πDTV: A Client-Based Interactive DTV Architecture

Edward Y. Chang
Department of Electrical & Computer Engineering
University of California, Santa Barbara
echang@db.stanford.edu

Abstract

In this study we present a client-based architecture that supports time-shift Digital-TV (DTV) features (i.e., pause, replay and fast-forward) and interactive applications. We call this architecture πDTV for Personalizable Interactive Digital-TV architecture. Our study focuses on devising effective techniques for managing the client's data (e.g., textual, binary, and video/audio data) under the local resource constraints. We outline the challenges that πDTV faces and sketch data and resource management policies that maximize the client's QoS adaptively based on each individual viewer's quality requirement. We present a prototype that have been built and describe an array of applications that πDTV can help realize.

1 Introduction

Many studies have proposed server-based interactive DTV architectures (e.g., [1, 2, 3, 5, 6, 7, 8, 9, 10, 11, 12]), in which a server schedules its resources (e.g., memory and disk bandwidth) to service interactive VCR-like requests, such as pause, replay and fast-forward. In a server-based architecture, the clients are assumed to be passive—simply receiving bits and rendering frames. However, because of the typical long end-to-end transmission delay between a server and a client and the Interet's limited bandwidth, it is practically infeasible for the server to support "real-time" VCR-like interactive operations for tens of thousands of simultaneous users. Furthermore, on a broadcast channel (e.g., CNN), one simply cannot request the server to pause or to replay a program.

In this study, we present a client-based interactive DTV architecture. We call this architecture *πDTV*, for Personalizable Interactive DTV. Equipped with a large and inexpensive disk, a πDTV client can cache a large amount of media data [4]. This economical caching together with the random access capability of the disk enables a πDTV client to support time-shift operations. A viewer can pause a live DTV program to take a break from viewing while the broadcast stream continues arriving and being written to the local disk. The viewer can resume watching the program after the pause with a delay, or fast-forward the program to get back in synch with the broadcast stream. These time-shift functions allow one, for example, to do one's own instant replay during a sports program or to watch a remote lecture at one's own pace. One can also record programs at multiple channels at the same time (which a tape-based VCR cannot do), filter out unwanted content and produce a customized program.

Adding an Internet back-channel in addition to the disk to a πDTV client allows TV content to be customized in many ways. While a given broadcast stream remains the same for all viewers, various data objects can be transmitted to individual viewers via the Internet channel. Data objects can be textual (e.g., an HTML page), binary (e.g., a java appelet), or continuous (e.g., a video/audio stream). Image-based rendering techniques can overlay a number of data objects with the broadcast stream to provide a customized program. For instance, a viewer can query and change the attributes (e.g., color, shape, position and history) of these data objects. A broadcaster can target individual families with tailored advertisement via the back-channel. Many other applications exist (more examples are listed in Section 5).

Managing heterogeneous DTV data objects (e.g., video, audio, texts, 3D graphics objects, etc.) at the client side to support interactive DTV features faces two major challenges: 1) different clients may have different available resources and 2) different viewers may have different viewing preferences. A resource manager at the client side must allocate and schedule available resources adaptively to maximize each individual viewer's quality requirement. In this paper, we thus propose techniques to manage DTV data objects under the constraints of the local resources to maximize the client's QoS. We start with proposing a quantitative model that describes the size and latency characteristics for each data object. A viewer can assign a QoS factor to a data object to convey that object's scheduling priority to the resource manager. The resource manager then schedules the local resources for the data objects in an order that maximizes the total QoS. Through quantitative analysis, experiments and implementation we show that our client-based architecture can support interactive DTV features effectively with very low memory requirement and hence at a low cost. We believe that this client-based approach is a practical architecture to realize interactive DTV for both point-to-point and broadcast networks.

The rest of this paper is organized as follows: Section 2 depicts a typical πDTV pipeline. In Section 3 we characterize πDTV data objects. Section 4 describes how resources are scheduled to maximize QoS under constraints. Section 5 presents the digital-VCR prototype that we have bulit. Finally, we offer our conclusion in Section 6.

2 System Overview

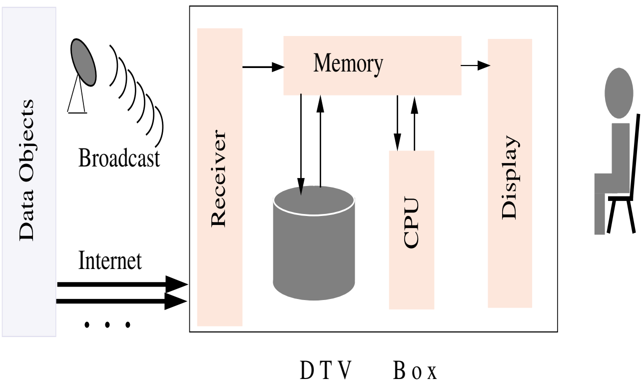

Figure 1: The πDTV Pipeline

Figure 1 depicts a πDTV pipeline. On the left-hand side of the figure, πDTV data objects are transmitted via broadcast and point-to-point channels to a πDTV box. The πDTV box receives network packets in its main memory. If the data are not needed shortly (e.g., the viewer paused the playback), the data are written to the box's local disk to conserve memory. The data are made available to the CPU before they are needed. The CPU processes (e.g., computes, decodes, renders, etc.) the data in main memory, and finally the processed data are played back to the viewer on the right-hand side of the figure.

A πDTV pipeline consists of two major components: πDTV data objects and system resources, including CPU, memory, and a local disk. (We assume that a πDTV box has only one local disk for economical reasons.) In the following sections we characterize πDTV data objects and system resources in more detail.

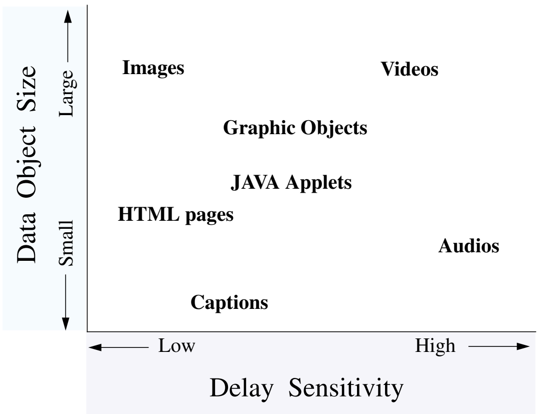

Figure 2: The Characteristics of πDTV Data.

3 Characterizing Data Objects

Data objects can be continuous, textual, binary, and so forth. They are best characterized by their *sizes* and *latency constraints*. Continuous data are audio/video streams, which can be voluminous (video streams) and delay sensitive. Textual data include captions, HTML and XML documents, etc. Textual data in general occupy far less space than continuous data and are not delay sensitive. Other data, such as images and applets, can be very copious or scant and may or may not be delay sensitive, depending on the nature of the application. Figure 2 shows the characteristics of these data objects. The x-axis in the figure represents the delay sensitivity from low (on the left-hand side) to high (on the right-hand side); the y-axis gives the size of an object. At one extreme, videos are large and very sensitive to delay. At the other extreme, captions are small in size and not very sensitive to delay.

The size of a data object is measured by the amount of storage space it needs and is denoted by S_i, where i stands for the i^{th} object. (We show how S_i is computed for a continuous data object in [4].) To quantify the delay sensitivity of a data object, one can specify a value function for it. The function can be defined in many ways: Figure 3 shows four representatives. The x-axis in the figure depicts delay; the y-axis depicts the value of the object normalized to from zero (worthless) to one. Function F_a depicts an object that can tolerate delay up to a period of time, t_δ, then becomes worthless. Function F_b shows the value of an object that decays linearly after t_δ. Functions F_c and F_d have other value decay patterns. Function F_a is a good value function for an audio frame. Function F_b may be a good value function for a video frame since slight delay of a frame display may be tolerable. Function F_b may be a good value function for a subtitle. Function F_d, which shows that a user's patience decays exponentially after t_δ, may be a good value function for a Web page. Without losing generality, we define the value function for object i as $f_i(t)$, where t denotes the span between the time the object is requested and the time the object is available in main memory for processing.

In short, we characteristize data objects as follows:

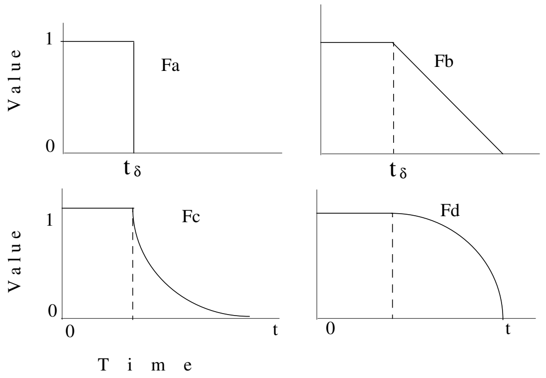

Figure 3: Delay Function Examples.

- S_i represents the i^{th} object's storage requirement, and
- $f_i(t)$ depicts the i^{th} object's delay sensitivity.

4 Scheduling Resources

The local resources include CPU, memory, disk bandwidth and network bandwidth. The local resources may not be adequate to support a particular interactive scenario. Therefore, it is critical for the resource manager to be adaptive to the resource constraints and to degrade, if degradation is unavoidable, in a graceful manner. Given limited resources, the design objective of a πDTV box is to prioritize resource allocation in order to maximize the user's satisfaction.

To measure a user's satisfaction, we must allow the user to have a say about what is important and what is not. We thus associate each data object with a *QoS parameter*. A viewer can assign a QoS value to each data object, either implicitly or explicitly. For instance, if a user decides to turn off all interactive features, the resource manager can assign a QoS factor of zero to data objects that are needed to support interactive features. Regarding the data objects that are needed to support the playback, higher QoS can be assigned to mission-critical data objects, such as broadcast video and audio streams, and lower QoS can be assigned to optional data objects such as captions and applets.

Given N_{all} requested data objects and M available memory, the goal of the resource manager is to schedule N data objects ($N \leq N_{all}$) to maximize the total QoS under the resource constraints. We use α_i to denote the QoS requirement assigned to the i^{th} data object ($0 \leq \alpha_i \leq 1$) and τ_i to denote the latency needed to stage the i^{th} data object in main memory. (The value of τ_i depends on the local disk bandwidth and network bandwidth, and the size of the data object. We show how to determine τ_i in the full version of this paper.) Let j represent the scheduling order for N data objects. We would like to schedule the service to these N data objects in the order that maximizes the sum of the QoS. The objective function can be written as:

$$Max \sum_{j=1}^{N} \alpha_j \times f_j(t - \sum_{k=1}^{j} \tau_k), \qquad (1)$$

Figure 4: A Digital-VCR Screen Shot.

which is subject to the memory constraint

$$\sum_{j=1}^{N} S_j \leq M.$$

We show how the optimal schedule can be solved efficiently in the full version of this paper.

5 A Digital-VCR Prototype

To demonstrate that the πDTV architecture is practical, a Digital-VCR has been built. The prototype is implemented on a Pentium II 450 MHz Windows 98 workstation with a Quantum Viking II disk and 256 MBytes DRAM. The main memory and disk are linked with a SCSI fast bus and managed by an Adaptec AIC-$789x$ based PCI Ultra2 SCSI controller, both of whom have much higher data transfer rates than the disk.

The Digital VCR supports VCR-like interactive functions including *pause*, *fast-forward* and *instant replay*. Figure 4 shows a screen shot of the user interface (i.e., *codec*) and display. At the bottom of the figure is the codec through which a viewer issues interactive commands and monitors the system. To issue a command, a viewer selects and presses one of the control buttons on the upper-left corner of the codec (described shortly). To monitor the system, the codec uses trackbars to display various statuses. The five trackbars from the top to the bottom on the codec keep track of the amount of memory use, the amount of disk use, the accumulated number of hiccups, the playback progress, and the broadcast progress.

When a viewer watches a TV program live, memory is used to buffer the mismatch between the data transmission rate and the playback rate. The memory trackbar keeps track of the current memory use. The disk use is zero when a program is watched live. When the pause button is pressed, the excessive data that the DRAM cannot stage is written onto the disk. Using the disk as a part of the cache, the VCR can allow the viewer to pause for a substantial amount of time (a 9.1 GBytes disk can buffer more than one hour of 19.2 Mbps HDTV program). After returning from the break, the viewer can press the *play* button to resume the playback. At this time, the decoder consumes data from the disk while the broadcast data continue arriving and being appended to the tail of the disk file. The viewer has the option to perform *fast-forward* to catch up with the live program. When the fast-forward button is pressed, the Digital VCR retrieves data from the disk at a faster pace and the decoder decodes and displays key frames only. The data cached on the disk are eventually used up and the playback returns to the live mode.

The viewer can replay from the start of a recorded video sequence by pressing the reply button. We envision that to support user-friendly replay, the Digital VCR should automatically generate thumbnails to help the viewer select a desired video sequence (e.g., a sequence showing McGwire's record breaking homerun). Implementing video indexing and enhancing the replay function are in the project plan.

Potential Applications

The Digital-VCR prototype supports VCR-like features without the need to interfere with the server. This is because the large local disk provides an inexpensive buffer to cache a large amount of data to support interactive features. The followings are some applications that can already take advantage of this prototype (without an Internet back-channel):

- Self-paced distance learning: One can view a lecture at one's own pace.
- Personalized instant replay: One can replay a video or an audio segment of a live program.
- Personalized news program: One can record news playing on multiple channels at the same time and compose a customized program of ones desire (e.g., skipping all financial news).

As discussed in Section 1, with image-rendering techniques, πDTV can allow the viewer to influence TV content at the frame level. For example, a virtual museum program may deliver data at very high rates but the client decodes, stitches, and displays only the frames that the user views. Children would be able to change the attributes (e.g., colors, size, position and even the personality) of the "bunny," "big bird" and other characters in the *Sesame Street* program. A grade school student can learn American states by interacting with a map. Adding an Internet back-channel in addition to a large disk to the πDTV box makes many more applications possible: e.g., customized news, soaps, ads, and enhanced home-shopping channels.

6 Conclusion

We have presented a client-based πDTV architecture that supports personalizable interactive DTV features. The πDTV architecture adds a large disk and a back-channel to the broadcast DTV channels at the client side and manages data objects under local resource constraints to maximize each client's quality requirement. The Digital-VCR prototype shows that πDTV can support interactive operations effectively. Its data management and resource scheduling components are being enhanced currently to support some advanced applications mentioned in the paper. We believe that πDTV is a practical approach that can turn the passive TV viewing experience into a personalizable and interactive one.

7 Acknowledgements

I would like to thank professors Hector Garcia-Molina and Patrick M. Hanrahan at Stanford University for their helpful comments and suggestions. I would also like to thank Serge Rutman and Milton Chen at Intel for their hardware/software support in building the digital-VCR prototype.

References

[1] E. Chang and H. Garcia-Molina. Bubbleup - Low latency fast-scan for media servers. *Proceedings of the 5th ACM Multimedia Conference*, pages 87–98, November 1997.

[2] E. Chang and H. Garcia-Molina. Effective memory use in a media server. *Proceedings of the 23rd VLDB Conference*, pages 496–505, August 1997.

[3] E. Chang and H. Garcia-Molina. Reducing initial latency in media servers. *IEEE Multimedia*, 4(3):50–61, July-September 1997.

[4] E. Chang and H. Garcia-Molina. Medic: Memory and disk cache for media servers. *Proceedings of IEEE International Conference on Multimedia Computing and Systems*, pages 493–499, June 1999.

[5] E. Chang, H. Garcia-Molina, and C. Li. 2d BubbleUp - Managing parallel disks for media servers. *Proceedings of the 5th Foundations on Data Organization*, pages 221–230, November 1998.

[6] M.-S. Chen and P. Yu. Storage and retrieval methods to support fully interactive playout in a disk-array-based video server. *ACM Multimedia Systsmes*, 3(3):126–35, July 1995.

[7] S. Ghandeharizadeh. Continuous retrieval of multimedia data using parallelism. *IEEE Transactions on Knowledge and Data Engineering*, 5(4):658–69, 1993.

[8] T. Johnson and A. Zhang. A framework for supporting quality-based presentation of continuous multimedia streams. *Proceedings of the 4th IEEE Conference on Multimedia Computing and Systems*, pages 169–176, June 1996.

[9] R. Ng and J. Yang. Maximizing buffer and disk utilizations for news on-demand. *Proceedings of the 20th VLDB Conference*, pages 451–462, 1994.

[10] B. Ozden, A. Biliris, R. Rastogi, and A. Silberschatz. A low-cost storage server for movie on demand databases. *Proc. VLDB*, September 1994.

[11] B. Ozden, R. Rastogi, and A. Silberschatz. A framework for the storage and retrieval of continuous media data. *Proc. IEEE Multimedia*, pages 2–13, May 1995.

[12] Y. Wang, J. Liu, D. Du, and J. Hsieh. Efficient video file allocation schemes for vod services. *ACM Multimedia Systems*, 5(4), September 1997.