Multimedia Systems (2002) 8: 295–314

Multimedia Systems

JINSIL: A middleware for presentation of composite multimedia objects in a distributed environment

Junehwa Song[1], **Asit Dan**[2], **Dinkar Sitaram**[3]

[1] EECS, Korea Advanced Institute of Science and Technology, 373-1, Kusong-dong, Yusong-gu, Taejeon, 305-701 Korea
e-mail: junesong@cs.kaist.ac.kr
[2] IBM T.J. Watson Research Center, Yorktown Heights, NY 10598, USA
e-mail: asit@us.ibm.com
[3] Andiamo Systems, India; e-mail: dsitaram@andiamo.com

Abstract. In a distributed environment, the presentation of structured, composite multimedia information poses new challenges in dealing with variable bandwidth (BW) requirements and synchronization of media data objects. The detailed knowledge of BW requirements obtained by analyzing document structure can be used for efficient utilization of system resources. A distributed multimedia environment consists of various system components that are either dedicated to a client (e.g., client buffer space and BW) or shared across multiple clients (e.g., server buffer space and BW). A shared server could benefit from fine granularity advanced reservation of resources based on true BW requirements. Prefetching by utilizing advance knowledge of BW requirements can further improve resource utilization. The prefetch schedule needs also to be aware of the BW fragmentation in a *partitioned server*. In this paper, we describe the JINSIL middleware for retrieval of a composite document that takes into account the available BW and buffer resources and the nature of sharing in each component on delivery paths. It reshapes BW requirements, creates prefetch schedules for efficient resource utilization, and reserves necessary BW and buffer space. We also consider good choices for placement of prefetch buffers across client and server nodes.

1 Introduction

The rapid evolution of multimedia technologies has made feasible new ways of creating and presenting complex multimedia documents. Such documents consist of media objects of various types and granularities that are organized into meaningful chunks; for example, a slide presentation or a story consisting of multiple (and even simultaneous) images, text data, as well as video and audio clips [10, 15, 20, 24]. Two examples of such composite presentations and resulting variations in the data consumption rates are shown in Figs. 1 and 2. The first example is a report of a swimming competition where, in addition to the global view, small video windows containing close-ups of the leaders are shown toward the end of the report. The second example mingles audio, images and narration of the interesting sights around Washington, DC (details are provided later).

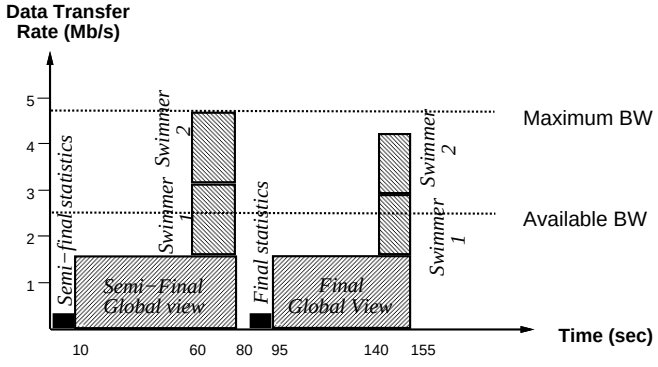


Fig. 1. Olympic swimming competition

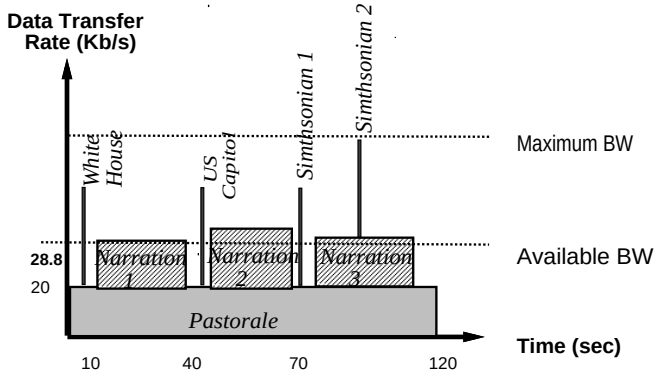


Fig. 2. Tour of Washington, DC

An efficient presentation of such structured, composite multimedia information (i.e., retrieval and synchronous playback) gives rise to new challenges. In a distributed multimedia environment, some or all pieces of a composite presentation object may reside in one or multiple remote systems away from client presentation systems (see Fig. 3). To avoid jitter in a presentation, appropriate resources need to be reserved on various data paths from the respective sources to the client systems [2, 3, 6, 8, 11, 12]. For a composite document, the instantaneous total data consumption rate that needs to be supported will vary over time depending on the structure of the presentation [16, 17]. Table 1 shows the data consumption rates for the above two presentation examples.

Table 1. Presentation schedules: **a** Olympic competition; **b** tour of Washington, DC

Object id	Start time	Duration	Data rate
image 1	0	10	30 Kb
narration 1	3	6	20 Kb/sec
video 1	10	70	1.6 Mb/sec
video 2	60	20	1.5 Mb/sec
video 3	60	20	1.6 Mb/sec
image 2	85	10	28 Kb
narration 2	87	6	24 Kb/sec
video 4	95	60	1.5 Mb/sec
video 5	140	15	1.4 Mb/sec
video 6	140	15	1.4 Mb/sec

Object id	Start time	Duration	Data rate
White house	5	30	30 Kb
US Capitol	40	30	28 Kb
Smithsonian 1	75	20	29 Kb
Smithsonian 2	95	15	30 Kb
Narration 1	10	22	15 Kb/sec
Narration 2	43	20	25 Kb/sec
Narration 3	78	24	16 Kb/sec
Pastorale	0	120	20 Kb/sec

The variable data rates make the task of efficient allocation of resources in a shared component more complex.

One simple resource allocation policy for applications with variable bandwidth (BW) requirements is to reserve a constant BW (equal to the maximum instantaneous BW required by the presentation) for the entire duration of the presentation. However, there are several disadvantages to this simple approach. First, in many commercial environments (e.g., cable or phone connection to home) the BW is limited at the final stage of the network. This is referred to as the "last mile problem". Hence, it may be impossible or difficult to support presentations of complex media documents that require multiple streams of video, audio, image or text data (even for a short duration). Higher up the data delivery paths (e.g., in the server), the BW may be shared across multiple presentations. Allocating the peak BW reduces the number of presentations that can be admitted.

In this paper, we describe the scheduling and retrieval policies of the *JINSIL* retrieval system that, in a distributed multimedia environment, allocates appropriate resources and creates an object delivery schedule in each component on data delivery paths. The *resource allocation* and creation of a *prefetch schedule* in JINSIL address various issues, including obtaining a document's variable BW requirement by analyzing the document's structure, determining delivery paths based on the locations of media objects, dedicated vs. shared resources on these paths, and available buffer and BW resources. Note that BW variability in a presentation can come from two sources: variations in compression ratio of a stream [9, 19] and composition of different multimedia objects [16–18, 21–23]. The JINSIL system deals with the second case, and can use the solutions proposed in [9, 19] for dealing with changing compression ratios.

The remainder of the paper is organized as follows. Section 2 describes typical workloads, and a brief overview of the JINSIL system. Section 3 introduces the general problem of BW and buffer satisfiability for a given presentation with a variable BW requirement and describes the scheduling and retrieval policy used by the JINSIL system. In Sect. 4, we extend the problem to the case in which media objects are stored in multiple remote storage systems and describe JINSIL's policy in that case. A performance study of the proposed algorithms using simulation is presented in Sect. 5. Finally, Sect. 6 delineates the contributions of this paper and describes its relationship to earlier work. Finally, conclusion of the paper is presented in Sect. 7.

2 Distributed client–server multimedia environment

In this section, we present two motivating examples of composite documents and describe the requirements imposed on underlying systems. We then describe the components of JINSIL, the required data structures, and, finally, the operation of the system.

2.1 Motivating examples

A composite multimedia document consists of a mixture of media types (text, audio, video, image, etc.) which are to be presented in some prespecified temporal relationship. This relationship can be described in an *object map* (described in detail in Sect. 2.2). To illustrate the concept of an object map, we describe the object maps for the Olympic swimming competition and the tour of Washington, DC below.

Olympic swimming competition. This example shows the highlights of Olympic swimming competitions. It is composed of seven atomic media objects as shown in Table 1a. Image 1 presents the statistics of the semi-final and the competitors in the game. Along with Image 1, a narration (Narration 1) describes the statistics. After the image and the narration, Video 1 is played to show the global view of the competition. Near the end of Video 1, two small screens are opened in the middle of the monitor, to show close-up views of the two best competitors (Video 2 and Video 3). The playback of Video 2 and Video 3 overlaps with that of Video 1. At the end of the game, the last frame of Video 2, which shows the winner of the competition, fills the whole monitor and stays till the playback of the final competition. The final competition is similarly presented by Image 2, Narration 2, Videos 4, 5 and 6, etc. Figure 1 shows the variability in the data consumption rates of the presentation over time.

DC tour. The second example provides a guided tour of Washington, DC. It shows three popular places in DC – the White House, the US Capitol, and the Smithsonian Mall. The playback starts with some background music, Beethoven's Pastorale, which is played till the end of the presentation. The first image shows the White House and also gives a narrative description. After some moments, the second image (of the Capitol building) is provided, along with its corresponding narration. Soon after, the third and fourth images (of the Smithsonian Mall) are presented one after the other, along with a narration for both. Figure 2 shows the variability in data consumption rates during this presentation.

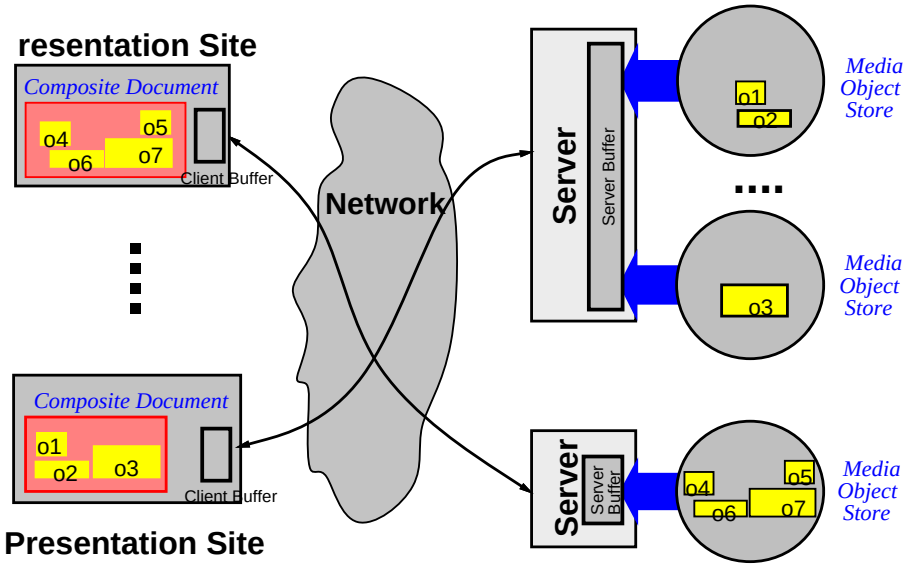


Fig. 3. A distributed multimedia environment

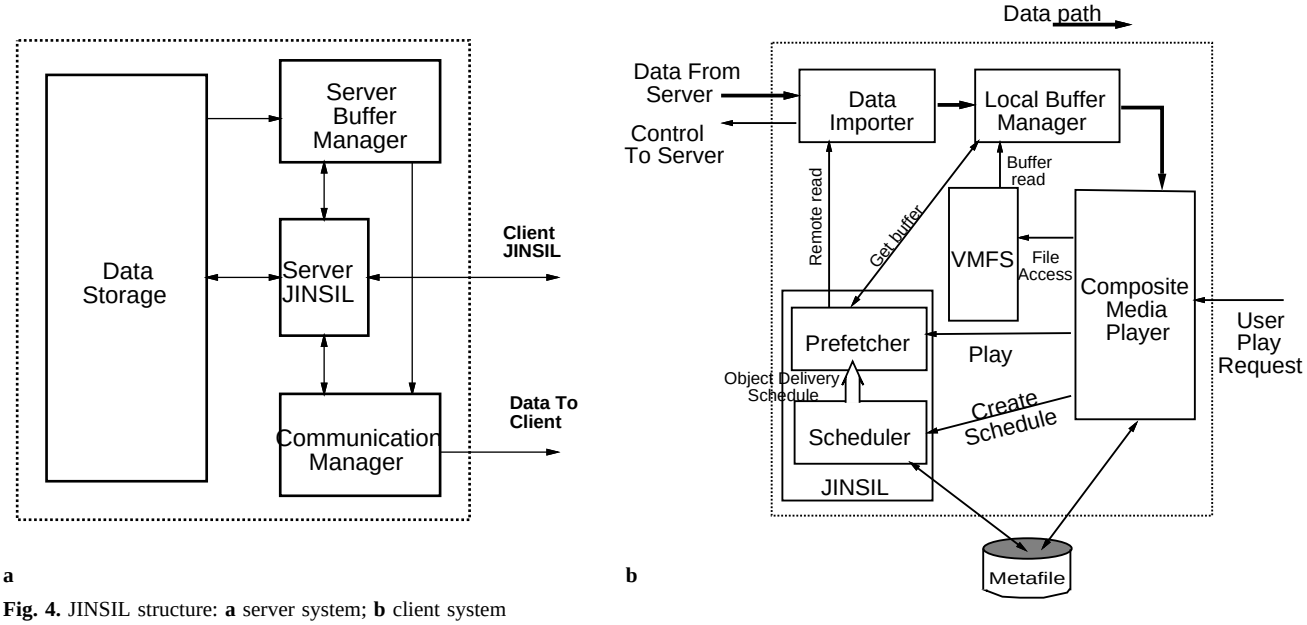


Fig. 4. JINSIL structure: **a** server system; **b** client system

2.2 JINSIL middleware

The JINSIL middleware provides services for creating efficient object delivery schedules based on presentation structure, and for the allocation of appropriate resources. Figure 4 shows the interactions in the JINSIL layer in a client and a server system. The authoring system stores the structure of a presentation in a metafile [15, 24]. Upon receiving a presentation request from a user, the *composite media player* invokes JINSIL to retrieve the required multimedia object. This request is received by the JINSIL scheduler. The JINSIL scheduler then generates an *initial object delivery schedule* and a *bandwidth requirement profile*. This initial object delivery schedule and BW requirement profile reflect the data consumption requirement of the media player. Then, the JINSIL scheduler in a client system uses this information to test the feasibility of the presentation with the available client buffer and BW. If the presentation is feasible, the scheduler reserves the required resources and constructs a new *object delivery schedule* which will be used for a data delivery request to a server. An object delivery schedule specifies a schedule for retrieving the atomic objects, and may include prefetching of the atomic objects, delay in starting the presentation, or delay of the atomic objects. The delay is estimated as the minimum required delay to avoid rejection of the presentation request due to insufficient resources. The new object delivery schedule is given to the server and is used, in turn, by the scheduler at the server, to generate a new BW requirement profile and, if feasible, to reserve appropriate resources. Note that in a shared or partitioned system component (e.g., server), the scheduler will trade off BW and buffer space to maximize the throughput of the component. Details of the algorithms for feasibility

Table 2. An object map holds information of each object stored in the server storage system including object id, type, size, etc.

ObjId	Type	Rate	Start time	Duration	Object location
X_1	JPEG	k_1 Mbps	t_1 sec	1 sec	//tj.watson.ibm.com/a
X_2	MPEG1	k_2 Mbps	t_2 sec	30 sec	//andromeda.watson.ibm.com/b
...	...	...	...	...	
X_N	MPEG2	k_n Mbps	$t_n$sec	5 sec	//foraker.watson.ibm.com/c

test and resource allocation will be described in Sects. 3 and 4.

Once the processes of resource allocation and creation of object delivery schedules are successful, the composite media player starts playing back the presentation. The composite media player starts accessing a virtual media file system (VMFS). The data is inserted into the file system buffer by the local or remote JINSIL prefetcher, depending upon whether the delivery mode is push or pull.[1]

2.2.1 JINSIL resource management tables

JINSIL maintains tables to keep track of the structure of composite objects, the schedules for retrieval or delivery of the objects, the resulting BW requirements, and the availability of resources (BW and buffer).

Object map. This defines the structure of a composite presentation. Note that a composite object may be created from atomic objects or other composite objects. For each component object, the map contains the object id, its type, data rate, relative start time for display, duration, size and location. This is illustrated in Table 2.

Object delivery schedule (ODS). For each stage on a data delivery path, the system maintains an object delivery schedule required from that component to the component downstream. Additionally, each stage also generates a modified object delivery schedule representing the schedule required by the next upstream stage. For example, a client system has the initial object delivery schedule used between itself and a composite media player. It generates a new object delivery schedule needed by a server system. An example object delivery schedule is shown in Table 3. The first row contains the total reserved bandwidth in each time interval. Succeeding rows contain the scheduled data transfer rates in each time interval for each component object.

Bandwidth requirement profile. A bandwidth requirement profile represents the instantaneous BW requirement resulting from an object delivery schedule. Formally, a BW profile is a list C,

$$C = \langle (t_1^c, C_1), \ldots, (t_L^c, C_L), (t_{L+1}^c, 0) \rangle,$$

where the data consumption rate between two time points t_l^c and t_{l+1}^c is C_l. Each time point t_l^c where the consumption rate changes is referred to as a *breakpoint*. [2]

[1] The difference between the two modes is that, between any two stages, the object delivery schedule is used either by the source prefetcher to deliver or by the destination prefetcher to prefetch.

[2] We use a lower-case letter to represent an arbitrary index and an upper case letter an end point. For example, t_L^c represents the last breakpoint where the data consumption rate is not zero and t_{L+1}^c is used to represent the last breakpoint where the data consumption rate becomes zero. t_l^c and t_{l+1}^c designate breakpoints such that $t_1^c \le t_l^c \le t_{l+1}^c \le t_{L+1}^c$.

Table 3. Object delivery schedule

Time	15	25	30 (Mbps)	34	40
Reserved BW	0.4	1.4	0.5	0.3	0.4
$object_1$	0.4	0.6	–	–	–
$object_2$	–	0.8	0.5	0.3	0.4

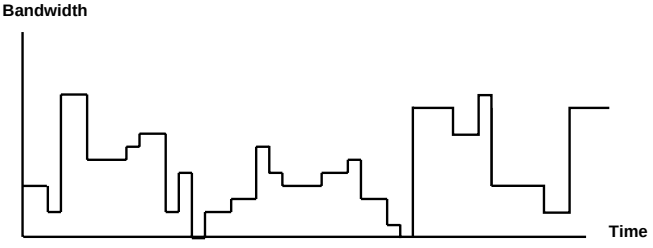


Fig. 5. Time-dependent bandwidth availability

Bandwidth and buffer availability. Each system component maintains time-dependent lists of available BW and buffer (see Fig. 5). Bandwidth availability is represented by a list $\lambda = \langle (t_1^\lambda, \lambda_1), \ldots, (t_M^\lambda, \lambda_M), (t_{M+1}^\lambda, 0) \rangle$, where the available BW between two time points t_m^λ and t_{m+1}^λ is λ_m. Similarly, buffer space availability is represented by a list $B = \langle (t_1^b, B_1), \ldots, (t_N^b, B_N), (t_{N+1}^b, 0) \rangle$.

Scheduling for a presentation of a composite multimedia object is initiated upon the user's request for playback of that object. In generating an initial object delivery schedule, JINSIL simulates static data types as continuous data types. Consider an image $image_i$ of size L KB. Let the start time of the $image_i$ be s_i, and the play duration d_i. The entire image data of L KB should be available by s_i and is consumed at once. JINSIL simulates it as a continuous media object by giving a pseudo start time as $s_i - \delta$ and a pseudo duration δ with rate L/δ for a small positive number δ.

3 Resource allocation scheme

In this section, we first introduce the notions of *fine granularity advanced reservation (FGAR)* and *generalized bandwidth and buffer constrained scheduling (GBBCS)*. Subsequently, we describe the scheduling algorithms in a step-by-step manner.

To maximize resource utilization in a shared component, it is necessary to take into account the variability in resource availability and resource requirements. To guarantee continuous delivery of data, the FGAR policy reserves a different amount of resources for different time periods rather than a

```
GBBCS(C, λ, B) {
    FeasibilityWithMinimumPrefetching(C, λ, B);
    if not schedulable,
        min_delay = ComputeMinDelay (C, B, λ);
        C_delay = DelayReq (C, delay);
    else C_delay = C;
    C_reshape = RsvMinPeakBW (C_delay, B, λ));
    return(min_delay,C_reshape,λ, B);
}
```

Fig. 6. Generalized bandwidth buffer constrained scheduling

```
FeasibilityWithMinimumPrefetching(C', λ',B') {
  B°_P = 0;
  For p = P − 1, . . . , 1
    { if (C'_p + B°_{p+1}/(t_{p+1} − t_p)) ≤ λ'_p,
        {B°_p = 0;  λ°_p = C'_p + B°_{p+1}/(t_{p+1} − t_p);}
    else { λ°_p = λ'_p;
        B°_p = B°_{p+1} + (t_{p+1} − t_p)(C'_p − λ'_p);
        if B°_p > B'_p, return (p,B°_p); } /* overflow at t_p */
    }
    return(0,B°_1);  /* if B°_1 == 0, request is feasible,
  }                     else initial prefetching is required */
```

Fig. 7. Algorithm for testing feasibility

constant (i.e., the maximum) amount throughout the whole session.

If a fixed amount of BW and buffer space are dedicated to a single client, a scheduler can test the feasibility of a requested presentation by using the algorithm proposed in [17]. However, the availability of resources will be time-dependent in the system components shared by multiple clients. In such environments, a scheduling policy needs to reflect the time-dependent availability of the shared resources. The proposed GBBCS policy takes into account both the BW requirement profile and the time-dependent availability of resources in creating a delivery schedule for the atomic objects in a presentation.

3.1 Overall structure of GBBCS

Figure 6 shows the broad steps of the GBBCS policy used *by client and server systems.* The parameters of the GBBCS policy are the earlier defined BW requirement profile (C), and the availability lists of bandwidth (λ) and buffer (B) of an invoking system component. The GBBCS policy first tests the feasibility of a presentation for the given C, λ and B. If the presentation is not feasible, it then checks for the feasibility with a presentation delay. The *ComputeMinDelay* step is invoked to compute the minimum time by which the request has to be delayed for resource availability. Depending on the buffer availability, the presentation delay is used either to prefetch a sufficient amount of data for a jitter-free presentation, or to wait until enough BW and buffer space are available. The GBBCS policy then modifies the BW requirement profile as C_{delay} by shifting the BW profile C by the required amount of delay. Once the presentation is feasible, the step *RsvMinPeakBW* is invoked to generate a reshaped BW requirement profile, $C_{reshape}$, for the current presentation request. The reshaped BW requirement profile uses prefetching whenever possible, to minimize the peak BW required by the request C_{delay}. The above procedure *RsvMinPeakBW* also reserves the required bandwidth and buffer space, and updates the availability lists λ and B. The details of these steps are given in Sects. 3.2–3.5.

Note that in a dedicated system component where λ and B are constant values, the smoothing algorithm proposed in [19] can also be used to create $C_{reshape}$. The algorithm in [19] has an additional objective of minimizing the variance in the reshaped BW requirement profile.

The GBBCS policy has additional advantages. First, as shown later in Sect. 5, a variable BW requirement may lead to periodic exhaustion in server BW. This also results in *a waste* of server resources, since a request cannot be scheduled until enough resources are available, and the available resources remain idle. By reducing the variability in BW requirements, the GBBCS policy reduces this waste of server resources. Also, the GBBCS policy is run on both client and server systems. Note that even when enough resources are available in a client system, the reshaping of BW requirement profiles by JINSIL reduces fluctuations seen by the server system.

The remainder of this section describes in more detail the steps of the GBBCS. We first set up a fundamental equation which describes the relationships among the time-dependent resource availability and the requests in Sect. 3.2. This relationship forms the basis for all the steps used in the JINSIL scheduler. In Sect. 3.3 we describe an algorithm for testing the schedulability of a request. Then, in Sects. 3.4 and 3.5, we describe the steps for computing the minimum delay and those for the minimum peak BW reservation.

3.2 Relationship amongst consumption rate, reserved bandwidth and buffer

Let T^C, T^λ and T^B be the set of breakpoints in a BW requirement profile, a BW and a buffer availability, respectively. Also, let $T^C = \{t^c_l, l = 1, \ldots, L\}$, $T^\lambda = \{t^\lambda_m, m = 1, \ldots, M\}$, and $T^B = \{t^b_n, n = 1, \ldots, N\}$. To obtain the relationships among C, B, and λ in a feasible allocation, we consider the combined set of the breakpoints $T = T^C \cup T^\lambda \cup T^B$. Let P be the cardinality of the set T, that is, the total number of breakpoints in T. We redefine the data consumption rate, the available BW and buffer space with respect to the combined set of breakpoints T, by C', λ' and B', respectively, where $C' = \langle (t_1, C'_1), \ldots, (t_P, C'_P), (t_{P+1}, 0)\rangle$, $\lambda' = \langle (t_1, \lambda'_1), \ldots, (t_P, \lambda'_P), (t_{P+1}, 0)\rangle$ and $B' = \langle (t_1, B'_1), \ldots, (t_P, B'_P), (t_{P+1}, 0)\rangle$. For multimedia presentations with continuous data consumption, the data should be delivered without causing any buffer underflow or overflow. Let λ°_p be the data delivery rate between the breakpoints t_p and t_{p+1}. Also, let B°_p and B°_{p+1} be the amount of prefetched data at breakpoints t_p and t_{p+1}, respectively. The relationship among C'_p, λ°_p, B°_p, and B°_{p+1} can be characterized by

$$B^\circ_{p+1} = B^\circ_p - (t_{p+1} - t_p)(C'_p - \lambda^\circ_p), \quad \text{where}$$
$$0 \le \lambda^\circ_p \le \lambda'_p \quad \text{and} \quad 0 \le B^\circ_p \le B'_p. \tag{1}$$

The second term in the above equation represents the net data reduction (or accumulation) from the buffer during this interval. Any allocation of BW and buffer (i.e., λ° and B°) satisfying the above equation is referred to as a feasible allocation.

3.3 Testing for schedulability of a request

Given a BW and a buffer availability (λ' and B'), the algorithm in Fig. 7 tests for the feasibility of a retrieval request with a BW requirement profile (C'). (Note that there always exists a feasible solution if the available BW or buffer space is unlimited, whereas the case we treat here is that the BW and buffer space are constrained possibly in a time-dependent fashion.) The algorithm identifies the feasibility in linear time ($O(P)$) if and only if there exists a feasible allocation. If feasible, the algorithm also computes the minimum amount of data that needs to be prefetched at each breakpoint. Otherwise, it returns the breakpoint at which the feasibility condition (equation (1)) is violated. (This breakpoint (t_p) and the prefetching amount (B_p°) would be used to compute the minimum delay to remove the identified overflow as described in Sect. 3.4.) The algorithm steps through all breakpoints, starting at the last (t_P). At each iteration, it computes the minimum amount of data that needs to be prefetched at each breakpoint t_p to satisfy the data consumption rate during the interval $\langle t_p, t_{p+1}\rangle$. This is done using equation (2) as described in the next paragraph. If the required prefetching amount is greater than the available buffer size, the presentation is not feasible under the currently available BW and buffer space. The procedure terminates at this point and returns the index of this breakpoint (i.e., at which a buffer overflow occurs). The normal termination of the algorithm implies a feasible schedule has been found. If the initial prefetching amount (B_1°) is non-zero, the presentation has to be delayed to prefetch this amount of data.

Computation of the minimum prefetching amount. Assume that B_{p+1}^θ denotes the minimum amount of prefetching at t_{p+1} to satisfy a partial BW requirement profile $\langle (t_{p+1}, C'_{p+1}), (t_{p+2}, C'_{p+2}), \ldots, (t_P, C'_P), (t_{P+1}, 0)\rangle$. The minimum amount of prefetching required at t_p (i.e., B_p^θ) to satisfy the partial BW profile starting from t_p and the corresponding allocation of BW (i.e., λ_p^θ) can be computed by the following equation:

$$\begin{aligned} B_p^\theta = 0 \quad \text{and} \quad \lambda_p^\theta = C'_p + \frac{B_{p+1}^\theta}{t_{p+1}-t_p}, \\ \text{if } C'_p + \frac{B_{p+1}^\theta}{t_{p+1}-t_p} \le \lambda'_p \\ B_p^\theta = B_{p+1}^\theta + (t_{p+1}-t_p)(C'_p - \lambda'_p) \\ \text{and} \quad \lambda_p^\theta = \lambda'_p, \quad \text{otherwise.} \end{aligned} \tag{2}$$

Equation (2) is derived as follows. At each breakpoint t_p, if the minimum prefetch requirement B_{p+1}^θ at the next breakpoint t_{p+1} is given, the required BW to satisfy this prefetching amount is $B_{p+1}^\theta/(t_{p+1}-t_p)$. If this quantity plus the data consumption request C'_p on the current interval $\langle t_p, t_{p+1}\rangle$ is less than the available BW λ'_p, prefetching is not required at time t_p (i.e., $B_p^\theta = 0$), and the corresponding amount of bandwidth λ_p^θ is computed by equation (1). Otherwise, all the available BW needs to be allocated at time t_p for data transfer (i.e., $\lambda^\theta = \lambda'$) to minimize the prefetching amount at the current breakpoint. The corresponding amount of prefetching is also computed using equation (1).

3.4 Computation of minimum delay

Computation of delay under dedicated resources. If available BW and buffer space are fixed, it is straightforward to compute the minimum delay before a presentation can be started (min_delay). Assuming λ_{const} as the constant value of the available BW, then $min\_delay = B_1^\circ/\lambda_{const}$. Note, however, that delaying a presentation cannot remove a buffer overflow condition in such dedicated environments, since the available buffer space and BW remain constant.

Computation of delay under the FGAR policy. In a shared component with FGAR, the available resources are time-dependent. Hence, the computation of min_delay gets very complex. The algorithm to compute the minimum delay under the FGAR policy is shown in Fig. 8.

Given a BW requirement profile C, let $C(t)$ denote a shifted BW profile of C, where the start time is t. That is, $C(t)$ is the representation of C where the start time is parameterized as t. Further, let $B_1^\circ(t)$ denote the initial prefetching amount for a data request with the start time t, that is, the initial prefetching amount when $C(t)$ is applied. Similarly, $C(t+s)$ represents the BW profile after shifting $C(t)$ by s time units, and $B_1^\circ(t+s)$ represents the initial prefetching amount after such a shift. Finally, let $dB_1^\circ(t)/dt$ represent the rate of decrease in the required initial prefetch amount. The rate of decrease, $dB_1^\circ(t)/dt$, can be estimated as

$$\frac{dB_1^\circ(t)}{dt} = \frac{B_1^\circ(t+\delta) - B_1^\circ(t)}{\delta}, \tag{3}$$

where δ is a small positive number. Here the required prefetch amounts, $B_1^\circ(t)$ and $B_1^\circ(t+\delta)$, are computed using the procedure *FeasibilityWithMinimumPrefetching*. Now, assume a delay of Δt. With $dB_1^\circ(t)/dt$ computed as above, the initial prefetching amount after the delay (i.e., $B_1^\circ(t+\Delta t)$) can be estimated by

$$B_1^\circ(t+\Delta t) = B_1^\circ(t) + \frac{dB_1^\circ(t)}{dt}\Delta t. \tag{4}$$

Apart from equation (4), a separate condition on $B_1^\circ(t+\Delta t)$ is required in relation to the initially available bandwidth λ'_1. That is, this required prefetch amount of data should possibly be fetched for Δt time units using the initially available bandwidth λ'_1. In other words,

$$B_1^\circ(t+\Delta t) = \Delta t \cdot \lambda'_1. \tag{5}$$

Now, by relating equations (4) and (5), the required delay is computed by

$$\Delta t = \frac{B_1^\circ(t)}{\lambda'_1 - dB_1^\circ(t)/dt}. \tag{6}$$

Note that a time shift in $C(t)$ by Δt may result in a change in the relative orders of breakpoints (illustrated in Fig. 9). Once the relative order is changed, the dB_1°/dt computed by equation (3), and therefore the delay computation

MinInitialDelayUnderFGAR(t, C, λ, B) {
/* $C(t)$ denotes a shifted version of C, where the starting time is t, i.e., $t_1^{c(t)} = t$ */
[1] construct C', B', λ' over the combined breakpoints with $C(t), B, \lambda$;
compute $B_1^\circ(t)$ by *FeasibilityWithMinimumPrefetching*(C', λ', B');
[2] construct C', B', λ' over the combined breakpoints with $C(t+\delta_1), B, \lambda$;
compute $B_1^\circ(t+\delta_1)$ by *FeasibilityWithMinimumPrefetching*(C', λ', B');
/* δ_1 is a small positive number */
[3] $\dfrac{dB_1^\circ(t)}{dt} = \dfrac{B_1^\circ(t+\delta) - B_1^\circ(t)}{\delta_1}$;
[4] $\Delta t = \dfrac{B_1^\circ(t)}{\lambda_1' - dB_1^\circ(t)/dt}$;
[5] $\min T = \min\{t_i - t_j^c \mid t_i \in T^\lambda \cup T^B, t_j^{c(t)} \in T^{C(t)}, t_i > t_j^{c(t)}\}$;
[6] **if** $\Delta t \le \min T$,
construct C', B', λ' over the combined breakpoints with $C(t+\Delta t), B, \lambda$;
call *FeasibilityWithMinimumPrefetching*$(C'(t+\Delta t), \lambda', B')$,
if feasible, **return**(Δt);
else return$(\Delta t + MinInitialDelayUnderFGAR(t+\Delta t, C, \lambda, B))$;
[7] **else return** $(\min T + \delta_2 + MinInitialDelayUnderFGAR(t + \min T + \delta_2, C, \lambda, B))$
/* δ_2 is a small positive number */

Fig. 8. Algorithm to compute the minimum delay under the FGAR policy

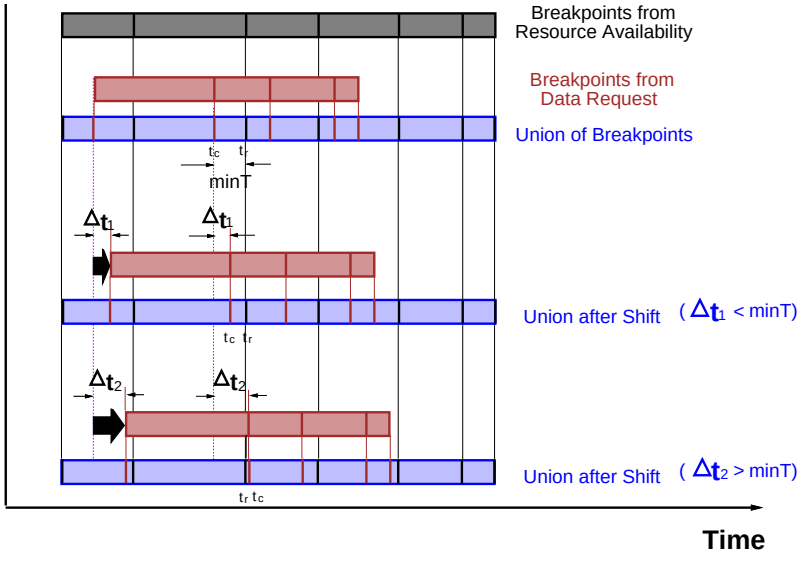


Fig. 9. Time shifts of a request and changes in breakpoints. The black stripes in the bars represent the breakpoints in a resource availability and the brown ones represent those in a BW profile. Shifting the BW profile by Δt_1 preserves the relative order among the breakpoints. However, shifting it by Δt_2 results in a change in the order between $t_r$and t_c

using equation (6), may not be valid. In Fig. 9, the stripes in the bars (solid or dotted) represent the breakpoints in resource (BW and buffer) availabilities ($T^B \cup T^\lambda$), BW profile ($T^{C(t)}$), and the union of these breakpoints before and after shifting of $C(t)$. Let $\min T$ be the maximum amount of shift which can keep the current relative order of the breakpoints, that is,

$$\min T = \min\{t_i - t_j^{c(t)} \mid t_i \in T^\lambda \cup T^B, t_j^{c(t)} \in T^{C(t)}, t_i > t_j^{c(t)}\}.$$

If $\Delta t \le \min T$ (Δt is obtained from equation (6)), shifting of $C(t)$ by Δt does not affect the relative orders of the breakpoints. Hence, Δt could be the required minimum delay. To check the satisfiability of the request $C(t)$ after a time shift of Δt, the algorithm in Fig. 7 is re-executed. If the shifted request is feasible, the computation of the minimum delay ends with Δt as the delay. Otherwise, the above steps (i.e., the computation of the minimum delay) are repeated, resulting in a further shift in the request.

If $\Delta t > \min T$, $dB_1^\circ(t)/dt$ no longer holds in the range $\langle t, t+\Delta t\rangle$. Here, the request is first shifted by $\min T + \delta'$, where δ' is a small number, and the steps for the minimum initial delay computation are repeated. The shift by $\min T + \delta'$ will result in changing the relative orders of the breakpoints. Finally, note that $dB_1^\circ(t)/dt$ in equation (4) may not always be defined, since $B_1^\circ(t+\delta')$ may be undefined.

As mentioned earlier, the computation of a delay is also required when a buffer overflow occurs at a breakpoint t_i during a feasibility test, that is, if $B_i^\circ > B_i'$. This can be done in a similar way to the computation of the initial delay. Here, a delay is introduced until the overflow is removed, that is,

$$B_i^\circ(t+\Delta t) = B_i'(t). \quad (7)$$

Therefore, from equation (7) and an equation similar to equation (4),

$$\Delta t = \frac{B_i^\circ(t) - B_i'(t)}{dB_i^\circ(t)/dt} \quad (8)$$

As before, the steps may need to be repeated if $\Delta t > \min T$.

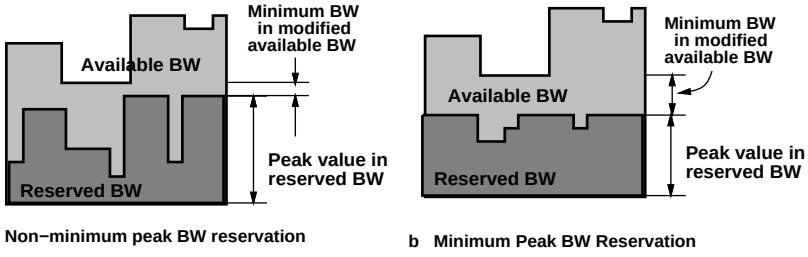


Fig. 10. Illustrations of reservations with and without minimum peak BW

3.5 Minimum peak bandwidth reservation

Once the feasibility test of a request is successful (after a possible delay), appropriate resources are reserved on behalf of this request. Equation (1) defines the set of feasible allocations. Among those, JINSIL selects an allocation where the peak value in the reserved BW (i.e., $\max_{1\le p\le P}\{\lambda^{\circ}_p\}$) is the minimum among all the feasible allocations (illustrated in Fig. 10). We call this allocation a minimum peak bandwidth reservation.

The computation of a minimum peak BW reservation is a complex process due to the time dependence of the resource availability and the BW requirement profile. We provide an iterative algorithm which computes a minimum peak BW reservation in $O(P^2)$. The algorithm is shown in Fig. 11. Initially, the algorithm computes a lower bound on the minimum peak bandwidth (λ_{lb}, detailed later) and sets the initial value of the peak bandwidth ($\widehat{\lambda}$) to this lower bound. The algorithm then iteratively computes the actual required minimum peak BW as follows. First, it tests the feasibility of the data consumption request in a similar way to *FeasibilityWithMinimumPrefetching*. During this feasibility test, the BW availability is changed to λ'_{new}, in which the available BW at each breakpoint is set to the minimum of the current $\widehat{\lambda}$ and the actual available BW. If the feasibility test is successful, the current $\widehat{\lambda}$ is taken as the required minimum peak BW. Alternatively, the failure implies a buffer overflow at a breakpoint, called an *overflow point* (OP). Hence, the algorithm computes a minimum increase in the required peak BW ($\Delta\lambda$) that will eliminate this overflow (detailed later). It subsequently increases $\widehat{\lambda}$ by $\Delta\lambda$. The algorithm then starts the next iteration, and ends when the feasibility test is successful. During each iteration, the computation avoids consideration of the breakpoints that are not affected by this increase in the peak BW. An *invariant point* or INV is defined as a breakpoint at which the required amount of data prefetching is not reduced by this change in $\widehat{\lambda}$. Hence, the computation restarts from an INV point.

Initial value of the minimum peak bandwidth. To compute the initial value of minimum peak BW, we consider the required minimum BW in each interval $\langle t_p, t_{p+1}\rangle$, assuming the maximum prefetching at t_p, that is,

$$\lambda_{lb} = \max_{0\le p<P}\{((t_{p+1}-t_p)*C'_p - B'_p)/(t_{p+1}-t_p)\}.$$

Identification of INV. The procedure *RsvMinPeakBW* keeps track of the last breakpoint (referred to as the *buffer empty point* or EP) where the required prefetch amount is zero (see Fig. 12). An EP itself is an invariant point, since the increase in the peak BW does not change the prefetch amount at an EP. The procedure *ComputeINV* determines an INV as follows. Starting from an EP, the procedure moves toward the OP to find the first breakpoint where the available BW is larger than the current peak BW (t_{n5} in Fig. 12). This is the first point at which the allocation of BW can be increased beyond the current peak BW and therefore the required prefetching amount would be affected. The INV we are after is the point where the prefetched amount remains unchanged (t_{n6} in Fig. 12).

3.5.1 Determining the minimum increase

Once a buffer overflow is detected during an iteration of the feasibility test, the current value of the minimum peak BW ($\widehat{\lambda}$) is increased. The selection of this increase $\Delta\lambda$ should meet the following two conditions. First, the increase $\Delta\lambda$ needs to be chosen small enough not to exceed the actual minimum peak BW. Second, it should be large enough for fast termination of the algorithm. By selecting $\Delta\lambda$ according to these two conditions, $\widehat{\lambda}$ will be monotonically increased and converge to the actual minimum peak BW. One obvious selection of $\Delta\lambda$ satisfying the two conditions is the minimum amount of increase which just removes the currently identified overflow. Let us refer to such an amount as the *overflow remove* (or *OR*) and denote it by Δ^{or}. With this selection of the minimum increase, the number of iterations required by the algorithm to find the actual minimum peak BW will be linear in the number of breakpoints since there will be at most as many overflow points as breakpoints. (This iteration means the outer-level iteration which is the iteration of the minimum increase computation, whereas the iteration mentioned below means the inner-level iteration which computes the minimum increase itself.) Unfortunately, however, the computation of the OR is not straightforward due to the interactions among the time-varying parameters over the breakpoints and may require multiple iterations. To avoid such a complication and to facilitate the computation of the minimum peak BW, we introduce an alternative quantity for the minimum increase by relaxing the OR. This selection of the minimum increase satisfies the two conditions mentioned above, guaranteeing convergence to the actual minimum peak BW. In terms of the number of (outer-level) iterations, this convergence is still guaranteed to occur in linear time. Moreover, this minimum increase can be efficiently computed in linear time (by scanning the breakpoints once). In the following, we detail the computation of the minimum increase in three steps. First, we observe the possible complication in the computation of an OR. Then we define the alternative quantity. In the last step, we detail how this al-

```
RsvMinPeakBW(λ', C', B') {
   λ̂ = max_{0≤i<P}{((t_{i+1} − t_i) * C'_i − B'_i)/(t_{i+1} − t_i)};
   MinPeakBW(λ̂, P);
   updateAvailability (λ, B, λ°, B°)
}

MinPeakBW(λ̂, l) {
    τ = 0; Δλ = ∞; ep = l;
    for p = l − 1, . . . , s {
      λ'_new = min{λ̂, λ'_p};
      λ°_p = C'_p + B°_{p+1}/(t_{p+1} − t_p);
      if (λ°_p ≤ λ'_new),                    /* case 1: buffer is emptied with the current λ̂ */
            { B°_p = 0;   ep = p; τ = 0; Δλ = ∞; }
      else if (λ°_p > λ'_new),   {           /* case 2: prefetching is required */
          λ°_p = λ'_new;
          B°_p = B°_{p+1} − (t_{p+1} − t_p) * (λ°_p − C'_p);
          if B°_p > B'_p        {            /* case 2-1: no overflow with the current λ̂ */
              if (λ̂ ≤ λ'_p),   { τ = τ + (t_{p+1} − t_p);   α = λ'_p − λ̂;  }
              else α = ∞;
              Δλ = min{Δλ, B°_p/τ, α };
              if (Δλ == B°_p/τ),   ne = p;
          }
          else,                  {            /* case 2-2: overflow with current λ̂ */
              op = p;
              if (λ̂ ≤ λ'_i),   { τ = τ + (t_{p+1} − t_p);   α = λ'_p − λ̂;  }
              else α = ∞;
              Δλ = min{Δλ, (B°_p − B'_p)/τ, α };
              if ((B°_p − B'_p)/τ == Δλ),   inv = computeINV(ep, op);      /* overflow removed */
              else inv = computeINV(ne, op);                  /* new empty point introduced */
              return (MinPeakBW(λ̂+Δλ, inv));
          }
      }
    }
    return (λ̂, λ°, B°);
}

computeINV (ep, op) {
   i = ep − 1;
   while ( i > op and λ'_i ≤ λ̂ ) i − − ;
   return (i+1);
}
```

Fig. 11. Algorithm to compute a minimum peak bandwidth reservation: the index variables ep, op, inv represent the indices of EP, OP, INV, respectively

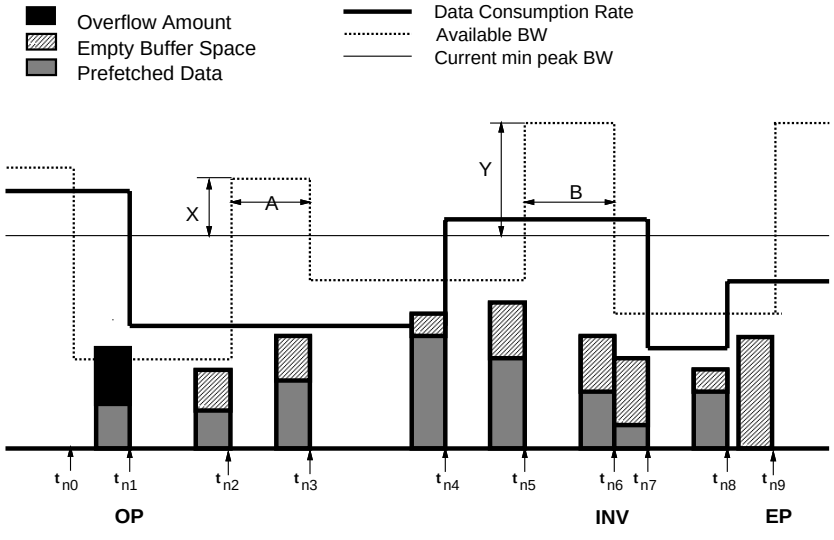


Fig. 12. Computation of the minimum increase: OP, EP and INV are overflow point, buffer empty point, and invariant point, respectively. A and B are the intervals where bandwidth allocation could be increased. X is the minimum of the differences between the current peak BW and available bandwidths

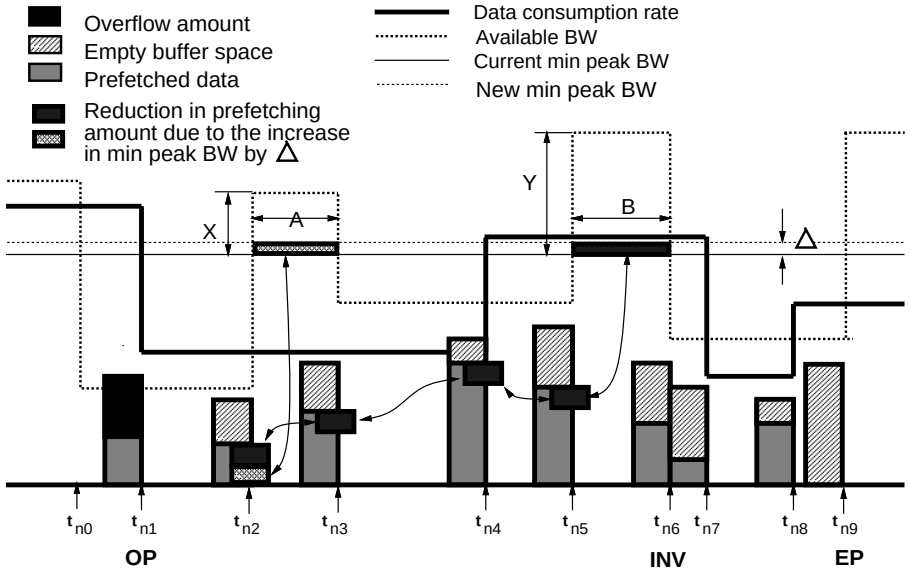


Fig. 13. Computation of the minimum increase: with the increase by Δ, more BW can be allocated in the intervals marked A and B. With the increase, the required prefetching amount is reduced at t_{n5} as marked by the overlaid rectangle. This reduction is propagated through t_{n4}, t_{n3}, and t_{n2}. At t_{n2}, additional reduction is made due to the BW increase in the interval $\langle t_{n2}, t_{n3}\rangle$, making t_{n2} a buffer empty point

ternative quantity can be easily computed. Figures 12 and 13 illustrate the notations introduced in the description.

Computation of overflow remove. Suppose that the current iteration started at t_b and identifed an overflow at t_a with the current peak BW $\widehat{\lambda}$. Let $T^+_{(x,b)}$ be the set of breakpoints, between the breakpoints t_x and t_b, where more BW can be allocated by increasing the current peak BW, that is, $T^+_{(x,b)} = \{t_p | \lambda'_p > \widehat{\lambda},\ x \le p \le b\}$. In Figs. 12 and 13, $T^+_{(n1,n6)} = \{t_{n2}, t_{n5}\}$, since at t_{n1}, t_{n3}, and t_{n4}, the available BW is already less than the current peak BW. Therefore, no more BW can be allocated by any increase of the peak BW at those breakpoints. Further, let $\tau_{(x,b)}$ be the total duration between t_x and t_{b+1} for which the additional allocation is possible. In other words, $\tau_{(x,b)}$ is the sum of the lengths of the intervals which are constructed by the breakpoints in $T^+_{(x,b)}$, that is,

$$\sum_{x \le p \le b, t_p \in T^+_{(x,b)}} (t_{p+1} - t_p).$$

In Figs. 12 and 13, A and B are the lengths of the intervals where the BW allocation can be increased between t_{n1} and t_{n6}, that is, $\tau_{(n1,n6)} = A + B = (t_{n3} - t_{n2}) + (t_{n6} - t_{n5})$.

Assuming that the increase $\Delta\lambda$ in the current peak BW is very small, the reduction ΔB°_i in the required prefetching at a breakpoint t_i, $a \le i \le b$, caused by this increase is

$$\Delta B^\circ_i = \Delta\lambda \cdot \tau_{(i,b)}. \tag{9}$$

Δ^{or} can apparently be computed using equation (9). However, equation (9) holds only under the following two conditions on $\Delta\lambda$. First, the increase of the peak BW by $\Delta\lambda$ should not introduce any buffer empty point between t_a and t_b. Second, for each interval where the BW could be increased, the allocation should be able to increase by at least $\Delta\lambda$, that is, $\lambda'_i \ge \widehat{\lambda} + \Delta\lambda$, for $t_i \in T^+_{(a,b)}$.

Relaxation of overflow remove as an alternative. To avoid this complication and simplify the computation, we define the minimum increase, $\Delta\lambda$, as follows:

$$\Delta\lambda = \min[\ \Delta^{or},\ \ \Delta^{be}_{(a,b)},\ \ \Delta^{\lambda'}_{(a,b)}\ ], \tag{10}$$

where

- $\Delta^{be}_{(x,b)}$, $a \le x \le b$, is the minimum amount of increase in the current peak BW to introduce a buffer empty point between breakpoints t_x and t_b. In Fig. 13, Δ is the exact amount required to make t_{n2} a buffer empty point. Also, no other breakpoint between t_{n1} and t_{n6} becomes a breakpoint by this amount of increase. Therefore, $\Delta^{be}_{(t_{n1},t_{n6})} = \Delta$.
- $\Delta^{\lambda'}_{(x,b)}$, $a \le x \le b$, is the minimum of the difference between the current peak BW and available BW at each breakpoint between t_x and t_b, that is,

$$\Delta^{\lambda'}_{(x,b)} = \min_{x \le p \le b, t_p \in T^+_{(x,b)}} \{\lambda'_p - \widehat{\lambda}\}.$$

  In Fig. 13, this is marked by X, that is, $\Delta^{\lambda'}_{(n1,n6)} = X$ since $X < Y$. In other words, $X = \lambda'_{n2} - \widehat{\lambda} < Y = \lambda'_{n5} - \widehat{\lambda}$.

$\Delta\lambda$ defined in equation (10) satisfies the minimality condition since it is bounded by Δ^{or}. It also guarantees the termination of the algorithm after at most $3P$ iterations (P is the number of breakpoints), since each of Δ^{or}, $\Delta^{be}_{(a,b)}$, and $\Delta^{\lambda'}_{(a,b)}$ guarantees the termination after at most P iterations. More importantly, as shown below, $\Delta\lambda$ defined in equation (10) reduces the complication in the interactions among the parameters and thus simplifies the computation. It is efficiently computed by scanning each breakpoint between t_a and t_b once.

Computation of the alternative quantity. To show how equation (10) is evaluated, we first define the following symbols.

- $\Delta^{be}_{x|(a,b)}$, $a \le x \le b$, is the minimum amount of increase in the current peak BW to make a specific breakpoint t_x a buffer empty point. Note that this is different from the previously defined $\Delta^{be}_{(x,b)}$ in that the latter is the minimum

amount of increase to make any breakpoint between t_x and t_b a buffer empty point.
- $\Delta_{(x,b)}$ is min [$\Delta^{be}_{(x,b)}$, $\Delta^{\lambda'}_{(x,b)}$], where $a+1 \le x \le b$.
- $\alpha()$ is a function such that

$$\alpha(x) = \begin{cases} x, & \text{if } x > 0 \\ \infty, & \text{otherwise.} \end{cases}$$

Using this notation, equation (10) can rewritten as:

$$\begin{aligned} \Delta\lambda &= \min[\ \Delta^{or},\ \ \alpha(\lambda'_a - \widehat{\lambda}),\ \ \min(\Delta^{be}_{(a+1,b)},\ \Delta^{\lambda'}_{(a+1,b)})\ ] \\ &= \min[\ \Delta^{or},\ \ \alpha(\lambda'_a - \widehat{\lambda}),\ \ \Delta_{(a+1,b)}\ ]. \end{aligned} \tag{11}$$

For the computation of $\Delta_{(a+1,b)}$, we set up a recurrence relation as follows:

$$\Delta_{(x,b)} = \min[\ \Delta^{be}_{x|(a,b)},\ \ \alpha(\lambda'_x - \widehat{\lambda}),\ \ \Delta_{(x+1,b)}]. \tag{12}$$

Assuming that $\Delta_{(x+1,b)}$ is available, the complication in evaluating equation (12) lies only in the computation of $\Delta^{be}_{x|(a,b)}$. However, it needs to be computed only when $\Delta_{(x+1,b)} > \Delta^{be}_{x|(a,b)}$ and $\lambda'_x - \widehat{\lambda} > \Delta^{be}_{x|(a,b)} > 0$. These conditions have the following implications. First, increasing the current peak BW by $\Delta^{be}_{x|(a,b)}$ meeting such conditions does not introduce any buffer empty point between t_{x+1} and t_b. Second, with this increase in the current peak BW, the allocation can be uniformly increased by $\Delta^{be}_{x|(a,b)}$ for each t_p in $T^+_{(x,b)}$. Therefore, the two conditions required to use equation (9) are satisfied. From the equation, the total amount of additional prefetching at t_x can be computed as $\tau_{(x,b)} * \Delta^{be}_{x|(a,b)}$. Thus, under these conditions,

$$\Delta^{be}_{x|(a,b)} = \frac{B^{\circ}_x}{\tau_{(x,b)}}.$$

Therefore, the above-formulated recurrence relation (equation (12)) can be computed by

$$\Delta_{(x,b)} = \min\left[\frac{B^{\circ}_x}{\tau_{(x,b)}},\ \ \alpha(\lambda'_x - \widehat{\lambda}),\ \ \Delta_{(x+1,b)}\right] \tag{13}$$

That is, $\Delta_{(a+1,b)}$ (= min [$\Delta^{be}_{(a+1,b)}$, $\Delta^{\lambda'}_{(a+1,b)}$]) in equation (11) can be computed in linear time by scanning the breakpoints from t_b to t_{a+1}.

Once $\Delta_{(a+1,b)}$ is available, $\Delta\lambda$ in equation (11) can also be computed in a similar fashion to equations (12) and (13). The complication in this case comes from the computation of Δ^{or}. Again, it needs to be computed only when $\Delta_{(a+1,b)} > \Delta^{or}$ and $\lambda'_a - \widehat{\lambda} > \Delta^{or} > 0$. As before, these conditions mean that increasing $\widehat{\lambda}$ by Δ^{or} does not introduce any buffer empty point and that at each breakpoint t_p in $T^+_{(a,b)}$, the increase by Δ^{or} can be fully reflected to the allocation. Thus, the resulting additional prefetching at t_a is $\tau_{(a,b)} * \Delta^{or}$. Therefore, under these conditions,

$$\Delta^{or} = \frac{(B'_a - B^{\circ}_a)}{\tau_{(a,b)}}.$$

Therefore, equation (11) can be computed by

$$\Delta\lambda = \min\left[\frac{(B^{\circ}_a - B'_a)}{\tau_{(a,b)}},\ \ \alpha(\lambda_a - \widehat{\lambda}),\ \ \Delta_{(a+1,b)}\right]. \tag{14}$$

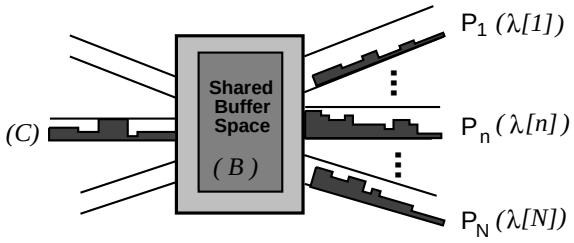


Fig. 14. Data retrieval on a partitioned system component

4 Resource allocation on a partitioned system component

In a distributed environment, multimedia objects are often stored in different data sources. For example, a server system may access an external storage (say, a tertiary data server) as well as its own storage unit. In other cases, the server storage may be composed of multiple independent devices with different BWs. For the presentation of a composite document, different atomic data objects may be retrieved from these different data sources.

Figure 14 shows a partitioned system component which is connected to multiple data paths, denoted as $P_1, P_2, \ldots, P_N$. (An example of such a system component is a partitioned server with multiple independent storage devices. A client can also be considered a partitioned component when it requires the retrieval of different media objects from multiple servers.) These different data paths may have their own resource characteristics. For example, they may have different total BW capacities. Also, in some data paths, the FGAR policy may be used for BW reservation, while in others a simple constant BW reservation policy is used. The system component may also be equipped with a buffer space which is shared by the paths. In such a partitioned system component, efficient resource allocation and data retrieval become more complex. First, separate data consumption requests need to be constructed for different data paths considering the locations of the different atomic objects. In addition, from these separate data consumption requests, separate object delivery schedules need to be constructed, and separate resource allocation needs to be made for the different paths. In doing so, all different resources in different data paths, possibly with different characteristics, need to be collectively considered along with their possible interactions. Most importantly, separate schedules thus generated should guarantee a synchronous presentation of the requested document.

In this section, we describe the resource allocation policy in JINSIL on such a partitioned system component. The resource allocation policy on a partitioned system component inherits the advantages of FGAR and GBBCS in the data retrieval on a single path system component. In addition, the difference in loads on different paths is taken into account to reduce possible *load imbalance* among different data paths. Note that highly loaded paths could result in the creation of a bottleneck of the whole system due to the synchronization requirement among the atomic media objects. In the rest of the section, we first give an overview of the GBBCS on a partitioned system component. In Sect. 4.2, we characterize the feasible resource allocations on a partitioned system component. We then describe the steps for the feasibility test

```
GBBCSOnPartitioned(C, λ, B) {
    FeasibilityOnPartitioned(C, λ, B);
    if not schedulable,
        min_delay = ComputeMinDelayOnPartitioned (C, B, λ);
        C_delay = DelayReq (C, delay);
    else C_delay = C;
    C_reshape = RsvMinBalancedPeakBW (C_delay, B, λ));
    return(min_delay,C_reshape,λ, B);
}
```

Fig. 15. GBBCS on a partitioned system component

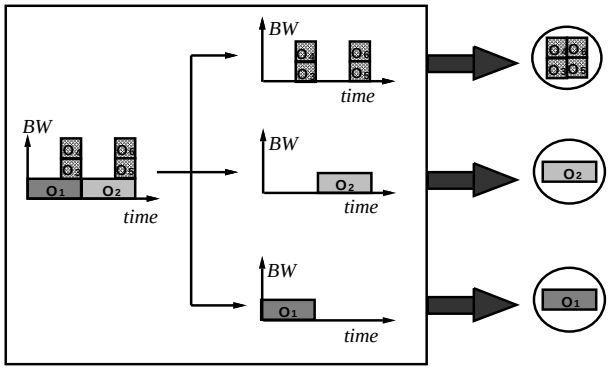


Fig. 16. Partitioning an object delivery schedule

and the minimum delay computation. Finally, in Sect. 4.3, we describe the balanced minimum BW reservation.

4.1 GBBCS on a partitioned system component

Consider again the partitioned system component in Fig. 14. The component maintains a bandwidth λ and a buffer space availability B as in a single path system component. As for the BW availability, it needs to maintain the availability separately for each path. Therefore, the BW availability λ is denoted by a list, $\lambda = \langle \lambda_{(1)}, \lambda_{(2)}, \ldots, \lambda_{(N)} \rangle$, where $\lambda_{(n)}$ represents the BW availability from a path P_n, $1 \leq n \leq N$. Note that when FGAR policy is supported for a path P_n, the BW availability $\lambda_{(n)}$ is a time-dependent list as before:

$$\lambda_{(n)} = \left[\langle t_1^{\lambda_{(n)}}, \lambda_{(n,1)} \rangle, \ldots, \langle t_{M_n+1}^{\lambda_{(n)}}, \lambda_{(n,M_n+1)} \rangle \right].$$

Upon receipt of a data consumption request (i.e., receipt of an object delivery schedule ODS from a previous system component), the resource allocation takes place as follows. The system component first generates a sequence of *partitioned object delivery schedules* $ODS_1, ODS_2, \ldots, ODS_N$ (see Fig. 16). This is done by inspecting the input ODS and the location of each atomic object. From each separate ODS_n, $1 \leq n \leq N$, a corresponding *partitioned BW requirement profile* is constructed. We represent the partitioned BW requirement profile corresponding to the path P_n by $C_{(n)}$. Then, the steps for *GBBCS on a partitioned system component* take place as shown in Fig. 15.

As shown in the figure, *GBBCS on a partitioned system component* is composed of the steps which are parallel to those on a single path system component. Each step of the algorithm is a generalization of that for the single path toward handling the parameters from multiple data paths and their interactions. In the following, we focus on the differences in the respective parallel steps of the two algorithms.

4.2 Feasibility of a presentation

Let us first set up an equation similar to equation (1) to capture the relationships amongst the time-dependent resource availability and requests. The additional difficulty here is how to incorporate the possible interactions among the resources and the requests in the different data paths.

Let T^C and T^B be the set of breakpoints from a BW requirement profile C and a buffer availability B. Similarly, let $T^{\lambda_{(n)}}$ be the breakpoints from the partitioned BW availability $\lambda_{(n)}$ of the path P_n. We take the combined set of breakpoints $T = (\cup_{1 \leq n \leq N} T^{\lambda_{(n)}}) \cup T^B \cup T^C$. Note that, unlike the case of a single path system component, all breakpoints from all partitioned BW availabilities are considered. By taking such a union, the interactions among parameters from different paths can be captured. As before, new resource availabilities and a new BW requirement profile, B', $C'_{(n)}$, and $\lambda'_{(n)}$ are defined over this combined set of breakpoints T. Also, the allocated BW and buffer space for a path P_n at a breakpoint t_p, $t_p \in T$, are denoted by $\lambda^{\circ}_{(n,p)}$ and $B^{\circ}_{(n,p)}$, respectively. Then, the relationships amongst the parameters in a feasible allocation can be characterized by

$$B^{\circ}_{(n,p+1)} = B^{\circ}_{(n,p)} - (t_{p+1} - t_p)(C'_{(n,p)} - \lambda^{\circ}_{(n,p)}), \quad \text{where}$$
$$0 \leq \lambda^{\circ}_{(n,p)} \leq \lambda'_{(n,p)}, \quad 0 \leq \sum_{n=1}^{N} B^{\circ}_{(n,p)} \leq B'_p,$$
$$1 \leq p \leq P, \text{ and } 1 \leq n \leq N. \tag{15}$$

Note that the overflow condition captured in equation (15) is different from that in equation (1). That is, in equation (15), the summation of the prefetched data for each partitioned BW requirement profile needs to be less than the available shared buffer space.

Now let us consider how the feasibility of a presentation can be tested. This is done by extending the algorithm *FeasibilityWithMinimumPrefetching*. Let $B^{\theta}_{(n,p)}$ denote the minimum amount of prefetching required at t_p for a path P_n to satisfy a partial data consumption request $[\langle t_p, C_{(n,p)} \rangle, \langle t_{p+1}, C_{(n,p+1)} \rangle, \ldots, \langle t_P, C_{(n,P)} \rangle, \langle t_{P+1}, 0 \rangle]$. Suppose that $B^{\theta}_{(n,p+1)}$, the minimum prefetching at t_{p+1}, has been computed for each n, $1 \leq n \leq N$, without identifying any overflow. Then $B^{\theta}_{(n,p)}$, the minimum prefetching at t_p, can be recursively computed without considering other paths P_m, $m \neq n$, in exactly the same way as in equation (2). This is due to the fact that $B^{\theta}_{(n,p)}$ is independent of any parameters of other paths and depends only on those of the path P_n (i.e., $B^{\theta}_{(n,p+1)}$, $\lambda'_{(n,p)}$, t_{p+1}, t_p, and $C'_{(n,p)}$). (Note that the possible interactions among different paths have been exposed by taking the union of the breakpoints over all paths in equation (15).) Therefore, the minimum amount of prefetching for the entirety of paths at a breakpoint t_p can be computed by

$$B^{\theta}_p = \sum_{n=1}^{N} B^{\theta}_{(n,p)}.$$

Consequently, the infeasibility can be identified by testing if an overflow has occurred, that is, $B_p^{\theta} > B'_p$.

Computation of the minimum delay is also done in a similar way as in a single path system component. Let $B^{\circ}_{(n,1)}(t)$ be the initial prefetching amount for a partitioned BW requirement profile $C_{(n)}$ with start time t as before. The rate of decrease in the required prefetching amount, $dB^{\circ}_{(n,1)}(t)/dt$, can be estimated as before by $(B^{\circ}_{(n,1)}(t+\delta) - B^{\circ}_{(n,1)}(t))/\delta$ for a small number δ. The required delay for the partitioned BW requirement profile $C_{(n)}$ can be computed, similarly to equation (10), by

$$\Delta t_{(n)} = \frac{B^{\circ}_{(n,1)}(t)}{\lambda'_{(n,1)} - dB^{\circ}_{(n,1)}(t)/dt}$$

Since the initial prefetching for all paths needs to be satisfied, we take $\max_{1\le n\le N}\{\Delta t_{(n)}\}$ as Δt. As in Sect. 3.4, the process continues depending on $\Delta t < \min T$.

The computation of the minimum delay to remove an overflow can also be computed in a similar way as in a single path system component. The required delay can be represented by

$$\sum_{n=1}^{N} B^{\circ}_{(n,p)}(t+\Delta t) = B'_p(t). \tag{16}$$

An equation corresponding to equation (8) can be constructed to compute Δt using equation (16).

4.3 Bandwidth reshaping and load balancing

Once a request to a partitioned system component is feasible, JINSIL reshapes each partitioned BW requirement profile and accordingly reserves the resources to further increase the system utilization.[3] In reshaping the partitioned BW requirement profiles, JINSIL notes two aspects. First, similarly to the case in a single path component, JINSIL bounds the peak value in the allocation of BW for each path. This will increase the utilization of the path. In addition, JINSIL balances the peak BW of each path with an estimate of the relative load of the path and selects the allocation which minimizes the balanced peak BW among all possible allocations. That is, JINSIL selects the allocation which minimizes

$$\max_{1\le n\le N}\left[\max_{1\le p\le P}\left\{\frac{\lambda^{\circ}_{(n,p)}}{\gamma_n}\right\}\right], \tag{17}$$

where γ_n represents a load factor of a path P_n.[4] By considering the balancing factor among different paths and minimizing the balanced peak BW, the possible imbalance of loads among different paths can be reduced. We call such an allocation a *balanced minimum peak bandwidth* reservation.

The balanced minimum peak BW reservation is again a generalization of the minimum peak BW reservation and can be iteratively computed in a similar fashion. The idea is to maintain a balanced peak bandwidth, $\widehat{\lambda}$, as a global upper bound to all the *weighted* available BWs on all the paths. (A weighted available BW is defined as the available BW divided by the load factor of a path, i.e., $\lambda'_{(n,p)}/\gamma_n$.) At each iteration, a separate peak BW, $\widehat{\lambda_{(n)}}$, for each path is decided proportional to the balanced peak BW. Upon an overflow, the minimum increase $\Delta\lambda$ is computed for the balanced peak BW, $\widehat{\lambda}$. As before, it starts by computing a lower bound, λ_{lb}, on the balanced minimum peak BW and sets the initial value of $\widehat{\lambda}$ to λ_{lb}. One such lower bound is

$$\lambda_{lb} = \max_{1\le n\le N}\max_{1\le p<P}\left[\left\{C'_{(n,p)} - \frac{B'_p}{t_{p+1}-t_p}\right\}/\gamma_n,\ \ 0\right].$$

Also, at each iteration, the test of feasibility is made in a way similar, in this case, to *FeasibilityOnPartitioned*.

The algorithm maintains a separate invariant, INV_n, for each partitioned BW requirement profile $C_{(n)}$, $1 \le n \le N$. INV_n is defined as a breakpoint where any increase in the current balanced peak value cannot reduce the required prefetching amount for $C_{(n)}$. For P_n, a buffer empty point (EP) is also defined as the breakpoint where no prefetching is required for $C_{(n)}$. As before, an EP of $C_{(n)}$ itself is an INV_n. At each iteration, the computation for each path P_n is restarted at its own invariant point.

Deciding the minimum increase. Suppose that the current iteration for each partitioned BW requirement profile $C_{(n)}$ started at a breakpoint t_{b_n} (i.e., t_{b_n+1} is an INV for $C_{(n)}$) and identified an overflow at t_a. The minimum increase in the balanced peak BW can be defined as follows:

$$\Delta\lambda = \min\left[\Delta^{or},\ \min_{1\le n\le N}\{\Delta^{be}_{(n,a,b_n)}\},\ \min_{1\le n\le N}\{\Delta^{\lambda'}_{(n,a,b_n)}\}\right], \tag{18}$$

where

- Δ^{or} is the minimum increase in the current $\widehat{\lambda}$ to remove the currently identified overflow.
- $\Delta^{be}_{(n,a,b_n)}$ is the minimum increase in the current $\widehat{\lambda}$ which introduces an EP for a path P_n between (t_a and t_{b_n}).
- $\Delta^{\lambda'}_{(n,a,b_n)}$ is the minimum difference between the current $\widehat{\lambda}$ and the weighted available BW at each breakpoint between t_a and t_{b_n} for a path P_n, that is,

$$\Delta^{\lambda'}_{(n,a,b_n)} = \min_{a\le p\le b_n, t_p\in T^{+}_{(n,a,b_n)}}\left\{\frac{\lambda'_{(n,p)}}{\gamma_n} - \widehat{\lambda}\right\}.$$

See Table 4 for other notations.

In equation (18), the minimality of $\Delta\lambda$ is guaranteed by Δ^{or} as in Sect. 3.5.1. The equation also says that an increase in the balanced peak BW by $\Delta\lambda$ either (1) removes the identified overflow, (2) advances the INV for at least one partitioned BW requirement profile (since an EP is an INV) or (3) makes $\widehat{\lambda}$ one step closer to the maximum weighted

[3] Finding an optimal load balancing strategy itself is a hard problem, which may involve the prediction of future presentation requests. The purpose of the discussion in this paper is to raise the issues of load balancing along with a pragmatic strategy rather than any optimal solution. Issues concerning optimal reshaping remain interesting research problems.

[4] There are many ways to abstract the load of a path. In JINSIL, we currently use $\gamma_n = \dfrac{\lambda_n^{cap} - \lambda_n^{peak}}{\sum_n(\lambda_n^{cap} - \lambda_n^{peak})}$, where λ_n^{cap} is the capacity of the path P_n and λ_n^{peak} is the peak value in the currently allocated BW. Therefore, a large value of γ_n represents a path with relatively high capacity or a lowly loaded path. The algorithmic framework presented in this section can be generally applied to any abstraction of loads by a constant value.

Table 4. Notation

$\Delta^{be}_{x\|(n,a,b_n)}$	the minimum increase in the current $\widehat{\lambda}$ to make t_x an EP of the path P_n, where $a \le x \le b_n$.
$T^{+}_{(n,x,b)}$	the set of breakpoints where available BW is larger than the current peak BW for $C_{(n)}$, i.e., $\{t_p \| x \le p \le b, \lambda'_p > \widehat{\lambda_{(n)}}\}$
$\tau_{(n,x,b)}$	the total duration between t_x and t_b for which the additional reservation is possible by increasing the current balanced peak BW, i.e., $\sum_{x \le p \le b, t_p \in T^{+}_{(n,x,b)}} (t_{p+1} - t_p)$
$\Delta_{(n,x,b)}$	$\min\{\ \Delta^{be}_{(n,x,b)},\ \Delta^{\lambda'}_{(n,x,b)}\}$, where $a+1 \le x \le b$

available BW ($\max_{1\le n\le N, 1\le p\le P}\{\lambda'_{(n,p)}/\gamma_n\}$) which is an upper bound for the actual minimum value of the balanced peak BW. Therefore, termination of the algorithm is guaranteed after $O(NP)$ iterations.

For the computation of $\Delta\lambda$, equation (18) can be rewritten as:

$$\Delta\lambda = \min\left[\Delta^{or}, \min_{1\le n\le N}\left\{\alpha\left(\frac{\lambda'_{(n,a)}}{\gamma_n} - \widehat{\lambda}\right)\right\}, \min_{1\le n\le N}\{\Delta_{(n,a+1,b_n)}\}\right] \tag{19}$$

Similarly to equation (12), $\Delta_{(n,a+1,b_n)}$ can be computed by the following recurrence relation:

$$\Delta_{(n,x,b_n)} = \min\left\{\Delta^{be}_{x\|(n,a,b_n)}, \alpha\left(\frac{\lambda'_{(n,x)}}{\gamma_n} - \widehat{\lambda}\right),\ \Delta_{(n,x+1,b_n)}\right\} \tag{20}$$

Assuming that $\Delta_{(n,x+1,b_n)}$ is available, $\Delta_{(n,x,b_n)}$ can be computed by

$$\Delta_{(n,x,b_n)} = \min\left[\frac{B^{\circ}_{(n,x)}}{\tau_{(n,x,b_n)}}, \alpha\left(\frac{\lambda'_{(n,x)}}{\gamma_n} - \widehat{\lambda}\right),\ \ \Delta_{(n,x+1,b_n)}\right] \tag{21}$$

Therefore, $\Delta\lambda$ can be computed by

$$\Delta\lambda = \min\left[\frac{B^{\circ}_a - B'_a}{\sum_{1=n}^{N}\tau_{(n,a,b_n)}},\ \ \min_{1\le n\le N}\{\alpha(\frac{\lambda_{(n,a)}}{\gamma_n} - \widehat{\lambda})\},\ \ \min_{1\le n\le N}\{\Delta_{(n,a+1,b_n)}\}\right] \tag{22}$$

The actual algorithm can be implemented as follows. A sorted list of $\{\Delta_{(n,p,b_n)} |\ 1 \le n \le N\}$ is maintained at each breakpoint t_p. Consider an expansion step from t_{p+1} to t_p. First, a test is carried out to see if the new breakpoint t_p is an overflow point. Suppose that it is not an overflow point. $\Delta_{(n,p,b_n)}$ is computed using equation (21) and the sorted list of $\Delta_{(n,p,b_n)}$ is constructed. On the other hand, if t_p is an overflow point, $\Delta\lambda$ is computed by equation (22) and the current $\widehat{\lambda}$ is increased. Now, since $\widehat{\lambda}$ has been changed, the sorted list of $\{\Delta_{(n,p+1,b_n)}\}$ needs to be updated. (After this update, the expansion step to t_p is retried.) This update is quite simple if the increase of $\widehat{\lambda}$ by $\Delta\lambda$ does not change, for any path P_n, the status of the breakpoints between t_{p+1} and t_{b_n} (i.e., $\Delta\lambda$ is taken as either of the first two terms in equation (22)). $\Delta_{(n,p+1,b_n)}$ is decreased by $\Delta\lambda$ for each P_n. However, the update is more complex if $\Delta\lambda$ is taken as the third term. Consider three different cases as follows.

- *Case 1*. A path P_m is selected to decide $\Delta\lambda$ and the increase introduces a new EP in P_m (i.e., $\Delta\lambda = \Delta_{(m,p+1,b_m)}$ and $\Delta\lambda = \Delta^{be}_{(m,x)}$ for some t_x, $t_{p+1} \le t_x \le t_{b_m}$). This means that the INV has been moved to the breakpoint t_x, which is the new EP. $\Delta_{(m,p+1,x)}$ is constructed from scratch by repeatedly applying equation (21).
- *Case 2*. A path P_m is selected to decide $\Delta\lambda$ as above. However, the increase is to the next level of the weighted BW availability (i.e., $\Delta\lambda = \Delta_{(m,p+1,b_m)}$ and $\Delta\lambda = \Delta^{\lambda'}_{(m,x,b_m)}$ for some t_x, $t_{p+1} \le t_x \le t_{b_m}$). In this case, the set of breakpoints where the BW can be increased is reduced, that is, $T^{+}_{(m,p+1,b_m)} = T^{+}_{(m,p+1,b_m)} - \{t_x\}$. $\Delta_{(m,p+1,b_n)}$ is constructed from scratch as in case 1.
- *Case 3*. For all the other paths, $\Delta_{(n,p+1,b_n)}$ is decreased by $\Delta\lambda$.

Let us consider the computational complexity of the algorithm. At an expansion step, if it is not an overflow, the computational cost is $O(N \log N)$, which is to construct the sorted list of $\{\Delta_{(n,p,b_n)} |\ 1 \le n \le N\}$. Since there are P breakpoints, the total cost for an expansion step which does not result in an overflow is $O(PN \log N)$. In case of an overflow, the bottleneck in the computation is the reconstruction of $\Delta_{(m,p+1,x)}$ or $\Delta_{(m,p+1,b_n)}$ for cases 1 and 2 above. This computation takes $O(P)$ and there are at most $O(NP)$ iterations of this kind. Therefore, the total cost of computing the balanced minimum peak BW reservation is $O(PN \log N + P^2 N)$.

5 Performance analysis

The performance of the GBBCS policy is evaluated in a two-stage client–server environment consisting of a set of clients and a server. The total number of clients is varied from 5 to 60. The clients are connected to the server via a dedicated network, and the server has a total BW capacity of 50 Mb/sec. We simulate various configurations with and without prefetch buffer space in the clients and server. The server prefetch buffer is shared by all clients connected to the server, while the prefetch buffer in a client is dedicated to a single user. We consider client prefetch buffer sizes up to 12 MB and server buffer sizes up to 250 MB.[5]

The BW requirement profile used in the simulation is derived from the Olympic swimming competition example (see Sect. 2). The clients are assumed to be watching different multimedia documents, however, with the same BW

[5] Note that apart from the prefetch buffers used for traffic reshaping, the system also maintains I/O and network buffers for continuous presentation to accommodate any jitter in the data retrieval or transmission [5, 19].

requirement profile. We also consider workloads resulting from presentations with different BW requirement profiles.[6] The system is modeled as a closed system where each client generates a new request immediately upon the service completion of its previous request. Here, the generation of a request means the generation of an initial object delivery schedule (and the corresponding BW requirement profile) for a composite multimedia object by a client which is passed to the server. Also, service completion means finishing the presentation of the client-requested composite object. Note that if a request cannot be satisfied, it is delayed until a feasible delivery schedule is found. The system performance is measured by the server throughput and the average delay introduced by the server scheduler, where the throughput of the system is estimated as the number of request completions per second in the steady state. For starting the system in a random state, the initial requests from each client are randomly spaced with an exponential distribution. Empirically, it was observed that the system became stable after about 15 minutes. Hence, the performance measures are taken starting from 20 minutes after start-up. Observations were made for approximately 1 hour.

5.1 Effects of fine granularity advanced reservation

We first test the efficiency of the FGAR policy without any prefetch buffer in the system. The delay and throughput under the FGAR policy are compared to those of a policy with peak BW reservation in Fig. 17. Figure 17a shows the average request delay as a function of the total number of clients in the system. The dotted curve represents the system with the FGAR policy, while the solid curve is for the peak BW reservation policy. In both cases the delay is zero for a smaller number of clients, increasing linearly beyond a certain point (15 for the FGAR policy and 10 for the peak BW reservation policy) with the number of clients. The FGAR policy clearly outperforms the peak BW reservation policy since it introduces a smaller average delay for the same number of clients. For 15 clients in the system, all requests can be served without delay under the FGAR policy whereas an average delay of approximately 80 seconds is required under the peak BW reservation policy. The difference grows larger as the system becomes further overloaded with an increasing number of clients. Figure 17b shows the corresponding throughput as a function of the number of clients. The achievable throughput under the FGAR policy is approximately 50% higher than that of the peak BW reservation policy. In both cases, the throughput rises linearly until the system is saturated. However, in the case of the FGAR policy, the throughput drops slightly after the saturation point. This is explained below. Once the system is overloaded, a presentation delay is introduced in all subsequent requests until enough resources start becoming available due to completion of ongoing presentations. This will result in periodic fluctuations in available BW and, hence, unpredictable throughput (e.g., a lower throughput in this example).

Figure 18 shows the variability in available server BW and presentation delay as a function of time for a system with 20 clients. Both the available server BW and hence, the presentation delay fluctuate periodically. There are time intervals where no client requests can be started, leading to wasted server BW. As shown later, prefetching of data can smooth out the BW fluctuations and reduce this wasted server BW. The presentation delay also fluctuates periodically, following the periodic fluctuations in the available BW (see Fig. 18b). The phenomenon of periodic exhaustion of server BW leading to cyclic variations in throughput and, hence, inefficient server utilization has also been observed in [1]. However, the system studied in [1] consisted of client requests for a single video object requiring fixed BW, and the server was subjected to a time-varying load by changing the arrival rate of client requests.

5.2 Effect of reshaping bandwidth requirement profile by clients

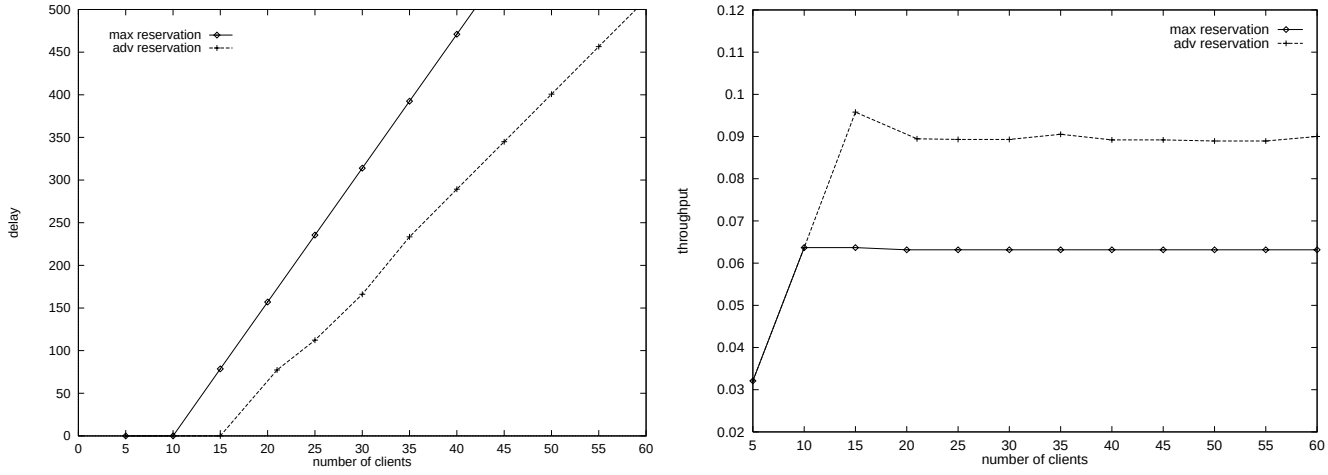
In the presence of client prefetch buffers, it is possible to prefetch portions of document objects to smooth out the BW requirement of a presentation. The relationship between the available buffer size and required peak BW for the chosen workload is shown in Fig. 19a. As the available client prefetch buffer increases, the required peak BW (in server and network) is reduced. Hence, prefetching into client buffers can be used to compensate for limited network BW.

Figure 19b shows the impact of client reshaping on the server throughput as a function of the total number of clients. Three cases with varying amounts of prefetch buffer per client (no buffer, 4 MB, and 11 MB) are shown. The required peak BW in these cases is 4.7 Mb/s, 3.0 Mb/s and 1.5 Mb/s, respectively. The server is assumed to have no prefetch buffer in these cases. For a larger number of clients, the throughput increases with better reshaping of BW requirement profiles, that is, reduction in peak BW requirement. (The highest BW utilizations for these cases are 55%, 72% and 99%, respectively.) Also, in all cases, the throughput increases with the number of clients and then saturates. However, for a smaller number of clients, one interesting phenomenon is observed by comparing the relative order of throughput for these three cases. When the system is lightly loaded, the server utilization is the same for all three cases. However, a larger presentation delay, and hence a larger total service time for achieving a larger reduction in a peak BW requirement, dominates the relative order in the server throughputs for these cases. The situation rectifies itself with a larger number of clients when the total service time becomes less relevant and a reduction in peak BW actually improves server resource utilization.

5.3 Effect of server reshaping

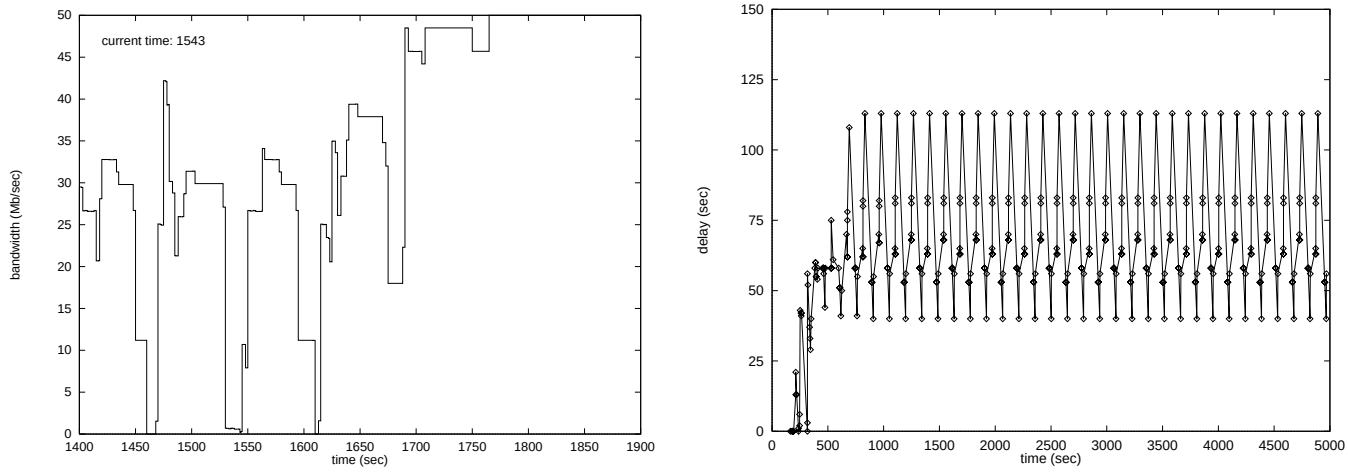
We next study the use of server buffers to prefetch data (from disks) and reshape BW requirement profiles. The experiments are conducted by varying the size of both client

[6] Issues related to multiple requests for the same document or different documents sharing common objects are beyond the scope of this paper. Various caching schemes introduced in the literature could be used to exploit such sharing [7, 14]. Furthermore, the current framework (i.e., prefetching) can be extended to integrate caching policies.



a **b**

Fig. 17. System performance under fine granularity advanced reservation: **a** average presentation delay; **b** system throughput



a **b**

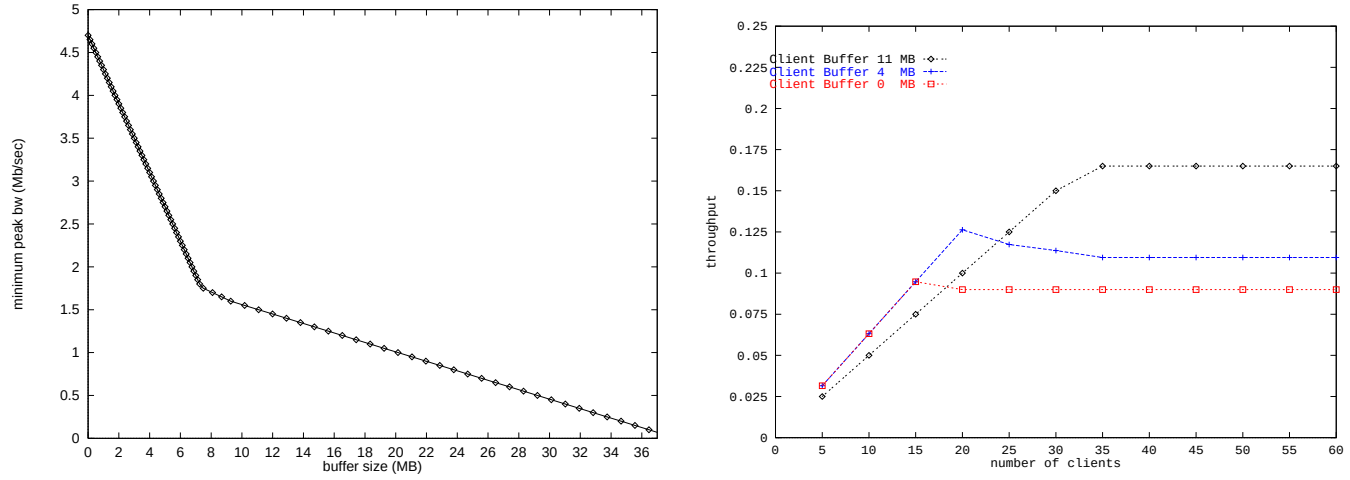
Fig. 18. Variability in **a** available server bandwidth and **b** presentation delay (20 clients)

and server prefetch buffers. We consider two server configurations with and without a server buffer of size 75 MB. For each server configuration, two different client configurations are considered (with and without a 4 MB client buffer). Figure 20 shows the required average presentation delay and throughput for the above four configurations. The dotted lines are for the cases with no server buffer, while the solid lines represent the cases with a 75 MB server buffer. The curves marked A represent the cases without a client buffer (i.e., reshaping), while those marked B represent the cases with a 4 MB client buffer. Comparisons of the solid lines with the corresponding dotted lines show that server prefetching reduces the average delay significantly in both client configurations (i.e., with and without client reshaping). In both cases, the server buffer increases the achievable maximum throughput by at least 50%.

5.4 Placements of prefetch buffers

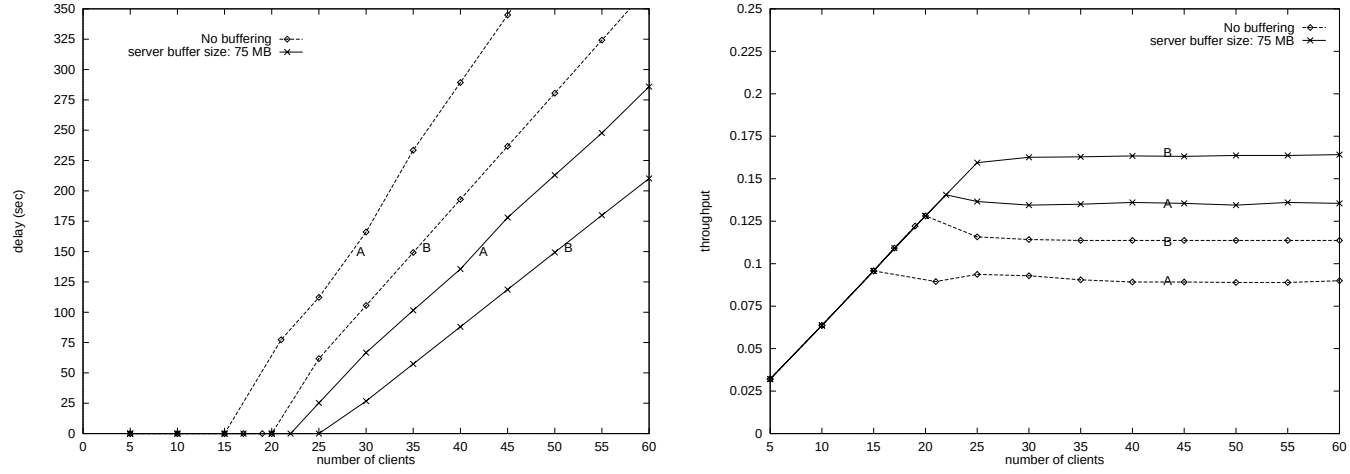
The experiments so far demonstrate the desirability of the FGAR policy combined with prefetching both on servers and clients. The efficacy of prefetching depends on the size of the prefetch buffers. Therefore, an important issue is how a given prefetch buffer should be distributed between the client and server systems. In the following experiments, the total buffer size in the system is kept fixed while the configurations are changed by varying the size of the client and server buffers. We start with a configuration where all buffer space is evenly distributed among 40 clients. We then reduce the buffer size in each client in steps of 0.25 MB and increase the server buffers size by 10 MB so as to keep the total amount fixed.

Figure 21 shows the effect of varying the distribution of the buffer space between the client and the server. The figure plots the average server throughput as a function of the amount of the buffer space in the server. Several curves, with total buffer size varying from 0 to 250 MB, are shown. In each case, the throughput rises as the amount of server buffer space is increased, indicating that the maximum throughput is achieved when all the buffer is in the server. This is because increases in server buffer space can be shared among all the streams in the system, whereas an increase in client buffer space benefits only one client. Hence, from the point of view of maximizing server throughput, it may be desirable to allocate all the buffer space to the server. However, a



a **b**

Fig. 19. Effect of client reshaping to advanced reservation: **a** client buffer size and corresponding minimum peak bandwidth; **b** throughput



a **b**

Fig. 20. Effect of server reshaping: **a** delay; **b** throughput. Cases without any client reshaping are marked A, cases with client reshaping with a 4MB buffer are marked B

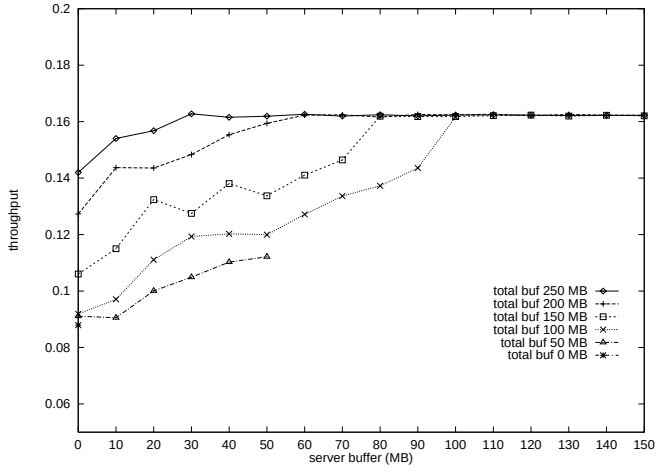
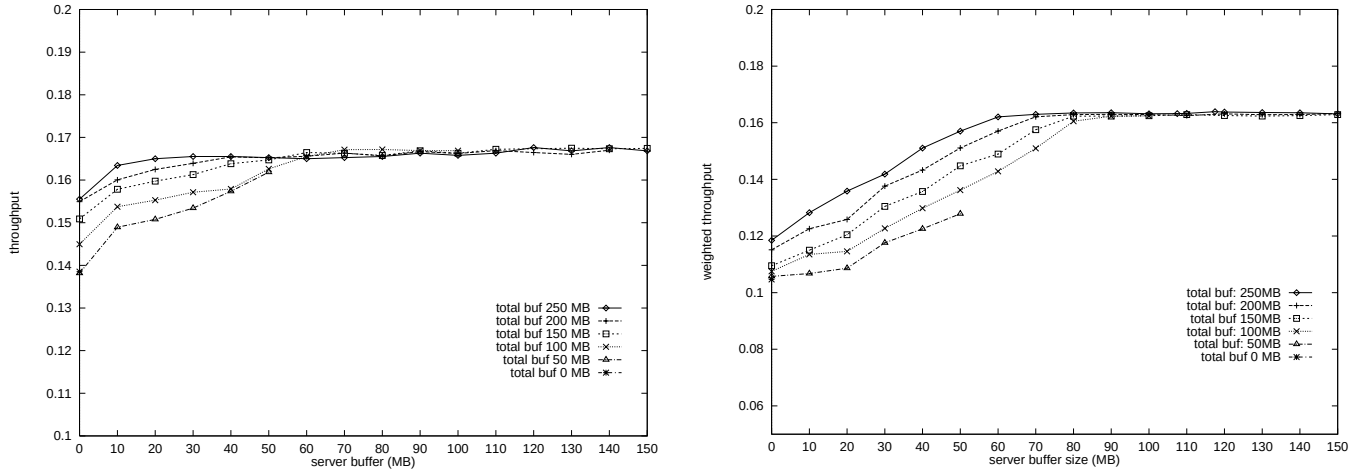


Fig. 21. Partitioning of prefetch buffers between clients and servers

certain amount of client buffer is necessary not only to avoid jitter in transmission, but also to make feasible presentations of some composite documents in the face of limited network BW (as in the "last mile problem"). One interesting point to note from the above figure is that the maximum server throughput cannot be achieved without a buffer size of 100 MB for this workload. This is also the configuration that maximizes the throughput while minimizing the total buffer requirement.

5.5 Mixed workloads

In our earlier experiments, we considered only a single document (and hence, a single BW requirement profile) accessed by all clients. We next consider mixed workloads where clients choose randomly from a set of BW requirement profiles. To make these profiles realistic, we have created five new profiles from the original profile. Profile 2 represents the same document with high resolution audio and video (e.g., MPEG-2). This is created by scaling up the profile by 3. Profile 3 is a low data rate document and includes only audio, image, and text data. This is created by scaling down the profile by 6. Profiles 4, 5, and 6 are created from profiles 1, 2, and 3 respectively by shortening the presentation dura-



a **b**

Fig. 22. Throughput under various partitioning of buffer space, mixed workloads: **a** two profiles (MIXED2); **b** six profiles (MIXED6)

tions by half. In the MIXED6 workload, the clients choose with equal probability one of the above six profiles upon a new request. Another workload, called MIXED2, was created by mixing two profiles where the clients choose, with equal probability, profile 1 and its reverse profile, that is, the BW requirement for playing the presentation backwards(!).

The experiments for studying the proper placement of prefetch buffer space reported in Sect. 5.4 are based on the workload composed of a single profile (i.e., profile 1). We now repeat these experiments for the MIXED2 and MIXED6 workloads. Figure 22 shows the results of these experiments. As before, the server throughput is measured by the average number of service completions per second. While in the MIXED2 workload both of the profiles retrieve an equal amount of data, the different profiles in the MIXED6 workload retrieve a different amount of data. Hence, to normalize the throughput across all workloads, a service completion of each request is weighted by the amount of data retrieved relative to that of profile 1. The results shown in Fig. 22 also confirm the observations made in the earlier experiments in Fig. 21. In both MIXED2 and MIXED6, the throughputs are increased (up to the maximum achievable value) when the server buffer space is increased. In both cases, the maximum throughput is achieved for the curves with total buffer size of 100 MB or more. However, this maximum throughput is achieved for a server buffer size of 60 MB for the MIXED2 workload.[7]

5.6 Resource allocation in a multihop environment

The JINSIL layer at each stage will pass an ODS to the next stage as a delivery request which is in turn used by the scheduler at that stage to generate a new ODS. However, resource allocation by applying GBBCS remains a challenging problem in a system where the data path passes through multiple stages (see Fig. 23). This is due to the fact that once a presentation delay, Δ, is introduced at a stage, S_i, other

[7] We conjecture that the server throughput is higher for the MIXED2 workload since profile 2 somewhat complements profile 1 and thus reduces the variability in available BW.

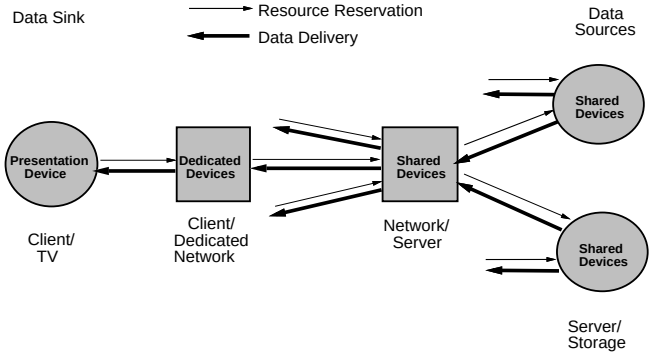


Fig. 23. Illustrations of multi-hop data delivery path

than the client (i.e., at some intermediate stage or in the data server) the satisfiability of the delayed presentation needs to be rechecked at all the earlier stages. If the resources dedicated to the requesting client are fixed at all times at all the earlier stages, such a delay, Δ, does not alter the satisfiability of the requested presentation at any of the earlier stages nor does it changes the BW requirement profile presented to the stage S_i. Therefore, the effect is a simple time shifting of the ODS at each of the earlier stages. However, if resources are shared at any of the previous stages, the schedule creation process needs to be started from the first affected stage (i.e., one with shared resources). Note that due to the variability in available resources at various stages, and hence their interactions via traffic reshaping, the amount of delay by which the presentation needs to shifted is not easy to estimate.

6 Contributions and relationship to earlier work

This paper is chiefly concerned with the scheduling and resource allocation issues addressed by JINSIL in a distributed multimedia environment. First, at each delivery stage (e.g., a client or server system) for a given allocation of BW and buffer space to a presentation, it creates a prefetch schedule for the atomic objects in the document. Second, if the server node supports FGAR, the scheduler matches the re-

source allocation (e.g., BW and buffer) to the variability in available resources and the exact prefetch schedule, that is, the traffic is shaped to match the *time-dependent* availability of BW and buffer. The resource allocation policies in JINSIL improve the efficiency of retrieval in a number of ways. The FGAR policy, together with the reduction of the peak BW requirement, increases the feasibility of a presentation and, hence, the number of streams that can be admitted to a server. Additionally, as shown in Sect. 5, the time-dependent variation in BW requirement may lead to periodic exhaustion of server BW and, hence, to inefficient server utilization. The resource allocation smoothes out the BW fluctuations and reduces such waste of server BW. Finally, if the objects are to be retrieved from multiple data sources, the reserved BW and buffer not only match the available BW in each of the sources but also reduce the *imbalance in peak loads* across those different sources.

In summary, the JINSIL retrieval system supports presentations of composite multimedia documents with variable BW requirements in both dedicated (e.g., client) and shared (e.g., server) system components, as well as both in a single storage system and in a distributed storage system. In a server environment, the resources are utilized by all streams delivered through that server. Hence, the resources available to a single stream are not fixed. Additional complexities are introduced in dealing with such an environment, especially in taking into account variable requirements of all streams and partitioning the available resources amongst the streams. When the media objects included in a composite document are distributed in multiple storage systems, the resource allocation further considers the synchronous as well as balanced utilization of resources on different paths.

It is very difficult to provide FGAR in an open environment such as the Internet. In such an environment, many different systems may compete for shared resources with many different applications. Some of the applications may generate best-effort traffic without requiring resource reservation, making a fine granuled control of the system resources more difficult. Maintaining a time-dependent resource availability on such a system may not be a practical assumption. The FGAR policy is more feasible in a rather closed system environment, where a fixed set of clients share the system resources with a predefined set of applications. In such a system, more complete control over the system resources would be possible. The performance study in Sect. 5 was also conducted in an environment which simulates such a closed client–server multimedia environment.

A number of papers have considered client prefetching from the view of a single client and in a single path environment where a constant amount of BW and/or buffer space is available per stream. Client prefetching mechanisms in such a dedicated, single path environment to smooth the burstiness introduced by data compression in a single long data stream are studied in [9, 19].[8] Both papers address optimal delivery schedules consisting of piecewise constant rates such that the prefetching of data does not overflow the client buffer, and the chosen delivery rate does not cause buffer underflow. The optimality condition is defined either as the minimum number of changes in the delivery rate [9] or as the least variability in delivery rates [19].

Bandwidth and buffer satisfiability issues for a composite multimedia document are studied in [16, 17]. This work studies the resource requirement (i.e., client buffer space and BW between a client and a server) from the view of a single client. We extend this work in several ways. First, we consider end-to-end scenarios of a client–server environment where the server system is shared, and hence efficient utilization of shared resources demands an object delivery schedule with minimum peak BW requirement. We also consider a server node that support the FGAR policy for BW and buffer. Structural analysis and time shifting (i.e., prefetching or delaying) have been used in the DEMON project [18] for presentations of composite documents in a networked environment. This has made better use of a fixed BW allocation per request by smoothing the bit rate requirements of a document. However, it has not exploited the FGAR policy for resource reservation. Neither of the papers considered the synchronization and resource allocation issues in a distributed storage server environment.

There is a lot of prior work on storage retrieval to avoid jitter in data delivery [3, 11, 12]. Continuous retrieval of a composite document using flash memory is considered in [20]. In [4], the authors study prefetching and delaying schemes to avoid contention in data retrieval from disk space. A characterization and workload of hypermedia applications is proposed in [13]. The workload is used to study (via simulation) the performance of the Continuous Media File System (CMFS).

7 Conclusions

A composite multimedia document may consist of multiple video and audio clips, images and other data objects. The synchronous presentation of such structured, composite multimedia information poses serious challenges in a distributed environment where all or various pieces of a composite presentation document may reside in one or multiple remote systems away from a client representation system. Appropriate resources need to be allocated in various data paths from the respective sources to the client system. Even if all the data are stored in a single system, the instantaneous data consumption rate will vary over time depending on the structure of the presentation. A presentation of a complex media document may require multiple streams of video, audio, image or text data for a short duration.

In this paper, we describe the *JINSIL* retrieval system that addresses issues for a presentation in a shared environment. It allocates appropriate resources on delivery paths and creates object delivery schedules for both client and server systems. The *resource allocation* and creation of *object delivery schedules* are based upon document structure, the locations of objects and resulting delivery paths, dedicated vs. shared resources on these paths, and available buffer space and BW.

There are a number of interesting research issues to be done in the proposed data retrieval and resource allocation

[8] In [19] performance of the proposed policy is studied in a shared network environment. However, the proposed algorithm did not explicitly take into account the available BW in the shared component (i.e., network) in reshaping the traffic.

framework. One of them is to examine a more extensive performance comparison of the various possible resource allocation schemes. For example, the minimum peak bandwidth allocation policy can be compared with other policies proposed in [9, 19]. The policies in [9, 19] were proposed in the context of scheduling the presentation of a single long video stream. With appropriate modifications, the algorithmic framework will be able to be adopted for the scheduling of composite multimedia presentations.

In Sect. 5.3, we observed the effect of server prefetching and reshaping on the system througput. Also, in Sect. 5.4, we investigated the effect of different partitioning of the buffer space between the servers and the clients. From the experiment, we have also identified the size of server buffer space which maximizes the server througput. Related to these, another interesting issue will be to investigate more throuhgly the relationship between useful server buffer space and the number of simultaneous presentations.

Acknowledgements. Parts of this paper were presented in the ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems, 1997, and the IFIP WG 7.3 Workshop, 1997. The authors thank the anonymous reviewers for their valuable comments which improved this paper.

References

1. K. Almeroth, A. Dan, D. Sitaram, and W. Tetzlaff. Long-term channel allocation strategies for video applications. IBM Research Report, RC 20249, 1995.
2. David P. Anderson. Metascheduling of continuous media. ACM Trans Comput Syst 11(3):226–252, 1993.
3. S. Berson, L. Golubchik, and R. R. Muntz. Fault tolerant design of multimedia servers. In ACM SIGMOD, 1995.
4. S. Chaudhuri, C. Shahabi, and S. Ghandeharizadeh. Avoiding retrieval contention for composite multimedia objects. In Proceedings of VLDB Conference, 1995.
5. A. Dan, D. Dias, R. Mukherjee, D. Sitaram, and R. Tewari. Buffering and caching in large scale video servers. In Proceedings of IEEE CompCon, pages 217–224, 1995.
6. A. Dan, P. Shahabuddin, D. Sitaram, and D. Towsley. Channel allocation under batching and vcr control in video-on-demand systems. J Parallel Distrib Comput 30(2):168–179, 1995.
7. A. Dan and D. Sitaram. Multimedia caching strategies for heterogeneous application and server environments. Multimedia Tools Appl 4(3), 1997.
8. A. Dan, D. Sitaram, and P. Shahabuddin. Scheduling policies for an on-demand video server with batching. In Proceedings of ACM Multimedia, October 1994.
9. W. Feng, F. Jahanian, and S. Sechrest. An optimal bandwidth allocation for the delievery of compressed prerecorded video. Technical Report CSE-TR-260-95, University of Michigan, August 1995.
10. E.A. Fox. Advances in interactive digital multimedia systems. IEEE Comput 24(11):9–19, 1991.
11. J. Gemmel and S. Christodoulakis. Principles of delay sensitive multimedia data storage and retrieval. ACM Trans Inform Syst 10(1):51–90, 1992.
12. J. Gemmel, H. Vin, D. Kandlur, V. Rangan, and L. Rowe. Multimedia storage servers: A tutorial. IEEE Comput 28(6): 40–49, May 1995.
13. C. Gopal and J. F. Buford. Delivering hypermedia sessions from a continuous media server. In S. M. Chung, editor, Multimedia Information Storage and Management. Kluwer Academic Publishers, 1996.
14. M. Kamath, K. Ramamritham, and D. Towsley. Continuous media sharing in multimedia database systems. In 4th International Conference on Database Systems for Advanced Applications (DASFAA '95), April 1995.
15. M. Kim and J. Song. Multimedia documents with elastic time. In ACM Multimedia Conference '95, 1995.
16. T. D. C. Little and A. Ghafoor. Multimedia synchronization protocols for broadband integrated services. IEEE J Selected Areas Commun 9(9):1368–1382, 1991.
17. T. D. C. Little and A. Ghafoor. Scheduling bandwidth-constrained multimedia traffic. Comput Commun, 15(5):381–387, 1992.
18. J. Rosenberg, G. Cruz, and T. Judd. Presenting multimedia documents over a digital network. Comput Commun 15(6), 1992.
19. J. Salehi, J. F. Kurose, Z. L. Zhang, and D. Towsley. Supporting stored video: Reducing rate variablity and end-to-end resource reservation through optimal smoothing. In Proceedings of ACM SIGMETRICS Conference on Measurement and Modeling of Coputer Systems, 1996.
20. C. Shahabi and S. Ghandeharizadeh. Continuous display of presentations sharing clips. Multimedia Syst, pages 76–90, 1995.
21. J. Song. Structured Composite Multimedia Documents: Design and Presentation in a Distributed Environment. PhD Thesis, University of Maryland, 1997
22. J. Song, A. Dan, and D. Sitaram. Jinsil: A system for representation of composite multimedia objects in a distributed environment. IBM Research Report, 1997
23. J. Song, A. Dan, and D. Sitaram. Efficient Retrieval of Composite Multimedia Objects in a JINSIL Distributed System. In Proceedings of the ACM SIGMETRICS Conference on Measurement and Modeling of Computer Systems, Seattle, Washington, June 1997
24. J. Song, G. Ramalingam, R. Miller, and B. Yi. Interactive authoring of multimedia documents in a constraint-based authoring system. Multimedia Syst, 7(5):424–437, 1999

Junehwa Song is an assistant professor, Dept. of EECS, Korea Advanced Institute of Science and Technology (KAIST), Korea. Before joining KAIST, he worked at IBM T. J. Watson Research Center, Yorktown Heights, NY as a Research Staff Member from 1997 to Sep. 2000. He received his Ph.D in Computer Science from University of Maryland at College Park in 1997. His research interest lies in Internet Technologies, such as intermediary devices, high performance Web serving, electronic commerce, etc, and distributed multimedia systems.

Dr. Dinkar Sitaram is an Andiamo Fellow at Andiamo Systems, responsible for architecture and technical direction. Previously, he was Director of the Technology Group at Novell Corp., Bangalore, one of the major research groups within Novell investigating the areas of networking security, multimedia, directory systems and distributed computing. The group has developed innovative products in addition to filing for many patents and standards proposals. Before that, he was a Research Staff Member at the IBM T.J.Watson Research Center, Yorktown Heights, where he worked in the areas of file systems, multimedia servers and E-commerce. He has received Novell's Employee of the Year award, holds several top rated patents, has received IBM Outstanding Innovation Award and several IBM Invention Achievement Awards, and received outstanding paper awards for his work. He is the author of the book "Multimedia servers" published by Morgan Kaufman, jointly with Dr Dan Dr. Sitaram received his Ph.D. from the University of Wisconsin-Madison and his B.Tech from IIT Kharagpur.

Dr. Asit Dan has been with IBM Research since 1990, and is at the forefront in the research and development of web services, transaction processing architectures and video servers. He holds several top rated patents in these areas and has received two IBM Oustanding Innovation Awards, seven Invention Achievement Awards, and the honor of Master Inventor for his work in these areas. Currently, he is managing the business-to-business integration department working on the development of infrastructure for supporting dynamic, and electronic SLA & trading partner agreement driven B-B e-commerce applications. Dr. Dan received a Ph.D. from the University of Massachusetts, Amherst. His doctoral dissertation on "Performance Analysis of Data Sharing Environments" received an Honorable Mention in the 1991 ACM Doctoral Dissertation Competition and was subsequently published by the MIT Press. He has also published extensively, including serveral book chapters, and a book on "Multimedia servers".