DVS: a System for Distribution and Management of Global Video on Demand Services (extended abstract)

Alexey Roytman, Israel Ben-Shaul, Israel Cidon*
Department of Electrical Engineering
Technion - Israel Institute of Technology
{aroytman, issy, cidon}.ee.technion.ac.il

Abstract

The Distributed Video Service (DVS) system provides a comprehensive solution for the management and distribution of global Video on Demand. DVS presents a scalable architecture that combines the global accessibility of the Internet with an acceptable quality of service through an optional private high-speed backbone, and by distributing movies off-line to servers that are closer to potential viewers. To cope with dynamic changes in movie demand and server load, DVS employs adaptive algorithms for server selection and for movie distribution. DVS was implemented in Java, integrated with a front-end playing software, and supports IP and ATM as the public and private networks, respectively.

1. Introduction and motivation

The availability on the Web of multimedia content in general, and video content in particular, has been rapidly growing recently. It is easy to see the huge potential in having a large-scale motion video base that is accessible through the public Internet as a source of information, entertainment, and education. Indeed, in the near future it is expected that multimedia providers will attempt to offer high quality *global* services, while minimizing the operational cost of their services.

There are two important characteristics that impact the design of a scalable global VOD service.

1. High bandwidth and high quality of service (QoS)
2. Very large size and storage requirements

These two characteristics present an inherent conflict from the system design perspective. To address the first requirement, it is desirable to distribute video content as close as possible to end-users by arranging servers in different geographical regions to enable users to access them with high bandwidth and low delay. This is particularly important when considering the unpredictable Internet for delivering the contents to end-users. The second requirement means, in contrast, that content distribution and replication is expensive, both in terms of communication as well as storage. Clearly, storage considerations favor a centralized approach, whereby video is played from remote servers, although load-balancing at least among co-located servers (possibly sharing the same movie base) must be considered (see [1] for such a solution). Thus, the effectiveness of a global system is largely determined by a solution that strikes a balance between these two requirements.

Another important property of the use of VOD is that the demand changes, both temporally (e.g., different movies are watched at different times of the day, and movie popularity changes over time), as well as spatially (e.g., different regions favor different movies). This implies that *dynamic* management of content distribution, placement and playing is highly desirable.

The Distributed Video Service DVS system addresses these challenges through a scalable distributed architecture, server selection (both in Section 2.) and movie distribution algorithm (Section 3.), which together provide managemant and distribution of video content among geographically-dispersed servers. DVS was implemented in Java and is integrated with a front-end video player software.

*also with Sun Microsystems Labs, CA

2. DVS Architecture

The networking infrastructure of DVS is bi-level, consisting of a private high-speed intra-server network (e.g., a 155Mbps fiber ATM network) with users connecting to servers through the public Internet. The high-speed network is mainly used for real-time playing of movies to remote users, through local Internet proxy servers to which users are attached. That is, the attachment point and the video source need not be the same node. This design greatly increases the flexibility in server selection and movie placement, but could be expensive if all nodes were required to be attached to the private network. Thus, we only require a small subset to be directly connected to the private network, as will be explained below. This also means that inter-server management and off-line movie transfer also operate over the public network.

The need to provide regional distribution and management of content implies a hierarchical structure of the intra-server network, with the edge nodes providing the actual VOD service and manager (internal) nodes performing resource management and allocation. Each region consists of a flat set of servers, and we make the reasonable assumption that the distance/bandwidth from each server in the same region to all users is roughly the same.

Each first level manager maintains both static and dynamic information on its managed servers. The static information consists of: maximum load of each server; a boolean flag that indicates whether the server can directly connect to the private network; and the server's storage capacity. The dynamic information consists of: the current load on each server, the demand per movie, and a global list of movies in the group, with each movie associated with a list of servers in the group that hold a copy of the movie. Managers at the second level and up keep similar information on demand and movie list. The demand of an internal node represents the sum of the demands on all its sub-nodes, recursively, and the movie list represents the union of the movie lists of its children, with each movie associated with a list of *children* nodes with a copy of the movie. As we go up the hierarchy the list of movies increases, but the entry per movie is kept small since it contains only (immediate) children nodes, not actual descendants with copies of movies. This in turn implies that a movie request at an internal node entails a traversal down the hierarchy in order to find the specific server with a copy of the movie, but only log N steps are needed.

The freshness of the dynamic information in first level managers is kept directly by the server selection and movie distribution algorithms. In contrast, managers above the first level need to poll their children periodically in order to stay up to date.

The movie playing service has two phases. 1) The user requests from DVS (via one of the regional servers) to watch a specific movie, DVS selects the proper server and returns its contact address to the user's client. 2) The client contacts the selected server and the movie is played to the client over the network from the remote server.

The selection process consists of at most three phases. First, if there are servers in the region of the client with a copy of the requested movie, the server with the lowest *relative load* is selected for playing, where the relative load is defined as the ratio between the server's current load (plus one) and its maximum load.

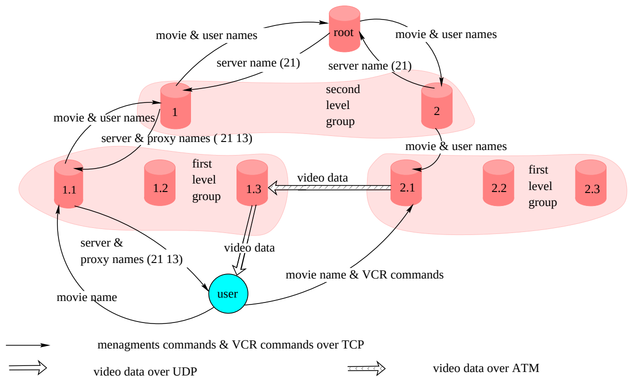

Figure 1. Sample movie requests

If no server can be matched in the local region, the second phase begins. Figure 2. illustrates an example of the second phase. The manager looks for the least-loaded server that can act as a proxy server, i.e., one that is connected to the private network. In the example, the selected proxy is named `1.3` (server naming is beyond the scope of this paper, see [2]). If a proxy is found, the manager requests from its own manager (server `1`) at the next level up to locate a remote server that has a copy of the requested movie and is also able to connect to the private network. The upper manager forwards the request to all other children nodes, in parallel. If such a server is found (`2.1` is such a server), DVS returns to the client a pair of addresses: the proxy's address, and the remote server's address. The client then contacts directly the remote server (`2.1`) over the public Internet and requests it to deliver the movie through the proxy (`1.3`). Notice that while the movie is transferred to the client through the proxy (in order to improve QoS and reduce costs), the client still communicates with the remote server, not with the proxy. This design enables the user to control the session parameters and issue VCR control operations such as pause, rewind, fast-forward etc. If no server is found by any of the sibling first-level managers, the request is forwarded one level up, recursively, until a server is found or until the root is reached. Thus, each iteration expands the range of the search.

The third and least desirable option is when even an available proxy server cannot be found. In this case the request is forwarded to the next-level manager as-is, in the hope to find a remote server that can still play the movie directly to the client through the public Internet with an acceptable quality of service.

There are three potential problems in the above procedure. First, a security problem might arise since there is no guarantee that the user will follow the system instructions (selected servers) or it might change its prior preferences (such as movie selections). The second problem is with potential discrepancy between the actual server loads and the information on their load as known at the regional manager. In particular, the load might change during the course of the request protocol, due to the possibly long delays between the assignment of servers to clients and their actual playing.

The second problem stems from the fact that the protocol actions may be interleaved with other concurrent requests from other clients. Thus, a standard solution involving locks on resources of the assigned server with a defined timeout is employed. Since typically the VOD service time is large, the wasted resource time of this locking operation is negligible. To address the security problem, the assigned servers may be passed user and session information which is kept with the lock and with other resources allocated for that user request. Upon the reception of the expected request, the server verifies its correctness and validity.

The third problem is reliability. The hierarchical structure of the intra-server network implies that each internal node is a single point of failure. DVS addresses this issue by allowing to grant dynamically the "management" role to each of the nodes in the level below it (using a leader election algorithm) as opposed to restricting it to a single dedicated node. A high priority can still be given to hosts that are intended to serve as managers, but this role can be temporarily assigned to other servers when a default manager fails.

3. Content distribution

The movie distribution process operates as follows. Each server maintains information about the *demand* per movie (i.e., number of requests in a time unit), and sends the information to its manager. The sum of the demands is collected and passed up by managers to higher level managers. The goal is to redistribute the movies to servers based on the demand and their (limited) storage capcity.

This task can be mapped to the "multiple knapsack 0/1" problem, which is known to be NP-hard. [3] describes a sub-optimal greedy algorithm with complexity $O(n^2)$, where n is the the number of movies.

We have adpated this algorithm in three aspects. First, since the maximum number of users that are supported by each server is bounded, it may be useful to keep multiple copies of popular movies in a single group. Thus, we have modified the greedy algorithm of [3] to the case of multiple choice constrains knapsack problem. Second, knapsack problems assume empty knapsacks whereas our servers generally store as many movies as possible, so the manager needs to also decide to *remove* movies from servers. Finally, since (off-line) transfer of a movie over the network is a costly operation, our algorithm also attempts to minimize the amount of movie transfers. The complete movie distribution algorithm is presented in [2].

4. Implementation

We have implemented all management activities of DVS in Java 1.1. Given that the computation cost is relatively small compared with the communication cost involved in the management activities (movie distribution and server selection), the current performance of Java did not present a practical problem. In contrast, we benefited greatly from its platform independence, built-in multi-threading support, and networking support (using RMI).

The current implementation supports TCP/IP as the

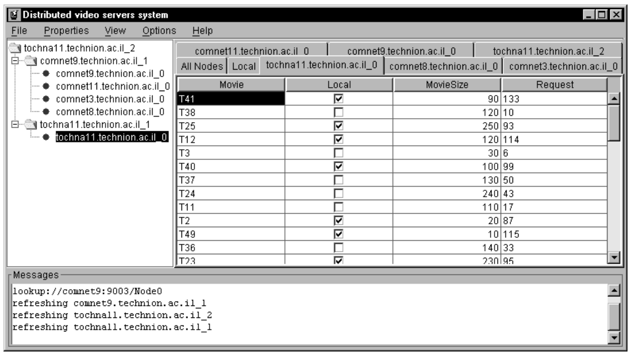

Figure 2. The DVS monitor

public network, and ATM as the high-speed network. VCR is mostly a "back-end" content manager. In order to actually play the video to end-users we have integrated the public domain VCR player from the Oregon Graduate Institute [4], and extended it with the capability to redirect its output to ATM, which is needed when the movie is played to a to a proxy over the private network).

Finally, DVS provides a runtime monitor that is capable of tracking dynamically various properties such as changing user demands and server loads, on all DVS servers. Figure 2. shows a snapshot of the monitor's GUI. The left frame lists the server groups and their hierarchical structure (two groups of 4 and 1 members, respectively), and the right frame shows the movie distribution in a selected server, `tochna11`, and the demand per-movie in that server. The monitor can be also used for manual movie transfer between servers, for human tuning and for testing and experimentation.

References

[1] Renu Tewari, Daniel Dias, Rajat Mukherjee, and Harrick Vin. Design and performance tradeoffs in clustered multimedia servers. In *In the Proceedings of IEEE-ICMCS -Tokyo*, June 1996.

[2] A. Roytman, I. Ben-Shaul, and I. Cidon. Dvs. a system for distribution and management of global vodeo on demand services. Technical report cc-270, Technion - Israel Institute of Technology, February 1999.

[3] S. Martello and P. Toth. *Knapsack Problems. Algorithms and Computer Implementations.* Wiley, 1990.

[4] S. Cen. *A Distributed Real-Time MPEG Video Audio Player.* ftp://ftp.cse.ogi.edu/pub/dsrg/Player.