from LBM by running up to a certain number of iterations without the breakout extension.

B. Queuing Simulation with Trace Replay

Now we replay the request traces in a round-based simulator (each round being one second) and simulate the queuing effect of the system under different placement strategies. Again, we divide the 22-hour trace into 11 2-hour measurement periods. For each measurement period, we adjust the block placement by LBM with the maximum iteration set to 20, using the request statistics from the previous 2-hour measurement period as the prediction of the request rates in the current measurement period. We feed the requests per second into the simulator assuming block unavailability happens randomly with a ratio of 5%. The request processing rate of each server is set such that the peak demand will utilize 70% of server capacity.

We initialize the placement by applying LBM to the statistics of the first 2-hour measurement period and evaluate the queuing performance in the remaining 10 measurement periods, where LBM is applied at the beginning of each 2-hour measurement period. We compare LBM to two schemes. One is the pure fixed random placement, which is a typical method adopted in industry today [1]. The other is dynamically adjusting the placement in each 2-hour measurement period by picking the best out of 1000 random placements, which we call the Best Random.

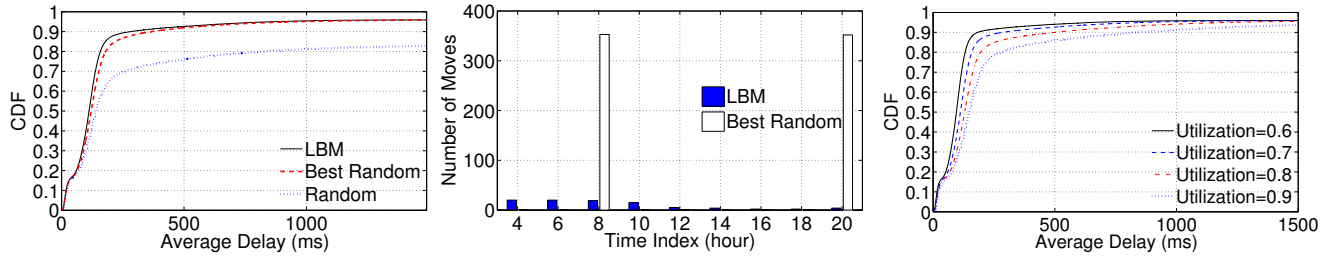
Fig. 3(a) plots the distribution of the average request delay, while Fig. 3(b) shows the number of block moves during each 2-hour measurement period. We can see that LBM can greatly reduce the access delay over the typical random placement. LBM can even beat the Best Random with only a small number of block moves in each measurement period, while the Best Random always needs lots of global shuffling to keep the placement optimal, bringing about high migration overhead, and is thus impractical. Fig. 3(c) shows the performance of LBM under different server processing rates, characterized by different levels of peak demand utilization.

Fig. 4 shows the average queue length of all the servers as time goes. We can see that with LBM, the queues will generally be stable, while the random placement will suffer under peak demands due to unbalanced server loads and poor utilization of server capacities.

V. Related Work

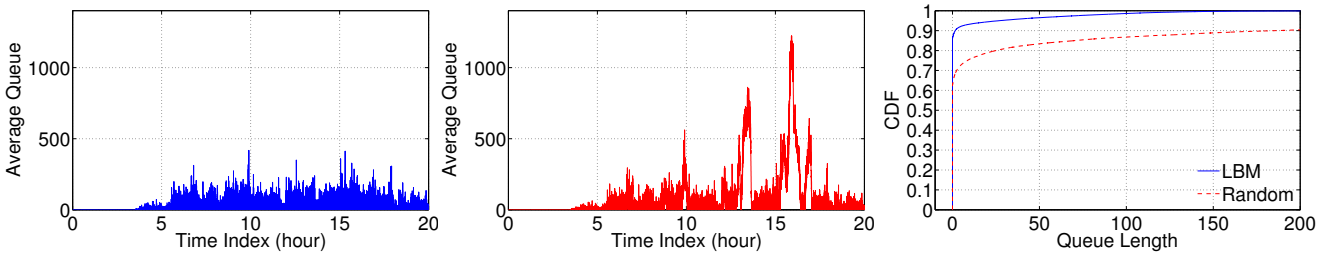
Abundant works are on enhancing the storage overhead and reducing the recover cost for the degraded reads. In [4], Local Reconstruction Code (LRC) is proposed to reduce the storage overhead. The works in [6], [19] focus the optimization of the degraded reads and better load direction scheme to boost the performance of the degraded reads was presented. M. Xia et al. in [7] use two different erasure codes that dynamically adapt to system load to achieve both the overall low storage overhead and low recovery cost. HitchHiker [8] propose a new encoding technique to improve recovery performance.

There are extensive works around the content placement problem in replication based systems with different desired QoS. In [20], Rochman et al. propose the strategies of placing the resources to distributed regions to serve more requests locally. In [21], Xu et al. propose a reasonable request mapping and response routing scheme to maximize the total utility of serving requests minus the cost. Bonvin et al. [22] propose a distributed scheme to dynamically allocate the resources of a data cloud based on net benefit maximization regarding the utility offered by the partition and its storage and maintenance cost. In [23], the automatic data placement across geo-distributed datacenters is presented, which iteratively moves a data item closer to both clients and the other data items



(a) Average request delay distribution (b) Number of block moves over time (c) Average delay (ms) under different utilization

Fig. 3. Performance of LBM, Best Random (the best out of 1000 random placements), and Random placement in the round-based trace replay.



(a) Local Block Migration (b) Random (c) Distribution of queue lengths

Fig. 4. The simulated queue lengths in each time slot (second).

that it communicates with. B. Yang et al. [24] study the content placement problem for systems when multiple items are needed in each request and the item size is small. They try to maximize the correlation of the contents stored on the same server to reduce the IO and the CPU overhead to fulfill a request at a time. On the contrary, our work focus on the applications in which the size of content block is large and each request only relates to one block. And we study the content placement for erasure coded systems.

In [9], [10] new parallel download scheme to optimize the delay performance of coded storage are proposed. Their work rely on parallel downloads to leverage a queueing-theoretical gain, where each request must access k or more servers, and abort the remaining download threads when k blocks are obtained.

In the line of the mathematical technique, the local search idea to solve the Max-2-Cut Problem is first proposed in [15] . It is improved with better efficiency in [16]. W. Zhu et al. [17] extend it to solve the mathematical Max-k-Cut Problem. In our work, we take the local search idea to solve the problem with challenging special constraints related to the real application and we provide a linear time searching scheme.

VI. Conclusion

In this paper, we study the problem of reducing access latency in erasure coded storage systems through block migration and content placement optimization. Based on request rate statistics, we have built a model to minimize the expected request waiting times, which is similar to the NP-Complete Min-k-Partition problem with a special additional constraint. We propose Local Block Migration which moves the block that reduces the latency objective the most at a time. We theoretically characterize the algorithm's worst-case approximation ratio, which depends on a demand variation measure across blocks. Through trace-driven simulations based on request traces from Windows Azure Storage, we show that in the presence of skewed demands, Local Block Migration can significantly reduce the access latency in erasure coded storage by only moving a few blocks once in a while, without global shuffling. Furthermore, the computation of such desired moves can be done within 1 second for 252 original blocks stored with a $(6, 3)$ RS code on tens of servers.

Acknowledgement

The authors would like to thank Dr. Cheng Huang for helpful discussions and for sharing the traces used in the evaluation.

References

[1] D. Borthakur, "Hdfs architecture guide," *HADOOP APACHE PROJECT http://hadoop. apache. org/common/docs/current/hdfs design. pdf*, 2008.

[2] S. Ghemawat, H. Gobioff, and S.-T. Leung, "The google file system," in *ACM SIGOPS operating systems review*, vol. 37, no. 5. ACM, 2003, pp. 29–43.

[3] B. Calder, J. Wang, A. Ogus, N. Nilakantan, A. Skjolsvold, S. McKelvie, Y. Xu, S. Srivastav, J. Wu, H. Simitci *et al.*, "Windows azure storage: a highly available cloud storage service with strong consistency," in *Proceedings of the Twenty-Third ACM Symposium on Operating Systems Principles*. ACM, 2011, pp. 143–157.

[4] C. Huang, H. Simitci, Y. Xu, A. Ogus, B. Calder, P. Gopalan, J. Li, S. Yekhanin *et al.*, "Erasure coding in windows azure storage." in *Usenix annual technical conference*. Boston, MA, 2012, pp. 15–26.

[5] H. Weatherspoon and J. D. Kubiatowicz, "Erasure coding vs. replication: A quantitative comparison," in *Peer-to-Peer Systems*. Springer, 2002, pp. 328–337.

[6] O. Khan, R. C. Burns, J. S. Plank, W. Pierce, and C. Huang, "Rethinking erasure codes for cloud file systems: minimizing i/o for recovery and degraded reads." in *FAST*, 2012, p. 20.

[7] M. Xia, M. Saxena, M. Blaum, and D. A. Pease, "A tale of two erasure codes in hdfs," in *To appear in Proceedings of 13th Usenix Conference on File and Storage Technologies*, 2015.

[8] K. Rashmi, N. B. Shah, D. Gu, H. Kuang, D. Borthakur, and K. Ramchandran, "A hitchhiker's guide to fast and efficient data reconstruction in erasure-coded data centers," in *Proceedings of the 2014 ACM conference on SIGCOMM*. ACM, 2014, pp. 331–342.

[9] G. Liang and U. C. Kozat, "Fast cloud: Pushing the envelope on delay performance of cloud storage with coding," *Networking, IEEE/ACM Transactions on*, vol. 22, no. 6, pp. 2012–2025, 2014.
[10] Y. Sun, Z. Zheng, C. E. Koksal, K.-H. Kim, and N. B. Shroff, "Provably delay efficient data retrieving in storage clouds," *arXiv preprint arXiv:1501.01661*, 2015.
[11] E. Schurman and J. Brutlag, "The user and business impact of server delays, additional bytes, and http chunking in web search," in *Velocity Web Performance and Operations Conference*, 2009.
[12] V. Kann, S. Khanna, J. Lagergren, and A. Panconesi, "On the hardness of approximating max k-cut and its dual," *Chicago Journal of Theoretical Computer Science*, vol. 2, p. 1997, 1997.
[13] R. M. Karp, *Reducibility among combinatorial problems*. Springer, 1972.
[14] V. Li, Q. SHUAI, and Y. Zhu, "Performance models of access latency in cloud storage systems," in *Proc. Fourth Workshop on Architectures and Systems for Big Data*, 2014.
[15] B. W. Kernighan and S. Lin, "An efficient heuristic procedure for partitioning graphs," *Bell system technical journal*, vol. 49, no. 2, pp. 291–307, 1970.
[16] C. M. Fiduccia and R. M. Mattheyses, "A linear-time heuristic for improving network partitions," in *Design Automation, 1982. 19th Conference on*. IEEE, 1982, pp. 175–181.
[17] W. Zhu, G. Lin, and M. Ali, "Max-k-cut by the discrete dynamic convexized method," *INFORMS Journal on Computing*, vol. 25, no. 1, pp. 27–40, 2013.
[18] P. Morris, "The breakout method for escaping from local minima," in *AAAI*, vol. 93, 1993, pp. 40–45.
[19] Y. Zhu, J. Lin, P. P. Lee, and Y. Xu, "Boosting degraded reads in heterogeneous erasure-coded storage systems."
[20] Y. Rochman, H. Levy, and E. Brosh, "Resource placement and assignment in distributed network topologies," in *INFOCOM, 2013 Proceedings IEEE*. IEEE, 2013, pp. 1914–1922.
[21] H. Xu and B. Li, "Joint request mapping and response routing for geo-distributed cloud services," in *INFOCOM, 2013 Proceedings IEEE*. IEEE, 2013, pp. 854–862.
[22] N. Bonvin, T. G. Papaioannou, and K. Aberer, "A self-organized, fault-tolerant and scalable replication scheme for cloud storage," in *Proceedings of the 1st ACM symposium on Cloud computing*. ACM, 2010, pp. 205–216.
[23] S. Agarwal, J. Dunagan, N. Jain, S. Saroiu, A. Wolman, and H. Bhogan, "Volley: Automated data placement for geo-distributed cloud services." in *NSDI*, 2010, pp. 17–32.
[24] B. Yu and J. Pan, "Location-aware associated data placement for geo-distributed data-intensive applications," in *Proc. of IEEE Infocom 2015*, 2015.

APPENDIX A

Proof of Proposition 1: It is not hard to figure out that (CP) and (CMKP+) have the same region of feasible solutions. We will show that every feasible solution achieves the same object value in both problems to complete the proof.

For any solution $\{y_1, y_2, \ldots, y_n\}$, consider each particular $k \in \mathbb{Z}^+$ with $0 < k \leq m$ and consider the corresponding group of coded blocks $\{i|y_i = k\}$, the contribution of the group to (1) is

$$\begin{aligned}\frac{1}{2}\mathbb{E}(L_k) =& \frac{1}{2}\mathbb{E}\big(\big(\sum_{j:y_j=k} D_j\big)^2\big)\\ =& \frac{1}{2}\sum_{i:y_i=k}\sum_{j:y_j=k;j\neq i}\mathbb{E}(D_i\cdot D_j) + \frac{1}{2}\sum_{i:y_i=k}\mathbb{E}(D_i^2)\\ =& \sum_{i<j:y_i=y_j=k}\mathbf{W}_{ij} + \frac{1}{2}\sum_{i:y_i=k}\mathbf{W}_{ii},\end{aligned}$$

which is exactly the contribution in (6). After summing up over k, we can get that the objective functions of (1) and (6) have the same value at the given feasible solution. □

Proof of Theorem 2: We will prove it by contradiction. Suppose $\{y_1, y_2, \ldots, y_n\}$ is a converged solution, in which, without loss of generality, there exists i_s and j_s such that $y_{i_s} = y_{j_s}$ and $G_{i_s} = G_{j_s}$.

We only consider the case that

$$\sum_{k:k\neq i_s; G_k\neq G_{i_s}}\mathbf{W}_{i_s k} \leq \sum_{k:k\neq j_s; G_k\neq G_{j_s}}\mathbf{W}_{kj_s},$$

since the proof is similar for the other way around. For simplicity of presentation, we define $S(i_s)$ as the set of the indices of servers on which no block from the same group of i_m is placed. Considering the sum of the gains by moving i_s to all the servers in $S(i_s)$, we have

$$\begin{aligned}&\sum_{k\in S(i_s)} g_k(i_s)\\ &\geq (m-\alpha+1)\sum_{k:y_k=y_{i_s};k\neq i_s}\mathbf{W}'_{i_s k} - \sum_{k:G_k\neq G_{i_s}}\mathbf{W}'_{i_s k}\\ &\geq (m-\alpha+1)\mathbf{W}'_{i_s j_s} - \sum_{k:G_k\neq G_{i_s}}\mathbf{W}_{i_s k}\\ &\geq (m-\alpha+1)\epsilon > 0,\end{aligned}$$

which indicates that there exists a $k \in S(i_s)$, such that $g_k(i) > 0$. This contradicts the assumption that $\{y_1, y_2, \ldots, y_n\}$ is a solution from Algorithm 1 since the algorithm will only terminate when there is no positive $g_j(i)$. □

Proof of Theorem 3: The proof will depend on the following lemma and properties:

Lemma 6. *The solutions from the Local Block Migration Algorithm are solutions to (MKC) with an approximation ratio of* $1 - 1/m$.

Proof. Suppose that $\{y_1, y_2, \ldots, y_n\}$ is a converged solution. Due to local optimality, $\forall k' \neq k$, we have

$$\sum_{i:y_i=k}\sum_{j:y_j=k'}\mathbf{W}'_{ij} \geq \sum_{i:y_i=k}\sum_{j:y_j=k;j\neq i}\mathbf{W}'_{ij}. \tag{21}$$

Since there are $m-1$ possible k' values, by adding up (21) over all $k' \neq k$, we have

$$\sum_{i:y_i=k}\sum_{j:y_j\neq k}\mathbf{W}'_{ij} \geq (m-1)\sum_{i:y_i=k}\sum_{j:y_j=k;j\neq i}\mathbf{W}'_{ij}. \tag{22}$$

Dividing (22) by $m-1$ and adding $\sum_{i:y_i=k}\sum_{j:y_j\neq k}\mathbf{W}'_{ij}$ on both sides, we have

$$\frac{m}{m-1}\sum_{i:y_i=k}\sum_{j:y_j\neq k}\mathbf{W}'_{ij} \geq \sum_{i:y_i=k;j\neq i}\sum_{j}\mathbf{W}'_{ij}. \tag{23}$$

By summing up (23) over all k and dividing it by 2, we have

$$\frac{m}{m-1}\sum_{i<j}\mathbf{W}'_{ij}(1-\delta(y_i-y_j)) \geq \sum_{i<j}\mathbf{W}'_{ij} \geq \mathrm{OPT}_C,$$

where OPT_C is the optimal value for the Max-k-Cut problem, completing the proof. □

Lemma 7. *Given $f_{ij}(\mathbf{W})$ defined in (18), for any arbitrary positive ϵ, we have*

$$\sum_{i\neq j}\mathbf{W}'_{ij} \leq \epsilon + \frac{m}{m-\alpha+1}\sum_{i\neq j}\mathbf{W}_{ij}.$$

Proof. By (9) and (18), we have

$$\begin{aligned}\sum_{i\neq j}\mathbf{W}'_{ij} \leq& \sum_{i\neq j}\mathbf{W}_{ij} + \sum_i \sum_{j:j\neq i;G_j=G_i}\mathbf{W}'ij\\ \leq& \sum_{i\neq j}\mathbf{W}_{ij} + \\ & \sum_i \sum_{j:j\neq i;G_j=G_i}\big(\epsilon + \frac{1}{m-\alpha+1}\sum_{k:k\neq i;G_k\neq G_i}\mathbf{W}_{ik}\big)\\ \leq& \sum_{i\neq j}\mathbf{W}_{ij} + \frac{\alpha-1}{m-\alpha+1}\sum_{i\neq j}\mathbf{W}_{ij} + (\alpha-1)n\epsilon\\ =& \epsilon' + \frac{m}{m-\alpha+1}\sum_{i\neq j}\mathbf{W}_{ij},\end{aligned}$$

proving the lemma. □

We have another property for the weight matrix $\mathbf{W}$:

$$\begin{aligned}&\sum_{i\neq j}\mathbf{W}_{ij}\big/\sum_i \mathbf{W}_{ii} = \sum_{i\neq j}\mathbb{E}(D_iD_j)\big/\sum_i \mathbb{E}(D_i^2)\\ =&\big(\mathbb{E}\big(\big(\sum_i D_i\big)^2\big) - \sum_i \mathbb{E}(D_i^2)\big)\big/\sum_i \mathbb{E}(D_i^2)\\ =&\frac{\mathbb{E}\big(\big(\sum_i D_i\big)^2\big)}{\sum_i \mathbb{E}(D_i^2)} - 1.\end{aligned} \tag{24}$$

We are now ready to prove Theorem 3. Let OPT_P be the optimal value for (MKP). By Lemma 6 and (16), we have

$$\sum_{i<j}\mathbf{W}'_{ij} - \sum_{i<j}\mathbf{W}'_{ij}\delta(y_i-y_j) \geq \Big(1-\frac{1}{m}\Big)\big(\sum_{i<j}\mathbf{W}'_{ij} - \text{OPT}_P\big). \tag{25}$$

Reducing (25) and adding $\frac{1}{2}\sum_i \mathbf{W}_{ii}$, we have

$$\begin{aligned}&\sum_{i<j}\mathbf{W}'_{ij}\delta(y_i-y_j) + \frac{1}{2}\sum_i \mathbf{W}_{ii}\\ \leq&\Big(1-\frac{1}{m}\Big)\Big(\text{OPT}_P + \frac{1}{2}\sum_i \mathbf{W}_{ii}\Big) + \frac{1}{m}\Big(\sum_{i<j}\mathbf{W}'_{ij} + \frac{1}{2}\sum_i \mathbf{W}_{ii}\Big).\end{aligned} \tag{26}$$

By the property of $\mathbf{W}$ in (24) and Lemma 7, we have

$$\begin{aligned}&\sum_{i<j}\mathbf{W}'_{ij} + \frac{1}{2}\sum_i \mathbf{W}_{ii}\\ \leq&\epsilon + \frac{m}{m-\alpha+1}\sum_{i<j}\mathbf{W}_{ij} + \frac{1}{2}\sum_i \mathbf{W}_{ii}\\ \leq&\epsilon + \frac{m}{m-\alpha+1}\Big(\frac{\mathbb{E}\big(\big(\sum_i D_i\big)^2\big)}{\sum_i \mathbb{E}(D_i^2)} - 1\Big)\cdot\frac{1}{2}\sum_i \mathbf{W}_{ii} + \frac{1}{2}\sum_i \mathbf{W}_{ii}\\ \leq&\epsilon + \Big(\frac{m}{m-\alpha+1}\Big(\frac{\mathbb{E}\big(\big(\sum_i D_i\big)^2\big)}{\sum_i \mathbb{E}(D_i^2)} - 1\Big) + 1\Big)\cdot\frac{1}{2}\sum_i \mathbf{W}_{ii}.\end{aligned} \tag{27}$$

By the feasibility of the solution, we also have

$$\sum_{i<j}\mathbf{W}'_{ij}\delta(y_i-y_j) = \sum_{i<j}\mathbf{W}_{ij}\delta(y_i-y_j). \tag{28}$$

From (26), (27) and (28), we have

$$\begin{aligned}&\sum_{i<j}\mathbf{W}_{ij}\delta(y_i-y_j) + \frac{1}{2}\sum_i \mathbf{W}_{ii}\\ \leq&\epsilon' + \Big(1 + \frac{1}{m-\alpha+1}\Big(\frac{\mathbb{E}\big(\big(\sum_i D_i\big)^2\big)}{\sum_i \mathbb{E}(D_i^2)} - 1\Big)\Big)\cdot\\ &\Big(\frac{1}{2}\sum_i \mathbf{W}_{ii} + \text{OPT}_P\Big).\end{aligned} \tag{29}$$

With the current setting of $\mathbf{W}'$, the optimal solution to (MKP) is also a feasible solution to the corresponding (CMKP+) problem, and also an optimal solution to (CMKP+). Combining it with the fact that ϵ is an arbitrary positive constant and the result in (29), we have completed the proof. □

Proof of Theorem 5: We show that in every iteration, if the solution from the previous iteration satisfies (7), in the current iteration, for every move that violates (7), there exists at least one other move that achieves a larger gain and yet does not violate (7).

Without loss of generality, consider all the gains achieved by moving the block i_s to other servers, respectively. There are two types of destination servers. One kind contains the blocks within the same group of i_s and the other kind does not. For all the destination servers not containing the blocks within the same group of i_s, denoted as $S(i_s)$, we have

$$\begin{aligned}&\frac{1}{m-\alpha}\sum_{k\in S(i_s)} g_k(i_s)\\ \geq&\frac{1}{m-\alpha}\Big((m-\alpha)\sum_{k:y_k=y_{i_s};k\neq i_s}\mathbf{W}'_{i_sk} - \sum_{k:G_k\neq G_{i_s}}\mathbf{W}'_{i_sk}\Big)\\ =&\sum_{k:y_k=y_{i_s};k\neq i_s}\mathbf{W}'_{i_sk} - \frac{1}{m-\alpha}\sum_{k:G_k\neq G_{i_s}}\mathbf{W}'_{i_sk}\\ =&\sum_{k:y_k=y_{i_s};k\neq i_s}\mathbf{W}'_{i_sk} - \frac{1}{m-\alpha}\sum_{k:G_k\neq G_{i_s}}\mathbf{W}_{i_sk}.\end{aligned}$$

By the definition of $\mathbf{W}'$ and (20), for any destination k_s containing a block j_s within the same group of i_s, we have

$$\begin{aligned}g_{k_s}(i_s) =& \sum_{k:y_k=y_{i_s};k\neq i_s}\mathbf{W}'_{i_sk} - \sum_{k:y_k=k_s}\mathbf{W}'_{i_sk}\\ \leq& \sum_{k:y_k=y_{i_s};k\neq i_s}\mathbf{W}'_{i_sk} - \mathbf{W}'_{i_sj_s}\\ \leq& \sum_{k:y_k=y_i;k\neq i}\mathbf{W}'_{ik} - \epsilon - \frac{1}{m-\alpha}\sum_{k:k\neq i_s;G_k\neq G_{i_s}}\mathbf{W}_{i_sk}.\end{aligned}$$

Therefore, we have

$$\frac{1}{m-\alpha}\sum_{k\in S(i_s)} g_k(i) > g_{k_s}(i),$$

which indicates that there exists at least one move with a better gain and yet does not violate the constraint, completing the proof. □