mathematics

MDPI

Article

A Novel Memory Concurrent Editing Model for Large-Scale Video Streams in Edge Computing

Haitao Liu [1,2], Qingkui Chen [1,3,*] and Puchen Liu [4]

[1] Business School, University of Shanghai for Science and Technology, Shanghai 200093, China; liuhaitao@lyu.edu.cn
[2] Office of Information, Linyi University, Linyi 276002, China
[3] School of Optical-Electrical and Computer Engineering, University of Shanghai for Science and Technology, Shanghai 200093, China
[4] Department of Statistics, Shanghai Polytechnic University, Shanghai 201209, China; pcliu@sspu.edu.cn
* Correspondence: chenqingkui@usst.edu.cn; Tel.: +86-131-2238-1881

Abstract: Efficient management and utilization of edge server memory buffers are crucial for improving the efficiency of concurrent editing in the concurrent editing application scenario of large-scale video in edge computing. In order to elevate the efficiency of concurrent editing and the satisfaction of service users under the constraint of limited memory buffer resources, the allocation of memory buffers of concurrent editing servers is transformed into the bin-packing problem, which is solved using an ant colony algorithm to achieve the least loaded utilization batch. Meanwhile, a new distributed online concurrent editing algorithm for video streams is designed for the conflict problem of large-scale video editing in an edge computing environment. It incorporates dual-buffer read-and-write technology to solve the difficult problem of concurrent inefficiency of editing and writing disks. The experimental results of the simulation show that the scheme not only achieves a good performance in the scheduling of concurrent editing but also implements the editing resource allocation function in an efficient and reasonable way. Compared with the benchmark traditional single-exclusive editing scheme, the proposed optimized scheme can simultaneously enhance editing efficiency and user satisfaction under the restriction of providing the same memory buffer computing resources. The proposed model has a wide application to video real-time processing application scenarios in edge computing.

Keywords: concurrent editing; memory models; edge computing; large-scale video streams

MSC: 68M07; 68M20

Citation: Liu, H.; Chen, Q.; Liu, P. A Novel Memory Concurrent Editing Model for Large-Scale Video Streams in Edge Computing. *Mathematics* **2023**, *11*, 3175. https://doi.org/10.3390/math11143175

Academic Editors: Ming Yang and Liqun Shan

Received: 28 June 2023
Revised: 13 July 2023
Accepted: 18 July 2023
Published: 19 July 2023

1. Introduction

With the deepening popularity and application of smart education, the cloud recording and live broadcasting system for university campuses has evolved into a modern teaching solution. Based on IoT technology, the campus cloud recording and broadcasting system is able to use high-speed transmission and edge network storage technology to achieve video capture and processing use, which is demanding for real-time computing capability and network transmission. In the application scenario of large-scale video streaming data processing, concurrent video editing plays a critical role.

The video editing systems widely used by content suppliers have existed for decades, including the tape-based linear editing system and the disk-based non-linear editing system. For those subject to very tight deadlines, requiring assigning work to multiple editors for concurrent processing, it is necessary to employ clusters of non-linear editing servers that support non-linear editing, where content can be shared among multiple editors to support effective concurrent collaboration [1].

In traditional team video creation, it is a tedious process to review and revise. The process requires multiple circulations of larger video files among team members, resulting in inconvenient communication for revision. Video concurrent processing supports different team members to co-create storyboard scripts and edit at different locations, marks comments frame by frame, and associates related personnel, which can effectively improve the efficiency of team video creation [2]. To highlight the characteristics of a long video creation process, such as many people in different links and irregular locations from shooting to editing operations, the concurrent collaboration function can provide a sharing space for team creators. It allows team users to realize multi-terminal cooperation in processing videos, from resource sharing to the whole process of creativity, script, shooting, editing, post-production, review and modification, and data operation. Through the concurrent collaboration function, the edited videos are stored in the cloud and can be shared with other users in the form of web pages by generating a sharing link with one click, thus solving the problems such as slow transmission of large files and repeated uploads and eliminating the problem of low efficiency in team collaboration material management and transmission [3–5]. When modifying, users can precisely position each frame for annotation, add arrows, selections, and comments, as well as edit records through the video timeline at any time to facilitate efficient team communication. As the size of the team expands, factors such as the time occupied by material copies, data management, and collaborative production take an increasingly important position in the team's production efficiency. We constantly receive demands from editing teams for creative collaboration, expecting to make it easy for every type of author to enjoy creating quality work in a consistent, simple, and efficient way.

There are three typical scenarios for concurrent editing of video streaming in smart education: (1) a recorded video is independently edited by one teacher user; (2) a recorded video is edited by multiple teacher users after segmentation; and (3) a recorded video is edited by multiple teacher users concurrently and collaboratively. Scenario ① and scenario ② are relatively simple, where teacher users select the corresponding time slots to complete editing according to the video content they want to edit. In scenario ②, the system provides a division function according to time slots in the editing task scheduling page. In addition, a complete video file is saved in small fragments on the cloud platform, creating conditions for later combinations into sub-video segments with varied time lengths. However, for scenario ③, collaborative editing among multiple teacher users becomes very complicated. First, the time when teacher users go online for editing work is random; and second, a conflict emerges when multiple teacher users co-edit, i.e., the same group of teachers will operate the same attribute of the video at the same time during the editing process and generate operation conflicts. Solutions for handling concurrent operation conflicts in real time include the following: ① Adding locks. When teacher A processes a video clip, it is locked in the video clip and only available to teacher A for editing, while other teachers need to wait until teacher A finishes processing and submits it before they can edit it. ② Overwrite. For teachers who process the same video segment i, whoever processes it later will have the final right to process the result, and other people's earlier operations are discarded. ③ Users handle conflicts by themselves. In case of a conflict, the teacher handling the operation will deal with the conflict himself/herself before merging the submission. In this paper, the large-scale distributed concurrent editing model and key algorithms are designed and implemented. Compared with previous work, the main contributions of this paper are as follows:

(1) Considering the constraints of the real editing concurrent requirements transferred to the editing buffer, utilizing the bin-packing, the idea of one-time loading of similar editing or multiple loading in successive sets is proposed. The model and algorithm to maximize the buffer space utilization through optimization are proposed, which provides an effective strategy for the buffer loading of complexity;

(2) By combining the above strategy with the ant colony algorithm, the shortcomings and deficiencies of the ant colony algorithm are effectively overcome by using heuristic information, inspiring a new idea for solving the buffer loading of complexity;
(3) The distributed video streaming online co-editing algorithm is proposed, where different teachers can process the video at the same time without locking and waiting, thus bringing faster, efficient video processing, and the overall processing time converges to that of an individual teacher's in the video source.

The framework of this paper is structured as follows: Section 2 focuses on the research foundation; Section 3 describes the related work; Section 4 is the dual-buffer co-editing model; Section 5 is the experimental design and analysis; and Section 6 details the conclusion.

2. Background

In the application of real-time processing of large-scale video streaming, the optimization of concurrent team creation efficiency of video grows into a significant challenge in video processing. Researchers design concurrent processing optimization algorithms to improve real-time processing efficiency. The main ideas of concurrent editing are as follows: A memory double buffer is created, and then the distributed storage slice file is transferred into memory, where the core data is transferred into the first buffer to complete the editing operation. When the buffer reaches saturation, the pointers of the double buffer are then exchanged, so the lock is released, and a thread finishes the work of depositing the data of the first buffer to disk. During this process, other threads can continue to acquire locks and quickly read and write memory, then release the locks so that other threads can continue to edit the second buffer.

The concurrent editing of large-scale video streaming in smart education applications requires arithmetic support provided by servers in the edge data center. The mechanism of concurrent editing operations on servers in the edge data center is as follows: Firstly, the cache layer uses a file system, which has better performance in handling random I/O operations. After each video completes the video slice, the media header information is copied before and after each slice video so as to create an independent video sub-file sequence. Before and after each slice file, a 4 M non-editable area can be added, and the split joint of sub-files belongs to the "no trace" state. The media header information of the sub-files is deleted directly after editing is completed, and the non-editable area remains unchanged and can be merged directly, which completes the process from splitting to merging large files. Secondly, the distributed files are operated in segments. The segmented video is transferred to the memory buffer for editing, and when the state of one buffer reaches saturation, the pointer is swapped, and the video data is written to disk. At the same time, another buffer continues to be unlocked and is open for other editing users to use.

Every edited segment forms an edit queue and is placed in a buffer for editing. Multiple users can edit concurrently at the same time. Regarding the barrel effect, if someone is slow to edit, other users completing segments editing have to wait for the delayed editing before they can be merged and written to disk. There are various data fragments, and one buffer for each fragment is not practical. The number of users editing concurrently is usually not too large, such as having dozens of users editing concurrently. The buffer queue can be made dynamically configurable, with several buffer queues for several people. Each data fragment is numbered first, then the fragments are evenly distributed to each person's buffer queue according to the idle algorithm, and the edited fragments enter the data merge queue. After each large segment of data fragment is edited, the merge queue is then merged and written to storage so that the respective edit queues are mutually independent. That is, some are busy, and some are idle, but the distribution and merging of data are imperceptible to other editors. Borrowing the thinking of MapReduce distributed/parallel computing [6], there are other editors who can queue up to enter the editing area [7]. The concurrent editing framework for video streaming is shown in Figure 1. (01, 02, . . . , 06) are the video slices numbers in the memory editing buffer, represented by

the original sequence numbers of the storage chain table obtained after the complete video slicing, which cannot be changed after being given.

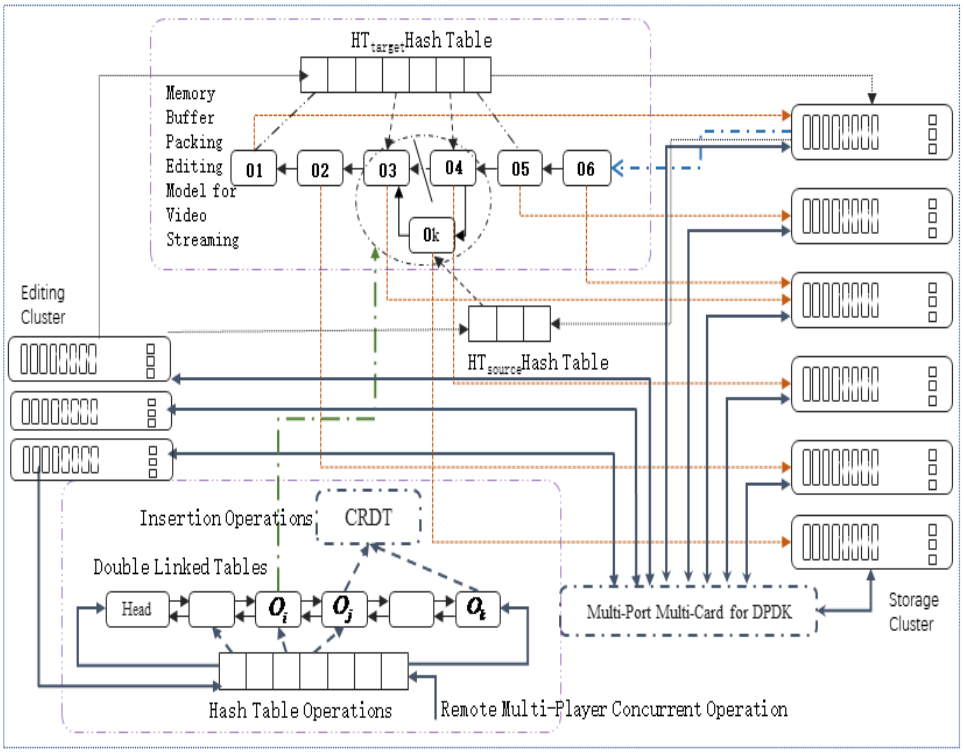

Figure 1. Concurrent editing framework for video streaming.

2.1. Video Streaming Media

Streaming media transmits audio, video, and multimedia files in stream form over a network. Streaming media compresses and stores continuous audio and video information on a web server, and users watch the video while downloading the data. This is achieved through the key technology of streaming, which converts multimedia files such as animation, audio, and video into a sequence of compressed packets by means of specific compression algorithms and transmits them in real time from the video server to the users.

2.2. Satisfaction Function

Satisfaction is a numerical way to measure and portray the psychological state of a customer's pleasure after their needs are satisfied. The quantification is a real number in the interval [0, 1]. A value of 1 indicates that the user's needs are fully satisfied, while a value of 0 indicates that the user's minimum needs are not satisfied. The satisfaction function is a mathematical expression of satisfaction, which is a mapping relationship established between the customer's needs and the psychological state of pleasure after the needs are satisfied. The mathematical description is the following: Let $M_i = \{M_1, M_2, \ldots M_n\}$ be the demand that the user is satisfied with n, and $S = S_1, S_2, \ldots S_n$ be the satisfaction set, then the satisfaction function F is defined as $F{:}M_i \rightarrow S$.

2.3. GOP Editing

According to the MPEG video coding standard, a group of images (GOP) is a set of contiguous images inside a video encoded in MPEG. Each video encoded in the MPEG standard consists of a variety of consecutive image sets. The length of the image collection is the number of frames between one I-frame and the next I-frame. The composition of the GOP is shown in Figure 2.

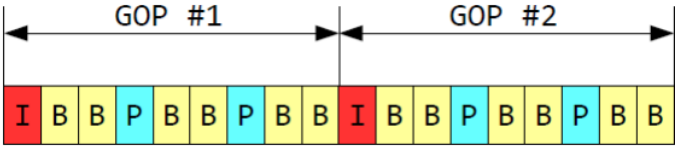

Figure 2. GOP composition.

MPEG coding classifies pictures (i.e., frames) into I, P, and B, wherein Frame-I is the internal coding frame, Frame-P is the forward prediction frame, and Frame-B is the two-way interpolation frame. Frame-I is the key frame, which can be understood as a complete picture, while Frame-P and B record the changes relative to Frame-I. Frame-P indicates the difference from the previous frame, and Frame-B indicates the difference between the front and back frames. Without Frame-I, Frame-P and Frame-B cannot be decoded, which is the reason why MPEG format is difficult to edit accurately, and why you have to fine-tune the head and tail.

The concurrent editing process of video transfers the slice file from distributed storage into memory, decompressing it into GOP, editing Frame-Is, exchanging the pointers of double buffers after the editing is completed, then compressing and writing it back to the hard disk.

3. Related Work

3.1. Video Transmission

The optimization of video transmission efficiency has become an important challenge in networks. Researchers have proposed many transmission structure optimization algorithms to improve network transmission efficiency. Hu has designed a video transmission optimization strategy that takes a reinforcement learning approach to improve video transmission efficiency and quality on an edge computing platform. However, the compression and decoding in video transmission optimization were not analyzed and did not directly reduce redundant traffic in video transmission [8]. Qi introduced DPDK to ensure that all video fragments have the same size so as to accelerate video transmission by bypassing the OS kernel [9]. Tashtarian proposed ROPL, a learning-based edge client request management solution; however, scalability was not achieved [10]. Park compared and analyzed network layer high-speed packet processing techniques in Linux, such as DPDK and XDP [11].

3.2. Collaborative Editing

Collaborative editing, as the most important research area in Computer Supported Cooperative Work (CSCW), aims to solve the problem of conflicts or data loss due to inconsistent operations between users working together in the same group. A well-established collaborative editing system can support harmonious human–computer interaction and allow online time-shared editing by multiple people in different locations. Therefore, the focus of collaborative editing research is on collaborative algorithms, which are mainly classified into OT (Operational Transformation) and CRDT (Conflict-free Replicated Data Type) [12–14].

The OT (Operational Transformation) algorithm is a real-time collaborative editing technique based on the idea that once a local editing operation is executed, the operation is immediately converted into a form adapted to the remote operation and transmitted to the server side. Therefore, the OT algorithm contains two core transformation functions: (1) Transform (clientoperation and serveroperation) converts the local operation of the client into the remote operation of the server, and (2) Transform (serveroperation and clientoperation) converts the server's remote operation to the client's local operation. The two conversion functions ensure that the OT algorithm enables multi-user real-time editing and that the content of documents available to different clients remains consistent. The two conversion functions ensure that the OT algorithm can achieve real-time editing by multiple users and that the content of documents is always consistent across clients. Despite the

advantages, the drawbacks shall not be neglected. The two transformation functions make the implementation of the OT algorithm relatively complex, and various combinations of operations and boundary problems need to be handled. Also, each operation has to be transformed. Again, the performance of the OT algorithm is negatively affected when more users are involved in the same group. Kumar divides the shared document into multiple sections and restricts each section to be edited by only one user, thereby addressing the issue of conflicts that may arise when multiple writers simultaneously write to the same location. LiteDoc, proposed by Kumar, is a simplified approach to achieve fast, scalable, and robust collaborative editing. This mechanism not only eliminates conflicts but also avoids the need for complex operational transformation (OT), differential synchronization, and rollback modules [15]. Under the primary limitations of the basic DAL approach, Fan proposed a shared lock method to fully support unconstrained real-time collaborative programming with semantic conflict prevention. Unfortunately, the DAL technology has not been further extended [16]. Bath designed and evaluated a web-based system for real-time collaborative editing of raster and vector images, which, however, did not address the existing limitations in synchronization and access conflicts [17].

CRDT (Conflict-free Replicated Data Types) is a conflict avoidance replicable data type, a data structure that solves the data synchronization problem in distributed systems. The copies do not require complex operational transformations and can be updated independently and in parallel without the occurrence of conflicts as assured. Unlike OT algorithms that focus on operation transformation, ORDT is a data structure in its definition, and its core idea is not how to resolve conflicts but to directly design a data structure that can avoid conflict generation [14,18–21]. Currently, CRDT algorithms are divided into two main categories: (1) State-based CRDTs (State-based CRDTs or Convergent Replicated Data Types, CvRDTs), which include the current state of the sending node copy in each message, and (2) Operation-based CRDTs (Operation-based CRDT or Commutative Replicated Data Types, CmRDTs), which will carry the updated operations performed by the local node from the last broadcast on each message. Compared with OT algorithms, CRDT algorithms do not need to design complex operation conversion functions and are relatively simple to implement. Meanwhile, CRDT algorithms can be decentralized using P2P technology and are more suitable for distributed systems. Zhao constructs a CRDT that enables exchangeable concurrent update operations by modeling a shared document as a linear text buffer and assigning globally unique and fully ordered identifiers to each basic element. Furthermore, Zhao also defines a set of correctness properties for malicious users and external adversaries and proves that the proposed Byzantine fault-tolerance mechanism guarantees the implementation of these properties and proposes a lightweight solution that achieves Byzantine fault-tolerance with a low runtime overhead [22].

At present, online collaborative editing algorithms mainly focus on text editing, table data editing, and image editing, and research on editing video streaming is still lacking. In this paper, a practical, efficient, and stable distributed video stream online co-editing algorithm is designed and abbreviated as the DVSCoE algorithm, based on CRDT to address the co-editing needs of teachers on campus cloud recording and live streaming platform.

4. Dual-Buffer Co-Editing Model

4.1. Dual-Buffer Optimization Scheme

In a live cloud recording and broadcasting platform, the video recorded by the teacher is divided into several small fragments, each of which lasts 3 to 5 s. These small fragments are stored on multiple servers in the edge cluster, and each fragment has a unique search number. In the video editing system, there are a large number of video streams that need to be transmitted concurrently at high speed and in real time. Usually, 1 G video files are transmitted to complete the read and write transmission work at once, and the video is divided into small-scale streams of continuous transmission of GOPs, which are transmitted concurrently using DPDK's multi-port multi-queue User Datagram Protocol

(UDP) protocol [23]. A video stream concurrent editing double-buffer model is established, as shown in Figure 3.

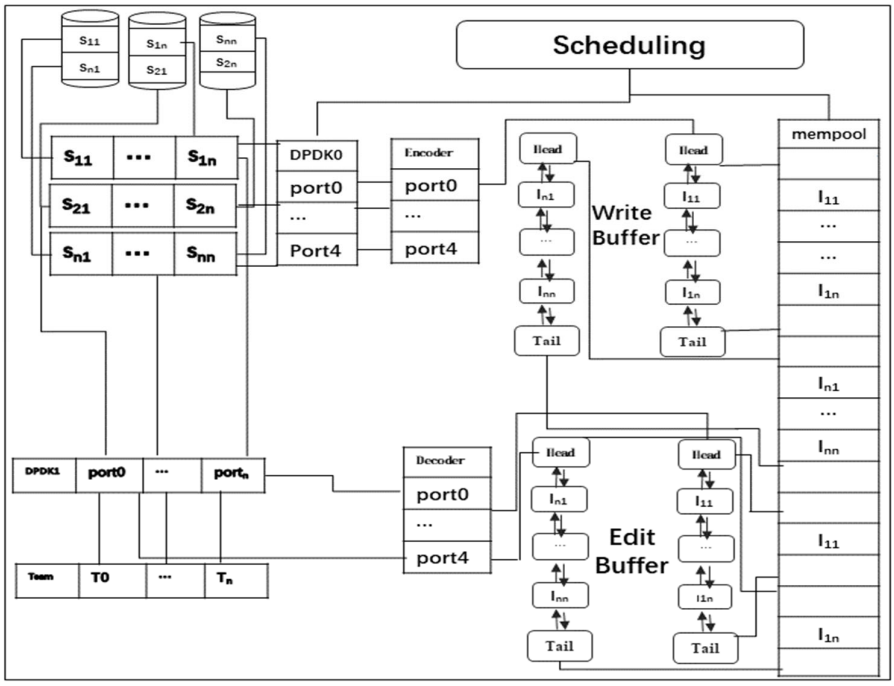

Figure 3. Video stream concurrent editing double-buffer model.

This section aims to allow multiple teachers to implement online collaborative editing of recorded videos. Memory management and other underlying operations are all practical applications of the boxing problem. Since there is no algorithm to find the exact solution in effective time for the current NP-complete problem, the solution to the boxing problem is extremely challenging. Hence, the boxing algorithms proposed successively are various approximate algorithms. Here, we draw on the ant colony algorithm to solve the memory concurrent editing buffer scheduling problem [24].

Ant colony optimization (ACO) metaheuristic probabilistically constructs solutions using a parameterized probability model, which is indicated by the pheromone trails. The solution-searching process is usually associated with the pheromone trail. Ants probabilistically add the pheromone trail to the partial solution until they generate a completely feasible solution. During these iterations, the pheromone values are dynamically updated based on the information derived from some high-quality solutions to force the search to concentrate on regions containing high-quality solutions in the solution space. It is a powerful algorithm to solve NP problems in the field of computing intelligence [22].

Hence, we leverage the ACO metaheuristic to solve the memory concurrent editing buffer scheduling problem [24]. To efficiently obtain an optimal online collaborative editing strategy, the memory buffer resource allocation algorithm is designed based on ACO (MEMO-ACO).

The basic symbols used in this article are described in Table 1.

An edit queue for each person is opened in a limited buffer. The large number of user editing requirements requires a large amount of buffer space, which is usually limited. Therefore, the buffer needs to be requisitioned multiple times to meet the user's editing needs. More users are selected to fill up the buffer, leading to the total number of allocated batches being minimal [25]. Our goal is to minimize the buffer requisition batches while achieving user satisfaction.

Table 1. Symbol descriptions.

Symbol	Descriptions
MEMO	Buffer space available
DEM	Collection of user editing requirements
m	Number of requirements
dem_i	The *i*-th user edit request
Degree	User satisfaction ratio limit
mem_i	Buffer space requirement for demand dem_i
P_{total}	Total number of batches using the buffer
P_{total}^{best}	Total number of batches using buffers under global optimal solution
d_{li}	Upper limit of waiting time for dem_i
P_{cur}^{i}	Number of batches that dem_i has waited before using the buffer
x_{ij}	1 for dem_i to use buffer in batch *j*; 0 otherwise
y_{ij}	1 for dem_i waiting time before using buffer not exceeding d_{li}; 0 otherwise
x_{ij}^{best}	dem_i into batch *j* under global optimal solution is 1; 0 otherwise
$Time_j$	Processing time of batch *j*

Goals:
min(P_{total}).
Constraints:

$$\sum_{dem_i \in DEM} (x_{ij} \cdot mem_i) \leq MEMO \tag{1}$$

$$\sum_{dem_i \in DEM} y_{ij} \geq m \cdot degree. \tag{2}$$

An edit queue for each person is opened in a limited buffer. The large number of user editing requirements requires a large amount of buffer space, which is often limited. Therefore, the buffer needs to be requisitioned multiple times to meet the user's editing needs. More users are selected to fill up the buffer, leading to the total number of allocated batches being minimal [25]. Our goal is to minimize the buffer requisition batches while achieving user satisfaction.

The updating of pheromone is defined as follows:

$\tau_{ij}(t+1)$ denotes the pheromone value of putting dem_i into batch *j* in round $t+1$ iteration.

$$\tau_{ij}(t+1) = (1-\rho)\tau_{ij}(t) + \Delta\tau_{ij}(t) \tag{3}$$

$$\Delta\tau_{ij}(t) = \frac{x_{ij}^{best}}{p_{total}^{best}} \cdot y_{ij} \cdot \omega_1 (\omega_1 : \text{coefficient factor}) \tag{4}$$

Heuristic information $\eta_{ij}(t+1)$ denotes the pheromone value of putting dem_i into batch *j* in round $t+1$ iteration.

$$\eta_{ij}(t+1) = \frac{p_{cur}^{i}}{dl_i} \cdot \omega_2 (\omega_2 : \text{coefficient factor}) \tag{5}$$

The transfer probability is defined as follows:

$$R_{ij}(t+1) = \frac{(\tau_{ij}(t+1))^{\phi} \cdot (\eta_{ij}(t+1))^{\varphi}}{\sum_{dem_i \in DEM} (\tau_{ij}(t+1))^{\phi} \cdot (\eta_{ij}(t+1))^{\varphi}} \tag{6}$$

In every iteration, it is necessary to evaluate the fitness of the solution. To realize the optimization goal, the following metrics shall be considered: total number of batches using the buffer and waiting time before using the buffer not exceeding the deadline. So, the

evaluation of fitness is a compound function of the two above. The evaluation function is defined as follows:

$$\text{fitness(t)} = \frac{1}{p_{total}(t)} \cdot \sum_{dem_i \in DEM} y_{ij} \cdot \omega_3 (\omega_3 : \text{coefficient factor}) \quad (7)$$

The detailed algorithm is shown in Algorithm 1.

Algorithm 1 MEM-ACO algorithm

Input: MEMO; DEM = {dem_1, dem_2, … dem_m}
Output: scheduling strategy for every dem_i

1. Set parameters, initialize pheromone trails, etc.
2. while termination condition not met do
3. for 1 to n (number of ants)
4. Set the buffer as empty
5. Set every dem_i as unselected
6. for every dem_i do
7. if already selected do
8. continue
9. else
10. while exists available space in the buffer
11. Calculate heuristic information according to (5)
12. Select some demand to enter the buffer with roulette algorithm according to (6)
13. Set the demand as selected and update buffer status
14. if no demand can be selected do
15. Update satisfaction degree matrix and switch to the next ant
16. Go to 4
17. else
18. Go to 6
19. end if
20. end while
21. end if
22. end for
23. Obtain batch scheduling strategy for every dem_i
24. if satisfy constraints do
25. if find a better solution do
26. Update best solution
27. end if
28. end if
29. Apply updating rule (3)
30. end while

As in Algorithm 1, the main idea of the MEM-ACO algorithm is described as follows:

(1) From line 2 to line 22, the MEM-ACO algorithm is used to search available solutions for buffer space allocation in every iteration. In line 4, initially, the buffer is set as empty for allocation. In line 5, every demand is set as unselected for allocation. Then, from line 6 to line 22, buffer space is allocated to unselected demands. Especially in line 11, the heuristic information is calculated, and some demands are selected to enter the buffer with roulette algorithm according to the transfer probability. After that, the satisfaction degree matrix is updated, and the next ant is switched to if no demand can be selected. Otherwise, continue to select some demand to enter the buffer;

(2) From line 12 to line 28, a new solution is obtained so as to decide whether to update the optimal solution.

Algorithm 1 runs until it converges or reaches the maximum iteration number. For each iteration in Algorithm 1, the time complexity of the path search for every evacuation transfer is O ($m \times n$).

In terms of buffer design, this paper uses a dual-buffer optimization scheme for separating IO processing from logical processing: the write buffer and the edit (read) buffer. Among them, the write buffer holds the operation-linked table of the video stream after collaborative editing by the client and the linked table of external video sources used during the operation. The write thread sequentially processes the video streams stored in the memory according to the linked table in the write buffer. The edit buffer reads the memory-linked table of the target video stream according to the client's demand, as well as constructs the operation-linked table for subsequent co-editing. This solution has the following advantages:

(1) Separate IO operations and logical processing threads;
(2) Enhance concurrency by enabling IO threads and logical processing threads to work simultaneously;
(3) Reduce the reading and writing of video streams in the disk by recording only operations and addresses;
(4) Decouple producers and consumers while providing underlying support for subsequent collaborative algorithms.

The two kinds of linked tables mentioned in the double-buffer optimization scheme serve as the basis of the algorithm for the concurrent editing of video streams proposed in this paper. Furthermore, the memory double-buffer slice and lock algorithm are used. When memory buffer 1 is found to be full, then the pointers to the two buffers are exchanged. Then, buffer 1 directly releases the lock, and then this thread flushes the data to disk, such that the process of writing back to disk does not occupy the lock. Then, all subsequent threads can continue to acquire the lock, write to memory quickly, and then release the lock. Through concurrency optimizations, disk swipes and memory writes can be performed fully in parallel at the same time. Considering that the core point here is to significantly reduce the time occupied by locks, and if only a single memory buffer is used, then the process of reading data from it and swiping it to disk will also require a lock. The thread requiring the lock to write to the memory buffer will be prevented from obtaining the lock. So, if only a single buffer is used, it will definitely lead to the process of reading memory data, and writing to disk will take a long time to occupy the lock. This leads to a large number of threads stuck in the lock acquisition, unable to obtain the lock, and then unable to write the data to memory.

4.2. Concurrent Editing Algorithm

In the live cloud recording platform, a complete recorded video is divided into 3–5 s unit segments, and the video stream selected for editing by the teacher is composed of multiple unit segments. The editing of the recorded video by the teachers of the same course focuses on the insertion and deletion of the unit segments. Therefore, the video stream independently processed by one teacher is considered linear data and represented as a double-linked table. The processing work on the video is defined only as the insertion and deletion operations on the double-linked table [26–28].

In the storage process on the distributed server, the storage information of the unit slice is represented as a data structure with four tuples, <unit slice ID, storage location, next pointer, and pre-pointer>, and a hash table with the unit slice ID as the key and the location in the information storage table as the value. Obviously, a centralized saving method is utilized for the recorded videos. During the collaborative editing process among the teachers in the same group, the client corresponding to each teacher copies a segment of the linked table from the central node for extracting the video unit piece from the cloud according to the current task content. In the case of the insert operation, the client copies the corresponding linked table segment from the source video (as insert material) and the target video, respectively. In addition, the client generates corresponding hash tables for both video chains, HT_{target} and HT_{source}.

Unique identifier: Each teacher is represented by a unique identifier. The identifier consists of both the user's $creatorID$ and the teacher's counter for the operation, i.e., $ID(creatorID, counter)$.

The DVSCoE algorithm represents the list of elements corresponding to the video streams as a double-linked table, which is called the operation table S in this chapter. Inspired by the YATA algorithm, each element in the operation table is represented by an insertion operation. When an insertion operation is deleted, it is not removed from the table itself, but marked as deleted instead, i.e., the tombstone method. This section defines each insert operation in the operation table as $O(ID, origin, originLeft, originRight, position, isDeleted, VideoID)$, where ID denotes the unique identifier of the operation; $VideoID$ denotes the ID of the new video unit slice inserted; $isDeleted$ is a Bool marker when $isDeleted = True$ denotes that the operation is deleted (otherwise, it is executed normally); $origin$ denotes the base point for creating the insert operation, here denoted by the original sequence number of the storage linked table obtained after the complete video slice, which cannot be changed after given; and $originLeft$ and $originRight$ denote that the insert operation is deleted when inserted and the other denotes the front and rear drive nodes in the double-linked table when the insertion operation is inserted simultaneously with other insertion operations. The values will change as the conflict is handled; $position$ denotes the execution position of the insertion operation on the storage linked table. When a new operation is inserted into the local operation linked table S, it is also broadcasted to other remote nodes [28,29].

Corresponding to the definition of operations above, we define HT_{target} and HT_{source}:

$$HT_{target} = \{origin : videoItem | origin \in \{0, \ldots, N\} \text{ and } vidoeItem \in \{0, \ldots, n\}\}$$

$$HT_{source} = \{videoID : videoItem | origin \in \{0, \ldots, M\} \text{ and } vidoeItem \in \{0, \ldots, m\}\}$$

where the origin and $VideoID$ denote the location of the data structures in the copy target and source linked tables in the original linked table in the distributed server, respectively; N and n denote the lengths of the target video original and copy linked tables, respectively; and M and m denote the lengths of the source video original and copy linked tables, respectively.

Intent retention: The intent of the current insertion operation is retained when and only when the insertion operation is executed in the linked table of video unit slice in the cloud server.

In order to realize the teacher's intention to retain the editing operation and ensure the real-time display during the operation, here utilize HT_{target} and HT_{source} to achieve a two-step synchronization: (1) synchronization for the real-time display of the client video, and (2) synchronization for the final consistency of the video in the cloud.

(1) Conflict

In order to achieve two-step synchronization, it is important to first understand the conflict when teachers collaborate in editing [29,30]. Since each teacher usually operates an ordered video list in terms of video unit slices during editing, for example, inserting a video list of length ≥ 1 into the original video linked table, the unit slices in the inserted ordered video list can form a meaningful video in order, and usually, the inserted video list is a fragment of a single material video linked table copied by the teacher on the cloud server (to protect the material video data from being corrupted by changes). During the collaborative editing process of the same group, different teachers intercept at different locations and at different lengths during the interception process, and the list of inserted videos intercepted is not necessarily the same. This brings about two types of conflicts:

① Multiple video lists are inserted distributively between left and right, with no overlapping unit slices between each video list;
② Multiple video lists are inserted distributively between left and right, with overlapping unit slices between the video lists.

To resolve the above two conflicts, the insertion operation of editing a single video list is transformed into an insertion operation with multiple video unit slices of the same

length. We define the function $<_c$, which specifies the position of O_{new} in the set of insert operations $(O_1.c, O_2.c, \ldots, O_n.c)$ where the conflict occurs. When $O_i.c <_c O_{new}.c <_c O_j.c$, O_{new} can be inserted between O_i and O_j. In addition, we also define that $<$ indicates the preposition on the double-linked table S (Note: When $O_i < O_j$, it indicates that in the double-linked table, O_i is on the left side of O_j, not necessarily adjacent), then O_i is the preposition of O_j. O_i is the preposition of O_j, or both are the same.

Inspired by the YATA algorithm, a strict full-order function $<_c$ for solving the conflict when collaboratively inserting video streams, together with three rules are proposed:

Rule 1: In $<_c$, keep transferability:

$$O_i <_{rule1} O_j \Leftrightarrow \forall O : O_i <_c O_j, O_j <_c O \rightarrow O_i <_c O \Leftrightarrow \nexists O : O_j <_c O <_c O_i.$$

Rule 2: The origin connection line crossings for conflicting operations are prohibited, i.e.:

$$O_i <_{rule2} O_j \Leftrightarrow O_i.c < O_j.origin \ \vee \ O_j.origin < O_i.origin.$$

Rule 3: For two conflicting operations with the same origin, the operation with the smaller videoID is placed on the left:

$$O_i <_{rule3} O_j \Leftrightarrow O_i.origin \equiv O_j.origin \ \wedge \ O_i.videoID < O_j.videoID.$$

Rule 4: When two conflicting insert operations with the same origin also have the same videoID, set the insert operation with the larger creator ID as the tombstone, i.e.:

$$O_i <_{rule4} O_j \Leftrightarrow O_i.origin \equiv O_j.origin \wedge O_i.videoID \equiv O_j.videoID \wedge O_i.ID.creator < O_j.ID.creator \wedge O_j.isDeleted = True$$

According to the proof process in the YATA algorithm, Rule 1, Rule 2, and Rule 3 can ensure that the order function $<_c$ is antisymmetric, transferable, and holistic when the inserted unit slice IDs are different. Rule 4 deals with the overlapping unit slices in the video list during insertion on the basis of Rule 3. It is still ordered in itself, only that in Rule 4, O_j, as a successor of O_i, is masked during execution. Therefore, the order function is still a strictly fully ordered function under Rule 1, Rule 2, and Rule 4.

(2) Insertion algorithm

For the client, when teachers perform insertion operations, it is represented in the visual operation as inserting a video segment into the current recorded video. And in the DVSCoE algorithm, the insertion process of a video segment of length n is firstly decomposed into n insertion operations, where n denotes the number of unit slices of the inserted video segment saved in the cloud. Each insertion operation has the same origin and videoID corresponding to the cell slice in the video segment. The specific insertion algorithm is shown in Algorithm 2.

(3) Two-step synchronization

Step 1: According to the above insertion algorithm, it is possible to generate an operation table on the teacher clients in the same group of video editing. Since the base point of the operation record in the operation table is based on the original video-linked table sequence, real-time collaborative work of different teachers can be achieved by HT_{target} and HT_{source}. It is worth noting that the length and starting point of the local video segment of the teachers working together can be different; that is, any teacher can be allowed to join or quit during collaborative editing.

Step 2: When each teacher finishes and saves editing and exits, the client will upload the local operation list to the central editing server node, and the central node will merge the operations according to their IDs. At the end of the editing of the final teacher in the same group, the operation list saved on the central node performs the editing operation.

The two steps, respectively, achieve (1) real-time display of the video during editing; and (2) final consistent synchronization of the original recorded video in the cloud. The

corresponding in the dual-buffer optimization scheme is that the read buffer provides (1) the double-linked table for editing and the address of the video stream clip for display; the write buffer records the final operation linked table and the address of the external video stream. The two buffers cooperate with each other to achieve the concurrent synergy of online editing of the final video stream.

Algorithm 2 Insertion of operation i into operation table L

Input: I; L = {L_1,L_2, ... L_m}
Output: L
1. insert (I, L):
2. i.c = 0
3. for o in L do:
4. // Rule 1
5. if (o.c < i.origin or i.origin $\leq$ o.origin)
6. and (o.origin ! = i.origin or o.videoID < i.videoID) do:
7. // Rule 2 and Rule 3
8. i.c = o.c + 1
9. else:
10. if (o.videoID == i.videoID) do:
11. // Rule 4
12. i.isDeleted = True
13. if (i.origin > o.origin) do:
14. break

5. Experimental Design and Analysis

An edge cluster was constructed using 10 servers and a 48-port Gigabit Layer 2 switching board. The configuration details of the servers are shown in Table 2. Each server is powered by an 8-core Intel Xeon E3-1230V2@3.3Ghz and two four-port Intel I350-T4 Gigabit network cards with eight network ports. The system is developed on the basis of DPDK version 17.11.3 (Data Plane Development Kit, Linux Foundation, San Francisco, CA, USA) and OVS version 2.9.0 (Linux Foundation, San Francisco, CA, USA). The DPDK consists of eight large memory pages, multiple ports receiving queues, and multiple send queues, allowing dynamic adjustment of the number of network ports to be accessed and forwarded. Table 2 shows the host configuration information.

Table 2. Host configuration information.

Name	Model/Version	Descriptions
CPU	Intel (R) Xeon (R) E31230 V2@3.3Ghz	2 × 4 cores with 8 threads
Memory	DDR31600Mhz	32 G
OS	Debian9.0/Centos7.0	-
kernel	Linux version 3.10.0862.14.4.e17.x86_64	-
NIC	I350-T4-4-port 1 Gb/s each port PCI-e X4	-
DPDK	DPDK17.11.3	8-Ports (2 × 4)

5.1. Transmission Rate

Each of the 10 servers in the video concurrent editing cluster has a different number of application connections, and the applications in the different nodes communicate with each other. The receiving server receives, filters, and re-parses these messages, which are then distributed to the destination applications to verify the optimization of the multi-NIC parallel transmission performance. A total of 2 NICs are designed here, each with 4 ports and 8 ports transmitting data separately. The transmission rate when transmitting 64 B small stream packets is tested by sending 10 GB of data 10 times, and the experimental results are shown in Figure 4.

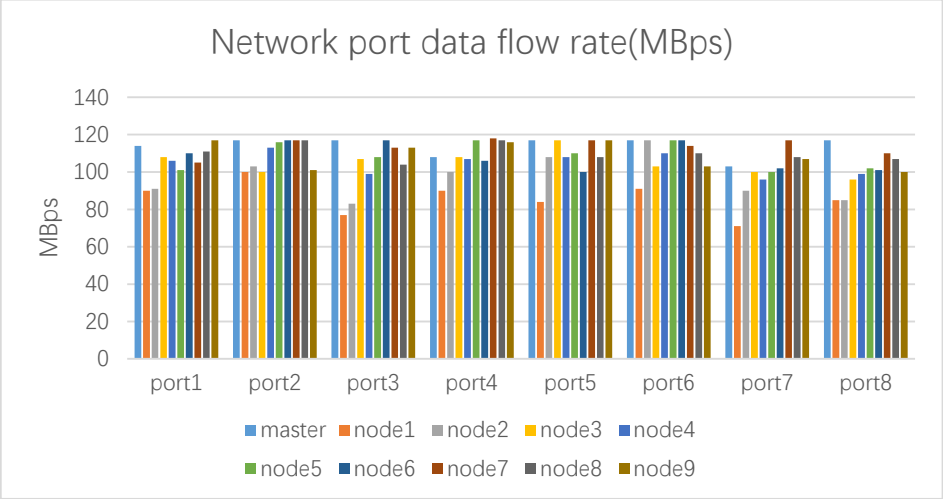

Figure 4. DPDK 8-port transmission rate.

5.2. Scheduling Batches

When the fixed buffer size is 512 MB, the number of comparison batches gradually increases with the increase in editorial staff demand, and the results are shown in Figure 5. When fixing the number of editorial staff demands, the number of batches varies inversely with the buffer size, and the results are shown in Figure 6.

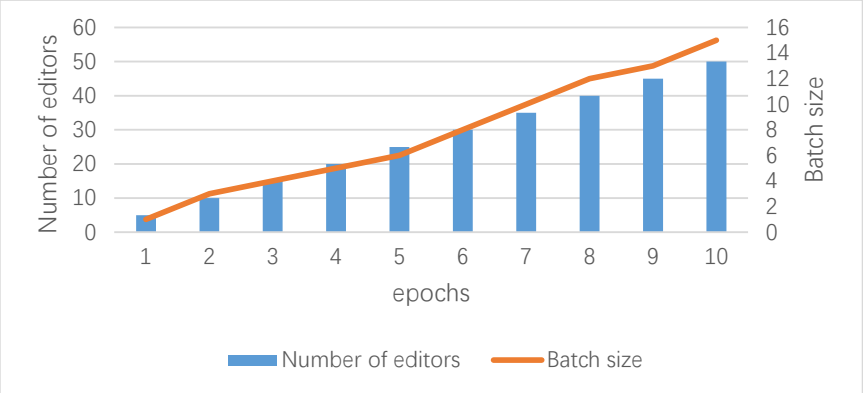

Figure 5. Editorial demand versus batch.

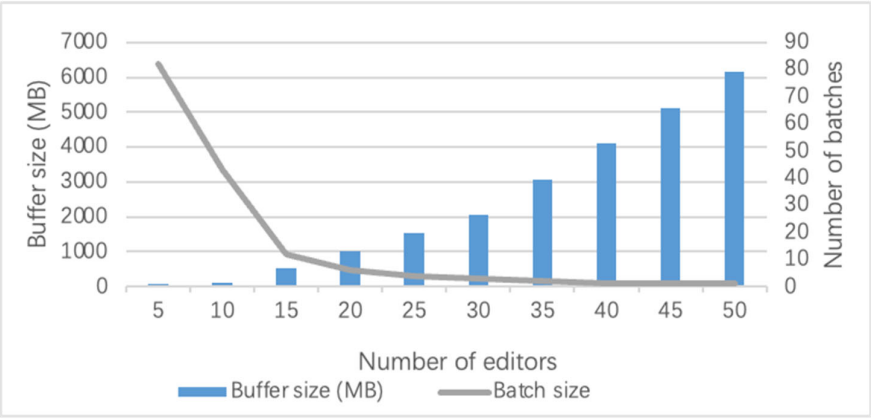

Figure 6. Relationship between editing batches and buffer size.

5.3. Satisfaction Experiment

Fifty college students majoring in new media are selected as editing members and concurrently edit the same video segments. The satisfaction is quantified to the interval of [0, 1]. The closer to 1, the better. After subjective training on their satisfaction, subjective satisfaction scores of the edited videos in the form of questionnaires are used to do subjective evaluation, and then statistical analysis is collected.

(1) The ant colony algorithm has 9.509% higher satisfaction average value and 0.946% smaller satisfaction variance compared with the traditional scheduling algorithm. It shows that the ant colony algorithm is more efficient and stable, and the results are shown in Table 3.
(2) When the data block size is 512 M, the satisfaction improves fastest when the edit buffer is 1024 M, and then it keeps increasing as the buffer increases. But the increase slows down after 4 times, indicating that the buffer size should be twice to three times the data block size for the best performance;
(3) The size of the data block and buffer will directly affect the size of the satisfaction level. The effect of these two parameters on editing satisfaction is tested and analyzed by varying the data block size and the buffer size allocated for each data stream.

Table 3. Satisfaction analysis.

Algorithm Comparison	Average Satisfaction Value	Satisfaction Variance
Ant colony algorithm	0.580738095	0.018091905
Unused ant colony algorithm	0.530309524	0.018263048
Comparison	9.509%	−0.946%

In the buffer size of a stream of data written to disk, the CPU needs to call the write disk pointer to write the location of the next stream of data corresponding to the location of the data block, and the pointer to move the offset is the size of the data block—the size of the buffer. Test data show that with a 64 MB~512 MB size of the data block, in the buffer size of 64 MB, relative to the 32 MB size of the data block, this time the buffer increases, the offset becomes smaller, so the write performance is improved, then the editing speed has been significantly improved. When the data block size is 64 MB~600 MB, the speed of concurrent editing of 50 data streams is firstly increased with the increase of buffer size, and the offset becomes smaller, so the speed of the writing disk is improved. However, as the buffer size continues to increase, the offset becomes very small, and when the last write disk is finished, and the next write disk is called, the disk has already been turned to the location to be written, and it needs to turn one more time to write, which makes the write performance decrease. A larger data block size does not always work. When the data block size increases to 600 MB, the offset becomes very large, and the resulting seek time overhead makes the write performance drop significantly, dragging down the whole editorial satisfaction experience. The results of the experiment are shown in Figure 7.

The data block and buffer size will directly affect the buffer pool performance. The impact of these two parameters on buffer editing performance is tested and analyzed by varying the data block size and the buffer size allocated for each data stream. Fifty video streams are launched for concurrent editing on three storage servers, and the bit rate size of the data streams is not limited in order to test the maximum throughput rate. The test results are shown in Figure 8.

As shown in Figure 9, the time to retrieve a one-minute video clip using the block-based (BB), frame-based (FF), and blended (BF) methods is shown. The x-axis of this figure indicates the size of a block with BB, measured by the number of frames that fit into the block. The y-axis indicates the time to retrieve a one-minute video clip. Note that FF is a special case of BB or BF when the number of frames in the block is 1. Due to excessive seek, FF requires longer service time compared to BB and BF. In short, BF is the best choice for FF and BB.

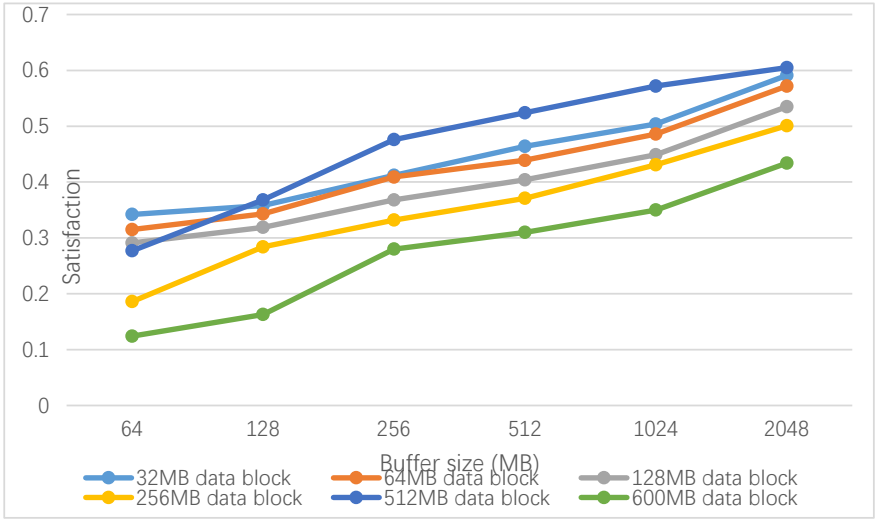

Figure 7. Impact of data block size on satisfaction.

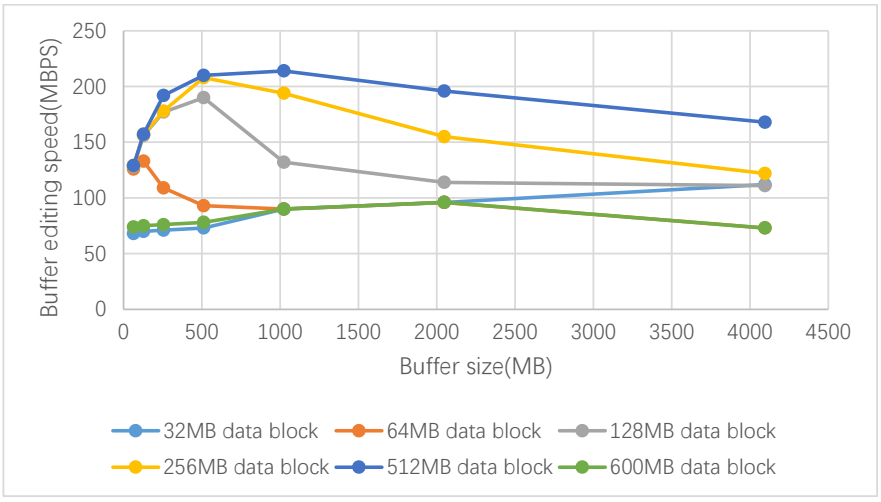

Figure 8. Effect of data block and buffer size on editing performance.

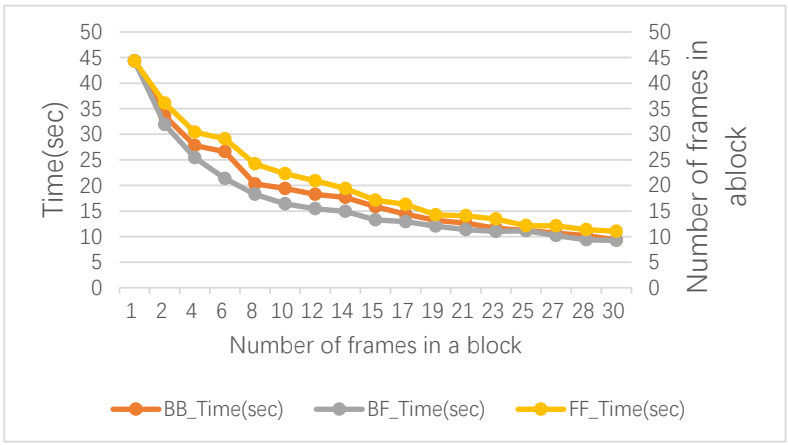

Figure 9. Time to retrieve a one-minute video segment.

As shown in Figure 10, when the number of video channels is 15, the equalization load ratio of the buffer reaches a maximum of 93.8%. When the number of video channels rises to 20, the buffer equalization load ratio drops sharply to 68%. As the number of video

channels continues to rise, the buffer scheduling rate decreases significantly. As the number of video channels rises and the editor stores a large number of concurrent video streams, the system needs to maintain more open file table entries, the memory overhead becomes larger, and the file descriptors to be written need to be switched back and forth, and the heads move more frequently, resulting in a significant degradation in editing performance.

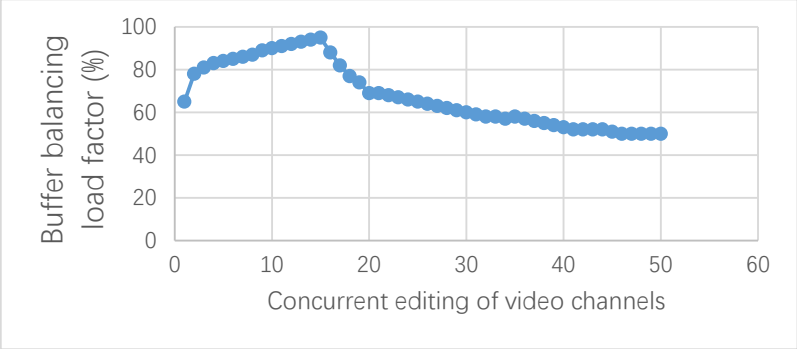

Figure 10. Effect of concurrent video count on buffer balancing load factor.

5.4. Algorithm Comparison

To evaluate the superiority of the algorithms, four algorithms, namely, the heuristic algorithm, the genetic algorithm, the greedy algorithm, and the ant colony algorithm proposed in this paper, are used. For a large video file of 1 G, 50 times of concurrent load scheduling editing are performed, and the obtained average calculation results are shown in Table 4. Among them, the heuristic strategy schedules the edits but does not consider the loading constraints. In contrast, the algorithm considers the constraints, and the two are easy to compare. The genetic algorithm considers a limited number of constraints. According to the above results, the heuristic algorithm has the highest space utilization, indicating that the space decomposition strategy can effectively use space. The space utilization and stability of the genetic algorithm are inferior to the ant colony algorithm, which is related to the defects of the genetic algorithm itself. The ant colony algorithm proposed has improved in both space utilization and stability. The reasons for this are the use of the space division method and concurrent editing one-time loading idea, which effectively avoids the waste of layout space. The use of dynamic information, such as heuristic information and initial value, cleverly overcomes the shortage of pheromones at the beginning of the ant colony algorithm's search. The constraints considered in the optimization process based on the algorithm of this paper are more comprehensive and consistent with the actual working conditions. Its calculation results show that space utilization has been improved, and the practicality is also enhanced. In addition, it can be seen from the table that the space utilization of the buffer decreases with the increase of concurrent edit types, which is due to the increasing complexity of creating and the decreasing completeness of the remaining space.

Table 4. Algorithm comparison.

Algorithms	Average Buffer Utilization%	Standard Deviation
Heuristic Algorithms	84.12	0.181
Genetic Algorithm	78.25	0.186
Ant Colony Algorithm	81.38	0.157
Greedy Algorithm	63.22	0.189

5.5. Concurrent Editing Experiments

To verify the effectiveness of the DVSCoE algorithm, Intellij IDEA functions as the experimental platform, and Java language is used to simulate the application environment of the algorithm to test the performance of the algorithm. Meanwhile, the whole experiment

is carried out on a cluster of 10 dual-way Intel (R) Xeon (R) E31230 V2@3.3 Ghz CPU servers. Multiple automated programs capable of random insertion or deletion are enabled on each server. Each program simulates a teacher who edited it online. In addition, five videos of 30 min in length are used as source videos during the experiments and divided into video fragments of 1 s in length for storage according to the characteristics of the linked table fragmentation in the algorithm.

In order to compare the improvement of video processing efficiency by the collaborative property of DVSCoE algorithm proposed in the paper, a lock-based collaborative editing method for multiple teachers (LCE) is also designed. Its core lies in the fact that when a teacher is editing a video segment, the client containing the same video segment editing will be locked until the previous teacher finishes editing.

In the experimental simulation of a multi-teacher collaborative editing scenario, the single video source is first randomly assigned video streams of 5 to 30 s in length according to the number of collaborating teachers, and there is a repeated overlay between the video streams. Figure 11 shows the trend of the number of different editing teachers compared to the average time required for editing when simulating the processing of 5 videos. As shown in Figure 10, under the collaborative algorithm with locks, in the beginning, the time required for editing decreases as the number of users increases. But after the number of users exceeds 20, the time required surges due to a large number of collaborators, bringing too much lock waiting time. On the other hand, in the DVSCoE algorithm proposed in this paper, different teachers can process the video at the same time without lock waiting, thus bringing more efficient video processing speed. The overall processing time tends to equal the processing time of a single teacher in the video source. Later, as the number of users increases, more conflicts arise during the collaborative processing, so the processing time for a single user shows a slight increase. It should be noted that in practical situations for the processing of a single video source, the number of participating teachers generally does not exceed 20, and the best results are achieved when 15 teachers edit concurrently. Thirty to fifty users participate in the experiments in order to try to improve the chances of conflicts in the experiments for overload testing (to verify the stability of the algorithm in extreme cases). The experimental results show that the method proposed in this paper can efficiently and feasibly achieve distributed collaborative video editing work for multiple teachers in practical situations.

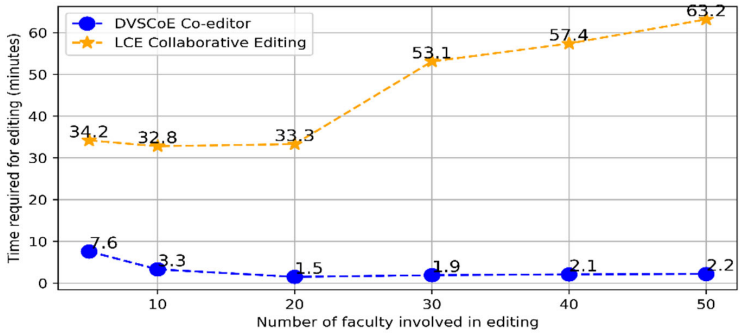

Figure 11. Trends in average video editing time as the number of teachers increases.

6. Conclusions

A large-scale distributed concurrent editing model and key algorithms are designed and implemented to address the need for real-time concurrent editing of large-scale streaming video data present in edge computing applications. Under the constraint of considering concurrent edits to be transferred to the server memory buffer, a strategy of loading similar edits at once or concentrating on multiple loads is proposed. A novel model and related algorithms are designed to maximize the buffer space utilization through optimization to provide an effective solution to the buffer loading problem of the edge server memory. An ant colony algorithm solves the problem, and heuristic information effectively overcomes the shortcomings and deficiencies of the ant colony algorithm. In the proposed DVSCoE

algorithm, different teacher users are allowed to concurrently process video data at the same time. The lock waiting problem of concurrent processing is solved, and then the efficiency of concurrent processing of large video data is improved, which makes the processing time of concurrent users converge to the processing time of a single user. The experimental results show that the proposed method can efficiently and feasibly implement distributed collaborative video editing for multiple teacher users in an edge cluster. The current limitations of difficult algorithm parameter search and slow convergence speed are present, and the future features of concurrent editing computational load are combined to further optimize the performance of the concurrent editing algorithm as well as improve the convergence speed of the optimized algorithm.

In the broader context of concurrent video editing, the best place to implement the process is in an edge data center, as this meets the need for multiple concurrent editors to edit video quickly, without prejudice to the consistency of video co-editing and low latency in writing to disk. The process can also be extended to servers in cloud data centers or personal high-performance editing computers, which need to meet appropriately high bandwidth and low latency. Thus, the proposed solution is applicable not only to very specific cases but also to a number of other broader scenarios.

Author Contributions: Conceptualization, H.L.; methodology, Q.C.; software, H.L.; validation, H.L. and P.L.; formal analysis, H.L. and P.L.; investigation, H.L.; resources, H.L. and Q.C.; data curation, H.L.; writing—original draft preparation, H.L.; writing—review and editing, H.L. and P.L.; visualization, H.L.; supervision, Q.C.; project administration, H.L.; funding acquisition, Q.C. and H.L. All authors have read and agreed to the published version of the manuscript.

Funding: This work was supported by the Shanghai Key Science and Technology Project (19DZ1208903); National Natural Science Foundation of China (Grant Nos. 61572325 and 60970012); Ministry of Education Doctoral Fund of Ph.D. Supervisor of China (Grant No. 20113120110008); Shanghai Key Science and Technology Project in Information Technology Field (Grant Nos. 14511107902 and 16DZ1203603); Shanghai Leading Academic Discipline Project (No. XTKX2012); Shanghai Engineering Research Center Project (Nos. GCZX14014 and C14001); Introduction and Cultivation Program for Young Innovative Talents of Universities in Shandong (2021QCYY003); and Natural Science Foundation of Shandong Province (No. ZR2023MF090).

Data Availability Statement: Data are available in a publicly accessible repository that does not issue DOIs. Publicly available datasets were analyzed in this study.

Acknowledgments: The authors would like to thank the editors and anonymous reviewers for their valuable comments and suggestions and acknowledge the support from the Flow Computing Laboratory at the University of Shanghai for Science and Technology.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Ghandeharizadeh, S.; Kim, S.H. Design of multi-user editing servers for continuous media. *Multimed. Tools Appl.* **2000**, *11*, 101–127. [CrossRef]
2. Dambra, S.; Samela, G.; Sassatelli, L.; Pighetti, R.; Aparicio-Pardo, R.; Pinna-Déry, A.-M. Film editing: New levers to improve VR streaming. In Proceedings of the 9th ACM Multimedia Systems Conference, Amsterdam, The Netherlands, 12–15 June 2018. [CrossRef]
3. Cao, S.; Hua, Y.; Feng, D.; Sun, Y.; Zuo, P. A high-performance distributed storage system for massive HD video data. *J. Softw.* **2017**, *28*, 1999–2009. [CrossRef]
4. Liu, Y.; Cao, X. Research on performance optimization methods for distributed storage of massive video data. *Comput. Appl. Res.* **2021**, *38*, 1734–1738. [CrossRef]
5. Luo, S.Q.; Zhu, X.R. Adaptive transmission control method for multi-stream concurrent transmission of HD video based on multi-terminal collaboration. *Telecommun. Sci.* **2015**, *31*, 42–50.
6. Sun, X.; He, Y.; Wu, D.; Huang, J.Z. Survey of Distributed Computing Frameworks for Supporting Big Data Analysis. *Big Data Min. Anal.* **2023**, *6*, 154–169. [CrossRef]
7. Zhang, S.; Zhang, Y.; Wang, Q.; Wang, Z.; Li, M. Energy consumption management strategy of cloud computing center based on M/M/c queuing model. *Comput. Meas. Control* **2020**, *28*, 193–197+237. [CrossRef]
8. Hu, N.; Cen, X.; Luan, F.; Sun, L.; Wu, C. A Novel Video Transmission Optimization Mechanism Based on Reinforcement Learning and Edge Computing. *Mob. Inf. Syst.* **2021**, *2021*, 6258200. [CrossRef]

9. Qi, Z. A novel video delivery mechanism for caching-enabled networks. *Multimed. Tools Appl.* **2020**, *79*, 25535–25549. [CrossRef]
10. Tashtarian, F.; Falanji, R.; Bentaleb, A.; Erfanian, A.; Mashhadi, P.S.; Timmerer, C.; Hellwagner, H.; Zimmermann, R. Quality Optimization of Live Streaming Services over HTTP with Reinforcement Learning. In Proceedings of the 2021 IEEE Global Communications Conference (GLOBECOM), Madrid, Spain, 7–11 December 2021; pp. 1–6. [CrossRef]
11. Park, P.K.; Moon, S.; Hong, S.; Kim, T. Experimental Study of Zero-Copy Performance for Immersive Streaming Service in Linux. In Proceedings of the 2022 13th International Conference on Information and Communication Technology Convergence (ICTC), Jeju Island, Republic of Korea, 19–21 October 2022; pp. 2284–2288. [CrossRef]
12. Ignat, C.L.; Oster, G.; Fox, O.; Shalin, V.L.; Charoy, F. How do user groups cope with delay in real-time collaborative note taking. In Proceedings of the 14th European Conference on Computer Supported Cooperative Work, Oslo, Norway, 19–23 September 2015; Springer International Publishing: Cham, Switzerland, 2015; pp. 223–242.
13. Ng, A.; Sun, C. Operational transformation for real-time synchronization of shared workspace in cloud storag. In Proceedings of the International Conference on Supporting Group Work ACM, Sanibel Island, FL, USA, 13–16 November 2016; pp. 61–70.
14. Lv, X.; He, F.; Cai, W.; Cheng, Y. A string-wise CRDT algorithm for smartand large-scale collaborative editing systems. *Adv. Eng. Inform.* **2017**, *33*, 397–409. [CrossRef]
15. Kumar, S.; Pan, H.; Wang, R.; Tseng, L. LiteDoc: Make Collaborative Editing Fast, Scalable, and Robust. In Proceedings of the 2020 IEEE International Conference on Pervasive Computing and Communications Workshops (PerCom Workshops), Austin, TX, USA, 23–27 March 2020; pp. 1–6. [CrossRef]
16. Fan, H.; Zhu, H.; Liu, Q.; Shi, Y.; Sun, C. Shared-locking for semantic conflict prevention in real-time collaborative programming. In Proceedings of the 2017 IEEE 21st International Conference on Computer Supported Cooperative Work in Design (CSCWD), Wellington, New Zealand, 26–28 April 2017; pp. 174–179. [CrossRef]
17. Bath, U.; Shekhar, S.; Egbert, J.; Schmidt, J.; Semmo, A.; Döllner, J.; Trapp, M. CERVI: Collaborative editing of raster and vector images. *Vis. Comput.* **2022**, *38*, 4057–4070. [CrossRef]
18. Lv, X.; He, F.; Cai, W.; Cheng, Y.; Wu, Y. CRDT-based conflict detection and reso-lution for massive-scale real-time collaborative CA-D systems. In Proceedings of the 12th Chinese Conference on Computer Supported Cooperative Work and Social Computing, Chongqing, China, 22–23 September 2017; pp. 185–188.
19. Guidec, F.; Mahéo, Y.; Noûs, C. Supporting conflict-free replicated data types in opportunistic networks. *Peer Peer Netw. Appl.* **2022**, *16*, 395–419. [CrossRef]
20. Rinberg, A.; Solomon, T.; Shlomo, R.; Khazma, G.; Lushi, G.; Keidar, I.; Ta-Shma, P. DSON: JSON CRDT Using Delta-Mutations for Document Stores. *Proc. VLDB Endow.* **2022**, *15*, 1053–1065. [CrossRef]
21. Guidec, F.; Launay, P.; Mahéo, Y. Causal and Delta-Causal Broadcast in Opportunistic Networks. *Future Gener. Comput. Syst.* **2021**, *118*, 142–156. [CrossRef]
22. Zhao, W.; Babi, M.; Yang, W.; Luo, X.; Zhu, Y.; Yang, J.; Luo, C.; Yang, M. Byzantine fault tolerance for collaborative editing with commutative operations. In Proceedings of the 2016 IEEE International Conference on Electro Information Technology (EIT), Grand Forks, ND, USA, 19–21 May 2016; pp. 246–251. [CrossRef]
23. Liu, H.; Chen, Q.; Liu, P. An Optimization Method of Large-Scale Video Stream Concurrent Transmission for Edge Computing. *Mathematics* **2023**, *11*, 2622. [CrossRef]
24. Tian, R.; Sun, L.; Wang, N.; Li, B. A multi-pheromone ant colony algorithm for solving the multi-unload point vehicle crating problem. *Comput. Eng.* **2015**, *41*, 156–161.
25. Yan, J.; Wang, H.; Li, X.; Yi, S.; Qin, Y. Multi-objective Disaster Backup in Inter-datacenter Using Reinforcement Learning. In *Wireless Algorithms, Systems, and Applications*; Yu, D., Dressler, F., Yu, J., Eds.; Lecture Notes in Computer Science; Springer: Cham, Switzerland, 2020; Volume 12384. [CrossRef]
26. Wu, Y.; He, F.; Zhang, D.; Li, X. Service-oriented feature-based data exchange for cloud-based design and manufacturing. *IEEE Trans. Serv. Comput.* **2018**, *11*, 341–353. [CrossRef]
27. Gao, L.-P.; Gao, D.-F. Research on consistency maintenance of real-time collaborative graphic editing in mobile cloud environment. *J. Chin. Comput. Syst.* **2018**, *39*, 173–178.
28. Gao, L.; Yu, F.; Chen, Q.; Xiong, N. Consistency maintenance of do and Undo/Redo operations in real-time collaborative bitmap editing systems. *Clust. Comput.* **2016**, *19*, 255–267. [CrossRef]
29. Gao, L.P.; Xu, X.F. Consistency maintenance of exchangeable replicated data model in large-scale real-time graphics editing. *Small Microcomput. Syst.* **2019**, *40*, 1361–1367.
30. Wei, E.; Zong, X.; Gao, L.P. Consistency maintenance of a real-time collaborative editing system for two-dimensional tables. *Small Microcomput. Syst.* **2023**, *44*, 1–10.