Dynamic Proxy-Cache Multiplication Inside LANs

Claudiu Cobârzan*

"Babeş-Bolyai" University, Computer Science Department,
Mihail Kogălniceanu 1, 400084 Cluj-Napoca, Romania
claudiu@cs.ubbcluj.ro

Abstract. Proxy-cache deployment in LANs has become a current practice with well known benefits. For situations when a proxy-cache comes under constraints, due to increased load, and has to drop requests or perform cache replacement, we propose the alternative solution of cache-splitting. This means to dynamically deploy additional proxy-caches inside the LAN, and divert towards them some of the requests addressed to the original proxy-cache(s). By doing this, better response time, load balancing, higher availability and robustness of the service can be achieved than when using a single proxy-cache.

1 Introduction

The constant increase in both volume and demand of multimedia data in the Internet tends to stress the existing infrastructure. The main factors are the characteristics of multimedia data (e.g. size, bandwidth requirements) which highly differ from those of typical web data. The traditional way to cope with such situations is to deploy proxy-caches at LAN edges. Under certain conditions a single proxy-cache does not suffice, so multiple proxy-caches have to be used. Cooperative caching has been introduced for web caches, e.g. Harvest [4] and Squid [18], and for video caches as well, e.g. by Brubeck and Rowe [3] and MiddleMan [1].

Our paper proposes a novel proxy-cache system that is able to "spawn" new proxies via split operations whenever the actual situation demands it. Examples of such situations include extremely high load and severe storage constraints on the proxy. In those cases, one additional proxy-cache in the LAN would help lower the load on already running proxy-caches as well as increase the capacity of the "federate" cache.

The system we propose dynamically adjusts the number of running proxies in the LAN, depending on the load and on client request patterns, by either spawning new proxies on periods with high activity or putting them in a "hibernate" state or even stopping them on periods with low activity.

* The work was done during the stay at Klagenfurt University, Austria, within the CEEPUS (Central European Exchange Program for University Studies) framework, in winter 2004-2005.

J.C. Cunha and P.D. Medeiros (Eds.): Euro-Par 2005, LNCS 3648, pp. 890–900, 2005.

2 Proxy-Cache Splitting

There are situations when having a single proxy-cache in the LAN does not suffice, for example when servicing large, popular content to many clients, or when the volume of requested data puts the proxy-cache under constraints (cpu, mem, storage, etc.). In those cases, requests have to be rejected in order to lower the load on the machine and cache replacement has to be performed in order to free disk space. We state that in some circumstances it would be more beneficial to just deploy an additional proxy-cache inside the LAN. This new proxy-cache could take over some of the load on the existing proxy-cache(s) and by doing so, avoid both request dropping and cache replacement. On the other hand, if current and maybe predicted future load could be handled by a smaller number of proxy-caches than those currently active, then some of them could enter a "hibernating" state or could be shut down (stopped).

The distributed architecture we propose, assumes the deployment of two types of entities: the `dispatchers` and the `daemons`. The `dispatchers` are processes/threads that run on the same node as the proxy-cache and can be seen as front-ends of the proxy-caches which:

- *handle incoming requests* - serve them either from the local cache, or from the origin server; if this is not possible, the requests are forwarded to other active `dispatchers`/proxies in the LAN or they are discarded; a request is forwarded to the best candidate (the `dispatcher` that has a cached copy of the requested object or the one with the smallest load);
- *manage the proxy code* - archive the proxy code and send it to the location on which a new proxy-cache is to be spawned (using the `daemon` running on the selected target)
- *manage the "child" proxy-cache processes* - the `dispatchers` are responsible with stopping/ pausing/ restarting a "child" proxy-cache depending on various conditions (global load, volume of the clients' requests, volume of streamed/stored data, etc.)

The `daemon` processes/threads run in the ideal case on every node of the LAN and are responsible for:

- *managing clients' requests* - the `daemon` either directly receives, or it intercepts the client requests and then decides to forward them to the appropriate proxy-cache, depending on the local available knowledge about the global state of the proxy-cache "federation"
- *managing the proxy code* - the `daemon` receives/compiles the code sent by a `dispatcher` that initiates a proxy split operation;
- *managing the local proxy-cache process* - stops/ pauses/ restarts it, either as a result of incoming requests from its "parent" proxy, or depending on specific local conditions (load, storage capacity).

In the case the `daemon` thread/process hosted on a certain node crashes, the clients from that node still have access to the federate cache as long as the proxy-cache(s) set as default in the client's browser is/are still running. This is also

true for the clients from nodes with no running `daemons` at all. The `daemon` is essential in the proxy-cache splitting process, as it is used by the "parent" to transfer its code if it is not already available at the selected node.

2.1 The Model of the Proposed Distributed Proxy-Cache Architecture

We consider that the number of nodes in the LAN is **n**. Let **P** be the set of available proxy-caches (there is at least one running proxy cache in the LAN):

$$P = \bigcup_{i=1}^{k} P_i, k = |P|, 1 \leq k \leq n$$

A proxy-cache $\mathbf{P_i}$ is defined as follows:

$$P_i = (maxResources_i, minResources_i, LC_i), i = 1..k$$

where:

- $maxResources_i$ - represents the maximum amount of resources that can be used by the proxy-cache:
  $maxResource_i = (maxCpu_i, maxMem_i, maxCapacity_i, maxLan_i)$
  namely the maximum amount of CPU power, memory, storage space and external bandwidth;
- $minResources_i$ - represents the minimum amount of resources that have to be used in order to serve any client's request. It is defined in a similar mode with the $maxResources_i$;
- LC_i - the content of the local cache
  $LC_i = \{c_{ij}, j = 1..q\}$, q = the number of cached objects

An object c_{ij} is defined as:

$$c_{ij} = (size(c_{ij}), timeLastAccess(c_{ij}), hitCount(c_{ij}), qualityValue(c_{ij}))$$

where $size(c_{ij})$ is the size of the object, $timeLastAccess(c_{ij})$ indicates the last time the object has been requested, $hitCount(c_{ij})$ shows the number of times the object has been served from the cache and $qualityValue(c_{ij}) \in [0..1]$ is the measure of the object's quality (based on the actual characteristics of the video object, such as resolution, color information, etc.)

The $qualityValue$ is a relative value that shows the degree in which the cached object matches the desired quality of a certain class of users. A value equal or close to 1 corresponds to the objects that have exactly or almost the desired quality, while values close to 0 are assigned to objects that show the most drastic difference between actual and desired quality. High absolute quality does not necessarily mean that the $qualityValue$ is close to 1. For example, if the vast majority of the users have only limited display size, say 800x600, a video object encoded at 1280x1024 will have a qualityValue closer to 0 than to 1, because further operations (e.g. transcoding) have to be performed in order to deliver the object to the requesting clients.

For each object c_{ij}, a *utility* value can be computed using a function $u : LC_i \rightarrow \mathbf{R}$:

$$u(c_{ij}) = const_1 * size(c_{ij}) + const_2 * \frac{1}{timeLastAccess(c_{ij})} + \\ + const_3 * hitCount(c_{ij}) + const_4 * qualityValue(c_{ij})$$

where $const_1, const_2, const_3, const_4 \in [0, 1]$ and $const_1 + const_2 + const_3 + const_4 = 1$ ($u(c_{ij})$ is computed as a weighted average of the different characteristics of the cached video object).

Those constants can be fixed when the proxy-cache is started and remain the same during the run period of the proxy-cache. Another possibility that needs further investigation would be to dynamically modify those values when traffic conditions, load level, request rate, etc. reaches certain values, in order to maximize the byte hit ratio. The utility value of the cached objects is used to decide which objects get discarded when performing cache replacement.

We use **D** to denote the set of `dispatchers`:

$$D = \bigcup_{i=1}^{k} D_i, k = |P|, 1 \leq k \leq n.$$

As each `dispatcher` corresponds to a certain proxy-cache, there is a function f (bijection), $f : D \rightarrow P, f(D_i) = P_i, \forall i \in \{1, .., k\}$ (a proxy P has exactly one `dispatcher` D). One `dispatcher` $\mathbf{D_i}$ is defined as follows:

$$D_i = (P_i, GC, GU, siblings_i), \forall i \in \{1, ..., k\}, k = |P|, 1 \leq k \leq n$$

where:

- P_i - the corresponding proxy
- GC - the content of the global cache (viewed as the union of all local caches)

$$GC = \bigcup_{i=1}^{k} LC_i, \forall i \in \{1, ..., k\}, k = |P|, 1 \leq k \leq n$$

- GU - the *utility* values for the objects in GC

$$GU = \bigcup_{i=1}^{k} LU_i, \forall i \in \{1, ..., k\}, k = |P|, 1 \leq k \leq n$$

  where LU_i = the set of utility values for the objects in LC_i
  $LU_i = \{u(c_{ij}) | c_{ij} \in LC_i, j \in \{1, ..., q\}, q = |LC_i|)\}$, $i \in \{1, ..., k\}$
- $siblings_i$ - the rest of the running proxies/dispatchers ($siblings_i = P \setminus \{P_i\}$).

We denote by **A**, the set of `daemons`, ideally running on each node of the LAN.

$$A = \bigcup_{i=1}^{n} DA_i$$

There is a function g (bijection), $g : [1..n] \rightarrow A, g(i) = DA_i, \forall i \in \{1, .., n\}$ which assigns each node in the LAN a running `daemon`. One `daemon` $\mathbf{DA_i}$ is defined as follows:

$$DA_i = (P\prime_i, LOAD(P\prime_i), \bigcup_{p \in P\prime} MLC_p(m)), \forall i \in \{1, ..., n\}$$

where:

- $P\prime_i$ - a subset of the proxy-cache set P ($P\prime_i \subseteq P$)
- $LOAD(P\prime_i)$ - the load of the proxy-caches in the subset $P\prime$

$$LOAD(P\prime_i) = \bigcup_{p \in P\prime} LOAD(p)$$

  where $LOAD(p)$ represents the current load of the proxy $p \in P\prime$
- $MLC_p(m)$ - the most "useful" m objects stored in the cache $p \in P\prime$

$$MLC_p(m) = \bigcup_{i=1}^{m} c_{ij}, u(c_{ij}) \geq u(c_{ij+1}), \forall j \in \{1, ..., q-1\}$$

  where c_{ij} represents the cached object and $u(c_{ij})$ the value returned by the utility function defined above for the object.

2.2 Proxy Splitting Scenarios

As mentioned before we intend to perform a splitting operation under two conditions: when the proxy-cache is under storage constraints or under load constraints. The question is how to decide that a splitting operation is more appropriate than performing cache replacement or reject the incoming requests? We propose the following two conditions:

A. In the Case of Storage Constraints
If $\forall i \in \{1, .., k\}, \forall m, s \in \{1, .., q\}(m \neq s), k = |P|, q = |LC_i|$

$$|u(c_{im}) - u(c_{is})| < \delta \tag{1}$$

then perform splitting, else perform cache replacement.

In other words, splitting is performed when all cached objects are essentially "equally" useful - the difference between the utility of all objects in the cache is smaller than a certain fixed limit δ. The condition could be relaxed, if considering that not for all, but for a certain fraction of the cached object set, the above mentioned condition holds.

If the condition does not hold, than cache replacement should be performed with regard to the utility of the objects. As an observation, if $const_1 = const_3 = const_4 = 0$ then the cache replacement strategy is basically LRU (Least Recently Used), and if $const_1 = const_2 = const_4 = 0$, the cache replacement strategy is LFU (Least Frequently Used).

We present a short example of how the values of those constants could influence the decision of making either a split operation or perform cache replacement. Consider that the cache contains only 5 objects with the characteristics described in Table 1. Consider now Table 2 with four values configurations for the constants that appear in the definition of the utility function (see Subsection 2.1).

Table 1. Characteristics of the cached objects

Cached objects	Size (MB)	Time of last access	Hit count	Quality value
c_1	100	1	60	1
c_2	100	5	70	1
c_3	100	10	80	1
c_4	100	20	90	1
c_5	100	30	100	1

Table 2. Values for the coefficients used by the *utility* function u

Configuration	$const_1$	$const_2$	$const_3$	$const_4$
$conf_1$	0.25	0.25	0.25	0.25
$conf_2$	0.10	0.40	0.40	0.10
$conf_3(LRU)$	0	1	0	0
$conf_4(LFU)$	0	0	1	0

The graphical representation of the utility values corresponding to the data in Table 1 and Table 2 can be seen in Figure 1.

It can be seen that the decision to perform either cache replacement or a split operation highly depends on the value configuration of the coefficients. For example, if $\delta = 15$ and the proxy is under storage constraints, cache replacement will be performed if configuration 3 or 4 are used, but a split operation will be initiated if configuration 1 or 2 are considered.

B. In the Case of Load Constraints

When servicing a request for an object c_{ij} a certain amount of resources must be available. If $\forall P_i \in P$ the available resources are not enough to service a request r_i, than r_i is discarded and the particular time t_i is marked.

If $\forall i \in \{1, .., p-1\}$ (p fixed) we have

$$t_{i+1} - t_i < \xi \tag{2}$$

(the time interval between p consecutive discarded requests is smaller than a fixed threshold ξ), than we make a split operation.

It is to investigate in a real time environment how different values for δ and ξ influence the dynamics of the system.

2.3 Additional Costs Induced by the Proposed Architecture: Best-Case/Worst-Case Scenarios

Inside the system, the message exchange cost can be viewed with regard to the required time to transmit a message, with regard to the amount of data that is transferred, or as a combination of the two (both time and data volume).

In the following, we give a short analysis of the best/worst case scenarios from the point of view of the latency perceived by the client. For a similar analysis on the amount of data transferred within the system, please see [5].

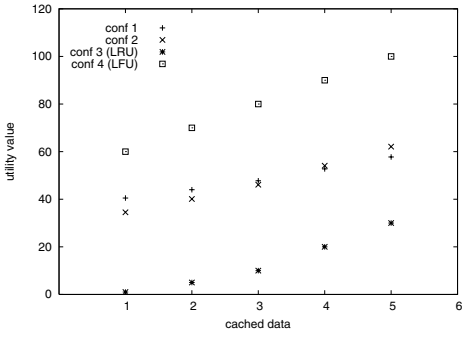

Fig. 1. The utility values for the cache configuration from Table 1 computed using the constants values from Table 2

The delay perceived by the client depends on the delay introduced by the LAN communication, the one introduced by the WAN communication, as well as on the delay introduced by searching the local caches and the server repository.

We make the following notations:

- $Delay$ - the total delay as perceived by the client
- d_{lan} - the delay introduced when transmitting a message in the LAN;
- d_{wan} - the delay introduced when transmitting a message in the WAN;
- d_{cache} - the delay introduced when searching the local cache;
- d_{server} - the delay introduced when searching the server repository/performing admission control;
- $d_{timeOut}$ - the time out interval fixed for the proxy-server communication

We propose the following forwarding algorithm for requests passed from one proxy to another inside the LAN: when a proxy receives a request, it first checks the local cache and returns the appropriate object in case of a hit. Otherwise (local miss) it checks the list with cached objects at siblings' sites in order to see if the requested object is cached in the federate cache. If it does, it marks the request and sends it to the appropriate sibling. The decision to forward a request to a certain proxy is made based on the locally available information on the global state of the federate cache. It may happen that this information is outdated and that by the time a forwarded request reaches the sibling, the requested object does not exist anymore on the sibling site. The *worst case* would be when a request received by a `dispatcher` D_i is forwarded from one dispatcher to the other until it returns to D_i. In this case, supposing that the client didn't cancelled the request, it is forwarded by D_i to the origin server S.

Suppose there are k active proxy-caches, and using the above mentioned notations, we distinguish the following two *worst cases*, when it comes to the user perceived latency:

- bouncing request and server down

$$Delay = (k+1)d_{lan} + kd_{cache} + d_{timeOut}$$

– bouncing request and server can't serve incoming requests

$$Delay = k(d_{lan} + d_{cache}) + 2(d_{lan} + d_{wan}) + d_{server}$$

The *best case* is of course when the first proxy receiving the client request, can serve it from the local cache. In this case we have:

$$Delay = 2d_{lan} + d_{cache}$$

Assuming the following two configurations, *conf1* with $d_{lan} = 0.1, d_{cache} = 0.001, d_{wan} = 0.5, d_{server} = 0.005$, and *conf2* with $d_{lan} = 0.01, d_{cache} = 0.001, d_{wan} = 0.05, d_{server} = 0.005$ (measurements in seconds) the maximum introduced delay in the case up to 11 proxy-caches are active inside a LAN is showed in Figure 2.

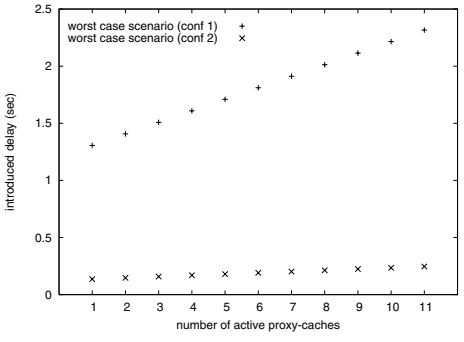

Fig. 2. Maximum introduced delay

It can be seen from the above example that, if the load on both proxy-caches and server(s) is more or less constant, then only variations in network conditions (local and external) makes the delay vary. In a well connected high speed LAN/WAN, the more realistic configuration would be one similar to *conf2*, but when no control over internal/external network can be assumed, a configuration like *conf1* is very probable. The values for *conf1* and *conf2* were measured at Klagenfurt University during normal working hours.

This means that even without constraints regarding the available external bandwidth, it is highly probable for the maximum number of active proxy-caches to be limited by the additional latency that would be induced in the worst case scenario. This holds especially if the network conditions are not very good (we have high induced latencies for both LAN and WAN) as the delay is highly dependable on those conditions.

We have performed a series of simulation experiments [6] using synthetic log traces generated with WebTraff [11]. Figure 3 shows the variation of byte-hit ratio with the number of active proxy-caches inside the LAN (after up to four split operations). The log we used for this particular simulation consisted of 1000 requests following a Zipf distribution with $\alpha = 0.3$ for a number of 300 video

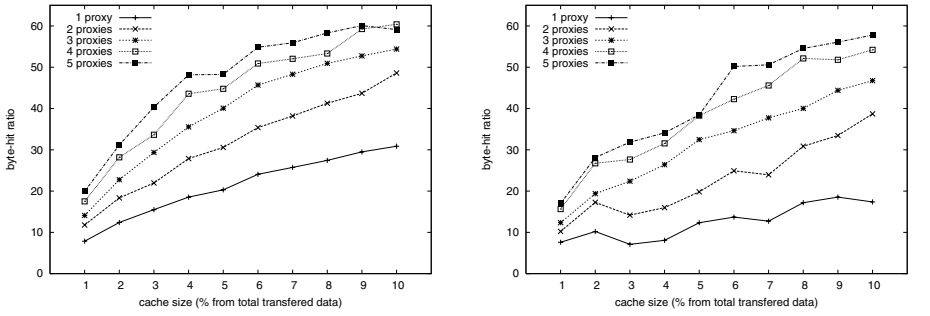

Fig. 3. The variation of the byte-hit ratio with the number of active proxy-caches and cache size. **(Left)** Cache replacement strategy set to LRU. **(Right)** Cache replacement strategy uses the utility values of the objects

objects. From those, 70% were one-timers while their size was approximately 3GB and followed a Pareto distribution with the tail index set to 1.2.

We simulated two replacement strategies, LRU and one strategy that used the utility values of the cached objects (objects with the lowest utility values are discarded when cache replacement has to be performed). The utility of the cached objects was computed with the value 0.25 set for all four coefficients. It can be seen that as the size of the deployed caches increases so does the byte-hit ratio but, more important, the values obtained when using the two above-mentioned cache replacement strategies are pretty close. Figure 3 also seems to suggest that the benefits obtained from adding new proxy-caches inside the LAN tend to diminish as the number of active proxies increases (the increase in byte-hit ratio is greater when moving from 1 to 2, or even from 2 to 3 active proxies than it is when moving from 3 to 4 or from 4 to 5 active proxies).

There is a trade-off between costs and benefits, the best *cost/benefit* ratio seems to be achieved at a moderate number of proxies, as Figure 2 and Figure 3 suggest. We intend to validate this assumption in a real time environment, once our implementation of the system (which is based on the existing implementation of the QBIX proxy-cache [16]) is completed.

3 Related Work

The last few years have brought an increasing interest in video caching as a result of the rising popularity and availability of multimedia content on the Web. The vast majority of the research concentrates on partial video caching, approach that considers specific parts of videos or is done with respect to the quality of the videos. Examples of proposals for partial video caching include caching of a prefix [17], caching of a prefix and of selected frames [10], caching of a prefix combined with periodic broadcast [8], caching of hotspot segments [7]. Other approaches consider the caching of a prefix based on popularity [12], segment-based prefix caching [19] or variable sized chunk caching [2].

Quality based video caching proposals include periodic caching of layered coded videos [9], adaptive caching of layered coded videos in combination with congestion control [14], quality adjusted caching of GoPs (group of pictures) [15] or simple replacement strategies (patterns) for videos consisting of different quality steps [13].

Regarding distributed video caching we have, among others, the work of Brubeck and Rowe [3] proposing multiple video servers accessible via the web and which manage tertiary storage systems as well as the MiddleMan [1] system which proposes a cooperative caching video server.

Our proposal, though having similarities with that in [1], differs from previous work by the fact that our system is dynamic and able to adjust the number of running proxy-caches in the LAN in a fully distributed fashion depending on a number of factors including current load, storage constraints, request patterns.

4 Conclusion and Future Work

We have presented a distributed proxy-cache architecture which aims at providing better service to LAN clients. The feature that distinguishes our proposal from those made in the past is the dynamic characteristic of our system which is able to adapt itself to changes in access, request and response patterns as well as to changes in network conditions.

Future work will focus on finishing the implementation of the system, evaluating its performance in real-life situations and compare the performance with the case in which a single proxy-cache is used. Other points of interest are represented by the conditions triggering the split, hibernate and shut down operations. Another interesting problem is what happens in a system like the one we described, when multiple outgoing links with different capacities are available.

Acknowledgements

I would like to thank my supervisor from Klagenfurt University, prof. dr. Laszlo Böszörmenyi for his constant help, support and guidance during my research work.

References

1. Acharya, S., Smith, B.: Middleman: A Video Caching Proxy Server. In: Proceedings of the 10th International Workshop on Network and Operating System Support for Digital Audio and Video (2002)
2. Balafoutis, E., Panagakis, A., Laoutaris, N., and Stavrakakis, I.: The impact of replacement granularity on video caching. In: IFIP Networking 2002. Lecture Notes in Computer Science, vol. 2345. Springer-Verlag, Berlin, Germany, (2002) 214-225
3. Brubeck, D.W., Rowe, L.A.: Hierarchical Storage Management in a Distributed VOD System, In: IEEE MultiMedia, Fall 1996, Vol. 3, No. 3

4. Chankhunthod, A., Danzig, P, Neerdaels, C., Schwartz, M., Worrell, K.: A Hierarchical Internet Object Cache. In: Proceedings of the 1996 USENIX Technical Conference (1996)
5. Cobârzan, C., Böszörmenyi, L.: Dynamic proxy-cache multiplication inside LANs. Technical Reports of the Institute of Information Technology, University Klagenfurt, TR/ITEC/05/2.02
6. Cobârzan, C., Böszörmenyi, L.: Measurements on byte-hit ratio variation in LANs deploying multiple proxy-caches. Technical Reports of the Institute of Information Technology, University Klagenfurt, TR/ITEC/05/2.05
7. Fahmi, H., Latif, M., Sedigh-Ali, S., Ghafoor, A., Liu, P., Hsu, L.H.: Proxy Servers for Scalable Interactive Video Support. In: IEEE Computer, 43(9): (2001) 54-60
8. Guo, Y., Sen, S., Towsley, D.: Prefix Caching Assisted Periodic Broadcast for Streaming Popular Videos. In: Proceedings of ICC (International Conference on Communications) (2002)
9. Kangasharju, J., Hartanto, F., Reisslein, M., Ross, K.W.: Distributing Layered Encoded Video through Caches. In: Proceedings of IEEE INFOCOM (2001)
10. Ma, W.-H., Du, D.H.-C.: Reducing Bandwidth Requirement for Delivering Video over Wide Area Networks with Proxy Server. In: IEEE International Conference on Multimedia and Expo, (2000) 991-994
11. Markatchev, N., Williamson, C.: WebTraff: A GUI for Web Proxy Cache Workload Modeling and Analysis. In: IEEE International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunications Systems, Vol. 10, (2002) 356-363
12. Park, H. S., Chung, K.D., Lim, E.J.: Popularity-based Partial Caching for VOD Systems using a Proxy Server. In: Workshop on Parallel and Distributed Computing in Image Processing, Video and Multimedia (2001)
13. Podlipnig, S., Böszörmenyi, L.: Replacement strategies for quality based video caching. In: Proceedings of the IEEE International Conference on Multimedia and Expo (ICME). Vol. 2. IEEE Computer Society, Piscataway, NJ, (2002) 49-53
14. Rejaie, R., Kangasharju, J.: A Quality Adaptive Multimedia Proxy Cache for Internet Streaming. In: Proceedings of the International Workshop on Network and Operating Systems Support for Digital Audio and Video (2001)
15. Sasabe, M., Wakamiya, N., Murata, M., Miyahara, H.: Proxy Caching Mechanisms With Video Quality Adjustment. In: Proceedings of the SPIE Conference on Internet Multimedia Management Systems (2001) 276-284
16. Schojer, P., Böszörmenyi, L., Hellwagner, H, Penz, B., Podlipnig, S.: Architecture of a quality Based Intelligent Proxy (QBIX) for MPEG-4 Videos, World Wide Web Conference, (2003) 394-402
17. Sen, S., Rexford, J., Towsley, D.: Proxy Prefix Caching for Multimedia Streams. In: Proceedings of the IEEE INFOCOM99. (1999) 1310-1319
18. Wessels, D.: Web Caching. O'Reilly, 2001
19. Wu, K.-L., Yu, P.S., Wolf, J.L.: Segment-Based Proxy Caching of Multimedia Streams. In: Proceedings of the Tenth International World Wide Web Conference (2001)