Log-Analysis based Characterization of Multimedia Documents for Effective Delivery of Distributed Multimedia Presentations

Maria Luisa Sapino
Università di Torino
mlsapino@di.unito.it

K. Selçuk Candan
Arizona State University
candan@asu.edu

Paola Bertolotti
Università di Torino
bertolot@di.unito.it

Abstract

In a distributed environment, media files have to be downloaded, coordinated, and presented to the clients, according to the specification given by the authors of multimedia documents. We present a server-side logO *middleware which* learns *the structure of a multimedia presentation, without accessing the author's specification, and constructs an automaton which captures the dynamic aspects of the evolution of the document, relevant for the optimization of the presentation delivery, prefetching, and scheduling.*

1 Introduction

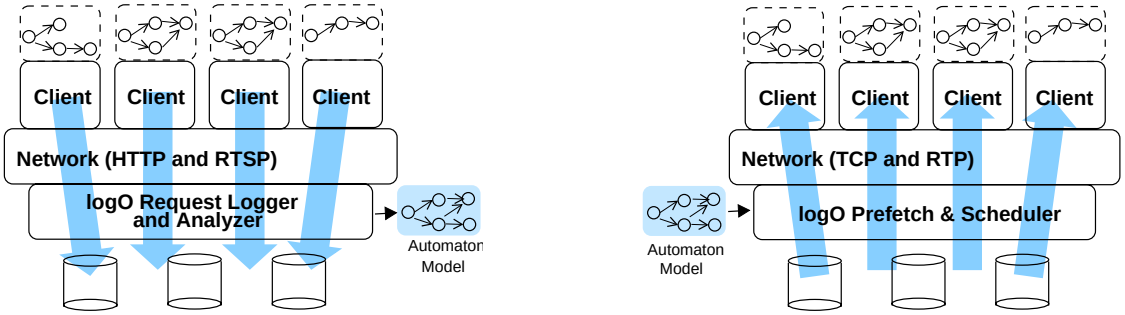
Multimedia documents are collections of media objects, synchronized through temporal and spatial constraints. In a distributed environment, where media servers and clients are dispersed, objects have to be requested, received, coordinated, and presented to the clients. Content servers are largely ignorant of the high-level presentation semantics, but can only observe media files that are requested by and delivered to the client for presentation through lower-level network protocols. These include control protocols, such as Realtime Streaming Protocol (RTSP) and Realtime Control Protocol (RTCP), and transport-level protocols, such as Realtime Transport Protocol (RTP). In particular RTSP [20], an application-level client-server protocol, allows timing and control of multimedia streams. After receiving information about the available media (such as addresses, ports, formats) through a DESCRIBE request and after setting up a session through a SETUP message, the client can request media (or parts of the media) through PLAY requests. The media are then streamed to the client using RTP.

A high-level understanding of the synchronization semantics, on the other hand, can be essential for effective delivery of presentations [**?**, 19, 24, 22, 9, 6]: improving the delivery of presentations through intelligent scheduling of data accesses to the secondary storage, prefetching and buffering of the media objects, or even adapting/replacing presentation fragments can be possible only if advance knowledge about the presentation specifications is available. Since this information is not available below the presentation level in the hierarchy, traditional work in this area either (a) assumed that intelligent scheduling is being done at the client level [6] or (b) relied on less-informed access-clustering or popularity-based prefetching (such as [16]) techniques to help improve content delivery.

In this paper, we note that both of these approaches have their shortcomings. The first approach is naturally more informed than the second one; however, it is limited to only client-side optimizations. Furthermore, such analysis can be extremely difficult in the presence of documents specified according to different reference models. Most importantly, even when presentation constraints are available, when there are alternative presentation plans available to the users, such user preferences can not be directly estimated using the client-side presentation semantics.

We, on the other hand, highlight that there is a third alternative which brings together the best of the approaches: (a) a high-level understanding of presentation synchronization characteristics, (b) capturing user preferences in terms of popularity of alternative presentation pathways, and (c) independence from specific high-level sychronization models, enabling analysis even if presentations are defined with different languages. In particular, we propose to *learn* presentation structure and the user preferences directly from its execution traces, observable as RTSP protocol requests.

Contributions. We propose a server-side middleware, *logO*, for log-analysis based characterization of multimedia documents with the ultimate goal of improving the delivery of distributed multimedia presentations (Section 2). Note that, unlike traditional log-analysis approaches used in various application domains, such as web-page delivery, where the goal is to learn the correlation between object requests, the *logO* middleware also has to learn the temporal synchronization characteristics underlying the multimedia presentations. For this purpose *logO* relies on RTSP request logs which contain information about the status of the media objects and the time instants in which changes take place. Then, by using the trace available in the logs, *logO* constructs an automaton capturing the dynamic evolution of a document, relevant for effective delivery (Section 3).



(a) Request event logging (b) Model-based informed delivery

Figure 1. Overview of the *logO* middleware: (a) the request logger and analyzer listens to the RTSP requests that servers are receiving and integrates these requests into a state automaton and collects appropriate statistics, (b) this automata and the associates statistics are then used for giving intelligent prefetching and scheduling decisions for effective delivery of presentations

A particular challenge *logO* faces is that different executions of the same presentation can vary in their evolution traces. Thus, *logO* first approximates traces by collapsing to the same time instant events possibly occurring at very close, but distinct, times. To identify quite similar, but slightly different executions of the same presentation, *logO* relies on appropriate temporal similarity metrics. Whenever possible, *logO* merges the paths that are the same up to a certain state and introduces non-linear alternatives whenever different evolutions are identified.

2 Overview of the *logO* Middleware

logO is a server-side middleware. In order to learn the presentation structure, it listens to the RTSP requests that servers are receiving and integrates these requests into an integrated state automaton (Figure 1(a)). As it constructs the automaton, *logO* collects appropriate statistics regarding clients' *preferred* access patterns over this structure, to enable effective prefetching and scheduling decisions for improved delivery of presentations (Figure 1(b)).

Presentations can contain static (such as images and text) or streaming objects, which need to be presented smoothly to the user. The objects can be fetched using different transport protocols: static objects can be requested using HTTP and delivered using TCP. Since HTTP requests are not referenced against a timeline, streaming objects (or objects that need to be synchronized against a timeline), on the other hand, are generally requested using RTSP and delivered using RTP. A sample RTSP request is presented below:

```
PLAY rtsp://aria.asu.edu/audioobj RTSP/1.0
     CSeq: 715
     Session: 12567
     Range: npt=10-25;time=20060324T164800Z
```

Here, *CSeq* is a counter incremented for each distinct request, *Session* is the session ID for this request, and *Range* states that the server will play seconds 10 through 25 of the media. The *Time* entry specifies when the server should start the playback (in this case at 16:48 on 03/24/2006). Therefore, by listening to the requests, *logO* can observe the timing characteristics of the multimedia presentations.

The *logO* middleware needs to accommodate for the facts that (a) some media objects in the presentation may be requested by the client through other sources that are not being logged by *logO* and (b) request traces can vary due to user interactions or dynamicity of the system states. The first issue is naturally handled by *logO* in that the automaton created using available logs will reflect the portion of the media presentation that need to be delivered from relevant servers. The second issue, on the other hand, requires an automata merging mechanism which can handle variations in request traces (Section 3.2).

3 Learning Presentation Structures

Without getting into the details of the HTTP and RTSP request formats, let us consider a simplified log format, where each entry is a record of the form $\langle t, m, ev \rangle$. Here, t is a (middleware) time instant, m is a media objects, and $ev \in \{st, end\}$ is an event[1] that has occurred, possibly changing the state of the media m. As an example, consider the following event trace:

Example 3.1 (Event Trace)

```
15:11:03:863 intro st     | 15:29:07:019 descr end
15:11:03:891 logos st     | 15:29:07:052 image1 st
15:14:05:129 intro end    | 15:29:07:059 image2 st
15:14:05:133 logos end    | 15:35:07:021 image2 end
15:14:05:150 artist st    | 15:35:07:128 image1 end
15:19:06:997 artist end   |  15:35:07:400 concl st
15:19:07:016 descr st     | 15:40:08:638 concl end
```

- The record `hh:mm:ss:ms m st` means that media m has started at time instant `hh:mm:ss:ms`.
- The record `hh:mm:ss:ms m end` means that presentation of the media m has ended at time instant `hh:mm:ss:ms`.

Events are *simultaneous* if they occur at the same time instant. Unfortunately, the sequential nature of the request

[1]For simplicity of discussion, in this paper, we ignore complex events, such as the distinction in RTSP between *termination* vs. pause.

processor can associate distinct request time instants to events, even when they need to occur at the same time instance according to the synchronization specifications. Thus, the event traces are generally interpreted at a lower granularity (a system parameter) than the millisecond level log entries and *logO* treats as simultaneous all events whose recorded times are within the granularity threshold.

In the above trace example, the time instants refer to the middleware's internal clock. Naturally, to extract a uniform, normalized representation of the presentation dynamics, *logO* associate times 0 to the starting instant of any presentation, and shifts all the time instants accordingly.

3.1 Trace Finite State Automata

To represent the information obtained from the trace, *logO* introduces the notion of *state of the presentation at any point in time*. A presentation's state represents the set of media that are active (i.e., being delivered) at a given time instant. Thus, state changes occur when at least one active media ends or one new media starts; i.e., they occur as a consequence of at least one media event in the trace.

Given an event trace for a presentation P, the states and state transitions form a trace automaton. To associate events to their media we use the functional notation: $ev(m)$ represents the fact that the event ev occurs on the media m.

Definition 3.1 (Trace Automaton) *Let log_P be the* normalized *log file tracing the events of a multimedia presentation. Its associated trace automaton is the* 5–tuple $AUT(log_P) = \langle S,\ s_0,\ S_f,\ \mathcal{TR},\ next\rangle$. *In the automaton* **(i)** *S is the set of possible states of the presentation - each state is a pair $\langle id, \mathcal{AM}\rangle$, where id is a globally unique identifier and $\mathcal{AM}$ is a set of active media;* **(ii)** *s_0 is the initial state, $s_0 = \langle id_0, \emptyset\rangle$;* **(iii)** *The set of final states is the singleton $S_f = \{s_f = \langle id_f, \emptyset\rangle\}$;* **(iv)** *$\mathcal{TR}$ is the the set of symbols that label possible transitions. $\mathcal{TR} = \{\langle ev(m),\ inst\rangle \mid$ the record $\langle inst, m, ev\rangle \in log_P\}$[2]. **(v)** $next : S \times \mathcal{TR} \rightarrow S$ is the transition function. Given a state $s = \langle id, \mathcal{AM}\rangle$, a set of events ev, and a time instant $inst$, $s' = next(s, \langle\{ev\}, inst\rangle) = \langle id', \mathcal{AM} \setminus \{m \in \mathcal{MI} \mid end(m) \in \{ev\}\} \cup \{m \in \mathcal{MI} \mid st(m) \in \{ev\}\}\rangle$;*

Intuitively, the new state s' reflects the items which have terminated or have been stopped at time instant $inst$ (and are deleted from the set of active media, accordingly) and the items which are starting at the same time (and thus, are inserted in the set of active media).

The states of the trace automaton are implicitly associated with a timeline: each state is entered at a specific time instant associated with the events that cause the transition from the previous state to it and is exited when the next set of events (if more than one, simultaneously) occurs.

[2]We denote *simultaneous* events with a unique compact transition $\langle\{e_0(m_0),\ldots, e_n(m_n)\}, inst\rangle$, where $\{e_0(m_0),\ldots, e_n(m_n)\}$ is the set of all the simultaneous events occurring at the same time instant $inst$.

Example 3.2 *The automaton for the trace in Example 3.1 has the following media sets and durations:*

$\mathcal{AM}_0 = \emptyset$	$\mathcal{AM}_1 = \{intro, logos\}$ d_1 = 182
$\mathcal{AM}_2 = \{artist\}$ d_2 = 31	$\mathcal{AM}_3 = \{descr\}$ d_3 = 600
$\mathcal{AM}_4 = \{img1, img2\}$ d_4 = 360	$\mathcal{AM}_5 = \{concl\}$ d_5 = 300
$\mathcal{AM}_6 = \emptyset$	

In a sense, the trace automaton describes the evolution of the delivery as consequence of events in the trace.

Theorem 3.1 *The automaton associated to a single (normalized) trace log_P recognizes exactly one word,*

$$\langle e_0(m_0), inst_0\rangle \ldots \langle e_n(m_n), inst_n\rangle,$$

which is a compact representation of the event trace log_P. Compactness comes from associating to a single transition, the multiple events at the same normalized time instant.

3.2 Merging Trace Automata

The trace automaton defined above is a chain of states which reflects the information extracted from the trace of a **single** presentation. Thus it recognizes a single word, which is exactly the sequence of records appearing in the given trace. Effective prediction for multimedia delivery, on the other hand, requires knowledge from multiple execution instances. Therefore, for proper modeling of the overall structure of a multimedia document and for capturing the *popularity* of alternative execution plans, we need to be able to merge related trace automata. In general, *logO* relies on two alternative schemes for merging. *History-independent merging:* In this scheme, each state in the original automata is considered independently of its history. Thus, to implement history independent merging, an equivalence relation ($\equiv_{log}$), which compares the active media content of two given states, s_i and s_j, is sufficient for deciding which states are compatible for being merged. The merge algorithm produces a new automaton in which the media items in the states are (representatives of) the equivalence classes defined by the $\equiv_{log}$ relation. The label of the edge connecting any two states s_i and s_j includes (i) the event that induced the state change from a state equivalent to s_i to a state equivalent to s_j in any of the merged automata, (ii) the duration associated to the source state, and (iii) the number of transitions, in the automata being merged, to which (i) and (ii) apply. The resulting automaton may contain cycles. Note that the transition label includes the counting of the number of logged instances where a particular transition occurred in the traces. The count labels on the transitions provide information regarding the likelihood of each transition. In a sense, the resulting trace automaton is a *timed* Markov chain, where the transitions from states have not expected trigger times, but also associated probabilities. Therefore, given the current state, the next state transition is identified probabilistically (as in Markov chains) and the corresponding state transition is performed at the time associated with the chosen state transition.

History-dependent merging: In this scheme, two states are considered identical only if their active media content **as well as** their histories (i.e., past states in the chains) are matching. The equivalence relation, $\equiv_{log}$, compares not only the active media content of the given states, s_i and s_j but also requires their histories, $hist_i$ and $hist_j$, to be considered identical for merging purposes. In particular, to compare two histories, *logO* uses an edit-distance function. Possible temporal similarity functions have been described in [25]. Unlike history independent merging, the resulting trie-like merged automaton will not contain any cycles and the same set of active media can be represented as different states, if the set is reached through differing even histories.

4 Use of Merged Automaton for Prefetching

Supposing that the playout of the interactive multimedia presentation has reached to a state s in this probabilistic automaton; to determine which objects may be needed in the future, *logO* must look ahead in the automaton and estimate what the probabilities of possible execution paths are and which objects are contained in these paths (and when). Reachability and state likelihood analysis for caching and prefetching is commonly done for Markov chains [12]. In trace-automata used in *logO*, however, one has to consider not only the state transition probabilities, but also (a) the *temporal* aspects of the transitions as well as (b) the fact that multiple states can contain the same object. Therefore, *logO* cannot don traditional Markov chain analysis techniques.

Once a trace-automaton is identified, *logO* uses the probabilistic prefetching framework we introduced in [7]. In particular, it computes concept of j-bounded object appearance probability, where if we "look ahead" j state transitions in the automaton then the probability that an object o will be needed in those j levels is the sum of the probabilities of all paths containing object o. Based on this, *path prefetching* and *object prefetching* strategies are developed: in the former case, the middleware fetches as many objects along the most probable paths in the trace trie as will fit into the prefetch buffer. In the later, *logO* prefetches those objects that will most probably be required (within the limitations imposed by the size of the prefetch buffer). The experiment results showed that object-based prefetching (though is more expensive to compute) can provide better hit ratios.

5 Related Work

Temporal models include instant-based [11, 5] and interval-based [2, 13] models. Timed Petri nets are used to describe interval-based compositions [14, 8]. Automata based approaches to temporal modelling are presented in [4, 3]. [10, 19, 24, 22, 9, 6, 15] show that knowledge about the temporal characteristics of presentations can lead to significantly better request and delivery schedules. Yet, since they assume direct knowledge of the presentation characteristics, these schemes are not directly applicable in *logO*. Also most schemes for prefetching and buffering of audio/video streams [17] rely on the assumption that clients will follow a linear sequence. Existing analysis-based schemes (for web pages, documents, or database objects) consider object reference and access patterns [16, 1, 12, 18]. But, they do not capture the underlying temporal structures.

6 Conclusions

We presented a server-side middleware, *logO*, for log-analysis based characterization of multimedia documents. *logO* constructs a probabilistic finite state automaton, capturing the dynamic evolution of the presentation.

References

[1] J.-H. Ahn and H.-J. Kim. *SEOF: An Adaptable Object Prefetch Policy for Objectoriented Database Systems*. ICDE 1997.

[2] J.F. Allen. *Maintaining Knowledge about Temporal Intervals*, CACM, vol. 26, no. 11, pp. 832-843, November 1983.

[3] P. Bertolotti, O. Gaggi, M.L. Sapino *Dynamic Context Adaptation in Multimedia Documents*, ACM SAC06

[4] P. Bertolotti and O. Gaggi. *A Study on Multimedia Documents Behaviour: a Notion of Equivalence*, accepted to MTAP.

[5] K.S. Candan, *et al. A Framework for Supporting Multimedia Document Authoring and Presentation*, ACM Multimedia 1996.

[6] K.S.Candan *et al.Retrieval Schedules Based on Resource Availability and Flexible Presentation Specifications.* Multimedia Sys. 1998.

[7] K.S. Candan, E. Lemar, V.S. Subrahmanian.*View Management in Multimedia Databases.* VLDB Journal 9(2): 131-153, 2000.

[8] S.M.Chung and A.L.Pereira *Time Petri Net Representation of SMIL* IEEE Multimedia, pp.64-72, 2005.

[9] M.L. Escobar-Molano *et al. Retrieval Scheduling for Multimedia Presentations*, MIS01.

[10] J.W.Ham. *A Pre-Scheduling Mechanism for Multimedia Presentation Synchronization*. ICMCS'97.

[11] M.Y. Kim and J. Song. *Multimedia Documents with Elastic Time*, ACM Multimedia Conference '95, 1995.

[12] A.Kraiss and H.Weikum.*Integrated Document Caching and Prefetching in Storage Hierarchies Based on Markov-Chain Predictions.* VLDB Journal 1998.

[13] L. Li *et al. Multimedia Teleorchestra With Independent Sources:Part 1-2*, ACM Multimedia Systems, 1(4), 1994.

[14] T.D.C. Little and A. Ghafoor. *Synchronization and Storage Models for Multimedia Objects*, IEEE JSAC, 8(3), April 1990.

[15] C.Lohr and J.Courtiat. *From the Specification to the Scheduling of Time-Dependent Systems*. LNCS vol. 2469. 2002.

[16] A. Nanopoulos *et al. Exploiting Web Log Mining for Web Cache Enhancement*. LNCS vol. 2356. 2002.

[17] R.Ng and J.Yang. *An Analysis of Buffer Sharing and Prefetching Techniques for Multimedia Systems*, Multimedia Systems, 1996.

[18] R.H.Patterson *et al. Informed Prefetching and Caching.* ACM Symposium on Operating Systems Principles, 1995.

[19] B. Prabhakaran and S.V. Raghavan. *Synchronization Models for Multimedia Presentation With User Participation*, ACM Multimedia Systems, vol.2, no. 2, August 1994, pp. 53-62.

[20] RTSP, `http://www.rtsp.org/`

[21] SMIL, `http://www.w3.org/TR/REC-smil/`

[22] Y. Song *et al. NetMedia: Synchronized Streaming of Multimedia Presentations in Distributed Environments*, ICMCS'99.

[23] M. Vazirgiannis and S. Boll. *Events in Interactive Multimedia Applications: Modelling and Implementation Design*, ICME 1997.

[24] S. Wirag, *Scheduling of Adaptive Multimedia Documents*, ICMCS'99, pp. 307-311, June 1999.

[25] Prakash Yamuna and K.S. Candan. *Similarity-based Retrieval of Temporal Documents in Digital Libraries*, accepted to MTAP.