Computer Communications 31 (2008) 3604–3612

Contents lists available at ScienceDirect

Computer Communications

journal homepage: www.elsevier.com/locate/comcom

Analysis and placement of storage capacity in large distributed video servers ☆

Vicent Cholvi [a,*], Juan Segarra [b]

[a] Universitat Jaume I, Campus del Riu sec s/n, 12071 Castellón, Spain
[b] Instituto de Investigación en Ingeniería de Aragón (I3A) and HiPEAC, Universidad de Zaragoza, Spain

ARTICLE INFO

Article history:
Received 1 February 2008
Accepted 16 June 2008
Available online 24 June 2008

Keywords:
Video on demand
Quality of service
Multimedia
Broadband
Multicast

ABSTRACT

In this paper, we study how to distribute storage capacity along a hierarchical system with cache-servers located at each node. This system is intended to deliver stored video streams in a video-on-demand way, ensuring that, once started, a transmission will be completed without any delay or quality loss. We use off-line smoothing for videos, dividing them into CBR video parts. Also, our request rates are distributed following a 24 h audience curve. In this system, when a request is received, the server reserves the required bandwidth at the required time slots, trying to serve the video as soon as possible.
We perform a detailed analysis by means of simulations of the start-up time delay for some storage distributions. It shows that an adequate storage distribution can increase performance about 25% with respect to a uniform distribution and about 47% with respect to one in which all the storage is attached to the gateway routers that connect the final users. We also analyze bandwidth usage, comparing the behavior of these storage distributions.
Finally, we present a method which allows dynamic and transparent video reallocations when their popularity changes.

1. Introduction

Currently, the delivery of stored digital video is becoming commonplace and a significant source of network traffic. When a customer demands a video file, one of the servers of the system will provide a copy which can be watched by the customer. However, according to the tight bandwidth and latency constraints of video systems, rigorous analysis is necessary before its deployment [1].

Perhaps the most simple technique consists in using a centralized single-server intended to deliver all video requests to the users. However, it is known that this approach does not scale well as the load (i.e. the number of users and videos) grows.

In order to solve this problem when considering video delivery over wide area networks, some authors propose the use of a number of cache-servers distributed along the communication network. These cache-servers will store a subset of those videos so as to provide users with direct access to them (see for instance [2–4]). Such an approach has been found to be preferable to using a centralized server.

Similarly, the same conclusion has been reached when considering networks intended not only to provide good performance of the overall system but also to guarantee QoS and real-time video delivery, e.g. hierarchical networks [5,6]. In those networks, it has been also shown that the best placement of videos (i.e. the best way of distributing the videos along the cache-servers) consists in placing the most popular videos as close to the users as possible.

On the other hand one major problem about this kind of systems is the large amount of information a video stream contains. Compression formats allow an important reduction of these data size, but they increase the bit rate variability of these data [7,8]. Transmitting real-time VBR flows is no trivial matter, because a different amount of bandwidth will be needed in each moment of the transmission. Naive approaches to this problem use a pessimistic evaluation, as they reserve enough bandwidth for the stream to be transmitted supposing it is always the worst case. Obviously, this is quite inefficient. One of the methods to improve these transmissions is the *smoothing* [9,10] of streams before their transmission. This is done by transmitting frames into the client playback buffer in advance of each burst so that peak and rate variability requirements are minimized. Rexford and Towsley show how to compute an optimal transmission schedule for a sequence of nodes by characterizing how the peak rate transmission rate varies as a function of the playback delay and the buffer allocation at the server and client nodes [11]. By also using smoothing, several techniques have been proposed which develop a *transmission plan* [10] consisting of time periods so that, in each period, the transmission may be performed by using a constant-bit-rate (CBR) network service. In [4], the authors present a technique that makes use of this approach. They also develop a video delivery technique via intelligent utilization of the links bandwidth and

☆ This work has been partially supported by Bancaixa under Grant P1-1B2007-44, by the MEC of Spain under Grant TIN2007-66423 and by the DGA under Grant "Grupo Consolidado de Investigación".
* Corresponding author. Tel.: +34 964728332.
E-mail address: vcholvi@uji.es (V. Cholvi).

0140-3664/$ - see front matter © 2008 Elsevier B.V. All rights reserved.
doi:10.1016/j.comcom.2008.06.012

storage space available at the proxy servers (i.e., servers which are attached to the gateway router that connects the final users). In [12], the authors show that any (including optimal) off-line schedulings of the periodic broadcasting methods can be improved by on-line schedulers in video-on-demand.

In our study, we use a *true* video-on-demand system. This means that quality of service is completely guaranteed. Bandwidth requirements are reserved when a video is requested, trying to offer the minimum possible starting delay and using multicast transmissions if possible. This means that video playing will not suffer any break or quality loss, and also, reservations guarantee a fixed and known starting delay. This is translated into instant server responses notifying users about their reservation and playing start time of their request.

Using this system, we make a detailed analysis of the storage distribution in a tree structured video server. Previous studies on hierarchical systems have been obtained assuming either that all cache-servers have the same storage capacity (*uniform size distribution*) or that they are attached only to the gateway router that connects the final users (*proxy-like size distribution*). In this paper we focus on studying how the storage distribution in the network affects its performance.

These results provide a guidance as to how to distribute storage capacity in a system to achieve the best performance. Also, they allow us to predict, when there is a change in the system parameters, how such a change will improve or degrade the overall performance.

Additionally, we propose a reallocation method, which allows to change the place where videos are stored. This provides a dynamic behavior, making this study suitable for application in real systems. Basically, this reallocation method allows to change the parameters of several videos, and redistributes the videos along the structure according to the new parameters. This is done while the system is on-line, and also in a transparent way for users.

The rest of our paper is organized as follows. In Section 2 we describe the scenario we use. In Section 3 we present our video-on-demand delivery system. Then, in Section 4 we show our results. Finally, our conclusions are presented in Section 5.

2. Scenario

For our study, we use a hierarchical network architecture. This architecture is used nowadays in most cable networks and has been demonstrated to be adequate for VoD systems [13,6]; so it meets our needs. Our structure is a binary tree with four levels of depth, that is, one central server plus three cache levels. We consider this system a representative one, because we can evaluate the storage size near the server, near the client, and in between, and show how the storage size should be distributed.

Cache-servers at leaves have two branches with the same number of users. Links between nodes have a bandwidth of 1.5 Gbps downstream. A moderate bandwidth is also needed upstream in order to send user requests to the central server. Note that final users share a bandwidth of 1.5 Gbps because they share the link to their nearest node. Fig. 1 shows two final client branches of this tree structure.

We use scenarios containing from 1000 to 10,000 videos. Each video information has been constructed making variations over the example presented in [10], resulting in (smoothed) videos of a mean duration of about 85 min and a mean bandwidth requirements of about 1850 Kbps in fragments ranging from 96 to 4608 Kbps.

In our approach to the video transmission problem we studied the performance over a 24 h period. Previous studies on video demand rates [14,15] have determined the behavior of these rates in this period. Our simulations work with a rate distribution according to these studies. We also consider that a user makes a request only if he or she is not waiting for or receiving a video (i.e. nobody can receive more than one transmission at a time). In Fig. 2 we can see the request rate distribution having one request per day for each user. This is what we assumed in our study (actual VoD trials show that the average number of user requests per week is about 2–3).

Using this request rate, videos are requested following a *popularity* factor, which represents the probability of a video to be requested according to Zipf's law [16]. This distribution has been found to statistically fit video program popularities estimated through observations from video store statistics [17].

3. Video allocation and reservation algorithm

In this section we first describe how to distribute videos throughout the whole system. Since bandwidth usually becomes a bottleneck in stream transmissions, the placement policy is a major issue. Then, we explain our reservation algorithm.

In order to avoid the rate variability problem described in the introduction, we use smoothed versions of video streams [10]. In this way, each video is characterized by a collection of tuples $\langle time, rate \rangle$, whose first parameter denotes a time period and whose second one is the CBR rate associated with such a time period.

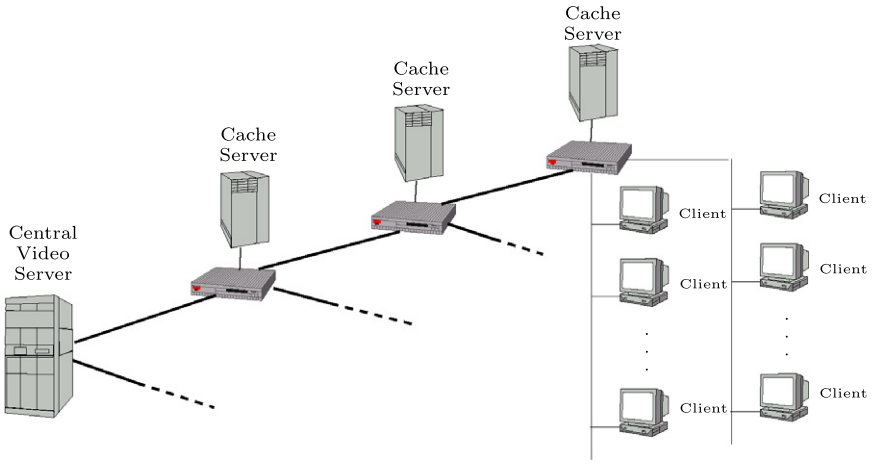

Fig. 1. Representation of two final branches of our tree structure.

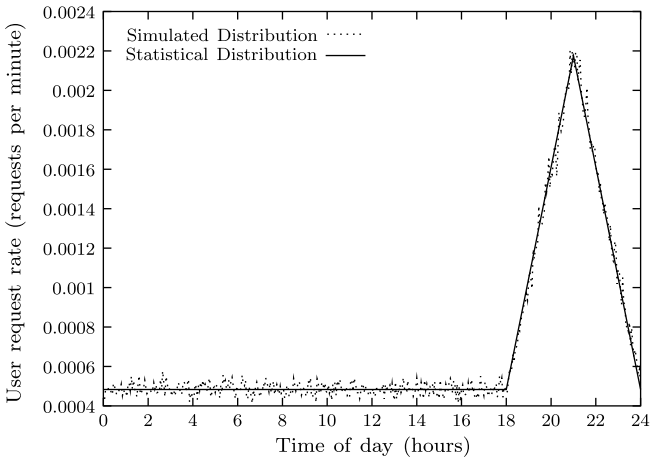

Fig. 2. Request rate distribution in a 24 h period.

Our distribution policy is applied off-line and it acts on the different video parts corresponding to the tuples of each video. Basically, it first assigns, at start up, a weight value to each video. That weight value is based on the video *popularity*, according to Zipf's law [16] as above. Once such an assignment is done, it distributes the video parts in the tree, sorting them out by weight value. The higher the weight value, the nearer to users they are placed. When storing these video parts in places other than the main server, they are replicated in all other caches at the same level. Note that, working with smoothed video parts, each one can be stored independently. That is, despite all parts of the same video will have (in this case) the same weight value, they can be stored in different cache levels when caches become full.

The good results obtained using the popularity value as the weight parameter can be easily explained [5,6]. The main goal of our system is to reduce the high bandwidth requirements that a centralized system would have. By using this weight value, the most requested videos are placed close to users, so their transmission affects only the links at leaves, and not the full tree structure. On the other hand, the less popular videos, which will hardly be requested, can be transmitted directly from the main server. Thus, bandwidth costs produced by a few popular videos is reduced by using storage size to replicate them close to the users.

For the above proposed system it is necessary to use a *reservation algorithm*. Such an algorithm manages video requests, guaranteeing that, once a video is accepted (maybe after some start-up time delay), it will be delivered without interruption to the final user(s).

Our reservation algorithm is constructed around a *bandwidth reservation table* which represents the bandwidth that will be used by each link at each time unit. In our case these time units (*time-slots*) have a duration of 1 min.[1] Additionally, each request has a process time of 60 s. This means that when a request is received, the system processes it during 60 s and its transmission could start the next time-slot after this processing time.

The management of each request is the following one. When a video request is received, we first map out the route which each video part will follow. Then, we test (for each video part) whether it is already planned for transmission at the same time by the same links. If so, that request is added to the multicast transmission of this video part without any additional bandwidth requirements. Otherwise, we check whether there is enough bandwidth in each link of the route during the slots of the transmission. If so, bandwidth is reserved. Otherwise, the system repeats this process trying to place the transmission plan one further time-slot. Obviously, in such a case the video starting time is also delayed one time-slot. Thus, when there is not available bandwidth, transmissions of new requests are delayed until there is enough bandwidth. Fig. 3 shows the code of this algorithm.

There are several features in this algorithm which must be taken in consideration. First of all, when placing a new request, the system looks for a time to serve it and makes the transmission plan. This means that the system can respond immediately indicating when the requested transmission will start. With this information, and knowing that the starting delay is not too large, we assume that there will not be transmission rejections.[2] Additionally, by making full reservations for each request, there is no need of any renegotiation.

In reference to the multicast transmissions, note that we assume all nodes in the tree belong to the same system and can be managed by any convenient software. Thus, multicast transmissions can be reduced to the user links, because node-to-node they could be implemented by means of unicast transmissions. In this way, only the last level of cache would need to be aware of which users are requesting each video. Nevertheless, we talk about multicast when a transmission serves several users, despite of the implementation of this transmission.

It must be pointed out that, in general, it may be necessary to synchronize the different video parts that make up a video during the playback. That is because they may be placed at different servers along the system. In our case the problems which could be caused by this are very limited. Note that all our transmissions are reserved in advance and there are no external ones so there cannot be congestion problems. Moreover, we use a time-slot of 60 s, which is fairy high compared with high speed transmission times. Additionally, users would probably have a minimum playback buffer in order to avoid sporadic glitches. Thus, using a time synchronization protocol in nodes (such as NTP [18]) would probably provide enough precision. Nevertheless, stream synchronization could be done in a number of ways by using protocols such as RTP [19] and RTSP [20]. In this paper we will not dig into details about how to do it.

4. Results

In this section we analyze storage size distribution in our tree structure. For doing this, we use the *starting delay* as the main metric, that is, the time between a request and the beginning of its video transmission. As we understand, the cost of a dedicated network of these characteristics is basically to deploy it (or contract a fixed amount of bandwidth), and also to acquire the videos and the storage size. We consider these aspects as a fixed cost, so that transmissions are made using these "already paid" resources. Following this point of view, if the video-on-demand company has invested in those resources, the cost of using or not using them is the same, and its goal is to offer the best possible response to its clients. That is why we test several scenarios (which vary the fixed system costs) and also focus on starting delays for evaluating the system performance.

Our experiments are the following. We firstly analyze how the system performance varies depending on the number of users and videos in two basic size distributions, which will give a preliminary view of the importance of caching in VoD systems. Then, we fix several scenarios (fixing both the number of users and vid-

[1] Considering the request rate presented above, a system with 5000 users/branch has no more than around 100 video requests each minute during the high audience peak, so the computational cost of such a time-slot is relatively low.

[2] In a real system, the procedure in a video rejection would simply be removing unused reservations.

When a new video request is received:

Set the video starting time to the next time-slot after the processing time.

While the video transmission is not reserved **do**

For each video part **do**

Calculate the transmission path of this video part and the exact time when it has to be transmitted.

For each link in the transmission path **do**

If there is a previous reserve for this video part in this link at the same time **then**

Add the video part transmission as a multicast.

Else

If there is available bandwidth for this video part in this link at the same time **then**

Reserve it.

Else

Cancel previous reservations of this request.

Set the video starting time to one time-slot further.

Exit both *for* loops.

Return the video starting time.

Fig. 3. Algorithm for the bandwidth reservation.

eos) and perform a series of simulations varying the size distribution. This will show which size distributions are better in each case, and also how better they are in relation to other ones. Next, we perform a bandwidth analysis of the system comparing several scenarios and size distributions, which will show why some are better than others. Finally, we present a mechanism for automatically managing the contents of our system while it is on-line.

4.1. Comparison of several known distributions

For our work, we will use a centralized system as a reference model. As it has been pointed in Section 1, both a proxy-like and an uniform size distributions improve the performance of a centralized system. Furthermore, it has also been pointed out that an uniform size distribution offers better results than a proxy-like one.

Figures in this section show the system performance when varying the number of users.[3] The following results have been grouped based on the number of videos in the repository. There are 1000 videos in Fig. 4, 5000 in Fig. 5 and 10,000 in Fig. 6. These three figures contain results in a centralized system, in a proxy-like size distribution (all storage size allocated in the caches close to users) and in a uniform size distribution (all caches have the same size). The total storage size in the cache system is 1400 GB, so in the uniform size distribution caches have 100 GB each one, and in the proxy-like size distribution, caches close to users have 175 GB and 0 size the other ones.

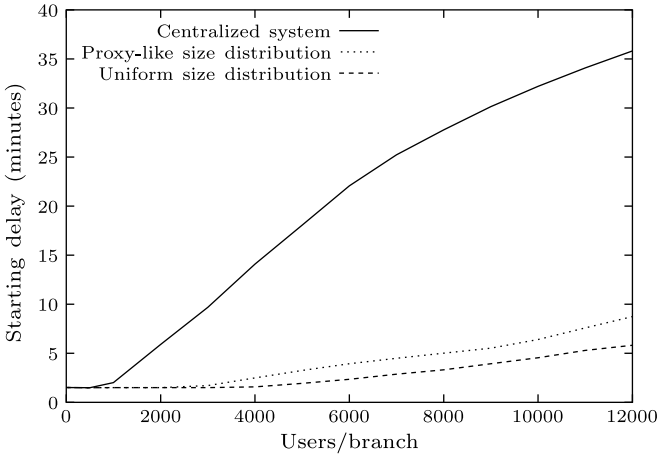

Fig. 4. System performance of a system with 1000 videos.

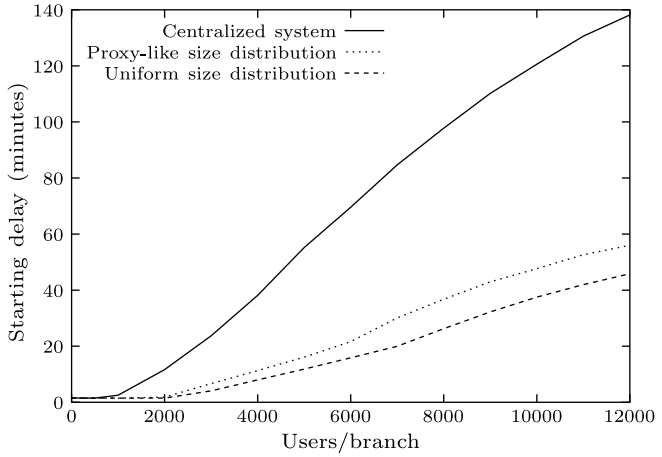

Fig. 5. System performance of a system with 5000 videos.

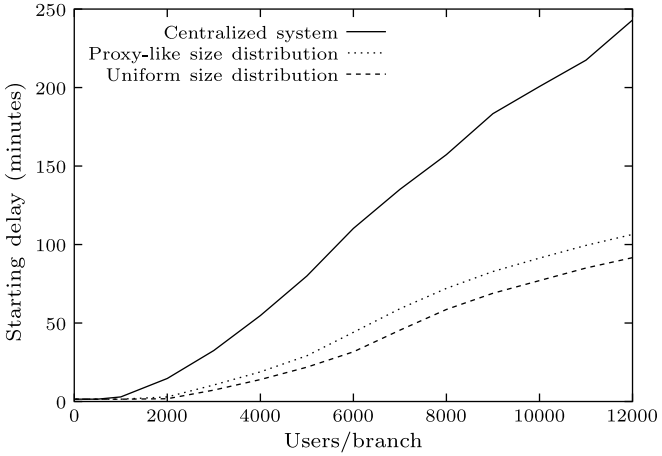

Fig. 6. System performance of a system with 10,000 videos.

In the first place it can be seen how the centralized systems performs much worse than the others. Uniform and proxy-like distributions perform in a similar way, being the uniform one better the proxy-like. The minimum delay is 1.5 min in all figures, which corresponds to the 60 s processing time plus half a time slot.

[3] The *x* axis is measured in *users/branch* instead of *users* because in this way it also provides information about how many users are sharing each final link. This tree has 16 final branches, so the total number of users can be obtained by multiplying by 16.

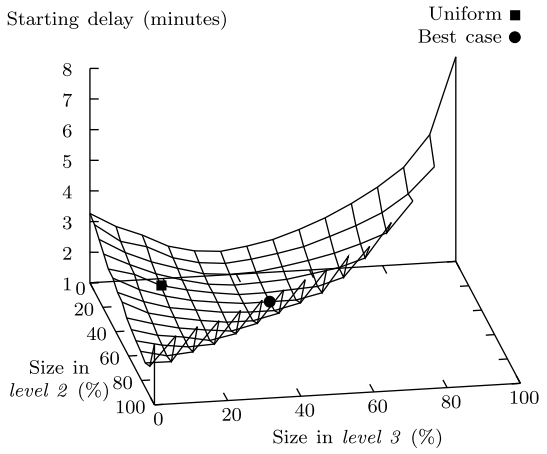

Fig. 7. Level size distribution for 5000 users/branch and 1000 videos.

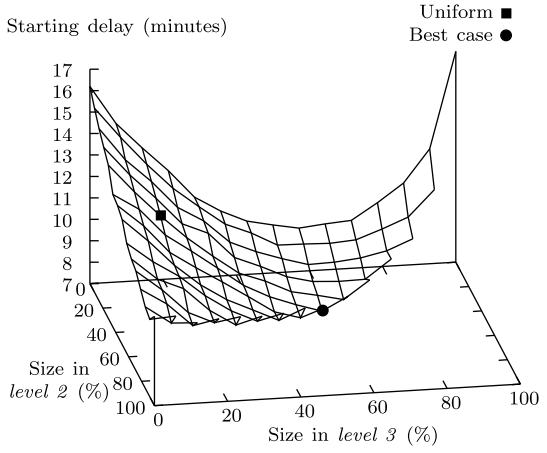

Fig. 8. Level size distribution for 5000 users/branch and 5000 videos.

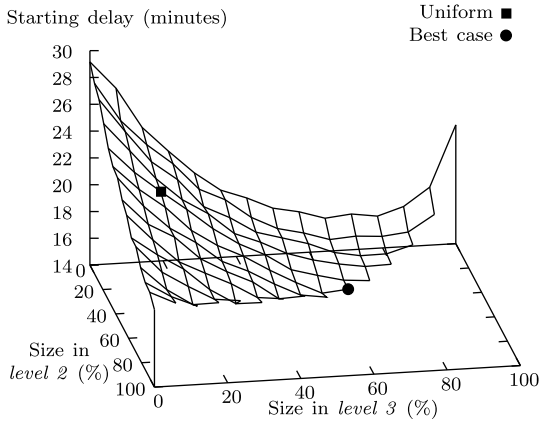

Fig. 9. Level size distribution for 5000 users/branch and 10,000 videos.

Although it is not clearly visible in these figures, plots present an inflexion point. It is located around 6000 users/branch in the proxy-like distributions, and around 7000 in the uniform one. We call these points *saturation* points, and they indicate the situation where the main server links reach an utilization of 100% independently of the time of day, i.e. these links are being used at full capacity always. A system with a load (n. users and videos) over these points should be considered as overloaded. However, this overload is translated into a faster increment of the starting delays, but not into rejecting requests. Moreover, the multicast capability limits the amount of transmission requests, because the number of videos awaiting for being transmitted will never be higher than the number of videos in the repository. This means that overloaded systems can still be usable, although it would not be recommendable. It can be seen in the next section how overloaded systems present delays far higher than usable systems. In the bandwidth analysis section we also discuss this situation.

4.2. Size distribution in caches

In this section we have performed a study which reflects the system behavior having a total size in caches of 1400 GB. We use configurations with 1000, 5000 and 10,000 different videos and with 5000 and 10,000 users in each branch (which corresponds to 80,000 and 160,000 total users, respectively). With these six configurations we cover a wide range of possibilities, from systems which can store most of their videos in caches and have low user load to systems with high user load and a large video repository Figs. 7–12.

Figures presented in the next sections must be understood as follows. Considering that we have a tree structure with three cache levels, *level 3* indicates the (two) caches close to the main server, *level 2* is the intermediate one, and *level 1* is composed by the caches close to users. Thus, one size distribution can be specified by the percentage of size in *level 3* and the percentage in *level 2*. The remaining percentage corresponds to *level 1* (note that the sum of these three percentages is always 100%). The uniform distribution point and the best case point are also marked in all figures. The proxy-like distribution is not indicated, but it corresponds to the coordinates (0,0), that is, 0% at *level 3* and 0% at *level 2*. Also note that the number of nodes in each level grow exponentially in a tree, so levels with the same size percentage do not have the same size in their caches.

In the first place it can be seen that all figures have a similar pattern. Increasing the number of videos tends to pull up the (0,0) coordinates, which represents the proxy-like distribution. This means that when the number of videos in the repository grow, distributions close to proxy-like perform worse. Moreover, note that in the figures with 5000 and 10,000 videos, the best case configuration is located almost on the diagonal of the plot. This means that the best results in systems with a large number of videos are obtained when the size in *level 1* (caches close to users) is very low. This is a very interesting detail because, having a total cache size of 1400 GB, in these cases *level 1* has only about 7% of the total cache size, which corresponds to 12.5 GB in each cache close to users. In contrast, when there are only 1000 videos, all of them can be stored in caches and therefore the size distribution is not so important, resulting in flatter surfaces.

About storage size in *level 3* (caches close to the main server), we can see that all figures show an important U-shape curve. This means that bad results are achieved both when there is no storage size next to the main server (because bottleneck remains in the main server links) and when all the storage space is assigned to these two caches (because popular videos are served from these caches and require bandwidth in most of the links in the tree to reach users).

Size in *level 2* (intermediate level) has a more simple pattern. It presents a quite straight line: the more storage size located at this level, the lower delays are obtained.

Therefore, these results should be scalable as follows. In the first place, caches close to the main server are the most interesting ones, due to they produce the basic pattern that delays will follow. Moreover, the storage size allocated in these caches depends basically on the number of videos in the repository. Once these caches are fixed, most of the remaining size should be allocated on intermediate caches. Finally, a very small amount of storage size is required in caches close to users, which will contain the hot videos.

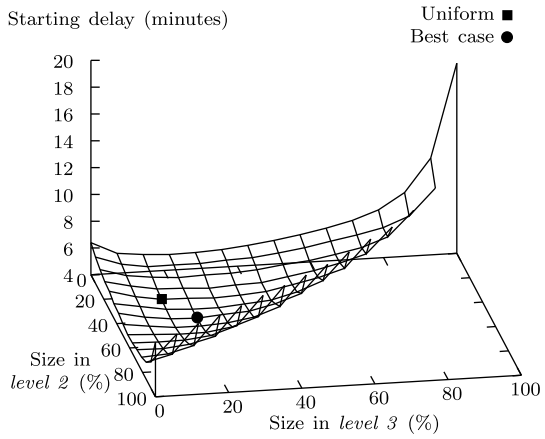

Fig. 10. Level size distribution for 10,000 users/branch and 1000 videos.

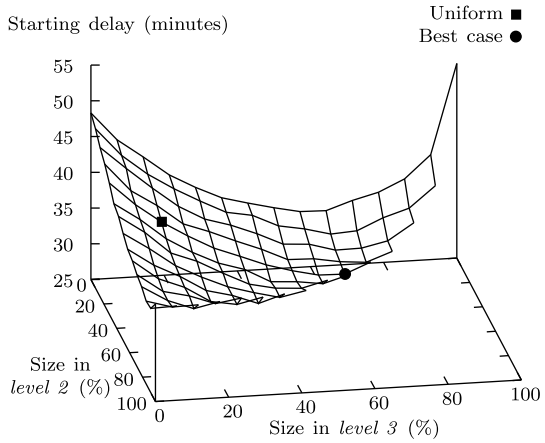

Fig. 11. Level size distribution for 10,000 users/branch and 5000 videos.

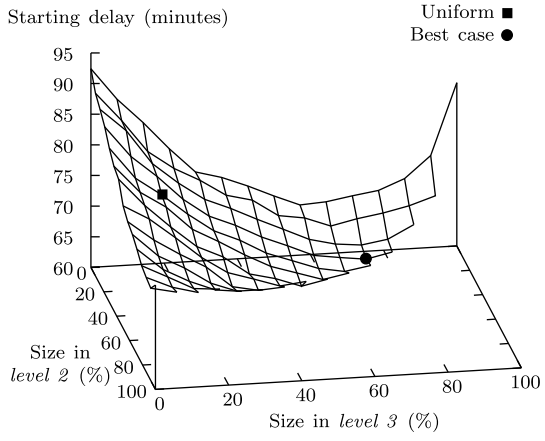

Fig. 12. Level size distribution for 10,000 users/branch and 10,000 videos.

Note also that both Figs. 11 and 12 are located over the saturation point, so they show overloaded systems. That is why their starting delays are so high.

4.3. Bandwidth usage analysis

As it has been pointed previously (see Fig. 2), the request rate varies through the day. Consequently, it seems quite reasonable to expect a different system behavior during a 24 h period. We have also shown in the previous section that different cache size distributions affect the final system performance. In this section, we analyze the system behavior through the day, focusing on the comparison between systems with uniform size distribution and with the best case distribution. This comparison shows the improvements in the bandwidth usage when we have an efficient size distribution.

For such a task, we take two systems with a total storage capacity in caches of 1400 GB and 5000 different videos, one with 5000 users in each final branch (i.e. 80,000 total users) and other one with 10,000 users in each final branch (160,000 total users). These two cases correspond to Figs. 8 and 11, respectively. Remember that this second system is an overloaded one, so it should not be desirable, but it allows us to show how a system would evolve when adding too much load to it.

Figures we show in this section represent the percentage of bandwidth usage in each level of the tree. We refer to the links which share each user branch (the ones between users and caches next to users) as *level 1* links. *Level 2* and *level 3* are the two intermediate link levels, and links at *level 4* are the two ones which communicate the main server with the two caches under it.

The first system is shown in Figs. 13 and 14. Fig. 13 represents a system where all caches have the same size, in this case 100 GB, and Fig. 14 is the same system with the best case cache size distribution.

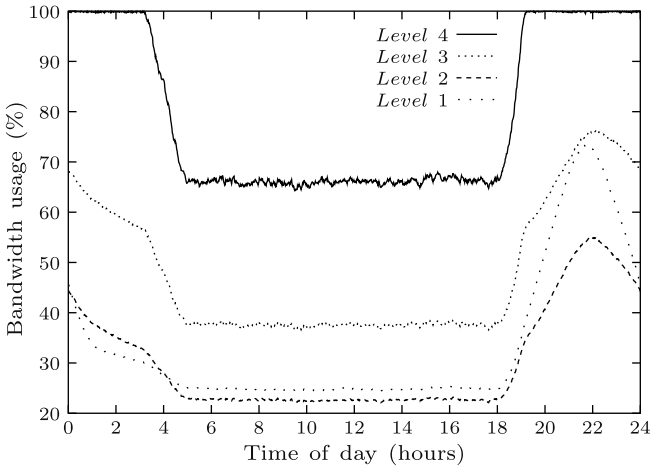

Fig. 13. Bandwidth usage with uniform cache size distribution for 5000 videos and 5000 users/branch.

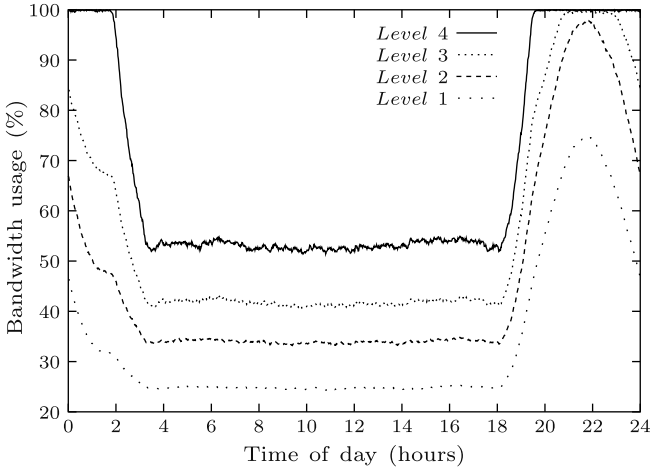

Fig. 14. Bandwidth usage with best case cache size distribution for 5000 videos and 5000 users/branch.

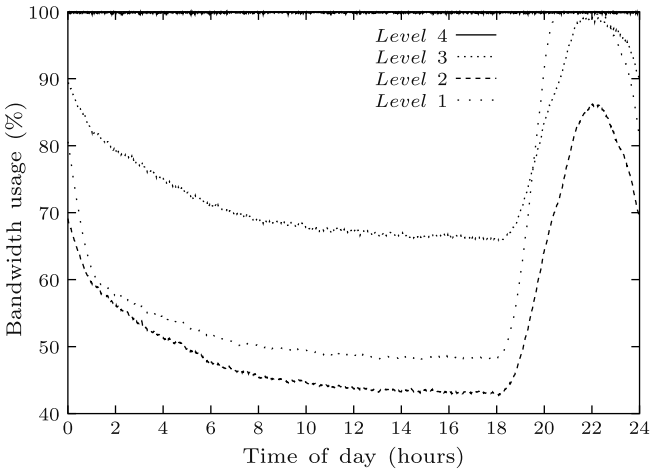

Fig. 15. Bandwidth usage with uniform cache size distribution for 5000 videos and 10,000 users/branch.

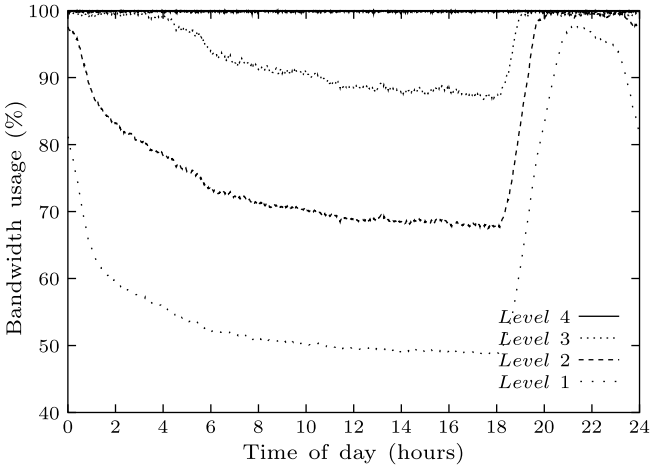

Fig. 16. Bandwidth usage with best case cache size distribution for 5000 videos and 10,000 users/branch.

One of the first differences we can see is the change in links utilization when the system is in a low load period. Whereas in Fig. 13 *level 4* has a high utilization Fig. 14 reduces links utilization at *level 4* by using more the links at *level 2* and *level 3*. This fact makes more uniform level loads and therefore the system in Fig. 14 can afford more load. In Fig. 13, *level 2* and *level 3* never reach utilizations above 55% and 75%, respectively. This indicates that in this system we are wasting much bandwidth. On the other hand, when the system has a high load with the best case cache size distribution, both these levels reach a very high usage.

As we said above, these simulations are made following the request rate of Fig. 2. During the high audience peak a high number there are a lot of requests and all them are served. Thus, both because of the duration of the requested videos and the delay introduced by the lack of available bandwidth, links usage is very high during some time after the high audience peak. This period can be seen during the first hours in all our bandwidth figures. However, in Fig. 14 this period is considerably shorter than in Fig. 13. This can be considered as a consequence of the previously pointed effects, resulting in a system with a size distribution which widely improves the uniform cache size distribution.

Figs. 15 and 16 represent a system with a very high user load. The first fact we can observe is that bandwidth used in *level 4* is always at 100%. *Level 4* is the one which connect the main server with the caches. In this case it is all 24 h saturated, which indicates that the main server is receiving more transmission requests than it can transmit.

In bottleneck situations, the saturated resource prevents other resources from being used as much as they could. Thus, one of the goals in these systems is to be able of using these other resources as well. This would be uniforming the use of resources in a non-saturated system, and it is what the best case distribution does. We can see that bandwidth usage in *level 2* and *level 3* is far more utilized in Fig. 16 than in Fig. 15. By the same reason, and like in Fig. 13, *level 2* bandwidth capacity is not being used as much as possible in Fig. 15 because it is always below 85%.

One last point to comment is the bandwidth usage in *level 1*. In our system, *level 1* corresponds to the links shared between users, and attached to the leaf caches. Thus, we can view links at this level as the path between the caches system and the users. With this point of view, it is easy to see that usage of links at this level is in some way independent to the server system, and just indicates the amount of bandwidth users are using for receiving videos. Of course this is not that simple, because the cache size distribution affects the delays in serving the requested videos, so bandwidth usage at *level 1* is also affected. However, it is reasonable to assume that *level 1* usage is independent of the cache size distribution. This is important because it limits the bandwidth required in this level. For example, we can see that both in Figs. 13 and 14, bandwidth at this level is always below 70%, so in order to reduce costs, we could use links with a capacity about a 30% lower in ılevel 1 without affecting the system performance.

However, comparing Figs. 15 and 16 we can see that *level 1* usage is not the same, it is a bit lower in Fig. 16 than in Fig. 15. This is because with a bad size distribution, there are many popular videos stored in caches close to users, so they are served using only links at *level 1*. This causes a congestion in that level. On the other hand, with a good size distribution, videos are located in higher caches, so their transmission can be done with a higher multicast degree. This, therefore, decreases the bandwidth usage in *level 1* increasing it in higher levels.

4.4. Delay distribution along a day

In previous sections we have seen how different size distributions affect our on-demand system, which distributions perform better and why perform this way. Here we present all this from a point of view near the users, where we can see how the system performs during a day. As above, we present two figures with 5000 users/branch and 10,000 users/branch of a system with 5000 videos. In Fig. 17 it can be seen how delays are the minimum possible when bandwidth is enough, and how they increase in the high audience period. These high delays decrease when the request rate descends and previous transmissions are being completed. However, in systems where the request rate is high (Fig. 18) delays cannot reach the minimum possible delay in this period because the request for transmissions exceeds the available bandwidth. This is why bandwidth usage at high levels is always around 100% and consequently delays are higher.

4.5. Reallocation of video streams

Whereas results in the previous sections are static, real applications usually require some mechanisms to manage the contents of the system. This is, basically, adding and removing videos, and changing the popularity of the existing ones. These changes should lead to automatically reallocate videos according to the new conditions.

In this section, we introduce a new management mechanism both for adding/removing new videos, and for changing the popu-

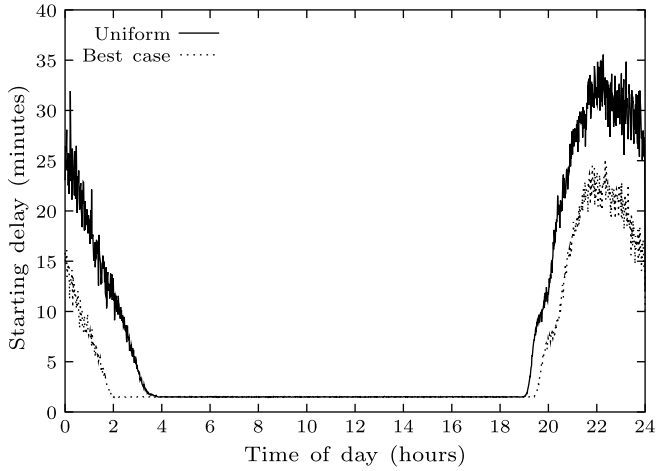

Fig. 17. Starting delay along a day with 5000 videos and 5000 users/branch.

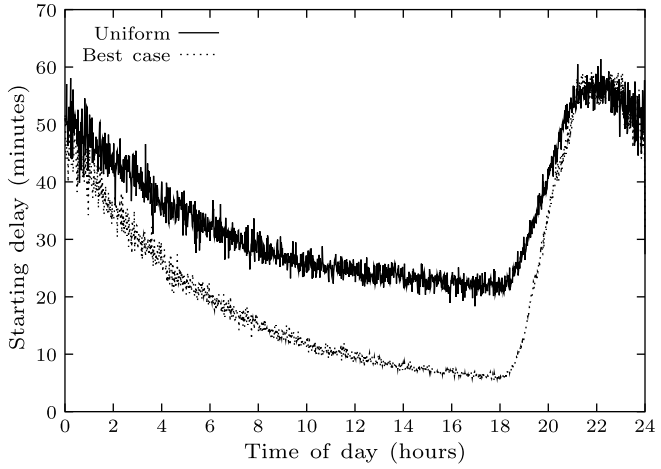

Fig. 18. Starting delay along a day with 5000 videos and 10,000 users/branch.

larity of the existing ones. Such a mechanism used a quite simple reallocation policy that, contrary to the previous distribution policy (which is intended to be used only at the very first time), can work on-line and in a transparent way to users. The first step is to find which video parts will need to be moved and where. Then, the reallocation process removes all video parts located at "incorrect" places, and transmits them downstream from the main server to the corresponding cache nodes.

However, the reallocation of video parts when the on-demand system is active can result in inconsistent services if the source data are being moved. Inconsistencies could be produced either when a requested video part has been removed and it is not yet allocated in its destination, or when a video part is removed while it is being transmitted to users. We solve that problem by marking the video parts which will be reallocated as unavailable (in their caches) during the reallocation time, being served by the main server during this time.

The main feature of this reallocation method is the speed; we can reallocate all affected video parts at the same time. Additionally, this method uses the same kind of downstream transmissions than the ones used to serve users, so no additional (upstream) requirements are needed. The obvious disadvantage is that all video parts are transmitted from the main server, which means a use of bandwidth directly proportional to the number of levels in the tree. However, such a reallocation can be done when the system load is low, so that we can afford "wasting" some bandwidth if that allows us to have a fast reallocation.

In order to show how our system behaves when we make use of our reallocation mechanism, we analyze the effect of adding a new video with maximum popularity to the system. This is the worst case, due to it will have to be located close to users and it will cause video movements in all levels. One detail in our study is that we maintain constant the number of available videos. This means that when we add a new video, another one (the less popular one, stored in the main server) is removed. This behavior allows a better analysis and representation of the obtained results, due to the number of videos in the system remains constant.

Figs. 19 and 20 show the bandwidth usage of a system with downstream reallocations at 12:00. Both these systems have 5000 videos and the best case storage size distribution. Fig. 19 has 5000 users/branch, and Fig. 20 has 10,000 users/branch. These parameters correspond to Figs. 14 and 16. It can be seen in both figures how reallocations represent a change in the bandwidth usage. In Fig. 19, the reallocation transmission uses some (unused) bandwidth, with no other effect on the system. In the system shown in Fig. 20, there is no unused bandwidth in the links attached to the main server, so the system behavior is different. In this case, reallocation transmissions will use a part of the bandwidth which would serve videos to users.

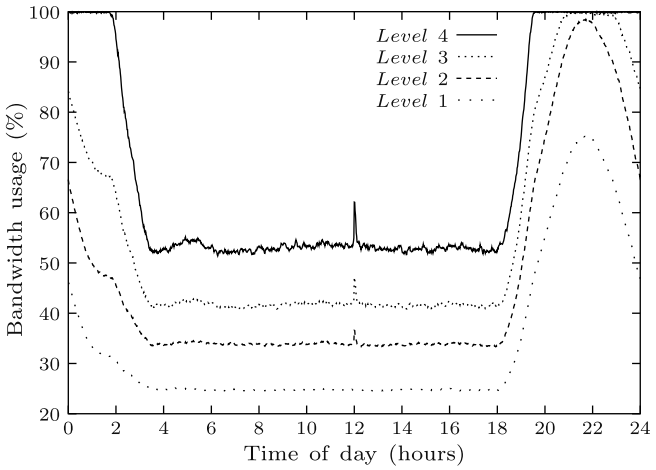

Fig. 19. Bandwidth usage with reallocations with best case cache size distribution for 5000 videos and 5000 users/branch.

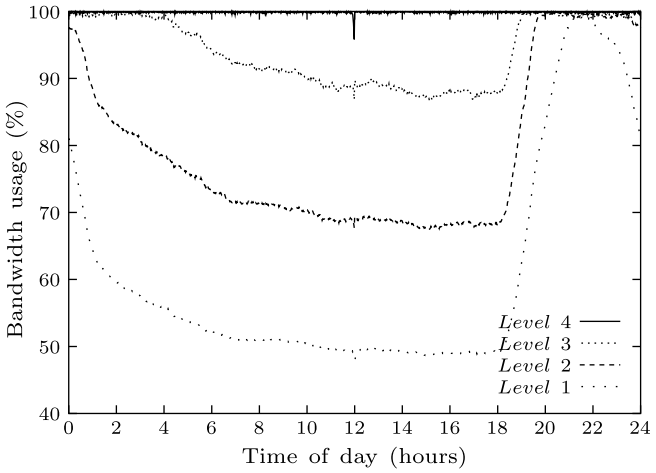

Fig. 20. Bandwidth usage with reallocations with best case cache size distribution for 5000 videos and 10,000 users/branch.

Thus, the system will be saturated earlier at reallocation time, and other requests will have to be reserved afterward. This can produce a light wasting of bandwidth the instants before reallocations. This effect can be seen in Fig. 20, where bandwidth usage decreases around 4% during the 2 min before the reallocations. A very light increment in starting delays at that time could also be expected. However, it is not numerically apreciable in our simulations.

5. Conclusions

In this paper we have carried out an analysis of the problem of storage size distribution. With this study we have obtained distributions which widely surpass the distributions proposed in previous works. Our best case distribution has a performance about 25% better than the uniform distributions. It also has improvements about 47% in comparison with proxy-like distributions. Moreover, our experiments show how the system performance changes when the size distribution is varied. We demonstrate that caches close to the main server are the ones which most significantly define the system performance. They produce a trade-off between not improving the main server bottleneck (when the caches close to this server are too small) and becoming a new bottleneck (when these caches are too large and receive most of the transmission requests). On the other hand, caches closest to users are the less important. Best case distributions have configurations with very low storage capacity in these caches. This cache level has about 7% of the total cache capacity, which corresponds to 12.5 GB in each cache close to users compared to a total cache size of 1400 GB. Finally, the behavior of size in intermediate caches is almost linear. The more storage size they have, the better performs the system.

We also demonstrate that bandwidth studies are important for understanding and improving the system. We have shown how it accurately reflects its state. This shows explicitly where the bottlenecks are and also which resources are being wasted.

Additionally, our reallocation policy provides a dynamic and automatic management of the videos stored in the tree in a way transparent to users. Also, used at low load periods, its bandwidth requirements do not represent a significant drawback.

Whereas the most critical resource is bandwidth (specially in those links attached to the main server) we have seen that the use of this resource depends very much on the storage size distribution, and obviously also on the place where each video is allocated. Thus, our study enlarges the efficiency of this kind of systems, allowing a considerable reduction of their response times.

Additionally, we have not used any improved transmission method in our studies. This means that our results represent a worst case. Using techniques such as stream merging [21,22] in combination with our storage allocation studies, bandwidth usage can be reduced in a very significant way.

References

[1] A. Abadpour, A.S. Alfa, J. Diamond, Video-on-demand network design and maintenance using fuzzy optimization, IEEE Transactions on Systems Man and Cybernetics 38 (2) (2008) 404–420.
[2] D.W. Brubeck, L.A. Rowe, Hierarchical storage management in a distributed video-on-demand system, IEEE Multimedia 3 (3) (1996) 37–47.
[3] A. Dan, D. Sitaram, A generalized interval caching policy for mixed interactive and long video workloads, in: Readings in Multimedia Computing and Networking, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2001, pp. 699–706.
[4] Y. Wang, Z.-L. Zhang, D.H. Du, D. Su, A network-conscious approach to end-to-end video delivery over wide area networks using proxy servers, IEEE/ACM Transactions on Networking 8 (4) (2000) 429–442.
[5] J. Segarra, V. Cholvi, Placement of storage capacity in distributed video servers, in: Proceedings of the IEEE International Conference on Communications, 2002.
[6] C. Vassilakis, M. Paterakis, P. Triantafillou, Video placement and configuration of distributed video servers on cable TV networks, Multimedia Systems (2000) 92–104.
[7] L. Berc, W. Fenner, R. Frederick, S. McCanne, Rpt payload format for jpeg-compressed video, Request for Comments 2035, Network Working Group, October 1996.
[8] D.L. Gall, MPEG: a video compression standard for multimedia applications, Communications of the ACM 34 (4) (1991) 46–58.
[9] W.-C. Feng, F. Jahanian, S. Sechrest, An optimal bandwidth allocation strategy for the delivery of compressed prerecorded video, Multimedia Systems 5 (5) (1997) 297–309.
[10] A. Solleti, K.J. Christensen, Efficient transmission of stored video for improved management of network bandwidth, International Journal of Network Management 10 (2000) 277–288.
[11] J. Rexford, D. Towsley, Smoothing variable-bit-rate video in an internetwork, IEEE/ACM Transactions on Networking (1999) 202–215.
[12] J. Segarra, V. Cholvi, Convergence of periodic broadcasting and video-on-demand, Computer Communications 30 (5) (2007) 1136–1141.
[13] C. Bisdikian, K. Maruyama, D.I. Seidman, D.N. Serpanos, Cable access beyond the hype: on residential broadband data services over HFC networks, IEEE Communications Magazine 34 (11) (1996) 128–135.
[14] B. Atlantic, Fact sheet: Results of Bell Atlantic video services. Video-on-demand market trial, Trial Results, 1996.
[15] P. de Haar et al., DIAMOND project: video-on-demand system and trials, European Transactions on Telecommunications 4 (8) (1997) 337–344.
[16] L. Breslau, P. Cao, L. Fan, G. Phillips, S. Shenker, Web caching and Zipf-like distributions: evidence and implications, in: Proceedings of the IEEE Infocom'99 Conference, 1999.
[17] A. Dan, D. Sitaram, P. Shahabuddin, Dynamic batching policies for an on-demand video server, Multimedia Systems 4 (1996) 112–121.
[18] J. Levine, D.L. Mills, Using the network time protocol (NTP) to transmit international atomic time (TAI), in: Proceedings of Precision Time and Time Interval (PTTI) Applications and Planning Meeting, 2000.
[19] H. Schulzrinne, S. Casner, R. Frederick, V. Jacobson, RTP: a transport protocol for real-time applications, Request for Comments RFC 1889, Network Working Group, 1994.
[20] H. Schulzrinne, A. Rao, R. Lanphier, Real-time streaming protocol (RTSP), Internet Draft, IETF, 1997.
[21] A. Bar-Noy, R.E. Ladner, Efficient algorithms for optimal stream merging for media-on-demand, SIAM Journal of Computing 33 (5) (2004) 1011–1034.
[22] D.L. Eager, M.K. Vernon, J. Zahorjan, Minimizing bandwidth requirements for on-demand data delivery, Knowledge and Data Engineering 13 (5) (2001) 742–757.