Semantic based Prefetching in News-on-Demand Video Servers

Ahmed Mostefaoui¹, Harald Kosch², Lionel Brunie¹

¹ Information Systems Engineering Laboratory, National Institute of Applied Sciences Lyon, France ; email: (Lionel.Brunie, Ahmed.Mostefaoui)@insa-lyon.fr
² Institute of Information Technology, University Klagenfurt, Austria ; email: harald.kosch@itec.uni-klu.ac.at

April 28, 2000

Abstract. In video databases, a video document has two abstractions. The high level abstraction corresponds to the view in which the contents of that video document are seen by end users, and the low level abstraction corresponds to the physical organization of that video document. Due to the huge size of continuous data, reducing I/O has become a key issue. The latter has been mostly addressed by developing appropriate buffering techniques. In addition, prefetching techniques play a major role to meet the video data requirements. In this paper, we propose a novel prefetching strategy based not only on run-time information (objects access frequencies for example) but also on knowledge about clips structures. The proposed technique merges the two views of a video document to trigger prefetching at the video server level. Simulation experiments for a News-on-Demand application performed on different request scenarios show an improvement of about 18% in the buffer hit-rate with respect, first to the available buffer size and second to the request arrival rate.

Keywords: News-on-Demand, Buffering Strategy in Video Servers, Video Databases

1. Introduction

Technological advances over the past years have made it possible to construct large digital video libraries comprising tens of terabytes of on-line or nearly on-line data (Flynn and Tetzlaff, 1998). As a result of the continued proliferation of video data, these libraries will grow significantly larger over the next few years. This gives rise to the need to efficiently query these archives. Thus in many video applications (TV archives, News-On-Demand, etc.), a video document has two abstractions: the High Level View (HLV) and the Low Level View (LLV) (Jiang and Elmagarmid, 1998). The first one corresponds to the end-users' perception, as the second one concerns the physical data organization. Usually, the HLV is a result of a video indexing

† Correspondance to: Harald Kosch

* This work is partially supported by France Télécom (through CNET/CCETT), research contract Nr. 96 ME 17 related to the SESAME Project.

process which segments a video document into clips\textsuperscript{1} in order to allow content-based retrieval. The mostly used approach is the so-called Stratification technique (Auguiere-Smith and Pinecver, 1991). In this technique, a video document is "cut" according to its contextual information into a set of overlapping strata. The overlap captures the shared contextual information between the streams which are textually described (annotated). The so-created annotations serve as a base for the content-based retrieval of streams. Figure 1 shows an example of an annotated scene that is composed of five overlapping strata.

In cooperation with the French institute, INA\textsuperscript{2} and the TV broadcasting corporation, FRANCE3, we are currently developing a video archive system where users (mainly reporters) can have on-line access to video archives. In this paper we present the novel prefetching approach we adapted in our video server to anticipate the load of future requested streams in the video server buffers.

1.1. Motivation

Typical Video-on-Demand (VoD) systems involve two actors, the video file server and the requesting client. Means for sophisticated search functionality is not really provided in such systems. This is due to the relatively simple structures of the stored movies (title, actors, etc.). On the contrary, many video applications (digital libraries, TV archives, etc.) need more elaborated systems with search functionalities. This calls for the addition of an annotation database. This database contains the descriptive information, i.e. high-level information of the stored video documents (Weiss et al., 1994; Jiang and Elmagarmid, 1998) and provides special query interfaces to the client. The annotation database can be integrated with the video server as is the case with IBM DB2 (Chamberlain, 1998) or be completely separated from it (Brubeck and Rowe, 1996). Our work is focused on the second category.

A typical request scenario (see figure 1) involves two steps. In the first step the user submits its query to the high-level view, i.e. to the annotation database. Actual implementations (Hjelsvold and Midtstraum, 1996) allow filter expression, or extended SQL-queries (Li et al., 1997). The result of the query usually returns a set of stream identifiers which are used in the second step to view the streams from the video server. Depending on the granularity of the annotating process, typically annotated streams have small lengths (less than 5 min-

\footnote{1 The terms clips, strata and streams are used interchangeably in the reminder of the paper.}

\footnote{2 INA (Institut National de l’Audiovisuel) is the French institute responsible for archiving all broadcasted programs.}
utes). Furthermore the annotated streams are almost contextually correlated (Hampapur, 1999). This correlation is accentuated in the case of the application studied i.e., TV archive management. Indeed, the latter presents the following features:

- **User access behavior**: users ask typically for streams related to a specific topic which is governed by actuality events (sports, politics, etc.). Hence, the contents of the streams of interest are very close and may overlap. Consider the example of figure 1. Let us assume a reporter is interested in viewing the stream about tracking. Afterwards, there is a high probability that she/he (or another reporter) will issue a request to view the thieves arrest stream. By anticipating the load of the unshared data between the two streams, the server makes the thieves arrest stream available in the buffer as soon as the tracking stream has been accessed.

- **User interaction**: the size of television archives grows steadily, thus the query may return an important number of streams. Fetching all streams at the normal rate is a time consuming process. That is why reporters, when viewing streams from the archive, make a frequent use of the VCR-like operations (mainly Fast-Forward and Fast-Rewind). This calls for caching video streams at the video server and for prefetching parts of videos closely related to the accessed one in order to serve subsequent requests.

At the lowest level, contextual correlation of streams is implemented as an overlap of video frames, i.e. an overlap relation captures the contextual information shared between these streams. In this context, our objective is to demonstrate that by considering the degree of overlapping between contextually correlated streams, a simple prefetching algorithm can be designed which anticipates the load of future accessed streams and therefore enhances the performance of the video server. We will show by simulation experiments that on average, compared to a non-prefetching strategy, a 18% higher buffer hit rate can be achieved without additional costs. Indeed, the number of delayed messages remains equal to 0 if the inter-arrival rate of the request remains above 2 seconds.

---

3 For instance, INA archives every year nearly 6 terabytes of compressed videos.
4 Video Cassette Recorder.
1.2. Related Work

Prefetching in multimedia servers has often been addressed in the literature within the general problem of buffer management. Many excellent research studies have focused on that problem (Rotem and Zhao, 1995; Kanath et al., 1995; Oezden et al., 1997; Oezden et al., 1996; Jadav et al., 1996; Ng and Yang, 1996; Ng and Yang, 1994). With respect to the topic area of this paper, these studies can be grouped into the following categories: (a) strategies promoting buffer sharing between clients by using caching techniques (Dan and Sitaram, 1997; Makaroff and Ng, 1995). In these approaches, the streams are cached in the server buffer in order to be used by subsequent requests; (b) strategies based on batching approach (Dan et al., 1996). In this method, requests are delayed until they can be merged with other requests for the same video; (c) strategies focusing on client interaction (VCR-like operation) to perform buffer replacement (Moser et al., 1995). In this technique, the buffer replacement policy is driven by the VCR buttons (Fast-Forward ...).

The majority of the proposed approaches were designed to handle Video-On-Demand requirements. Consequently, the prefetching was almost based on run-time information (prefetching a movie portion assuming that the client does not use the VCR buttons to alter the flow of the display). This was motivated by the fact that movies have long lengths and simple structures (title, duration, etc.). On the opposite, in News-On-Demand applications, data have more complex structures and their lengths are relatively short (typically less than 5 minutes (Hjelsvold and Midtstraum, 1996)). We argue that these features could help to develop more powerful prefetching techniques which use, in addition to run time information, semantic information about the structure of the requested streams. To the best of our knowledge, none of the previous works have addressed the potential benefits of prefetching in the context of short overlapping videos for the Stratification technique. In this context, our approach considers additional semantics from the annotation database i.e., mainly the overlapping degree of streams. Furthermore, we study the effectiveness of the proposed technique through extensive simulation. The results, as presented in section 4, show the effectiveness of the proposed technique under different workloads and different system configurations.

The remainder of this paper is organized as follows, section 2 briefly presents the adopted video server model. The prefetching strategy is formalized in section 3 and the implementation details of the proposed technique are discussed. Section 4 reports on its performance evaluation.
results. Section 5 concludes the paper and highlights future research directions.

Figure 1. Video Archive System with an example of an annotated scene.

2. Video Server Model

Figure 1 shows the overall model of the video server (Mostefaoui, 1997a; Breidler et al., 2000a). The server is composed of two main components: the Global Object Manager (GOM) and the Local Object Managers (LOMs), one for each node in the server. The architecture of the proposed model is based on a shared-nothing model, where each of the nodes in the system has private memory (buffer) and peripheral storage. The GOM is responsible for scheduling all incoming requests for streams onto storage components. Based on the available resources (network bandwidth, buffer space, etc.), it “decides” whether a request can be accepted or rejected (Mostefaoui, 1997b; Breidler et al., 2000b). Once a request has been accepted, the GOM delegates the responsibility of serving to the chosen LOM. The LOMs are responsible for serving interactively streams to clients (VCR operations). Streams are internally striped across the nodes in a round-robin fashion in order to cope with limited disk bandwidth problems (Ghandeharizadeh and Ramos, 1993).
The unit of striping is called a "Continuous Object" (CO). During an access request, COs have to be loaded into buffer before they can be transmitted to client modules\(^5\).

3. Prefetching Strategy

Prefetching techniques radically improve the performance of a client-server system by reducing the response time of queries. In order to explain the general idea of our prefetching approach for the Stratification technique, three points have to be handled: first determine which data has to be prefetched, second determine the moment for triggering a prefetching operation, and third locating the node for storing prefetched data.

The first point is related to the exploitation of the overlap relationship between streams. Here the criterion is based on the overlap relationship, i.e., the unshared parts of streams overlapping with previously preserved streams in the buffer are prefetched. However, a stream can overlap with more than one stream. The criterion used to select which streams are to be considered for prefetching is discussed in the next paragraphs. The second point leads to several possible solutions depending on the system resources. Obviously the more available resources (buffer space, network bandwidth, etc) the system has, the more frequently prefetching is triggered. Nevertheless, the system resources are limited and shared between activated queries and prefetching requests. Therefore, it is important to ensure a favorable balance between them. According to the idea of the proposed prefetching strategy to include information on the semantic relation (through the overlapping relationship) between the streams, the prefetching could only be triggered by considering the already loaded streams. Thus, we must find out which stream will be considered among all the available ones in the buffer or the just submitted ones. With regard to the user access behavior exposed above, we activated a prefetching by a new query admission for the following reason: as this stream being accessed currently, it is a subject of interest at that moment and consequently it is the most favorable candidate to trigger semantic prefetching. This choice proved to be efficient (see section 4).

The third point concerns the location of the prefetched data in the server buffers with regard to the server model. Since the prefetching is triggered by a new query admission, it is natural that it happens in the same buffer (node) as the query. This prefetching is called local

\(^5\) Examples for such modules are presentation or synchronization managers.
prefetching and is implemented in our system. Nevertheless, available buffer space is always unbalanced between the nodes. For example, it could happen that a node, which holds a stream, does not have enough buffer space to hold the prefetched data triggered by this stream. In such a case, it could be beneficial to locate differently prefetched data and queried data. This prefetching, called Global prefetching is not in the scope of this paper.

3.1. Analysis and Preliminary Definitions

At the video server level a video document consists logically of a set of streams and physically of a collection of Continuous Objects (COs). All the COs of a video document are time-indexed from 0 to \( n-I \) with \( n \) denoting the total number of COs in that video document. Assume a video document is composed of \( k \) streams and \( S \) denotes the set of these streams \( S = \{ S_0, \ldots, S_{k-1} \} \). Each \( S_i \) is associated with an interval \([CO_s, CO_f]\) where \( s < f \). To simplify the notation, we use \( S_i=[s_i, f_i] \) meaning that stream \( S_i \) is associated with an interval \([CO_s, CO_f]\). We define the overlap relationship between the streams of the same video document as follows:

**Definition 1.** We define the overlap relation \( O_v \) over \( S = \{ S_0, \ldots, S_{k-1} \} \) in the following way: two streams \( S_i, S_j \in S \) are said to overlap and we note \( S_i \sim O_v S_j \), iff one of the following hold:

\[ - \quad s_i \leq s_j \leq f_i \text{ or,} \]
\[ - \quad s_i \leq f_j \leq f_i . \]

If the two conditions hold, the overlap is total i.e., the stream \( S_i \) contains the stream \( S_j \). Note, that the overlap relation \( O_v \) is reflexive and symmetric. But it is not transitive, consider for instance the streams \( S_i=[10,20], S_j=[15,30] \) and \( S_k=[25,40] \), then \( S_i \sim O_v S_j \) and \( S_j \sim O_v S_k \) holds, but not \( S_i \sim O_v S_k \).

The overlap relation, as a result of the annotation process, is related to the contextual information shared between two streams. Obviously, from a high level viewpoint, the higher the overlap is, the closer the contents of the two overlapped streams are to each other. However, from a low level viewpoint, the overlap relation, as defined above, could lead to an inefficient prefetching. The reason is that it does not give any indication about the proportion of the data to be prefetched with regard to the shared data. For instance, in figure 1, the stream Alarm overlaps with the two streams Police intervention and Tracking. Despite the fact that the overlapped data size between the two streams, Alarm and Police intervention, is greater than the size of the overlapped data.
between the two streams, *Alarm* and *Tracking*, the size of the data needed to prefetch the stream *Police intervention* is greater than the size of data needed to prefetch the stream *Tracking*. For this reason, we introduced the *Degree of Overlap* between two streams.

With regard to the stream under consideration, $S_i$, we first introduce the *Overlapping Set* ($OS_{S_i}$) containing all sets $S_j \in S$ which overlaps with $S_i$, i.e. for which $S_i \cap S_j$ holds. Afterwards we formally define the *degree of overlap* of $S_i$ with each member of its related $OS_{S_i}$.

**Definition 2.** We define for one stream $S_i \in S = \{S_0, \ldots, S_k\}$ the set of streams $OS_{S_i}$ (overlapping set) which overlaps with $S_i$ by:

$$OS_{S_i} = \{S_j \mid S_i \cap S_j, 0 \leq j \leq k - 1\}.$$ 

**Definition 3.** Consider the overlapping set $OS_{S_i}$ for one stream $S_i \in S$. For each member $S_j$ of $OS_{S_i}$, the *degree of overlap* of $S_j$ with regard to $S_i$, noted $DoO(S_i, S_j)$, is defined as:

$$DoO(S_i, S_j) = \frac{|\text{overlap}(S_i, S_j)|}{|S_j|},$$

where $|\text{overlap}(S_i, S_j)|$ denotes the number of COs “shared” between the two streams $S_i$ and $S_j$ and $|S_j|$ denotes the number of COs of the stream $S_j$. If the $DoO$ is equal to 1, the stream $S_i$ contains the stream $S_j$.

Now we are able to define the set of COs not shared between $S_i$ and a member of its overlapping set. We will refer to it as *stream to prefetch*.

**Definition 4.** Consider the overlapping set $OS_{S_i}$ for one stream $S_i \in S$. For each member $S_j$ of $OS_{S_i}$, the *Stream to Prefetch* of $S_j$ with regard to $S_i$, noted $SP(S_i, S_j)$, is:

$$SP(S_i, S_j) = \{CO_p \mid CO_p \in S_j \text{ and } CO_p \notin S_i, 0 \leq p \leq n - 1\},$$

where $n$ denotes the total number of COs in this video document (see first paragraph of this section).

The *Stream to Prefetch* $SP(S_i, S_j)$ represents the set of COs of $S_j$ not shared with $S_i$. This means that if the stream $S_i$ is available in the buffer, the system has to prefetch $SP(S_i, S_j)$ to make the stream $S_j$ available in the buffer too.

Next we enlarge the notion of overlapping sets $OS_{S_i}$ by information on the overlapping degree and the stream to prefetch. We refer to it as *set of streams to prefetch* $SOSP$.

**Definition 5.** For every stream $S_i \in S$ we define the *set of streams to prefetch*, noted $SOSP_{S_i}$, as a set of triples:
\( SOSP_{S_i} = \{ < S_j, DoO(S_i, S_j), SP(S_i, S_j) > | S_i \cap S_j, 0 \leq j \leq k - 1 \} \).

Now we are able to formulate a criterion to define what member of the overlapping set \( OS_{S_i} \) of the actual scheduled stream \( S_i \) has to be prefetched. This criterion is based on the degree of overlap (DoO) component of \( SOSP_{S_i} \). On one side, the DoO represents the proportion of the contextual information shared between streams in the high level view. On the other side, from a low level viewpoint, the DoO allows to sort streams according to the size of the data to be prefetched.

### 3.2. Prefetching Algorithm

Before describing the prefetching algorithm, the server functionality has to be outlined. When a new query is submitted to the server, the latter uses a deterministic admission control policy: if there are enough resources (buffer space, disk bandwidth and network bandwidth), the query is accepted, otherwise it is rejected and queued. The node that has the lowest workload among all nodes is chosen to handle the query. Based on the available buffer space at the chosen node, a query can be either cached (i.e. fully stored in the memory) or pipelined (continuous flow from disks). When resources are released after the end of an interaction\(^6\), the server checks the waiting queue. If there is no waiting queries, the server waits for new incoming requests. As noted above, the prefetching is triggered only after the admission of a new cached query. The prefetched COs are loaded in the same node buffer as the accepted query. The admission controller makes a distinction between queries and prefetching requests because they do not have the same requirements. Indeed, prefetching requests do not have strict deadlines and the system therefore does not have to ensure a continuous service but “does its best”.

Now, assume that the stream \( S_i \) has been accepted and cached in the server. Assume also that the \( SOSP_{S_i} \) has been constructed from the annotation database and sorted on the DoO basis in a descending order. Now we should select the candidate members of the overlapping set \( OS_{S_i} \) of \( S_i \), by the information stored in \( SOSP_{S_i} \). In order to control the number and the quality of the candidates for prefetching, we first introduce a threshold on the overlapping value DoO, called \( DoO \_limit \), beyond which the prefetching is possible. Second we define an upper bound \( Upper(S_i) \) on the number of candidates for prefetching. Figure 2 presents the proposed prefetching algorithm. Note, that we

---

\(^6\) The interaction time (i.e. the time a user displays the stream) is not necessarily equal to the time length of the stream. In most cases, the interaction is longer than the length of the stream.
use a *Foreach* to iterate through the set. In each iteration, the triple
<s, overlap, sp> holds the actual value s of the overlapping stream,
the degree of overlapping overlap and the stream to prefetch sp. Note
that for the experiments we fixed the value of Upper(Si) to 1.

**Begin**

Let Si be the actual accepted stream.
Construct the SOSP<sub>i</sub> from the annotation database.
Nb = 0 /* count the number of prefetched streams */
**Foreach** <s, overlap, sp> ∈ SOSP<sub>i</sub> Do
  If overlap = 1 Do skip (<s, overlap, sp>) Endif
  If overlap ≥ DoO limit Do
    Check buffer space availability for sp.
    If enough buffer space available
      Perform prefetching bandwidth
        reservation for sp (do the best policy)
    Endif
    Prefetch sp
    Update
    Nb← Nb + 1
  If Nb ≥ Upper(Si)
    Exit
  Endif
Endif
**Endforeach**
**End**

*Figure 2.* Prefetching algorithm.

### 3.3. Prefetching Example

Let us reconsider the example video document of figure 1. We have
five annotated streams (total number of COs in the video document
is 50) with their respective time intervals in units of continuous objects
(COs):
Housebreaking=[1,22]
Police Intervention=[15,50]
Alarm=[9,26]
Tracking=[19,38]
Thieves Arrest=[33,50]
Let us suppose that the stream Tracking is the currently accepted one. Then, the set of streams to prefetch \(SO_{SP_i}\), sorted on the DoO basis in a descending order, expresses as:

\[
\{<\text{Police Intervention}, \frac{5}{9} \approx 0.556, 16>, <\text{Alarm}, \frac{1}{10} \approx 0.444, 10>, <\text{Thieves Arrest}, \frac{2}{9} \approx 0.333, 12>, <\text{Housebreaking}, \frac{2}{11} \approx 0.182, 18>\}.
\]

Let us suppose that the threshold on the overlapping value DoOl i m i t is set to 0.4, then we have two candidate streams Alarm and Police Intervention. Let the upper bound \(U_{pper}(S_i)\) on the number of candidates for prefetching be one and the buffer space availability for prefetching be limited to 12 COs, then only the Alarm stream can be prefetched, even if the Police Intervention has the higher degree of overlapping with the Tracking stream.

### 4. Experimental Results

This section describes the series of experiments we have performed in order to evaluate the effectiveness of the proposed prefetching technique. We have implemented a discrete-event simulation package based on the video server model discussed in section 2. We first introduce the experimental settings and then present the simulation results.

#### 4.1. Simulation Settings

Several parameters are distinguished and fall into four categories:

1. **Server parameters**: according to the video server model, four specific parameters are considered: the number of processing nodes in the server (equal to 8), the network interconnection bandwidth (equal to 376 Mb/s), the buffer space available on each node (varying from 32 to 256 MB, default 128 MB) and the disk bandwidth (equal to 72 Mb/s). The default values of these parameters are those of a real configuration based on a cluster of PC’s \(^7\) interconnected by a Myrinet network (Myricom Comp., 1999).

2. **Video database**: two parameters concerning the video database are considered: the size of the video database in terms of hours of video stored (50 hours, i.e. more than 3 month of daily news), and the type of the compressed data (MPEG-1 : 1.5 Mb/s)

\(^7\) The cluster, named POPC, was developed by MATRA Systems & Information.
3. **Annotation database parameters**: we have obtained from our partners some statistics on the news video annotation database. Five different parameters have to be considered: the number of news subjects\(^8\) (100 subjects), the number of annotated scenes per news subject (1-5), the number of streams per scene (7-11), the length of a scene (3-5 minutes) and finally the length of a stream (30-120 sec). This annotation corresponds to a basic annotation. It does not include high level constructions (composed streams or scenes), these information are supposed to be stored in a separate annotation database. The default values are those of typical News-On-Demand applications (Hjelsvold et al., 1995).

An important point concerns the annotation structures of a scene, i.e. the mapping of the generated streams within the scene. In order to be as close to reality as possible, we have worked with our partners on real examples. In real life, librarians usually begin by first segmenting a scene incrementally, starting from the beginning, in small streams which capture fine grained contents. These streams are called "Basic Streams" and do not overlap with each others. Next, they add streams to capture shared contextual information between basic streams. The added streams are called meta streams. In our experiments, we have followed this process. The number of meta streams is half of the number of basic streams.

4. **Request scenarios**: three parameters describe the request scenario: the arrival rate of requests, the total number of submitted requests and the streams chosen to be accessed. The arrival of requests is modeled as a *Poisson* process with mean inter-arrival time of *T* seconds (*T* will be varied from 2 to 8 seconds). Each simulation run involved 1000 requests. The request scenario (i.e. the process of choosing a stream to be accessed by a new request) is based on real life practices. Indeed, a client first issues a request about a subject which corresponds to the topic chosen. Afterwards, she/he selects a stream within this subject. In our simulations, we have followed the same process. A new arrival request first selects one subject among the subjects available in the annotation database. Next, a scene is randomly chosen from the scenes of the selected subject. Finally, a stream, over the streams of the selected scene, is randomly chosen. To provide more realistic scenarios, we varied the percentage of requests accessing the same topic, i.e. the same subject, from 1\% to 8\%.

---

\(^8\) For instance, political event, sport event, etc.
Table 1. Summary table of the default setting of the simulation parameters.

<table>
<thead>
<tr>
<th>Category</th>
<th>Parameter</th>
<th>Setting (default values)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Server</td>
<td>Number of nodes</td>
<td>8</td>
</tr>
<tr>
<td></td>
<td>Buffer size at each node</td>
<td>128 MB</td>
</tr>
<tr>
<td></td>
<td>Disk Bandwidth</td>
<td>72 Mb/s</td>
</tr>
<tr>
<td></td>
<td>Network Bandwidth</td>
<td>376 Mb/s</td>
</tr>
<tr>
<td>Video database</td>
<td>Size</td>
<td>50 hours</td>
</tr>
<tr>
<td></td>
<td></td>
<td>&gt; 3 month of daily news.</td>
</tr>
<tr>
<td></td>
<td>Data type</td>
<td>MPEG1 (1.5 Mb/s)</td>
</tr>
<tr>
<td>Annotation database</td>
<td>Number of subjects</td>
<td>100</td>
</tr>
<tr>
<td></td>
<td>Number of scenes per subject</td>
<td>uniform distribution (1-5)</td>
</tr>
<tr>
<td></td>
<td>Number of streams per scene</td>
<td>uniform distribution (7-10)</td>
</tr>
<tr>
<td></td>
<td>Length of a scene</td>
<td>uniform distribution (3-5) min</td>
</tr>
<tr>
<td></td>
<td>Length of a stream</td>
<td>uniform distribution (30-120) sec</td>
</tr>
<tr>
<td>Requests</td>
<td>Mean inter-arrival time (T)</td>
<td>4 sec</td>
</tr>
<tr>
<td></td>
<td>Number of request</td>
<td>1000</td>
</tr>
<tr>
<td></td>
<td>Access distribution</td>
<td>uniform distribution</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(number of subjects)</td>
</tr>
</tbody>
</table>

Note: The three parameters buffer size, request inter-arrival time and request scenario are the most important parameters to influence the ratio of the buffer hit rate metric for server caching with and without prefetching. Experiments performed on parameters concerning the annotation database: the number of subjects and the number of scenes per subjects did not influenced the performance comparisons. These results can be looked up in a previous paper (Mostefaiou and Brunie, 1999).

Table 1 summarizes the settings of the simulation parameters.

4.2. EXPERIMENTAL SCENARIOS

We conducted several series of experiments. Each experiment consisted of 20 simulation iterations and each iteration simulates 1000 requests. The confidence interval tests of the results (number of requests served from the buffers) indicate that 98% of the results are within 5% of the mean in almost all cases. All parameters are set to their default values as reported in table I unless explicitly indicated in the figures. We studied the effectiveness of the proposed prefetching technique by means of three parameters: the buffer space available at each node...
(varying from 32 MB to 256 MB), the request inter-arrival time (varying from 2 to 8 seconds), and the request scenario (varying from 1% to 8%). The metric chosen for the evaluation of the proposed prefetching is the Buffer Hit Ratio (BHR) which is commonly used for that purpose. The buffer hit ratio is defined as the ratio between the number of requests served entirely from the buffer and the total number of requests.

We first discuss the tests performed to determine the appropriate value for $DOO_{\text{limit}}$ (subsection 4.3). Then we analyse the performance of the system with and without using our proposed prefetching technique (subsection 4.4). Finally we compare the benefits of the proposed prefetching against its costs (subsection 4.5).

### 4.3. Determination of an Appropriate Value for $DOO_{\text{limit}}$

Based on the parameters listed in table I, we conducted experiments with three values of $DOO_{\text{limit}}$: 0.25, 0.50 and 0.75 against the varying parameters (the available buffer space, the request inter-arrival time and the request scenario). The impact of the $DOO_{\text{limit}}$ value on the prefetching performance is shown in figures 3, 4, and 5.

![Buffer Hit Ratio vs Buffer Size](image)

*Figure 3. Effect of varying the buffer size.*

Figure 3 displays the impact of the buffer space on the prefetching performance for the three value of $DOO_{\text{limit}}$. As expected, the buffer hit ratio increases as the available buffer space increases. For a buffer size of 256 MB, a $DOO_{\text{limit}}$ value equal to 0.25 gives the best buffer hit ratio. For lower buffer sizes, the performance of the prefetching technique for all $DOO_{\text{limit}}$ values is almost the same. Indeed, with a
**Figure 4.** Effect of varying the request inter-arrival time.

**Figure 5.** Effect of varying the request scenario.

DOO\textit{\textit{\textit{l}}imit} value equal to 0.25, the system performs more prefetching, consequently greater available buffer space leads to better performance. Next, we studied the impact of the request inter-arrival time on the three prefetchings. From figure 4, we can see that the performances are nearly the same when the request arrival rate is high. However, the performance of the prefetching with DOO\textit{\textit{\textit{l}}imit} equal to 0.25 is more noticeable when the arrival rate of requests is low. Finally, we studied the performance of the prefetching in the presence of different request scenarios. More precisely, we varied the percentage of requests accessing the same subject. The mean performance for prefetching is
the best in the case $DOO\_limit=0.25$ (9% and 10% of request on the same subject).

According to these results, the value of $DOO\_limit$ was set to 0.25 for the remainder of the experiments.

4.4. Validating the prefetching technique

This second experiment analyzes the benefits of our proposed prefetching technique. The results are given in three parts. In the first two parts we will demonstrate the performance gains: the first part examines the impact of the available buffer size on the performance and the second part studies the effects of varying the request arrival rate, both parts with respect to the percentage of accessing the same object. The third part analyzes the additional costs incurred by our prefetching technique.

4.4.1. Effect of varying the available buffer size

In these experiments, we varied the available buffer space on each node of the server from 32 MB to 256 MB. We compared the performance of the system with and without using prefetching. Figures 6, 7, 8 and 9 display the results of configurations with 256 MB, 128 MB, 64 MB and 32 MB respectively.

![Buffer Hit Ratio Graph](image)

**Figure 6.** Buffer Size = 256 MB.

As expected, independently of the request scenario, the performance of the system (with and without prefetching) increases with the increase of the available buffer space. For instance, when the percentage of
accessing the same subject is equal to 5%, the BHR (Buffer Hit Ratio) for a configuration with 128 MB buffer size, is about 32% and for a configuration with 256 MB buffer size is about 50%. This is because more accessed streams are cached in the buffers and therefore the BHR is increased. The results highlight the relevance of caching in multimedia servers. Indeed, better performance is reached with more available buffer space. However, when the buffer space is not significant i.e., 32 MB per node, (see figure 9), the BHR is not very noticeable i.e., less
than 15% in general. The performance of the system is better in the presence of "hot accesses". For example, in a scenario where 8% of the total requests access the same subject, the BHR is more than 55%.

In all request scenarios the prefetching improves the performance of the system. On average, the improvement rate is about 18.45%. The highest improvement (up-to 40%) rates are reached in the presence of "hot accesses", i.e. 8% of requests access the same stream. But even for lower request access rates, very important improvements can be shown, for instance when 2% of requests access the same stream and for a configuration with 128 MB of buffer size, the improvement is about 41.9%. Even in the case of very high buffer space availability a noticeable overall improvement rate of about 17.1% can be measured. This lower rate is due, as mentioned before, to the caching of hot streams. Indeed, the cached streams are preserved in the buffers and consequently the effect of the prefetching is attenuated. The more the streams are cached, the less the prefetching improvement in prefetching.

4.4.2. Effect of varying the request arrival rate

In these experiments, we studied the sensitivity of the proposed prefetching technique with respect to the variation of the request arrival rate. We varied the mean request inter-arrival time from 2 sec to 8 sec. Figure 10 and figure 11 illustrate the results for 2 sec and 4 sec respectively.

As it is the case in the previous subsection, the buffer hit ratio increases with the percentage of accessing the same subject indepen-
Figure 10. Buffer Hit Ratio for a varying percentage of accessing the same subject for mean request inter-arrival time equal to 2 sec.

Figure 11. Buffer Hit Ratio for a varying percentage of accessing the same subject for mean request inter-arrival time equal to 4 sec.

dently of the request arrival rate. This is illustrated in figures 10 and 11. The system performance with prefetching outperforms the one without prefetching in all cases. The average improvement is about 18.5%. In comparison to the buffer space impact, the system performance is less sensitive to the request arrival rate. The difference in the performance between different values of the mean inter-arrival time is not noticeable.
Note that there is no improvement incurred by the prefetching when the arrival rate of request is high i.e., when the request inter-arrival time = 2 sec, and when the request scenario is uniform, i.e., the percentage of accessing the same stream = 1% (see figure 10). This is because most of the requests are in competition for resources reservation with the prefetching requests. For mean request inter-arrival time greater than 8 sec, the improvement tends to a fixed value (see figure 12 where the absolute buffer hit ratio improvement for different request inter-arrival time is displayed). The reason is that beyond a certain request rate, the new requests do not affect the prefetching triggered by precedent requests i.e. the prefetching terminates before the arrival of new requests and therefore is not in competition for resources with them.

![Prefetching improvement](image)

*Figure 12. Absolute BHR prefetching improvement for different request inter-arrival time.*

4.5. **Cost of the Prefetching**

In this section the cost of the proposed prefetching approach is discussed and compared to its benefits. The metric we have chosen is the total number of requests that are not accepted immediately due to the triggered prefetching. The results are shown in figure 13. Note that only under extreme conditions (inter-arrival time equal to 2 sec), the prefetching alters the admission of new requests. Beyond this value, no effect has been noted.

From the figure, when the request scenario is uniform (i.e., there is no influence of the hot accessed streams), the amount of waiting request is about 2.6% , i.e. 26 requests out of 1000 requests were not accepted immediately. This is due to the high requests arrival rate.
As the prefetching consumes resources (mainly buffer space) the new arrived requests have to wait until the prefetching has been finished.

The number of delayed requests decreases considerably when the percentage of accessing the same subject increases. For example, when 5% of the requests access the same subject and with a request arrival rate equal to 30 requests/minutes (inter-arrival time equal to 2 sec), the cost of the prefetching is 0.5% where the improvement for the same request scenario is 16.01%. This shows clearly the benefit of the proposed prefetching technique.

5. Conclusion and future work

This paper proposes a new prefetching technique to handle the problem of buffering in multimedia servers. The prefetching issue described in related work has been often addressed based on run-time considerations (objects access frequencies, user VCR interaction, etc). Our technique, in addition to run time information, takes into consideration knowledge on the data structures and their relation to trigger prefetching of streams. The class of multimedia systems most amenable to the technique proposed here is the class of News-on-Demand systems. Indeed, in such systems, the streams have complex structures and are highly semantically correlated. We argue that by considering this available knowledge, the system can prefetch future accessed streams. To validate our proposal, we have studied a video repository application. The results clearly demonstrate the effectiveness of our approach (on average an improvement of 18% in the buffer hit-rate has been achieved).
argues for more in depth research in this direction by considering more complex structures.

In a future research work, we plan to generalize this proposal to high level semantic information from an annotation model (Prié et al., 1998). Furthermore, as video repository applications require tertiary storage, we plan to expand the proposed approach to handle I/O between the video server and the tertiary storage system.

References


