Design of a Modular Adaptive Virtual Video Server Architecture for On-Demand Video Services

Roland Tusch
Abstract

The unique characteristics of digital video streams concerning huge storage, high-bandwidth and real-time delivery requirements impose challenges in designing video server architectures. Efficient means for storing, retrieving, manipulating and sending video data are required, both concerning the server’s hardware and software. In this paper, a design of a modular video server architecture for use in a virtual environment is presented. The server’s components are finally mapped to services. This enables the server to adapt to different situations by migrating or replicating critical services to other machines. Thereby the quality criteria \textit{load-balancing} and \textit{scalability} are improved, but the performance behaviour of the system is still an open question.

\textbf{Keywords:} video server modules, stream and server-level adaptivity, virtual server
1 Introduction

The design of large-scale, on-demand video servers has been a well-discussed research area within the last years. Many proposals of server-architectures focused on design- and performance models for either single-server or distributed server environments. In many situations the distributed server-model outpaces the single-server model, but in certain circumstances this is not the case. This often depends on the missing flexibility of the server architectures concerning their ability to adapt themselves to different environmental conditions given by the nature of digital video streams. The storage, processing and delivery of videos imposes many challenging requirements for a large-scale, on-demand video server architecture, both on the level of hardware and of software.

Considering the hardware, huge amounts of disk and memory-space are required for storing and manipulating the streams. A high-speed I/O-subsystem, as well as a high-speed network-connection are necessary for real-time video delivery. On the software-level, a generic architecture of software components is required, performing tasks for an efficient storage, retrieval, manipulation and delivery of video data in large quantities.

However, an aspect not considered so far is the adaptivity of a video server not only concerning the delivery of videos with different qualities, but also migrating and/or replicating a server-component to a different location in certain circumstances. If e.g. a server has to stream the same video to a number of clients located far from the source-server, but closely aligned to each other, then it would make sense to do the buffering somewhere in the neighborhood of those clients. In order to be able to move a server-component to a different location, the server’s software architecture has to be designed modular. Moreover, it should be possible to move code or services from one machine to another, if e.g. the video needs to be decoded there before sending it to the clients. Movable services are called virtual services for the reason that their identity is not bound to a fixed physical machine.

This paper discusses the design of a modular server-architecture enabling dynamic expansion and shrinkage of a virtual video server to a number of physical nodes located somewhere in the net. The modules of a server may migrate from one node to another, depending on decisions made by control-components. In some cases a service might even migrate to the client in order to perform operations more efficiently.

The remainder of this paper is organized as follows. In section 2 a brief survey on related work concerning video server architectures is given. The design of an adaptive, module-based server architecture is discussed in section 3. The proposed model identifies the core modules of a video server architecture and describes their functionalities. In section 4, an execution environment for virtual servers is discussed. It is also illustrated, how to associate dynamic services to server modules. Section 5 concludes this paper and raises additional questions to be dealt with in future.
2 Related Work

In the past a number of design and performance models for large-scale, on-demand video servers were introduced. The common architecture shared by most VoD systems is the single-server model. Concerning their hardware they typically reach from standard PCs for small-scale systems, to massively parallel supercomputers with a number of processors for large-scale systems. E.g. in [1], the design and implementation of an enterprise video server using off-the-shelf hardware is described. It focuses on the three main subsystems video retrieval algorithms, buffer management and playback algorithms on client-side, and provides end-to-end performance guarantees to the client via a real-time ethernet implementation.

However, the single-server model has its limitations regarding scalability and server fault tolerance. Single-server models do not scale when the demand exceeds the server’s capacity, as the data either needs to be replicated, or partitioned among separate video servers. As consequence the system suffers from problems like increased storage overhead, missing load-balancing or single point of failure [7].

As a result, parallel (distributed) video server architectures have been designed. The main principle here is to stripe video data across multiple servers in order to achieve scalability and load-balancing. Some design and performance-models for distributed servers can be found in [2], [9] and [4]. An interesting framework for designing parallel video server architectures is given in [7], considering video distribution architectures, server striping policies and video delivery protocols.

In [8] it is claimed that nearly none of the models, neither single nor parallel, provides an integrated design across different server components. This is mainly due to the different hardware platforms and components they use. As a consequence, a generic software architecture for VoD servers is introduced. It is valid both for shared-memory and distributed-memory multi-processors. As the two main tasks of a video server are to retrieve and to deliver video blocks to the network with a guaranteed quality of service, they define control and access processes for storage and network devices as core components of their model. Control processes are required for transferring data to/from a storage device or a network device respectively. The access processes are communicating with each other and serve for handling multiple requests, as well as for bridging the physical distance of the processes in a distributed environment. Disk scheduling and flow control are implemented within these components. The whole system is controlled and monitored by a central control component, which also performs the admission control.

The term generic refers to the model’s flexibility to adapt to different hardware-platforms. In order to adapt the server’s behaviour to different constraints given at runtime, the software architecture itself needs to be modular and flexible. It should e.g. be possible to migrate the buffering component to a node somewhere in the network, if it denotes a better strategical position. In order to be able to do so, a modular design is required for increasing the server’s adaptivity. In the following section, a module-based, dynamic server-architecture is introduced.
3 A Modular Adaptive Server Architecture

The term *adaptivity* in existing architectures mostly belongs to their ability to deliver different qualities of video streams concerning color-depth and resolution. However, this kind of adaptation only belongs to streams and does not take into account runtime constraints as described before. This paper introduces a modular design enabling *server level adaptivity* for improving scalability and performance especially in a distributed environment.

A video server architecture, regardless whether it follows the single or distributed model, can be divided into four main modules: *request processing, server configuration, data repository* and *monitoring*. Each of these modules may consist of a number of sub-modules, as illustrated in figure 1. The model also shows the interactions between the modules, indicating a flow of control within the server. The *RequestProcessing* module covers all the functionalities for processing data-query or data-aquisition requests. The behaviour of handling requests is driven by the *ServerConfiguration* module. Here e.g. it is declared, whether the server is distributed or not, and which module-implementations are currently being used. The *DataRepository* module provides functionality and mechanisms to store and retrieve video data from the storage. And finally, the *Monitoring* module monitors the behaviour of the *RequestProcessing* module at runtime. The following subsections describe the functionalities to be provided by each of the modules. Note that each of the modules denotes an *interface*, which might have multiple implementations for testing the performance of different components. The proposed model will be managed by a virtual server environment using CORBA-facilities. Therefore, an object-oriented implementation of the modules is strongly recommended.

3.1 The Data Repository Module

A video server has to keep track of its stored media streams. The general technique in distributed server architectures is to stripe a single video over a number of server nodes in order to improve load-balancing and performance. In this case each node needs to know, which stripes of which stream it stores in which sectors, assuming direct access to the disk. As video streams may also be physically segmented into disjunct units, the repository might also keep track of video structures stored in the server. A model for video segmentation and structuring is presented in [10]. As a consequence, the repository module has to provide means for storing and retrieving video stripes, segments and streams. Further, it should be possible to change between time- and space-striping on different nodes. Which kind of striping actually should be used depends on the server’s configuration.

3.2 The Server Configuration Module

This module provides the most powerful functionalities concerning the server’s ability to adapt to different environmental conditions, as well as to improve performance and increase load-balancing. The configuration module has to provide means for exchanging implementations of server modules partially *on the fly*, partially off-line. This facility enables an evaluation of
Figure 1: Interacting modules of an adaptive video server
different algorithms in implementations at runtime. Furthermore, the distribution of the server is also specified here. This affects the server’s storage-layout concerning video streams, as well as the execution of certain services on certain nodes. If e.g. a distributed server architecture is preferred, the coordinating node (also called central server node) must have a different configuration concerning admission and service control, since it mostly delegates the request to the nodes storing the video data. Finally, it is the configuration module’s duty to provide means for improving the server’s reliability and for placing monitoring points within the server.

3.3 The Request-Processing Module

The module for request processing generally consists of the five sub-modules: contact management, admission control, service control, proxy and in its heart the resource management. All these sub-modules are briefly discussed in the next sub-sections.

3.3.1 ContactManagement

Each request, regardless if it comes from a client or another video server node, first is handled by the ContactManagement module. This module initially checks the type of the incoming request. Currently two different types of requests have been identified: video catalogue and video data requests. A VideoCatalogueRequest may query information on video data from the data repository like e.g. the used striping policy for a certain video, the segments of a given video or a video’s encoding type. Additionally, listings of all known video streams and their segments should be supported. A VideoDataRequest asks for processing a certain piece of video data with certain quality of service. If the requested stream is known by the repository, it is delegated to the admission control module. As a result, the contact manager notifies the requester, whether or not the request has been admitted. In the case of a distributed environment, a central server node has to collect all the required admissions from the respective nodes.

3.3.2 Admission Control

The admission control has to decide, whether a request can be handled or not. This is achieved by querying the resource management module, whether all the required resources themselves can handle the request regarding the request’s given QoS criteria. These criteria might include minimum and maximum bitrate, maximum delay, jitter and lossliness. The admission also should take into account the software overhead for processing the admission itself. On the level of implementation, different deterministic and statistical admission control policies should be provided. Again, which one is used at runtime is a matter of the configuration module. If the request is admitted, it is delegated to the service control unit for processing.
3.3.3 Service Control

The service control module’s task is to schedule the admitted requests and to control the servicing processes of the resource management module. At this time, all resources required by a request must have been reserved for each targeted resource type in the system. If under certain circumstances a resource becomes a bottleneck during request execution, it is this module’s duty to preempt the execution and to release all reserved resources.

The distinction between video-query and video-acquisition requests has also to be performed here. Requests for data acquisition are typically not as time-critical, as those for data retrieval are. Therefore, the scheduler should have some more space for scheduling acquisition requests. The service control should implement different scheduling strategies like e.g. earliest deadline first (EDF) for scheduling retrieval requests, and first come first serve (FCFS) for executing acquisition requests. A variety of service control implementations should enable a comparative study of the server’s behavior under different configurations.

3.3.4 Resource Management

The resource management module represents the central sub-module of the RequestProcessing unit. It has to provide efficient means for reserving and releasing resources for requests with a given quality of service. In certain cases of bottlenecks, resources should be able to adapt themselves to new situations. This should especially be the case with resources not needing a guaranteed QoS. A combination of the resource reservation and adaptation principle, called federative resource management [6], should be supported.

The resource management can be basically subdivided into five sub-modules: disk, buffer, network, bus and CPU management. Each of these management modules is driven by a periodical schedule initiated and controlled by the service control module. However, CPU and bus scheduling are usually controlled by the operating system or the hardware. But the resources CPU and bus still need to be managed, as they might impose constraints in servicing requests too.

The disk and network management modules should be divided into control and access units, as proposed in [8]. The storage control unit reads or writes video blocks from or to the storage subsystem. Multiple storage usage requests are handled by the storage access unit and arranged into disk serving schedules like FCFS or CSCAN. Similarly to the storage control unit, the network control unit reads or writes video blocks from or to the network interface. The network access unit’s due is to cooperate with the buffer management module by allocating the required buffers for serving the requests, and to handle multiple network usage requests. The buffer management module has to provide efficient means for dealing with send and receive buffers in a cycle-based system.
3.3.5 Proxy

As described in [7], the proxy module is responsible for resequencing and merging data from multiple servers into a coherent stream for delivery to the client. Additionally, the proxy can take advantage of data redundancy to mask server failures and achieve server-level fault tolerance. Another functionality of the proxy is to accommodate bit-rate fluctuations and network jitter, if the server is wide-spread and latencies are varying extremely. Finally, the proxy module should also support several protocols for video delivery.

In [7], three possible implementations of a proxy module are discussed: proxy at server, independent proxy and proxy at client. In the proxy at server model, the proxies are run on server-side and handling requests for multiple clients. Independent proxies denote separate computers in the network, which are running the proxy server, also serving requests for multiple clients. Finally, in the proxy at client model the proxy module is integrated into the client, meaning that it only handles requests for one client. The two main advantages of keeping the proxy module server-side or independent are that first, the complexity of communicating with multiple servers is hidden to the client, and second, the proxies are able to implement flow control mechanisms. Concerning an adaptive video server architecture it is essential to keep the location of the proxy module configurable.

3.4 The Monitoring Module

Finally, the monitoring module keeps track of the runtime behaviour of the request-processing module. It has to provide statistical information on all its sub-modules, which have at least one monitoring point associated. Note that the monitoring points are placed during the server configuration, as described in subsection 3.2.

The next section briefly describes and discusses the execution environment of virtual servers and some ideas on how to map the modules described above to services.

4 Runtime Environment for a Virtual Video Server

In order to achieve server-level adaptation, it might be a good idea to migrate or replicate certain services from one physical node to another. E.g. it would make sense to migrate the proxy module to an independent node in the neighborhood of closely aligned clients to increase performance and reduce network load. However, this is not a simple task, as each service has a set of data associated, which it needs to perform its work. For some of the modules described in section 3 it might not be worth to move through the network, as it would decrease the system’s performance extremely. In subsection 4.2, some ideas for linking modules and services are presented and discussed.

Dealing with virtual servers, an infrastructure for managing and executing distributed virtual services is required. This paper discusses the usage of the Symphony system, which represents a management infrastructure for executing virtual servers in internet settings. The following
subsection briefly presents its core components.

4.1 Symphony - A Management Infrastructure for Virtual Servers

Symphony represents an infrastructure for managing virtual servers in a distributed environment [3]. A virtual server denotes a server, whose identity is not bound to a fixed physical computer, but may span a dynamically changing number of physical nodes at runtime, depending on the system’s load. It follows a service-based design using CORBA technology in combination with group communication capabilities for added reliability and fault tolerance. Concerning group communication, Symphony is based on the Ensemble system [5].

The services are logically organized in a hierarchical structure, based on the operating system and Symphony’s runtime environment. The heart of the architecture is constituted by the management services, providing support for automatically replicating and migrating virtual servers. The management services take use of informational and code-handler services. Code-handler services are responsible for starting and stopping an image of a code on a given machine. Informational services collect and report information about the system’s load and topology, as well as the location of various applications. Both, the code-handler and the informational services are using basic services on the bottom of the hierarchy, like e.g. the naming, transaction and replication service. Finally, a virtual server is located on top of this hierarchy, using all its facilities either directly or via the management services.

4.2 Associating Services to Modules

Using Symphony as infrastructure for a virtual video server architecture requires an association of services to the modules described in section 3. Following the modular server design, the identification of such processes seems to be quite simple. Each module needs at least one driving process performing the tasks to cover the module’s functionalities. Modules with sub-modules may require one process per sub-module. E.g. the RequestProcessing module should be serviced by the following processes: a contact-handler for the contact manager module, an admission-handler for the admission control, a service-scheduler for the service control, a data-supplier for the proxy module, a data producer for disk management, and a data consumer for network management. However, actually there are no written rules and recommendations for mapping services to modules. It’s the designer’s liberty to perform the required tasks.

5 Conclusion

This paper introduced a modular adaptive video server architecture for execution in a virtual server environment. Many existing architectures are characterized by shortcomings concerning scalability, load-balancing, reliability and adaptivity. This is often due to a missing modular design of their architectures, as they typically represent monolithic systems with hard-coded components. This paper illustrated the core modules and its sub-components, a dynamic video
server should consist of. The modules are only described by their functionalities in order to allow a seamless mapping to interface declarations. Modules might have different object-oriented implementations and therefore different server configurations can be used for evaluation and testing purposes. Finally it is illustrated, how the model could be adopted to construct a virtual video server architecture running on top of Symphony. However, there are many questions not addressed so far. First, what are the costs for such a generic, adaptive server architecture? Second, how much effort does it require to port Symphony to use a real-time CORBA implementation? Third, what are the impacts of an operating system not capable of dealing with real-time constraints? And finally, how and when to migrate or replicate video data in a highly distributed environment?

References


