mProducer: Authoring Multimedia Personal Experiences on Mobile Phones

Chao-Ming Teng, Hao-hua Chu, Chon-In Wu
Department of Computer Science and Information Engineering
National Taiwan University
Taipei, Taiwan 106
{jt, hchu, r92079}@csie.ntu.edu.tw

Abstract

*This paper presents the **mProducer** tool that enables multimedia content authoring on mobile phones. It addresses three major challenges: limited storage space on mobile phones, costly and slow wireless links, and limited mobile user attention. Our contribution is twofold: (1) a novel technique called Storage Constrained Offloading (SCO) that intelligently uploads selected portions of contents to a remote storage server, so that the amount of contents a user can capture is not restricted by the size of the small mobile storage, and (2) two new UI techniques, the Key-frame based editing and automatically generated content navigation tree, to reduce the amount of user efforts for editing.*

1. Introduction

With the popularity of camera-equipped mobile phones, mobile users can easily capture and share their personal experiences as digital multimedia (video and audio) contents anytime, anywhere. To provide the convenience that a mobile user can capture, edit, and then share personal experience from the same mobile phone, we propose a mobile authoring tool, called **mProducer**, that allows a mobile user to perform multimedia content editing before sharing his/her personal experience with family and friends. This is an improvement over the existing practice of having to transfer the captured contents from a digital camera to a PC, then using a PC-based authoring tool to edit the content and share it. This practice is inconvenient because a user has to change devices between capturing, editing and sharing, and it limits the user to edit and share only when (s)he has access to a PC.

Developing a mobile authoring tool needs to consider the limitations of a mobile device and the mobile computing environment. We identify two challenges that **mProducer** needs to address:

1. The limited storage on a mobile phone restricts the size of contents a user can store. For example, the new Toshiba T08 mobile phone [10] provides a video editing tool. Given that it has only 8MB of mobile storage, it can only record about 3 minutes of video without audio. This small storage on a mobile device is hardly sufficient for a mobile user to record one complete experience. To address this problem, a mobile authoring tool should incorporate an offloading mechanism to upload captured contents to a remote storage server such that the amount of contents a user can capture is not limited by the mobile storage. One naive offloading approach is to upload every piece of content to the storage server immediately after it is captured, and then download it whenever the author needs to edit it. There are two potential problems in this naive approach. The first problem is that uploading the parts of contents that will later be cut by a user is a waste of network bandwidth. The second problem is that the low wireless link bandwidth is likely to result in slow downloading of contents, causing poor and non-interactive user editing experience. Therefore, we need a more intelligent offloading mechanism to determine when and what to offload from a mobile storage to a storage server.
2. The second challenge is the limited attention a mobile user gives to mobile applications. Under the mobile computing environment, real-world people, objects, and activities can contend for a user's attention at the same time (s)he is running the applications [13]. To address this issue, the mobile authoring tool and its editing UI need to be simple enough so that it demands less user efforts.

Our contributions are two innovative techniques that solve the two problems described above. (1) We design a novel offloading technique called *Storage Constrained Offloading (SCO)* that determines when and what contents to offload, while minimizing wireless network overhead between a mobile phone and a storage server. (2) We incorporate Key-frame based editing technique and navigation tree in editing UI in order to reduce user efforts in editing.

This paper is organized as follows. Section 2 describes the system setup for the **mProducer** tool. Section 3 discusses the SCO algorithm and its variants. Section 4 explains the editing UI of **mProducer**. Related work, and conclusion and future work are discussed in Sections 5 and 6.

2. The System Setup

We assume that a mobile user carries a mobile camera phone with limited storage space. This phone has Internet connectivity through a wireless network such as GPRS, 3G, or 802.11. A user operates this mobile phone to capture personal experience as digital multimedia data. When the mobile storage runs out of space, an offloading mechanism uploads contents to a storage server over the wireless link. The storage sever is assumed to have potentially infinite storage space. When a user finishes editing a multimedia clip and is ready to share it, this clip is also offloaded to the storage server. This allows mobile storage to reclaim space that can be used to accommodate new clips. The sharing, or distribution of personal experience contents, is provided by the storage server. In addition, we assume that a mobile phone is equipped with location sensing device (e.g., a GPS receiver), so that it is possible to determine the location where each piece of contents is captured.

There are two phases in personal experience authoring: *the capturing phase* and *the editing phase*. We assume that a mobile user would exhibit a repeating usage pattern of capturing one or more clip(s), followed by editing these clip(s), which frees up space in the mobile storage.

3. The Storage Constrained Offloading (SCO) Algorithm

An offloading algorithm makes the following two decisions: (1) when to offload, and (2) what portion of contents to offload. We design the SCO so that it can make good decisions to minimize the network communication (including both the uploading and downloading) in both phases of authoring. We describe the SCO algorithm by how it makes these two decisions.

SCO will not offload contents until the current storage space is full. The benefit is that we can avoid uploading frames that will later be cut by a user. SCO chooses frames for offloading based on an observation that there is a difference in quality requirements between *personal experience authoring tool* targeting *average consumers*, and so-called *mass media contents authoring tool* targeting *professional contents providers*. We believe that there is no need to provide a mobile personal experience authoring tool that can produce professional quality contents. In other words, fine-grained editing (e.g., frame-by-frame) used in a PC-based authoring tool for professional quality contents is in fact unsuitable for a mobile authoring tool, because they require both significant amount of user efforts and high resolution screens.

We define **Editing Granularity** as the level of details that an authoring tool allows a user to edit. Take MPEG video editing as an example, the finest granularity is frame-by-frame editing, where a user can preview and choose any arbitrary frames for cutting, adding text, etc. A coarser granularity is I-frames, where a user can preview and edit I frames only.

This observation leads to the fact that for average users, a portion of the frames can be offloaded without degrading the editing experience. For example, in MPEG video editing, if average users only require I-frame editing granularity, offloading non-I-frames does not affect user editing process and experience.

The SCO algorithm is based on a mapping between types of frames and priorities for offloading. In the above example, I-frames have higher priority than non-I-frames when it comes to offloading. In our current work, we design the SCO algorithm to prioritize frames into three levels based on their frame type:

Frame Priority	Frame Type
Level 1 (Low)	Non I and Key-frames
Level 2 (Medium)	I-frames
Level 3 (High)	Key-frames

We adopt the technique of Key-frame selection from the field of video summarization and set the Key-frames as the highest priority, because Key-frames are still images that best represents the contents of a video sequence [12]. As a result, Key-frames are never offloaded in order to guarantee a minimal Key-frame editing granularity.

The SCO algorithm is also based on a concept called **Storage Granularity**, which is about the types of frames that mobile storage can accommodate during the capturing phase:

Storage Granularity	Frames to Store
High	All frames
Medium	I and Key-frames
Low	Key-frames

Initially, a mobile storage is empty, so the SCO algorithm will store all types of frames in the mobile storage. The mobile storage is said to be at *high* storage granularity when it can accommodate all types of frames. As a mobile user captures new frames, mobile storage will eventually run out of free space at the current storage granularity. When a newly captured frame causes an overrun in mobile storage, the SCO algorithm will need to drop down a level to the *medium* storage granularity. From this point of time forward, it will store only new I/Key-frames, and offload all new non I/Key-frames to storage server. At the same time, it will also gradually offload existing non I/Key-frames to the remote storage, because they have lower priority level than what is allowed by the medium storage granularity. By offloading existing frames, it will create free spaces in the mobile storage for new I/Key-frames. Note that editing granularity cannot exceed the storage granularity. For example, to support I frame editing granularity requires I frame or above storage granularity.

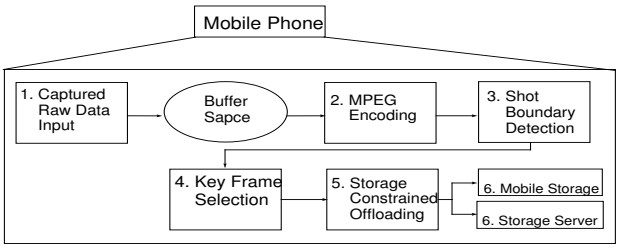

Figure 1. The Capturing Phase Processing of an MD using mProducer

3.1. Execution Flow Chart

Figure 1 shows the execution flow chart of **mProducer**, starting with data captured from a mobile camera phone, and finishing with storing the data either on a mobile storage or remote server storage. In the 1st step, camera and microphone on a mobile phone capture video and audio data, and the multimedia data is stored in a temporary buffer on a mobile phone. In the 2nd step, a MPEG encoder [2] reads these uncompressed frames from the buffer and outputs compressed frames. In the 3rd step, we apply Shot Boundary Detection (SBD) algorithm [4] on the multimedia frames to detect shots[1]. In the 4th step, we apply Key-frame Selection (KFS) algorithm [5] to identify representative Key-frame(s) in each shot. There are several well-known SBD and KFS algorithms in the literature. Since frames are compressed by MPEG encoder, we adopt the compressed domain techniques [3]. Note that **mProducer** is not about creating better SBD or KFS algorithms. It simply uses them to prioritize frames for offloading.

At the end of 4th step, we can attach the following information to each video frame: (1) whether it is a Key-frame or not, (2) its MPEG frame type (I, P, or B), and (3) its byte size. The SCO algorithm uses the information for offloading, which is in the 5th step described below in Section 3.2.

3.2. The SCO Algorithm

The SCO algorithm preserves two properties when offloading frames to remote storage. They are (1) *fairness to all clips*, and (2) *gradually offloading of frames*. If a mobile storage contains

1 A shot is defined as continuous frames from a single camera at a time. One easy detection method is to calculate the motion vector ratios for every B/P frame and use thresholds to detect the boundary [3].

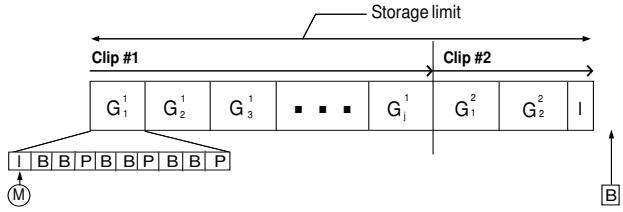

Figure 2. The storage view for illustrating SCO - Case I

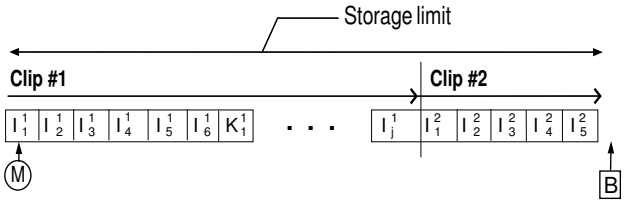

Figure 3. The storage view to illustrate SCO - Case II

multiple clips, the SCO algorithm should try to maintain fairness, meaning equal storage granularity, among all the clips currently in the mobile storage. This fairness property can ensure that **mProducer** tool can provide equal, consistent editing granularity for different clips in the mobile storage. When the SCO algorithm drops down one level of storage granularity (e.g., from high level to medium level), the offloading of frames should be done gradually and on an as needed basis, i.e., it does not offload all the non I/Key frames at once to remote storage. The reason for gradual offloading is to avoid unnecessary uploading of frames that will later be cut by users.

The example in Figure 2 illustrates how the SCO algorithm works. The mobile storage currently contains the entire clip #1 and clip #2 that is still under capturing, and it is completely full. The block G_j^i is the j-th group-of-pictures (GOP) of clip # i. Assume a new frame comes into the mobile storage, the SCO algorithm will offload all the frames except the I-frame and Key-frame (if any) of the GOP that is marked by the marker (Clip #1 in this case) to the storage server. This offloading frees up a block of space for new frames. This marker will then move to the next clip's (clip #2) foremost un-cleared GOP, where SCO will offload the non-I/Key-frames in this GOP in the next round. In order to achieve fairness among clips, the SCO algorithm offloads group of frames marked by the marker that moves in a round-robin fashion among all clips currently in mobile storage.

Suppose that all non-I/Key-frames are offloaded to a storage server. The SCO algorithm will offload I frames next. Figure 3 illustrates the state of a mobile storage at the "Medium Storage Granularity", where all the non-I/Key-frames are offloaded to a storage server to make space for I/Key-frames. I_1^1 to I_j^1 and K_1^1 to K_p^1 are I-frames and Key-frames of clip #1 respectively. Consider that a new frame is generated. If it is not an I or a Key-frame, it will be uploaded right away. If it is an I or Key-frame, the SCO algorithm drop to the "Low Storage Granularity", and it will start to offload I-frames to a storage server. Figure 3 shows an advancing marker that points to the next frame that will be offloaded next.

We design an "Offloading List" that computes the order of frames to be offloaded in the mobile storage. It sorts frames based on the frame priority first and then applying round-robin algorithm over clips. With this list, **mProducer** can simply look up the head of the list to get the frames for uploading. For example, in Figure 2, the 9 B and P frames of G_1^1 will be placed at the positions 1 to 9 on the Offloading List, and B and P frames of G_1^2 will be placed at positions 10 to 18 on the list, and so forth.

The main body of SCO algorithm is shown in Algorithm 1. We denote the free space in the storage as Z, the i-th frame of clip #j as f_j^i, its size as $S_{f_j^i}$, the newly coming frame as f_{new}^N, and N is the number of clips in the mobile storage.

Data: A new coming frame f_{new}^N *(size, type)*
Result: Frames to offload to storage server or save f_{new}^N
initialization;
if $S_{f_{new}^N} > Z$ **then**
  offload the frames in the order of the "Offloading List" until $Z > 0$;
  adjust the "Offloading List" accordingly ;
end
if f_{new}^N *is not offloaded* **then**
  save f_{new}^N and adjust the "Offloading List" ;
end

Algorithm 1: The basic SCO algorithm

In the current work, **mProducer** does not allow storage granularity to fall below the Key-frames level (i.e., the mobile storage must store all Key-frames). Therefore, there exists a limitation on the size of multimedia contents that a user can capture at any given time. The reason for this size limitation is that **mProducer** does not want to upload Key-frames to the storage server and then download them again during the editing phase. When this limit is reached, **mProducer** will inform its user to stop capturing new data and start to edit clips.

3.3. Variants of The SCO Mechanism

There are many possible variants to the SCO algorithm. We can use different priority metrics for incoming frames, which affect the ordering of frames in the offloading list. The priority metric can be based on the time of capturing (e.g., the later the higher priority), the size or fidelity of each piece of contents (e.g., the higher fidelity the higher priority), or the hierarchy of contents established by video indexing [7].

4. The Authoring User Interface

mProducer provides two UI techniques: (1) navigation tree, which is automatically generated using the context information attached to each clip (on the left of Figure 4) and (2) Key-frame based editing. Each of these two techniques is applied to simplify and reduce the amount of user efforts in each of two steps in the user editing process. The first editing step is searching for a clip to edit. As a user captures more contents, the number of clips increases, so it becomes more time consuming to scan through the long list to find the wanted clip on a small display. The navigation tree can help to speed up the search by organizing the clips into a content hierarchy based on their context information (GPS location or time). For example, if a user knows the location of a

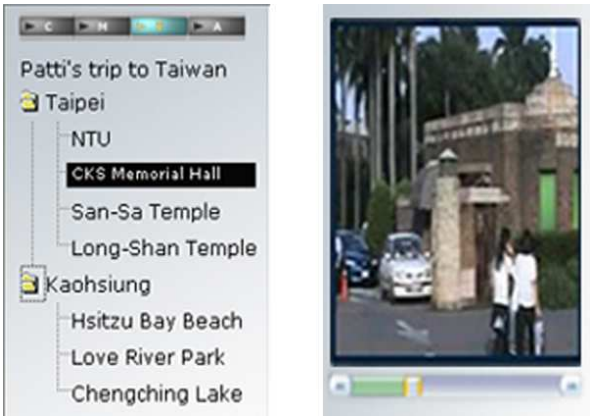

Figure 4. The integrated UI in mProducer

wanted clip, e.g., Taipei, he/she can quickly find it by scanning through only the clips under the Taipei content hierarchy.

The second editing step is previewing and editing a selected clip. Recall the observation in Section 3 that says average users do not need to create high quality contents using fine-grained, frame-by-frame editing granularity. Instead, average users are likely to prefer coarse-grained, key-frame editing with reduced user effort. As a result, **mProducer** sets the default editing granularity to Key-frames. It supports key-frame editing by presenting a slide show of key frames to a user. In general, the time duration of a slide is set to be smaller than the time duration of a shot, so playing a slide show is faster than playing a clip frame by frame. It also allows a user to quickly browse through different shots and clips. However, a user can select only key frames for editing, such as trimming, merging, embedding text or audio.

5. Related Work

Toshiba T-08 cell phone [10] is a commercial product that comes with a video editing tool. Since it does not provide any offloading algorithms, it allows users only to record around three minutes of video clips at 5 fps on its mobile storage. This is in contrast to **mProducer**, that allows users to capture almost unlimited contents.

Jokela [1] presents an overview of the key opportunities and challenges in developing tools for authoring multimedia contents for the mobile environment. However, it did not describe any solutions. Lara, *et. al.* in [8] describes a collaborative mobile authoring tool that allows different authors to download and edit different fidelity contents. They address the replica inconsistency problem when merging revisions at different fidelities on servers. However, they do not address the limited storage issue and up/downloading traffic overhead.

Hitchcock [6] is a desktop tool that uses Key-frames to speed up editing home video films. It displays full screen of Key-frames for selection, and a user can specify the length of playtime of each shot that each Key-frame represents. Since **mProducer** runs on a mobile phone with a smaller display, it cannot show too many Key-frames on the same screen. Instead, it uses slide shows to go through the Key-frames.

6. Conclusion and Future Work

In this paper, we present the **mProducer** tool for personal experiences authoring on mobile phones. We believe that **mProducer** is the first of its kind to address the problem of limited mobile storage and network overhead. Our contribution is twofold: (1) a novel technique called Storage Constrained Offloading (SCO) that intelligently uploads selected portions of contents to a storage server, so that the amount of contents a user can capture is not restricted by the size of the small mobile storage, and (2) two new UI techniques, the Key-frame based editing and automatically generated contents navigation tree, to reduce the amount of user efforts for editing.

For future work, we plan to design innovative ways of sharing these personal experiences. Current ways of sharing contents have some limitations. For example, the MMS [14] defined for cellular networks can only achieve "active sharing", which means that the contents are shared to audience in a limited scope (e.g., within one's contact list) only. Our future work will explore more interactive ways of sharing.

7. Acknowledgement

The authors would like to thank Dr. Shao-Yi Chien for the valuable discussion and support by NSC, Taiwan under grant 92-2218-E-002-036[2].

References

[1] Tero Jokela, "Authoring Tools for Mobile Multimedia Content," *Proc. of IEEE ICME*, 2003.
[2] MPEG Industry Forum, http://www.m4if.org
[3] J. Meng and S.-F. Chang "CVEPS - A Compressed Video Editing and Parsing System," *Proc. of ACM Multimedia*, 1996
[4] P. Browne, *et. al.*, "Evaluating and Combining Digital Video Shot Boundary Detection Algorithms," *Proc. of Irish Machine Vision and Image Processing Conference*, 2000
[5] H. Zhang, *et. al.*, "Video Parsing, Retrieval and Browsing: An Integrated and Content-based Solution," *Proc. of ACM Multimedia*, 1995
[6] A. Girgensohn, *et. al.*, "A Semi-automatic Approach to Home Video Editing," *Proc. of Symposium on User Interface Software and Technology (UIST)*, 2000
[7] C. Snoek and M. Worring, "Multimodal Video Indexing: A Review of the State-of-the-art," *Multimedia Tools and Applications*, 2004 (in press)
[8] E. de Lara, *et. al.*, "Collaboration and Multimedia Authoring on Mobile Devices," *Proc. of International Conference on Mobile Systems, Applications, and Services (MobiSys)*, 2003
[9] X. Gu, *et. al.*, "Adaptive Offloading Inference for Delivering Applications in Pervasive Computing Environments," *Proc. of IEEE Pervasive Computing and Communication (PerCom)*, 2003
[10] Vodaphone, Japan http://www.vodafone.jp/english/
[11] Synchronized Multimedia, W3C Interaction Domain, http://www.w3.org/AudioVideo/
[12] J. Casares, *et. al.*, "Simplifying Video Editing Using Metadata," *Proc. of Designing Interactive Systems (DIS)*, 2002
[13] P. Tarasewich, "Designing Mobile Commerce Applications," *Communications of The ACM*, Dec. 2003
[14] 3GPP TS 23.140, "Multimedia Messaging Service - Functional Description," *TSG Terminals*, stage 2, release 4, version 4.4.0, 2001-9

2 Any opinion, findings, and conclusions or recommendations expressed in this paper are those of the authors and do not necessarily reflect the views of NSC.