will be published in Proc. of the Int. Workshop on Multimedia Information Systems (MIS), Istanbul, September 1998, Springer Lecture Notes in Computer Science.

An Admission Control Framework for Applications with Variable Consumption Rates in Client-Pull Architectures*

Silvia Hollfelder, Karl Aberer

GMD - Integrated Publication and Information Systems Institute (IPSI)
Address: Dolivostraße 15, D- 64293 Darmstadt, Germany
E-Mail: {hollfeld | aberer}@darmstadt.gmd.de

Abstract. Highly interactive multimedia applications require variable data rates during their presentation. Current admission control mechanisms do not address the variable data rate requirements appropriately for the following reasons: (1) classical admission control mechanisms are based on the server-push approach, where the required data rate has to be estimated in advance, and (2) worst-case resource reservation is not economic. Client-pull models are more appropriate to serve these kinds of applications. At the current state, there are no suitable mechanisms that support admission control in client-pull architectures at the server. In this paper, a session-oriented framework for admission control is introduced that is based on two steps: (1) the admission of new clients and (2) the scheduling of the single requests of admitted clients to balance the load. The goal of this approach is to improve the server utilization and the Quality of Service. Evaluation studies demonstrate the benefit of the framework.

1. Introduction

Enhanced multimedia applications like computer-based training (CBT), product documentation, or digital libraries enable a flexible user behavior since the user has various options to realize its preferences interactively. Examples of interaction types that occur in such applications are the usage of VCR-functionality for the playback of continuous media, browsing in media archives [10] and the interactive control of preorchestrated composite multimedia presentations [1]. These different interaction types cause highly varying data consumption rates of the application and makes it difficult to specify resource demands in advance. Thus, the media storage components have to provide mechanisms that are able to deal with these characteristics of highly, interactive multimedia applications. These mechanisms differ from those required by

* This work has partly been funded by the European Union within the framework of the ESPRIT Long Term Research Project HERMES, No. 9141.

multimedia applications with more uniform access characteristics, like video-on-demand.

For illustration purposes, we discuss the characteristics of browsing applications in digital libraries as one typical example. In a digital library, the users have to deal with a huge amount of data, including video data. Therefore, they want to scan quickly through data that has been roughly specified by a query in order to inspect and compare it. In contrast to a video-on-demand application where the users generally want to view complete, single videos, in this scenario, the user makes full use of VCR-functionalities, like fast forward, rewind and pause, and might require to view several videos in parallel. Thus, within this application the data requests arrive aperiodically at the server, and the data rates vary within a wide spectrum. Another example for applications with variable data consumption are composite multimedia presentations where the user gets presented various media, like text, image, audio and video, simultaneously and the course of the presentation is controlled interactively by the user. The task of the multimedia data management system in such scenarios is to support the qualitatively adequate presentation of the required media data. Especially in distributed environments, this is a challenge.

Distributed multimedia applications require the support of continuous data flows from the server to the client satisfying Quality of Service (QoS) parameters. In order to achieve the required presentation quality, the clients compete for limited resources on the server. The basic strategies to deal with limited resources can be distinguished into optimistic and pessimistic ones.

With optimistic strategies all requests are served as well as possible (best effort). These strategies are typically used in client-pull architectures, where the client subsequently requests small chunks of media data during presentation. The server triggers the delivery of data only after a single, “small“ request has arrived. The data requests arrive aperiodically at the server [15]. In case of user interactions the client changes its request behavior, e.g., it will request larger blocks of a media or send more frequent requests. The client-pull architecture is suitable for interactive applications with varying resource requirements since it is not assumed that the required data rate will be constant for the duration of a presentation. Bottlenecks are dealt with either by the server or clients with various strategies, e.g., by means of quality adaptation mechanisms at the client [9] or at the server [18].

With pessimistic strategies resource reservations are made at the server, in advance. An admission control mechanism usually checks at the server, if enough resources for the adequate delivery of data to a new client are available. If there are enough resources available, the client is admitted and the resources are reserved for this client until the end of the presentation. Most approaches for admission control mechanisms consider the request of single media streams within a server-push architecture. Since the whole presentation is pre-specified by the request the server manages the delivery of the whole media stream and pushes the data continuously to the client. Admission control mechanisms for “pure“ server-push architectures assume that during a presentation the data rate consumed by the client will be nearly constant [15]. The available system resources are calculated by stochastic [11,19] or deterministic approaches [20,12]. Based on the knowledge about the already reserved

and freely available resources it is possible to reject requests in case of server overloads. Admission control mechanisms are harder to use for interactive applications since it is difficult to estimate server resources consumed by the applications. We discuss some of the strategies that can be considered.

A priori reservation. To guarantee a given Quality of Service worst case assumptions about the data rate required can be made. Obviously, the reservation of the worst case data rate wastes server resources and decreases the number of clients that can be served in parallel. Dey-Sircar et al. [5] reserve separate server bandwidth for VCR-interactions, using statistical methods. The drawback of their work is that they assume that interactions occur rarely.

Re-admission at interaction points. A straightforward way to use standard admission control policies with interactive applications is to perform admission control for each single media object request, that can occur as the result of an interaction as described in Gollapudi and Zhang [7]. One drawback of their approach is that each client request is subject to the admission control. For example, when the first scene of a video is admitted there is no guarantee for the admission of the subsequent scenes of the same presentation. This may lead to unacceptable delay in presentation when too many clients send requests. The main problem of their approach is that the admission is not performed for a client session. Thus, the admission of one continuous media stream of a multimedia presentation does not necessarily guarantee the admission of another continuous media stream that has to be synchronized with the already admitted streams. But, especially in composite, interactive presentations, temporal requirements between multiple media, like start_with, or finish_with, have to be considered.

Smooth the application data rates. Some approaches to admission control for interactive applications propose to 'smooth' the data rate deviations to achieve a relatively constant workload. Shenoy and Vin [16] reduce the high data rate for fast forward and fast rewind of MPEG-videos by encoding the stream in base and enhanced layers. The encoding of the base layer is done by reducing the temporal and spatial resolution. For fast forward only the base layer is used. Chen, Kandlur, and Yu [3] suggest segment skipping where a segment can be a set of Group of Pictures (GoP) of an MPEG-video. For fast forward or fast rewind some segments are skipped. Chen, Krishnamurthy, Little, and Venkatesch [2] change the order of MPEG-frames to a priority sequence. For fast forward and fast rewind only the most important frames (I- and P-Frames) are pushed to the client. The higher data rate is reduced by quality adaptation on the temporal dimension of other requests by a dynamic resource reservation. Reddy [14] reduces the latency of 'urgent' requests, but neglects varying bandwidth requirements. The smoothing approach is, however, restricted to relatively simple interactive scenarios where interactions take place within the presentation of one single media stream.

Reviewing the different approaches, up to date, no admission control mechanism has been developed which is applicable for varying resource requirements of highly interactive applications requiring a client-pull architecture. We presented first ideas on admission control for client-pull architectures in [8]. In this paper, we propose an admission control framework to support interactive multimedia applications,

assuming that no user profile is available that gives any information about the resources required by a pending client. It consists of (1) the admission of new clients, in the following called pending clients, when server resources are available and (2) the scheduling and adaptation of requests of admitted clients. We evaluate different strategies, both for admission and scheduling. With the approach we benefit from the relative safety provided by admission control, but still provide fallback strategies in the case of overloads. The admission control mechanism we propose is session-oriented such that requests related to the same presentation or session do not require separate admissions. This gives adequate support for composite multimedia presentations. The main problem to be adressed is the criterion upon which the admission decision can be performed. The inital request of a session does not provide the necessary information to predict future requests. Here, we use statistical features of the running sessions for determining the admission criterion. For a large number of parallel session the average client consumption is a good estimate for prediction. Data rate variations are accounted for by introducing a safety margin. Thus, an admitted client is supposed to obtain sufficient resources. If in spite of the admission control resource bottlenecks occur, strategies for rescheduling of requests are used to achieve high QoS by means of load balancing. In the worst case quality adaptations are required to enable guaranteed continuous delivery.

We first introduce the underlying system model in Section 2.1. Then we introduce the admission and scheduling strategies in Sections 2.2 to 2.4. In Section 3 we present the results of evaluation studies and conclude the paper with some remarks on future research directions.

2. Admission Control Framework

We describe a framework that is under implementation for a Multimedia Database Management System at GMD-IPSI. It is designed to support highly interactive applications with variable data consumption. The goal is to provide better QoS than best effort delivery and to achieve better server utilization than worst-case-based reservation.

2.1 System Model

The system model is based on a client-pull architecture. Clients request multiple chunks of data from the server within a single multimedia session. A multimedia session contains the presentation of at least one time-dependent media, like a video or an audio. Single media as well as composite media presentations are possible.

One task of the server is to limit the number of active clients by means of an admission control mechanism. At the beginning of a session, a client requests the server for admission (see Figure 1). Then the server informs the client about its admission. After a client has been admitted, it can send data requests to the server. When a client terminates its multimedia session it informs the server. The server

handles data requests from admitted clients within service periods of fixed length, which are indexed by $k \in \mathbb{N}$. Data requests specify which part of which media object is requested by the client and at which time it has to be delivered. Thus, a data request has the form <b, t>, with $t \in \mathbb{N}$.

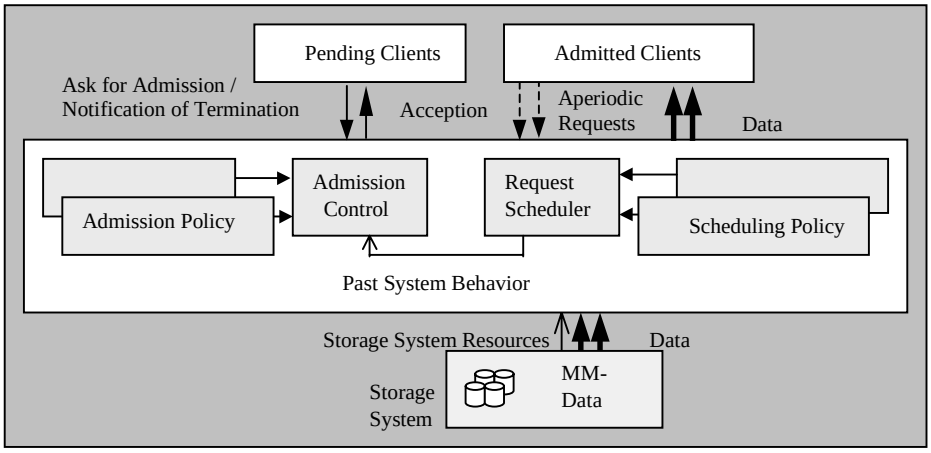

Fig. 1. The System Model

The specifications used for a block b can be of different types. In the simplest case, they indicate the beginning and the end of a block within a media stream. For example, for a video stream this can be the first and the last frame of a scene requested by the client. More complex cases are, for example, requests of data required for fast forward, which may require only every second frame of a video, and requests of parts of different media like an audio with the corresponding video. The server is responsible for mapping the specification of the data requests to the corresponding data accesses to the media data.

The deadline t specifies the time in number of service periods that a client is willing to wait until its request is scheduled. These deadlines may vary between the single requests of a client. The clients are able to specify deadlines since they have more information on the presentation state. For example, the buffer state of a client may be used as an indicator for the urgency of a request. By sending requests in advance the client can also account for delays of the request that can occur, for example, due to network transmission time, or can plan ahead for future presentation. This means that it has to send the data request in time to the server to avoid that data will be delivered too late. It overcomes deviations in delivery by using a local buffer [4]. The client assumes that requested data will be delivered within the given deadline, e.g., at the earliest within the next service period.

In each period k, the server collects the incoming requests, schedules the requests for the next service period, and when the system resources are not sufficient it delays the remaining requests for later periods. Since the delay of requests depends on the scheduling policy and not on the arrival time of the request, the arrival order does not

necessarily determine the scheduling of requests, even for the same client. Still the arrival time can be used as one priorization criterion.

In addition to scheduling single requests, the server has to be able to fragment single requests <b,t> into several disjoint requests <b_1, t>, ..., <b_n, t>, such that b_1, ..., b_n specify the same data as b, for two reasons. First, though we assume that single requests are small, it may happen that data requests are too large to be served within one service period. Second, for optimization it can be advantageous to split a request, such that part of the request is served with the remaining resources of a current service period and utilization is increased.

The storage system is responsible for the execution of the schedule. It maps the schedule to accesses to the storage system. This opens the opportunity to increase the throughput by optimizing the reading order for all requests scheduled for one service period. It is assumed that the request scheduler has a model for the storage system throughput and uses this for determining the scheduling.

The approach described in this paper abstracts from the properties of the storage system like disk scheduling policies, data placement strategies, and buffer size but it assumes that the throughput of the underlying storage system is given as a bandwidth value [5]. This assumption is realistic because todays‘ storage systems that consist of disk arrays do not offer access to single disks. The storage system reads the requested data from the disk, buffers it in the server cache, and sends it to the client over the network. It is also assumed that the network will be able to send the data appropriately to the client.

2.2 Admission of Pending Clients

In the following, several strategies for admission control are introduced. They are based on an admission criterion that automatically adapts to the system load. It has to adapt to different client consumption types, both for uniform and for bursty ones. The goal of the strategies is to achieve a high system utilization and to avoid overload situations since they degrade the temporal QoS by delayed requests.

Before a client session is allowed to issue data requests it requests admission from the server. This request is called admission request.

Let Res_{max} be the maximal amount of resources provided by the storage system. For each period k we introduce the following parameters: the set of currently admitted clients C_k, the set R_k of open requests that need to be handled by the server in the next period, and the set S_k of requests that are scheduled for the next period. Assume that for a given period k we have $R_k=\{r_1, ..., r_n\}$ and $S_k=\{s_1, ..., s_m\}$. The number of admitted clients is given by $| C_k |$. Then we define the server load l_k by

$$l_k = \frac{1}{Res_{max}} \sum_{i=1}^{n} res(r_i)$$

where $res(r_i)$ are the resources required to serve request r_i. Note that the server load can be larger than 1.0 thus not all requests can be handled within the next period.

Next, we define the server utilization u_k by

$$u_k = \frac{1}{\mathrm{Re}\, s_{\max}} \sum_{i=1}^{m} res(s_i)$$

where $res(s_i)$ are the resources required to serve the scheduled request s_i. Note that the server utilization always is smaller or equal to 1.

In the simplest case the admission control can admit new clients as long as storage system resources are available, i.e., $u_k \leq 1$, which basically amounts to a best effort strategy. However, the current server utilization in service period k is only a good prediction on the future server utilization, as long as the resource demands of single sessions vary only little. Thus, in approaches that smooth the resource demands for each session, as discussed in Section 1, the current server utilization is a viable measure for the admission, while for highly interactive applications, that we consider, it is not. Similarly, the current server load can be considered as an admission criterion, with $l_k \leq 1$.

In order to adapt to the long term system behavior, we introduce a 'lookback' value λ that determines the number of service periods we look back to observe the past system behavior. The choice of this value determines how fast the admission control can adapt to new load patterns, or how strongly it compensates for short term deviations in server load. Based on the quantities u_k and l_k we can now define two quantities that will be considered for an admission criterion.

The *average server load* $l_{k,\lambda}$ of the past λ periods is given by

$$l_{k,\lambda} = \frac{1}{\lambda} \sum_{i=1}^{\lambda} l_{k-i}$$

The *average server utilization* $u_{k,\lambda}$ of the past λ periods is given by

$$u_{k,\lambda} = \frac{1}{\lambda} \sum_{i=1}^{\lambda} u_{k-i}$$

In the following, we introduce two approaches for defining an admission criterion that considers the past system behavior. For these criteria we introduce a quantity $\tau \in [0,1]$ that is used as safety margin. This quantity determines how close the average utilization and load values may approach the maximum value of 1, and thus how much tolerance is available to compensate for short term deviations. High values of τ represent an aggressive admission policy, while low values of τ represent a defensive admission policy. In the first case, overload situations may happen more frequently, while in the latter case the utilization will be lower and less clients are admitted.

Strategy 1: Lookback to server behavior. This strategy considers the past server resource values as admission criteria. For the admission, both average server load and the average utilization can be used. Pending clients are admitted if the average server load or the average utilization are under the safety margin, i.e.,

$$u_{k,\lambda} < \tau \quad \text{or} \quad l_{k,\lambda} < \tau.$$

This strategy does not consider the information on the number of admitted clients.

Strategy 2: Lookback to client behavior. The idea is to estimate the client consumption by considering the average request or average consumption rate of the admitted clients, in the past. The advantage of this approach is that the number of admitted clients is considered for the admission criteria.

The average client request rate $r_{k,\lambda}$ of the past λ periods is given by

$$r_{k,\lambda} = \frac{1}{\lambda} \sum_{i=1}^{\lambda} \frac{l_{k-i}}{|C_{k-i}|}$$

The average client consumption rate $c_{k,\lambda}$ of the past λ periods is given by

$$c_{k,\lambda} = \frac{1}{\lambda} \sum_{i=1}^{\lambda} \frac{u_{k-i}}{|C_{k-i}|}$$

A pending client is admitted in service period k, if

$$c_{k,\lambda} * (|C_k| + 1) < \tau \qquad or \qquad r_{k,\lambda} * (|C_k| + 1) < \tau$$

With this admission criterion a pending client is admitted when in addition to the sum of the average consumption or average request rate of the admitted clients C_k resources for at least one new client are available. This means that resources required for a pending client are estimated to be equal to the average resources required by the admitted clients. Furthermore, it is expected that the average past behavior of admitted clients will be an indicator for the future. Note that we neglect their individual behavior since it is not necessarily representative for future resource demands.

2.3 Scheduling Policies for Data Requests

After pending clients have been admitted the server has to deliver the clients' requests in the forthcoming periods until their sessions are terminated. In each period k, the server collects the incoming requests I_k together with the requesting client and the time of arrival. Thus I_k consists of tuples $<c, t_{in}, b, t>$, $c \in C_k$, $t_{in} \in \mathbb{N}$, where C_k is the set of admitted clients and t_{in} the arrival time of a request at the server. By means of a scheduling policy the server generates a schedule S_k that determines which requests are scheduled for the next period, where $S_0 = \emptyset$. The set of open requests R_k in each period k is then determined by the incoming requests I_k and requests from previous periods that are not scheduled yet by $R_k = I_k \cup (R_{k-1} \setminus S_{k-1})$, where $R_0 = \emptyset$. During scheduling it may happen that the available system resources are not sufficient to handle all open requests. Then, the scheduling policy as a first option can move a request $<c, t_{in}, b, t>$ to period k+1 without harm as long as $t > k$.

As soon as the open requests cannot be satisfied within the forthcoming service period the scheduling algorithm has to priorize the open requests. In the following, we discuss different scheduling policies with two goals in mind: (1) serve open requests within their time constraints (see Section 2.1), and (2) balance the system load between different service periods. Load balancing is important since the system load varies in between consecuting periods. In contrast to the admission control mechanism, which targets at avoiding long term bottlenecks, short-term deviations in

system load can be smoothed through request scheduling. The following scheduling policies can be considered [8].

- First Come First Serve (FCFS): The simplest policy is to serve the requests by their arrival order. This policy has only small potential for load balancing since it does not consider the request urgency for scheduling.
- Resource-driven Scheduling: *LORF* (Low Resource Requests First Serve) reduces the negative effects of overload by first scheduling the requests with lowest resource requirements. This increases the number of requests that can be served. This strategy is useful in applications where the benefit of the system is related to the number of requests served within the time constraints.
- Earliest Deadline First Serve (EDF): *EDF* gives those requests priority that have the closest deadline. The further the deadlines of requests reach into the future, the higher is the potential of this policy. The goal of the policy is to achieve a high percentage of requests that have to be served within their deadlines, i.e., increase QoS in terms of jitter.
- Quality of Service-driven Scheduling: This policy requires a QoS-metric representing the relevant QoS parameters that determine acceptable quality ranges. High Quality First Serve (*QUF*) prefers requests that require high QoS parameters, assuming that applications with high quality requirements are most interested in good service.

When the advanced scheduling policy is not able to schedule open requests within their time constraints their service will be delayed. If the admission and load balancing techniques can no longer guarantee the full service quality, additional mechanisms for quality adaptation can be used to overcome bottleneck situations. The goal of quality adaptation is to balance quality degradation, of which lost or delayed requests is one type. With adaptation the required data volume is reduced by modifying the quality of the delivered data in the temporal and/or spatial dimension. For example, it is possible to modify the resolution of a video or audio, or drop frames of a video [9]. When the server decides to adapt it has the responsibility to reduce the data rate and deliver the data with reduced quality in a way such that the client is able to interpret the returned data, which then deviates from its original request. This might be a non-trivial problem for compressed stream data like MPEG with interframe dependencies. The quality adaptations are in the simplest case distributed over all clients equally, or can be based on individual QoS profiles of the clients. Related work on quality adaptation can be found, e.g., in [17].

Thus, the combination of an admission control mechanism, a scheduling mechanism with load balancing, and a quality adaptation mechanism is the way to enable the server to optimally adapt to the requirements of highly, interactive multimedia applications and at the same time achieve high resource utilization. The admission control and scheduling mechanism cannot guarantee that system overload does not occur but, by an appropriate choice of parameters, allow to reduce the number of overload situations.

2.4 Service Algorithm

We now give the detailed algorithm for admission control and request scheduling. The algorithm is given in pseudo code. In the algorithm, the average client consumption is used as admission criterion. The algorithm for the other admission criteria is analogous. For periods with $k < \lambda$ the value $c_{k,k}$ is used for the admission criterion, instead of $c_{k,\lambda}$.

```
S_0 := Ø; R_0 := Ø; C_0 := Ø; k:=1;
While(true)
Begin
P_k := requests for admission;
While(P_k ≠ Ø and c_k,λ (|C_k| +1) < τ)
      Begin
      C_k := C_k ∪ {first(P_k)};
      P_k := P_k \ {first(P_k)}; End;
I_k := requests from admitted clients;
R_k := I_k ∪ (R_k-1 \ S_k-1);
Sort R_k according to priority criterion;
Free_res = Res_max; S_k := Ø;
While(free_res > res(first(R_k)))
      Begin
      S_k := S_k ∪ {first(R_k)};
      R_k := R_k \ {first(R_k)};
      free_res := free_res - res(first(R_k)); End;
Execute scheduled requests in S_k;
Update c_k,λ;
Remove terminated clients from C_k;
k++; End;
```

Fig. 2. Pseudo Code of Admission Control and Scheduling Algorithm.

After performing admission control the algorithm schedules open requests until all resources of the next service period k+1 are reserved. After that, the schedule is sent to the storage system. In addition, the average client consumption is updated for the admission control mechanism in each service period.

3. Evaluations

In this section, we present experimental results that were obtained in a simulation environment implemented in the Mathematica [21] system. We compare two quantities for determining the quality of an admission and scheduling strategy, namely the average server utilization versus the number of delayed requests, which we use as a QoS criterion. In the course of this investigation a particular focus was on the stability of the admission control mechanism we propose. Stability means in this

context that the system is able to control itself in phases with low and high load. For example, in underloaded periods the danger exists that too many clients get admitted, and in overloaded periods too many pending clients get rejected which leads to system underload in the future. In the worst case, oscillating behavior of the system load occurs. This is a critical point for any feedback system.

We describe the configuration that was used for simulating client behavior. The client switches between periods, where it requests data, and idle periods, in which it does not request data (e.g., the user is inspecting discrete data). During the request periods the client requests a varying load of media data (a media stream). All duration parameters are generated using the exponential probability distribution, while the request sizes are generated using the normal distribution. The server is assumed to have 30000 units of resource within a service period. The duration parameters are chosen under the assumption that a service period has a length of one second. In the chosen configuration, there exist two types of clients. The first has an average request period duration of 10 service periods with an average request size of 1400 units of resource per service period, an average idle period duration of 10 service periods, and average total session duration of 100 periods. The second client has an average request period duration of 30 periods with an average request size of 4200 units of resources, an average idle period duration of 3 service periods, and an average total session duration of 200 service periods. While the first client is intended to simulate average quality presentations for a browsing application, the second client simulates a VCR access to a high quality video. Low quality clients are created five times as often than high quality clients. Furthermore, clients are created such that the server periodically has to deal with phases of low load and high load. One such phase has 100 periods. In this way, the server has to continuously deal with the stability problem. Our configuration allows only around 10 parallel clients. This low number of clients generates still a rather bursty distribution of requests and thus imposes a more difficult challenge to the admission control than a larger number of clients.

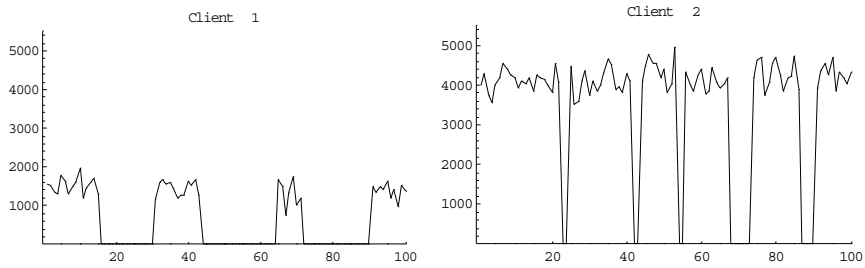

Fig. 3. Client Request Behavior of Type 1 and Type 2.

Figure 3 illustrates the typical behavior of both types of clients over their lifetimes. The x-axis represents the service periods and the y-axis gives the requested resource units. The graphs represent the request behavior of a client during its request and idle times. The request behavior varies around 1400 resource units in the left figure and around 4200 units in the right figure during a multimedia presentation. The clients request no data during their idle periods.

First we evaluate the different strategies that we have proposed. As a first experiment we take strategy 2 and run a prescreening for (λ, τ)-values for both, average client consumption rate $c_{k,\lambda}$ and average client request rate $r_{k,\lambda}$, as admission criterion. As scheduling policy we use EDF. The results are summarized in Table 1.

λ	τ	average utilization $c_{k,\lambda}$	variance of utilization $c_{k,\lambda}$	fraction of requests served in time ($c_{k,\lambda}$)	average utilization $r_{k,\lambda}$	variance of utilization $r_{k,\lambda}$	fraction of requests served in time ($r_{k,\lambda}$)
1	0.8	0.827701	0.132783	**0.774411**	0.856366	0.160017	**0.649223**
10	0.8	0.801515	0.168636	0.968677	0.818404	0.258037	**0.410026**
100	0.8	0.744060	0.164218	0.976362	0.705767	0.164979	0.977320
1	0.9	0.882038	0.152534	**0.246973**	0.924667	0.089782	**0.324022**
10	0.9	0.840137	0.141908	**0.890563**	0.793205	0.179114	**0.711840**
100	0.9	0.790747	0.140999	0.979186	0.719280	0.145321	0.976081
1	0.95	0.904935	0.122163	**0.438073**	0.883451	0.101035	**0.818325**
10	0.95	0.890717	0.119438	**0.521545**	0.814924	0.173354	**0.767916**
100	0.95	0.825157	0.157906	**0.712329**	0.732246	0.173858	0.900999

Table 1. Results of Strategy ´Lookback to Client Behavior´ (Strategy 2).

The left side of the table displays the results with $c_{k,\lambda}$ and the right one the results with $r_{k,\lambda}$ as decision criterion. This experiment shows that low values of λ (e.g., 1, 10) and high values of τ (e.g., $\tau = 0.95$) can immediately lead to unstable behavior (see bold values). This means that short lookback phases are not sufficient for the admission control because the system load varies in this experiment in larger time-windows. Further, in our configuration, a safety margin of $\tau = 0.95$ and higher is not sufficient to compensate short-term load deviations.

For better illustration we give one typical example of a stable system behavior for the values $\tau = 0.9$, and $\lambda = 100$, using $r_{k,\lambda}$ as a criterion for the admission which served more than 97% of the requests in time and achieved average utilization of almost 72% (see Figure 4). The best system utilization would be achieved by an average server load of value 1. The left figure shows that short term peaks in the server load can be dealt with. In the right figure, the dotted line gives the number of requests served within their deadline, and the solid line gives the number of delayed requests over the service periods. For example, it is shown how EDF balances the highest server load, occurred some periods just before service period 50. The request scheduler is able to avoid delayed requests at this period. This high server load leads to single delayed requests just after period 50.

Next, we evaluate alternative admission and scheduling strategies. The following experiment shows that average server utilization $l_{k,\lambda}$ and similarly average server load $u_{k,\lambda}$, used for strategy 1, are no adequate admission criteria. One reason for this worse behavior is that for a long lookback the values $l_{k,\lambda}$ and $u_{k,\lambda}$ increase and decrease very slowly when new clients get admitted or rejected since these values represent average values over the past. In this case, it may happen that with a long lookback, phases of low past server load lead to uncontrolled admission of new clients, and vice versa. This leads in the extreme case to a strongly oscillating behavior and extreme server

overload. On the other hand, short lookbacks do not sufficiently reflect the varying resource requirements. This worse behavior does not occur with strategy 2 since this admission criteria does account for the current number of admitted clients.

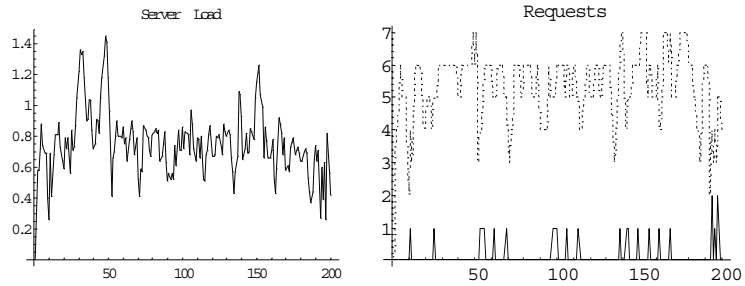

Fig. 4. Example of a stable System Behavior using ´Lookback to Client Behavior´ (Strategy 2).

We performed an experiment with (λ, τ)-values of (20, 0.9), as displayed in Figure 5. The left side of Figure 5 demonstrates the extremely high server load with multiple peaks (server load > 10). The right side shows that the number of delayed requests (solid lines) is extremely high, and only a few requests are served within their deadlines (dotted lines). With larger values of λ the behavior becomes even worse.

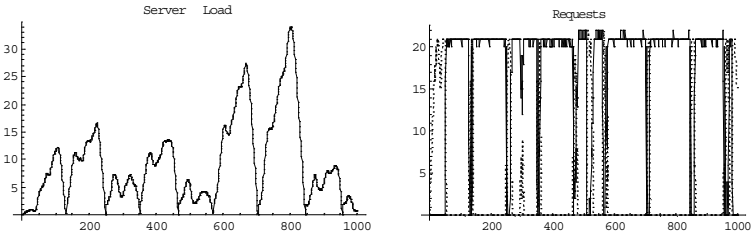

Fig. 5. Example Using ´Lookback to Server Behavior´ (Strategy 1).

The next experiment (Figure 6) illustrates the benefit of using the EDF scheduling strategy. We performed the experiment with (λ, τ) -values (100, 0.9) with the average client request rate as admission criterion, which behaved well when using EDF.

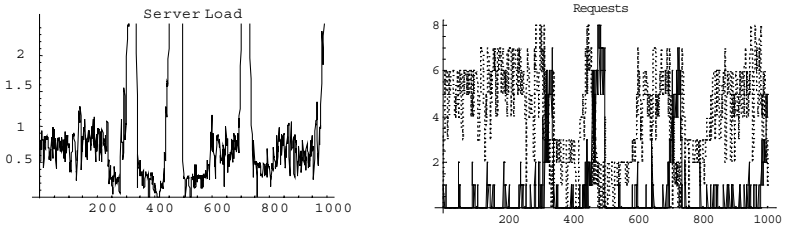

Fig. 6. Example Using ´Lookback to Client Behavior´ (Strategy 2) and FCFS.

For the same experimental parameters the FCFS scheduling strategy performs substantially high variations in server load (left figure). The temporal QoS, measured

in percentage of requests served in time, degraded to a rate of 0.85 in this experiment (right figure).
With regard to the question, whether average client consumption or average client request rate is the preferable admission criterion the results are not fully conclusive, yet. However, the Table 2 reports results of an experiment with $\lambda = 200$ that is one of several experiments, which indicate that the average request rate appears to be the admission criterion leading to more stable behavior. This experiment was run over 1000 periods.

decision criterion	λ	τ	average utilization	variance of utilization	fraction of requests served in time
ck,λ	200	0.8	0.753404	0.209823	0.802239
	200	0.9	0.797481	0.164365	0.85
rk,λ	200	0.8	0.703255	0.166676	0.972553
	200	0.9	0.73551	0.17511	0.958729

Table 2. Comparison of Client Consumption and Client Request Rate.

In order to verify our results for using the average client request rate as a criterion, we performed a long term run over 30.000 service periods with parameters ($\lambda = 200$, $\tau = 0.8$). In this experiment, too, the system behavior remained stable, with an average utilization of 0.75 and average Quality of Service of 0.97.

A question that requires further investigation is the dependency between the optimal safety margin τ and the variance of the average request rate and the average consumption rate. This quantities obviously depend on each other and a quantitative approximation of this relationship would be very useful. First experiments on relating the threshold value with the variance of the average request rate and the average consumption rate showed the following behavior: Using the variance of the average consumption rate as a safety margin for the admission criterion the system turned immediately unstable. Using the variance of the average request rate as a safety margin for the admission criterion showed acceptable, but still less stable behavior than in using predetermined values of τ. An alternative possibility would be to dynamically adapt the safety margin, i.e., to increase it when the quality decreases and, to decrease it when the server utilization decreases.

We also compared our admission control framework with a worst case based admission control. For that purpose we used a client that generates only few, but data intensive requests, and then remains idle for longer periods. In the configuration for this experiment the average request size is 4200, the average idle period 30, the average request period 3, and average total duration 200. Our admission control mechanism based on the average request rate achieved a utilization value of 0.67 and a QoS value of 0.92. A pessimistic admission strategy that admits only that many clients that can be served if all of them issue simultaneously requests, admits at most 7 clients at a time, since $7 * 4200 = 29400$ ($29400 < 30000$). This strategy achieved a utilization value of 0.09 (!) and QoS value of 0.93. Note that, due to random variations in the request size, overloads are still possible with the pessimistic

approach. Our strategy achieved a more than seven times higher utilization with the same QoS.

4. Conclusions

In this paper, a framework for an admission control mechanism for applications with variable data rates is introduced that is based on two steps: the admission of pending clients and the scheduling of the single requests of admitted clients. The admission of pending clients is based on the past client behavior. Various policies for the scheduling of single requests are used to balance the server load. Simulations show the benefit of the admission control framework for applications with variable data consumption with respect to QoS and server utilization. The experimental results can be summarized as follows:

- Using average server utilization and average server load are no adequate criteria.
- EDF achieves a substantial performance gain as compared to FCFS.
- Very short lookbacks do not work reliably.
- In general, the average client request rate behaves more stable in the long run and achieves comparably better performance than average client consumption.
- For uneven load patterns in the number of requesting clients the lookback should be at least over a whole period of load variation.

Currently, we are developing a distributed implementation of the admission control framework in Java that will be used both for evaluation purposes and for the implementation of the admission control module in a multimedia server.

Future work will include the evaluation of our approach within a multimedia database system. We also expect to obtain data on consumption rates of real world applications which we can use to further evaluate and to refine the admission control framework. However, we do not expect qualitative changes in its behavior since, in the experiments we performed, the admission control mechanism adapted very well to quite different types of load patterns. For the admission and scheduling method we see two important questions that need to be addressed. First, the dynamic adaptation of the safety margin values and, second, the use of dynamic QoS adaptations in the scheduling strategy.

References

[1] Susanne Boll, Wolfgang Klas, and Michael Löhr: Integrated Database Services for Multimedia Presentations. In: S. M. Chung (Ed), Multimedia Information Storage and Management, Kluwer Academic Publishers, 1996.

[2] Huang-Jen Chen, Anand Krishnamurthy, Thomas D. C. Little, and Dinesh Venkatesch: A Scalable Video-on-Demand Service for the Provision of VCR-Like Functions. In: Proc. 2nd Int. Conf. on Multimedia Computing and Systems, 1995.

[3] Ming-Syan Chen, Dilip D. Kandlur, and Philip S. Yu: Support for fully interactive Playout in a disk-array-based Video Server. In: ACM Multimedia 10/1994.

[4] Asit Dan, Daniel M. Dias, Rajat Mukherjee, Dinkar Sitaram, and Renu Tewari: Buffering and Caching in Large-Scale Video Servers. In COMPCON' 95: Technologies for the Information Superhighway, 1995.
[5] Jayanta K. Dey-Sircar, James D. Salehi, James F. Kurose, and Don Towsley: Providing VCR Capabilities in Large-Scale Video Servers. In: Proc. of ACM Multimedia, 1994.
[6] Craig S. Freedman and David J. DeWitt: The SPIFFI Scalable Video-on-Demand System. In: M. Carey and D. Schneider (Eds.): In: Proc. of Int. Conference on Management of Data (ACM SIGMOD), 1995.
[7] Sreenivas Gollapudi and Aidong Zhang: NetMedia: A Client-Server Distributed Multimedia Environment. In: Proc. of 3rd Int. Workshop on Multimedia Database Management Systems, 1996.
[8] Silvia Hollfelder: Admission Control for Multimedia Applications in Client-Pull Architectures. In Proc. of 3rd Int. Workshop on Multimedia Systems (MIS), 1997.
[9] Silvia Hollfelder, Achim Kraiss, and Thomas C. Rakow: A Client-Controlled Adaptation Framework for Multimedia Database Systems. In Proc. of Europ. Workshop on Interactive Distributed Multimedia Systems and Telecommunication Services (IDMS), 1997.
[10] Rune Hjelsvold, Roger Midtstraum, and Olav Sandstå: Searching and Browsing a Shared Video Database. In: Kingsley C. Nwosu, Bhavani Thuraisingham, and P. Bruce Berra (Eds), Multimedia Database Systems, Kluwer Academic Publishers, 1996.
[11] Guido Nerjes, Peter Muth, and Gerhard Weikum: Stochastic Performance Guarantees for Mixed Workloads in a Multimedia Information System. In: Proc. of the IEEE Int. Workshop on Research Issues in Data Engineering (RIDE), 1997.
[12] Banu Özden, Rajeev Rastogi, Avi Silberschatz, and P. S. Narayanan: The Fellini Multimedia Storage Server. In: S. M. Chung (Ed): Multimedia Information Storage and Management, Kluwer Academic Publishers, 1996.
[13] Seungyup Paek and Shih-Fu Chang: Video Server Retrieval Scheduling for Variable Bit Rate Scalable Video. In: Proc. of the IEEE Int. Conference on Multimedia Computing and Systems, 1996.
[14] Narasimha Reddy: Improving Latency in Interactive Video Server. In: Proc. of SPIE Multimedia Computing and Networking Conference, 1997.
[15] Siram S. Roa, Harrick M. Vin, and Asis Tarafdar: Comparative Evaluation of Server-push and Client-pull Architectures for Multimedia Servers. In: Nossdav 96, 1996.
[16] Prashant J. Shenoy and Harrick M. Vin: Efficient support for Scan Operations in Video Servers. In: Proc. of the Third ACM Conference on Multimedia, 1995.
[17] Heiko Thimm and Wolfgang Klas: Delta-Sets for Optimized Reactive Adaptive Playout Management. In Proc. 12th Int. Conference On Data Engineering (ICDE), 1996.
[18] Heiko Thimm, Wolfgang Klas, Crispin Cowan, Jonathan Walpole, and Calton Pu: Optimization of Adaptive Data-Flows for Competing Multimedia Presentational Database Sessions. In Proc. of IEEE Int. Conference on Multimedia Computing and Systems, 1997.
[19] Harrick M. Vin, Pawan Goyal, Alok Goyal, and Anshuman Goyal: A Statistical Admission Control Algorithm for Multimedia Servers. In: Proc. of the ACM Multimedia, 1994.
[20] Harrick M. Vin, Alok Goyal, and Pawan Goyal: Algorithms for Designing Large-Scale Multimedia Servers. In: Computer Communications, 1995.
[21] Stephen Wolfram: The Mathematica book, 3rd ed., Wolfram Media/Cambridge University Press, 1996.