To be published in: Computer Communications Journal, Special Issue on Interactive Distributed Multimedia Systems and Telecommunications Services 1998 (IDMS'98), Elsevier Science, Winter 99


# Operating System Support for Multimedia Systems

**THOMAS PLAGEMANN, VERA GOEBEL, PÅL HALVORSEN***
UniK- Center for Technology at Kjeller, University of Oslo, Norway
*{plageman, goebel, paalh}@unik.no*

**OTTO ANSHUS**
Department of Computer Science, Princeton University, USA
On leave from Computer Science Department, University of Tromsø, Norway
*otto@{cs.uit.no|cs.princeton.edu}*

**Abstract**

Distributed multimedia applications will be an important part of tomorrow's application mix and require appropriate operating system (OS) support. Neither hard real-time solutions nor best-effort solutions are directly well suited for this support. One reason is the co-existence of real-time and best effort requirements in future systems. Another reason is that the requirements of multimedia applications are not easily predictable, like variable bit rate coded video data and user interactivity. In this article, we present a survey of new developments in OS support for (distributed) multimedia systems, which include: (1) development of new CPU and disk scheduling mechanisms that combine real-time and best effort in integrated solutions; (2) provision of mechanisms to dynamically adapt resource reservations to current needs; (3) establishment of new system abstractions for resource ownership to account more accurate resource consumption; (4) development of new file system structures; (5) introduction of memory management mechanisms that utilize knowledge about application behavior; (6) reduction of major performance bottlenecks, like copy operations in I/O subsystems; and (7) user-level control of resources including communication.

*Keywords:* Operating systems, multimedia**,** Quality of Service, real-time

## 1 INTRODUCTION

Distributed multimedia systems and applications play already today an important role and will be one of the cornerstones of the future information society. More specifically, we believe that time-dependent data types will be a natural part of most future applications, like time-independent data types today. Thus, we will not differentiate in the future between multimedia and non-multimedia applications, but rather between hard real-time, soft real-time, and best effort requirements for performance aspects like response time, delay jitter, synchronization skew, etc. Obviously, all system elements that are used by applications, like networks, end-to-end protocols, database systems, and operating systems (OSs), have to provide appropriate support for these requirements. In this article, we focus on the OS issues on which applications, end-to-end protocols, and database systems directly rely. For simplicity, we use in this article the notion *multimedia systems and applications* which comprises also *distributed* multimedia systems and applications.

The task of traditional OSs can be seen from two perspectives. In the top-down view, an OS provides an abstraction over the pure hardware, making programming simpler and programs more portable. In the bottom-up view, an OS is responsible for an orderly and controlled allocation of resources among the various executing programs competing for them. Main emphasis of resource management in commodity OSs, like UNIX or Windows systems, is to distribute resources to applications to reach fairness and efficiency. These time-sharing approaches work in a best-effort manner, i.e., no guarantees are given for the execution of a program other than to execute it as fast as possible while preserving overall throughput, response time, and fairness.

Specialized real-time OSs in turn emphasize on managing resources in such a way that tasks can be finished within guaranteed deadlines. Multimedia applications are often characterized as soft real-time applications, because they require support for timely correct behavior. However, deadline misses do not naturally lead to catastrophic consequences even though the *Quality of Service* (QoS) degrades, perhaps making the user annoyed.

Early work in the area of OS support for multimedia systems focussed on the real-time aspect to support the QoS requirements of multimedia applications. Traditional

* This research is sponsored by the Norwegian Research Council, DITS Program, under contract number 119403/431 (INSTANCE project).

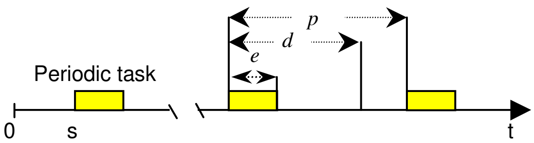


*Figure 1: Periodic task model*

real-time scheduling algorithms, like *Earliest Deadline First* (EDF) and *Rate Monotonic* (RM), have been adopted for CPU (and disk) scheduling. These scheduling mechanisms are based on a periodic task model [Liu et al. 73]. In this model, tasks are characterized by a start time $s$, when the task requires the first execution, and a fixed period $p$ in which the task requires execution time $e$ with deadline $d$ (see Figure 1). Often, the deadline is equal to the end of the period.

An 8 bit, 8 KHz PCM encoded audio stream is a good example for such a periodic task: the constant sampling rate and the fixed sample size generate a constant bit stream. In order to handle the stream more efficiently, samples of typically 20 ms are gathered in a packet. Thus, the system has to handle in each period, i.e., $p = 20$ ms, a packet before the next period. The fixed packet size requires a constant execution time $e$ per period. This periodic task model is attractive from an engineering point of view, because it makes it possible to predict the future: in period $k$, which starts at $s + (k-1)\,p$, the task with execution time $e$ has to be finished before $s + (k-1)\,p + d$.

However, experiences with multimedia applications including efficient variable bit rate (VBR) coding schemes for video, like H.261, H.263, MJPEG, and MPEG, lead to the conclusion that it is not that easy to foresee future requirements. Video frames are generated in a fixed frequency (or period), but the size of the frames and the execution times to handle these frames are not constant [Goyal et al. 96a], [Chu et al. 99]. It varies on a short time scale, between the different frames, e.g., I, B, or P frames in MPEG, and on a larger time scale, e.g., due to scene shifts such as from a constant view on a landscape to an action scene. Furthermore, the degree of user interactivity is much higher in recent multimedia applications, e.g., interactive distance learning, than in earlier applications, like video-on-demand (VoD). It is very likely that this trend will continue in the future and make resource requirements even harder to predict.

Latest developments in the area of multimedia OSs still emphasize on QoS support, but integrate often adaptability and support both real-time and best effort requirements. Furthermore, new abstractions are introduced, like new types of file systems and resource principals that decouple processes from resource ownership. Finally, main bottlenecks, like paging, copy operations, and disk I/O, have been tackled to fulfill the stringent performance requirements.

It is the goal of this article to give an overview over recent developments in OS support for multimedia systems. OS support for multimedia is an active research area, and therefore, it is not possible to discuss all particular solutions in depth. Instead, we introduce for each OS aspect the basic issues and give an overview and a classification of new approaches. Furthermore, we describe a few examples in more detail to enable the reader to grasp the idea of some new solutions. However, for an in depth understanding, the reader has to refer to the original literature, because this article is intended as a survey and to serve as an "entry-point" for further studies.

The rest of this article is structured as follows: Section 2 discusses general OS developments, and Section 3 summarizes the requirements of multimedia applications. The basic dependency between resource management and QoS is discussed in Section 4. Management of the system resources, like CPU, disk, main memory, and other system resources, are discussed in the Sections 5 to 8. New approaches to overcome the I/O bottleneck are presented in Section 9. Section 10 gives some conclusions.

## 2 Operating System Architectures

Traditionally, an OS can be viewed as a resource allocator or as a virtual machine. The abstractions developed to support the virtual machine view include a virtual processor and virtual memory. These abstractions give each process the illusion that it is running alone on the computer. Each virtual machine consumes physical resources like physical processor and memory, data-, instruction-, and I/O buses, data and instruction caches, and I/O ports. However, instead of allowing a process to access resources directly, it must do so through the OS. This is typically implemented as system calls. When a process makes a system call, the call is given to a library routine which executes an instruction sending a software interrupt to the OS kernel. In this way, the OS gets control in a secure way and can execute the requested service. This is a too costly path for some services, because trapping to the OS involves the cost of crossing the user-supervisor level boundary at least twice, and possibly crossing address spaces also at least twice if a context switch to another process takes place. In addition, there are costs associated with the housekeeping activities of the OS.

When several processes are executing, each on its own virtual processor, they will implicitly interfere with each other through their use of physical resources. Primarily, they will affect each other's performance, because applications are not aware of physical resources and of each other. A multimedia application can face a situation where it does not get enough resources, because the OS is not aware of each applications' short and longer term needs. This will typically happen when the workload increases.

The need to go through the OS to access resources and the way the OS is designed and implemented results in a system where low latency communication is difficult to achieve. It is also difficult to either have resources available when they are needed by a process; or have a process ready to execute when the resources, including cooperating processes on other processors or computers, are available or ready.

A traditional general-purpose OS, like UNIX or Windows NT, is not a good platform for the common case needs of multimedia applications. In these OSs, resource allocation for each process is based on general purpose scheduling algorithms, which are developed to provide a balance between throughput and response time, and to provide fairness. These algorithms get some limited feedback from the application processes on what they are doing, but basically, the OS has little or no understanding of what the applications are doing and what their requirements are. Also, the degree to which an application can directly control resources in a secure way, or provide the OS with hints for its resource requirements, has traditionally been very limited.

There are several OS architectures in use today of which the monolithic kernel and the μ-kernel architectures, or their hybrids, are the most common. In a monolithic OS kernel, all components are part of a large kernel, execute in a hardware protected supervisory mode, and can use the entire instruction set. Consequently, the kernel has total control over all resources. User processes execute in user mode and can therefore use only a limited instruction set and have only a limited control over resources. A user process cannot execute an instruction to switch to supervisory mode, enable or disable interrupts, or directly access I/O ports. When a user process needs OS services, it requests the service from the OS, and the kernel performs the requested service. Two crossings between user- and kernel-level take place, from the user process to the kernel and then back again when the service has finished.

In the μ-kernel architecture, the OS is divided into a smaller kernel with many OS services running as processes in user mode. This architecture is flexible, but has traditionally resulted in an increased overhead. The kernel sends a request for service to the correct user-level OS process. This creates extra overhead, because typically four crossings between user- and kernel-level take place, i.e., from the user process to the kernel, from the kernel to the user-level service, from the user-level service to the kernel, and finally, from the kernel to the user process. This can also result in memory degradation because of reduced instruction locality giving an increased number of cache misses. In [Chen et al. 96], a comparative study of three OSs, including NetBSD (a monolithic UNIX) and Windows NT (a μ-kernel like architecture), is presented. The monolithic NetBSD has the lowest overhead for accessing services. However, the overall system performance can significantly be determined by specific subsystems, e.g., graphics, file system, and disk buffer cache; and for some cases Windows NT does as well as NetBSD in spite of the higher overhead associated with its μ-kernel architecture.

Library OSs (also referred to as vertically structured systems) like the Exokernel architecture [Engler et al. 95], [Kaashoek et al. 97] and Nemesis [Leslie et al. 96] have been proposed as an alternative to monolithic and μ-kernel OSs. The basic idea is that those parts of an OS that can run at user-level are executed as part of the application processes. The OS is implemented as a set of libraries shared by the applications. The OS kernel can be kept very small, and it basically protects and exports the resources of the computer to the applications, i.e., leaving it to the applications to use the resources wisely. This allows for a high flexibility with respect to the needs of individual applications. At the same time it gives processes more direct control over the resources with better performance as a potential result.

However, the results presented in [Liedtke 95], [Liedtke 96], and [Härtig et al. 97] identify several areas of significance for the performance of an OS including the switching overhead between user and kernel mode, switching between address spaces, the cost of interprocess communication (IPC), and the impact of the OS architecture upon memory behavior including cache misses. These papers show how a μ-kernel OS can be designed and implemented at least as efficient as systems using other architectures. The SUMO OS [Coulson et al. 94] describes how to improve μ-kernels by reducing the number of protection crossings and context switchings even though it is built on top of the Chorus OS.

Threads can be used as a way of reducing OS induced overhead. Basically, threads can be user-level or kernel-level supported. User-level threads have very low overhead [Anderson et al. 92a]. However, they are not always scheduled preemptively, and then the programmer must make sure to resume threads correctly. User-level threads can also result in blocking the whole process if one thread blocks on a system service. Even if user-level threads will reduce the internal overhead for a process, we still have no resource control between virtual processors. Kernel supported threads have much higher overhead, but are typically scheduled preemptively and will not block the whole process when individual threads block. This makes kernel-level supported threads simpler to use in order to achieve concurrency and overlap processing and communication. Kernel-level supported threads can potentially be scheduled according to the process' requirements, but this is typically not done. *Split-level scheduling* [Govindan et al. 91] combines the advantages of user and kernel-level thread scheduling by avoiding kernel-level traps when possible. *Scheduler activations* [Anderson et al. 92a] is a kernel interface and a user-level thread package that together combine the functionality of kernel threads with the performance and flexibility of user-level threads.

Low context switching overhead can also be achieved when using processes. For example, Nemesis uses a single address space for all processes. This allows for a low context-switching overhead, because only protection rights must be changed.

The rapid development of commodity multiprocessors and clusters of commodity computers provides a scalable approach to separating virtual machines onto physical processors and memories and thereby reduces the interference between them. However, there are still shared resources that must be scheduled, including networks, gateways, routers, servers, file systems, and disks.

## 3 Multimedia Application Requirements

In this section, we briefly discuss the requirements that multimedia applications impose onto OSs. First, we exemplify typical requirements by introducing a multimedia application that is in productive use since 1993 for teaching regular courses at the University of Oslo. Afterwards, we give a more general characterization of multimedia application requirements.

### 3.1 Example: Interactive Distance Learning

The main goal of the electronic classrooms is to create a distributed teaching environment which comes as close as possible to a traditional (non-distributed) classroom [Bringsrud et al. 93]. The classroom system is composed out of three main parts: electronic whiteboards, audio system, and video system. At each site there is at least one large electronic whiteboard to display transparencies. The lecturer can write, draw, and erase comments on displayed transparencies by using a light-pen.

Main element of the audio system is a set of microphones that are mounted evenly distributed on the ceiling in order to capture the voice of all the participants. The video system comprises in each classroom three cameras and two sets of monitors. One camera focuses on the lecturer, and two cameras focus on the students. A video-switch selects the camera corresponding to the microphone with the loudest input signal. Two monitors are placed in the front and two monitors are placed in the back of each classroom displaying the incoming and outgoing video information. All video data is compressed according to the compression standard H.261.

During a lecture, at least two electronic classrooms are connected. Teacher and students can freely interact with each other regardless of whether they are in the same or in different classrooms. Audio, video, and whiteboard events are distributed in real-time to all sites, allowing all participants to see each other, to talk to each other, and to use the shared whiteboard to write, draw, and present prepared material from each site.

Detailed measurements are reported in [Plagemann et al. 99] and show that the audio system with 8 bit, 16 KHz PCM encoding generates a constant bitstream of 128 Kbit/s. The video stream, however, varies between 100 Kbit/s and 1.4 Mbit/s, because it depends on the activity in the classroom. The traffic pattern of the whiteboard fluctuates even more, because it solely depends on the interactivities of the users, i.e., teacher and students (see Figure 2). The large peaks are generated by downloading transparencies and range between 30 Kbit/s up to 125 Kbit/s. The small peaks of approximately 10 Kbit/s are generated by light-pen activities, like editing and marking text on transparencies.

These measurement results show that the size of video frames and corresponding execution times are not constant and that the whiteboard stream cannot be characterized as periodic task. Treating both as periodic tasks would require to perform pessimistic resource allocation for video and to install a periodic process that polls for aperiodic user interactions. However, both solutions result in poor resource utilization.

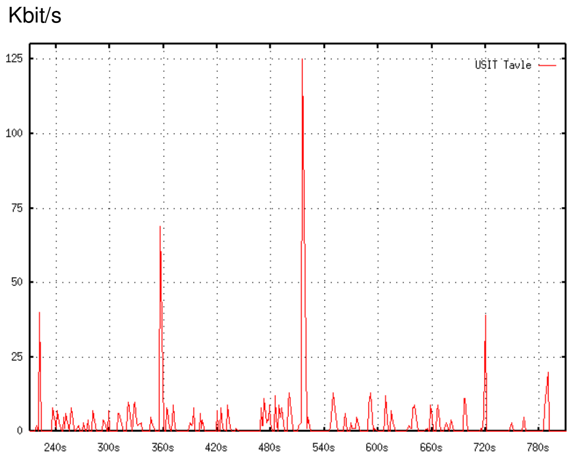


*Figure 2: Whiteboard stream*

### 3.2 Requirements in General

Generally, we can identify the following three orthogonal requirements of multimedia applications [Nahrstedt et al. 95], [Schulzrinne 96]:

- *High data throughput*: audio streams in telephony quality require 16 Kbit/s and in CD-quality 1.4 Mbit/s. Typical video data rates range from approximately 1.2 Mbit/s for MPEG, 64 Kbit/s to 2 Mbit/s for H.261, 20 Mbit/s for compressed HDTV, and over 1 Gbit/s for uncompressed HDTV.
- *Low latency and high responsiveness*: end-to-end delay for audio streams (which is a sum of network delay and two times end-system delay) should be below 150 ms to be acceptable for most applications. However, without special hardware echo cancellation, the end-to-end delay should be below 40 ms. Lip synchronization requires to playout corresponding video and audio data with a maximum skew of 80 ms. The maximum synchronization skew for music and pointing at the corresponding notes is +/- 5 ms. Audio samples are typically gathered in 20 ms packets, i.e., 50 packets per second have to be handled per audio stream.
- *QoS guarantees*: to achieve a quality level that satisfies user requirements, the system has to handle and deliver multimedia data according to negotiated QoS parameters, e.g., bounded delay and delay jitter.

Interrupt latency, context switching overhead, and data movements are the major bottlenecks in OSs that determine throughput, latency, and responsiveness. In [Araki et al. 98], it is documented that especially the interrupt handling is a major overhead. To implement QoS guarantees for these performance aspects, advanced management of all system resources is required. The need for advanced resource management has lead to the development of new OS abstractions and structures. In the following sections, we discuss basic resource management tasks and explain how the new OS abstractions and structures can reduce context switching overhead and data movement costs.

## 4 RESOURCE MANAGEMENT AND QOS

A computer system has many resources, which may be required to solve a problem: CPU, memory at different levels, bandwidth of I/O devices, e.g., disk and host network-interface, and bandwidth of the system bus. In Figure 3, the basic resource types CPU, memory, and bandwidth are partitioned into concrete system resources.

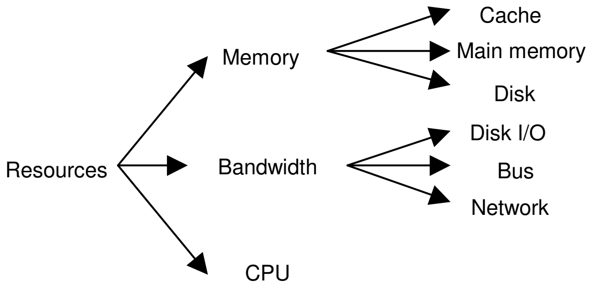


*Figure 3: Operating system resources*

One of the primary functions of an OS is to multiplex these resources among the users of the system. In the advent of conflicting resource requests, the traditional OS must decide which requests are allocated resources to operate the computer system *fairly* and *efficiently* [Peterson et al. 85]. Fairness and efficiency are still the most important goals for resource management in today's commodity OSs. However, with respect to multimedia applications, other goals that are related to timeliness become of central importance. For example, user interactions and synchronization require short *response times* with upper bounds, and multimedia streams require a minimum *throughput* for a certain period of time. These application requirements are specified as QoS requirements. Typical application-level QoS specifications include parameter types like frame rate, resolution, jitter, end-to-end delay, and synchronization skew [Nahrstedt et al. 99]. These high-level parameters have to be broken down (or mapped) into low-level parameters and resources that are necessary to support the requested QoS, like CPU time per period, amount of memory, and average and peak network bandwidth. A discussion of this mapping process is beyond the scope of this paper, but we want to emphasize at this point that such a specification of resource requirements is difficult to achieve. A constant frame rate does not necessarily require constant throughput and constant execution time per period. Furthermore, user interactions can generate unpredictable resource requirements.

In order to meet QoS requirements from applications and users, it is necessary to manage the system resources in such a manner that sufficient resources are available at the right time to perform the task with the requested QoS. In particular, resource management in OS for QoS comprises the following basic tasks:

- *Specification and allocation request* for resources that are required to perform the task with the requested QoS.
- *Admission control* includes a test whether enough resources are available to satisfy the request without interfering with previously granted requests. The way this test is performed depends on requirement specification and allocation mechanism used for this resource.
- *Allocation and scheduling* mechanisms assure that a sufficient share of the resource is available at the right time. The type of mechanism depends on the resource type. Resources that can only exclusively be used by a single process at a time have to be multiplexed in the temporal domain. In other words, exclusive resources, like CPU or disk I/O, have to be scheduled. Basically, we can differentiate between fair scheduling, real-time scheduling, and work and non-work conserving scheduling mechanisms. So-called shared resources, like memory, basically require multiplexing in the spatial domain, which can be achieved, e.g., with the help of a table.
- *Accounting* tracks down the actual amount of resources that is consumed in order to perform the task. Accounting information is often used in scheduling mechanisms to determine the order of waiting requests. Accounting information is also necessary to make sure that no task consumes more resources than negotiated and steals them (in overload situations) from other tasks. Furthermore, accounting information might trigger system-initiated adaptation.
- *Adaptation* might be initiated by the user/application or the system and can mean to downgrade QoS and corresponding resource requirements, or to upgrade them. Adaptation leads in any case to new allocation parameters. Accounting information about the actual resource consumption might be used to optimize resource utilization.
- *Deallocation* frees the resources.

Specification, admission control, and allocation and scheduling strongly depend on the particular resource type, while adaptation and resource accounting represent more resource type independent principles. Thus, the following two subsections introduce adaptation and resource accounting, before we discuss the different system resources in more detail.

### 4.1 ADAPTATION

There are two motivations for adaptation in multimedia systems: (1) resource requirements are hard to predict, e.g., VBR video and interactivity; and (2) resource availability cannot be guaranteed if the system includes best-effort subsystems, e.g., today's Internet or mobile systems. In best-effort systems, it is only possible to adapt the application, respectively the amount of application data the system has to handle. In OSs, both situations might occur, and it is possible to adapt both application and resource allocations. In [Gecsei 97], adaptation with feedback and adaptation without feedback are distinguished. Adaptation without feedback means that applications change only the functionality of the user interface and do not change any resource parameters. Therefore, we consider in this article only adaptation with feedback, i.e., feedback control systems. Figure 4 shows a simplified view of the collaboration between resource consumer, e.g. application, and resource provider, e.g., CPU scheduler, in adaptation with feedback [Lakshman 97]:

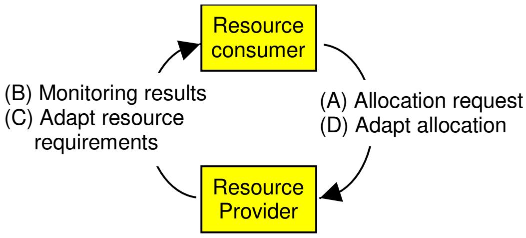


*Figure 4: Feedback control for adaptation*

(A) The resource consumer, or a management entity, estimates its resource requirements and requests the provider to allocate the resource according to its specification.

(B) After admission control is successfully passed, the resource utilization is monitored. The monitoring results can reflect the general resource utilization, e.g., the resource is under- or over-utilized, and the accuracy of the consumers' resource requirements estimations, e.g., it uses less or more resources than allocated. The monitoring results might trigger step (C) and/or (D).

(C) The provider requests the consumer to adjust its resource requirements, e.g., by reducing the frame rate of a video.

(D) The consumer requests the provider to adjust the allocation parameters.

Most of the recent results in the area of adaptive resource management discuss CPU management (see Section 5). Monitoring of actual execution times is supported in most of these systems. More general approaches for adaptive resource management include AQUA [Lakshman 97], SWiFT [Goel et al. 98], Nemesis [Leslie et al. 96], Odyssey [Noble et al. 97], and QualMan [Nahrstedt et al. 99]. The crucial aspects of adaptation are the frequency in which feedback control (and adaptation) is performed and the related overhead. For example, SCOUT uses a course-grained feedback mechanism that operates in the order of several seconds [Bavier et al. 98a]. On the other hand, the work presented in [Bavier et al. 98b] aims to predict the execution times of single MPEG frames. Whether fine-grained adaptation of allocation parameters results in better QoS and/or better resource utilization is still open. However, it is obvious that it requires frequent adjustment of allocation parameters, which must not impose much overhead.

### 4.2 New Abstractions for Resource Principals

Resource accounting represents a fundamental problem, because most OSs treat a process as the "chargeable" entity for allocation of resources, such as CPU time and memory. In [Banga et al. 99], it is pointed out that "a basic design premise of such process-centric systems is that a process is the unit that constitutes an independent activity. This gives the process abstraction a dual function: it serves both as a protection domain and as a resource principal." This situation is insufficient, because there is no inherent one-to-one mapping between an application task and a process. A single process might serve multiple applications, and multiple processes might serve together a single application. For example, protocol processing is in most monolithic kernels performed in the context of software interrupts. The corresponding resource consumption is charged to the unlucky process running at that time, or not accounted at all [Banga et al. 99]. µ-kernels implement traditional OS services as user-level servers. Applications might invoke multiple user-level servers to perform on its behalf, but the resource consumption is charged to the application and the user-level servers instead of charging it to the application only [Mercer et al. 94]. It is important to note that ownership of resources is not only important for accounting reasons, but also for scheduling. The resources a process "owns", e.g., CPU time, define also its scheduling parameter. For example, in commodity OSs with priority based CPU scheduling, each process is associated with a priority, which in turn determines when it is scheduled, i.e., receives its CPU time. Thus, a server or kernel thread that is performing a task on behalf of an application with QoS requirements should inherit the corresponding scheduling parameters for needed resources, e.g., the priority.

In [Jeffay et al. 98], the problem of resource ownership and the corresponding scheduling decisions is partially solved for a monolithic kernel by deriving weights of kernel activities from user weights. By this, the proportional share scheduling, in the extended FreeBSD kernel, is able to provide an appropriate share of CPU bandwidth to the kernel activity such that QoS requirements of the user process can be met.

We can identify two basic, orthogonal approaches to handle this problem: (1) introduction of a new abstraction for resource ownership; and (2) to provide applications as much control as possible over devices, as it is done in so-called library OSs, like Exokernel [Engler et al. 95], [Kaashoek et al. 97] and Nemesis [Leslie et al. 96]. In these systems, applications can directly control resource consumption for network I/O and file I/O, because network device drivers and disk device drivers are accessible from the application in user-space without kernel interference.

Obviously, resource consumption can be easily charged to the application.

In [Banga et al. 99], a quite extensive discussion of new abstractions for resource ownership can be found. These abstractions differ in terms of: (1) which resources are considered; (2) which relationships between threads and resource owners are supported; and (3) which resource consumptions are actually charged to the owner. In *Stride* and *Lottery* scheduling [Waldspurger 95], resource rights are encapsulated by *tickets*. Tickets are owned by clients, and ticket transfers allow to transfer resource rights between clients. In Mach [Ford et al. 94], AlphaOS [Clark et al. 92], and Spring [Hamilton et al. 93], *migrating threads* respectively *shuttles* correspond to threads and own resources. Migration of threads between protection domains enables these systems to account resource consumption of independent activities to the correct owner, i.e., thread. In [Ford et al. 96], threads can act as scheduler for other threads and can donate CPU time to selected threads. The *reservation domains* [Bruno et al. 98] of Eclipse, the *Software Performance Units* [Vergehese et al. 98], and the *scheduling domains* in Nemesis [Leslie et al. 96] enable the scheduler to consider the resource consumption of a group of processes.

The *reserve* abstraction is introduced in Real-Time Mach [Mercer et al. 94]. The main purpose of reserves is to accurately account CPU time for activities that invoke user-level services, e.g., client threads that invoke various servers (which are running in user-space, because Real-Time Mach is a μ-kernel). Each thread is bound to one reserve, and multiple threads, potentially running in different protection domains, can be bound to a single reserve. By this, computations of server threads that are performed on behalf of a client can be charged to the clients reserve and scheduling of the server computations is performed according to the reservations of the reserve. In [Rajkumar et al. 98], the concept of reserves is extended to manage additional resource types such as CPU, disk bandwidth, network bandwidth, and virtual memory.

*Resource containers* [Banga et al. 99] allow explicit and fine-grained control over resource consumption at all levels in the system, because it allows dynamic relationships between resource principals and processes. The system provides explicit operations to create new resource containers, to bind processes to containers and to release these bindings, and to share containers between resources. Resource containers are associated with a set of attributes, like scheduling parameters, memory limitations, and network QoS values, that can be set by applications and support the appropriate scheduling (decoupled from particular process information). Furthermore, containers can be bound to files and sockets, such that the kernel resource consumption on behalf of these descriptors is charged to the container.

In the Rialto OS, an *activity object* is the abstraction to which resources are allocated and against which resource usage is charged [Jones et al. 95], [Jones et al. 97]. Applications run by default in their own activity and typically in their own process. Activities may span over address spaces and machines. Multiple threads of control may be associated with an activity. The threads execute with rights granted by secured user credentials associated with this activity. The CPU scheduler treats all threads of an activity equal, because the assumption is that they cooperate toward a common goal.

The *path* abstraction in the SCOUT OS [Mosberger et al. 96] combines low-level de-multiplexing of network packets via packet filter with migrating threads. A path represents an I/O path and is executed by a thread. One thread can sequentially execute multiple paths. A newly awakened thread inherits the scheduling requirements of the path and can adjust them afterwards. The path object is extended in the Escort OS with a mechanism to account all resource consumptions of a path to defend against denial of service attacks [Spatscheck et al. 99].

Compared to the previously discussed approaches for resource accounting, a basically different approach is introduced in [Steere et al. 99]. Multiple threads, which may reside in different protection domains, are gathered in a job. Instead of measuring the resource consumption of these threads, Steere et al. [Steere et al. 99] monitor the progress of jobs and adapt, i.e., increase or decrease, the allocation of CPU to those jobs. So-called symbiotic interfaces link the notion of progress, which is depending on the application to system metrics, like a buffer's fill-level.

## 5 CPU SCHEDULING

Most commodity OSs, like Windows NT and Solaris, perform *priority scheduling* and provide time-sharing and real-time priorities. Priorities of threads in the real-time range are never adjusted from the system. A straightforward approach to assure that a time critical multimedia task receives sufficient CPU time would be to assign a real-time priority to this task. However, in [Nieh et al. 93], it is shown for the SVR4 UNIX scheduler that this approach results in unacceptable system performance. A high priority multimedia task, like video, has precedence over all time-sharing tasks and is nearly always active. Starvation of timesharing tasks, e.g., window system and basic system services, leads to poor QoS for the multimedia application and unacceptable performance for the entire system. Thus, the usage of real-time priorities does not automatically lead to the desired system behavior. However, other implementations, like those described in [Wolf et al. 96] and [Chu et al. 99], show that fixed real-time priorities can be utilized to successfully schedule real-time and best effort tasks in a general purpose UNIX system.

Several new solutions have been recently developed. The following section gives a general overview and classification of these solutions, and subsequent sections present two solutions in more detail.

### 5.1 Classification of Scheduling Mechanisms

Two of the most popular paradigms for resource allocation and scheduling to satisfy the contradicting goals of flexibility, fairness, and QoS guarantees are *proportional share resource allocation* and *reservation-based* algorithms [Stoica et al. 97]. In proportional share allocation, resource requirements are specified by the relative share (or weight) of the resource. In a dynamic system, where tasks dynamically enter and leave the system, a share depends on both the current system state and the current time. The resource is always allocated in proportion to the shares of competing requests. Thus, in pure proportional share resource allocation, no guarantees can be given, because the share of CPU might be arbitrary low. The basis for most proportional share resource allocation mechanisms are algorithms that have been developed for packet scheduling in packet-switched networks, like *weighted fair queuing* [Demers et al. 90], *virtual clock* [Zhang 91], and *packet-by-packet generalized processor sharing* [Parek et al. 93]. Examples for systems using proportional share CPU allocation include *Adaptive Rate Control* (ARC) [Yau et al. 96], SMART [Nieh et al. 97], *Earliest Eligible Virtual Deadline First* [Stoica et al. 97], *Lottery* and *Stride* scheduling [Waldspurger 95], and *Move-to-Rear List* (MTR-LS) scheduling in Eclipse [Bruno et al. 98].

In contrast to proportional share resource allocation, reservation-based algorithms, like RM and EDF scheduling, can be used to implement guaranteed QoS. The minimum share for each thread is both state and time independent. However, resource reservation sacrifices flexibility and fairness [Stoica et al. 97]. EDF scheduling is used in Nemesis [Leslie et al. 96], DASH [Anderson et al. 90], and SUMO [Coulson et al. 95]. The principles of RM scheduling are applied, for example, in Real-Time Mach [Mercer et al. 94], HeiTS [Wolf et al. 96], AQUA [Lakshman 97] and for the *Real-Time Upcall* in the MARS system [Buddhikot et al. 98]. To implement a feedback driven proportional allocator for real-rate scheduling, the work presented in [Steere et al. 99] uses both EDF and RM.

Most modern solutions of CPU scheduling for multimedia systems are based on either proportional share allocation, or are a combination of different allocation paradigms in hierarchies to support both real-time and best effort requirements. For example, the scheduler in the Rialto system [Jones et al. 97] and the soft real-time (SRT) user-level scheduler from [Chu et al. 99] combine EDF and round-robin scheduling. The Atropos scheduler in Nemesis [Leslie et al. 96] also applies EDF to sort waiting scheduling domains in different queues. The proportional share resource allocation *Start-time Fair Queuing* in [Goyal et al. 96b] is used in [Goyal et al. 96a] to achieve a hierarchical partition of CPU bandwidth in a general framework. For each of these partitions, arbitrary scheduling mechanisms can be used. Proportional share scheduling is also the primary policy in [Jeffay et al. 98]. When multiple processes are eligible, they are scheduled according to EDF. In the context of the Flux project, a *CPU inheritance scheduling* framework has been developed in which arbitrary threads can act as scheduler for other threads and widely different scheduling policies can co-exist [Ford et al. 96]. The scheduler in SCOUT, called BERT [Bavier et al. 98a], merges reservation-based and proportional share resource allocation in a single policy, instead of combining two or more policies in a hierarchical approach. Basically, BERT extends the virtual clock algorithm by considering deadlines for the scheduling decision and by allowing high-priority real-time tasks to steal CPU cycles from low priority and best effort tasks.

In addition to the resource allocation paradigm, i.e., proportional share (P), reservation-based (R), and hierarchical (H), Table 1 uses the following criteria to classify CPU scheduling approaches: (1) whether admission control is performed; (2) whether adaptation is supported; (3) whether a new abstraction for resource principal is introduced; and (4) what is the context of the scheduler, i.e., is it part of a new kernel, integrated in an existing kernel, or implemented on top of an existing kernel.

*Table 1: CPU scheduling approaches*

| System/Project | Paradigm | Admission Control | Adaptation Support | Resource Principal | Context |
|---|---|---|---|---|---|
| AQUA | R | Y | Y | N | Solaris |
| ARC | P | Y | Y | N | Solaris |
| Atropos | H | Y | Y | Y | Nemesis |
| BERT | PR[1] | Y | Y | Y | SCOUT |
| Flux | H | N | N | N | UL prototype[5] |
| [Goyal et al. 96a] | P | N | Y[2] | N | Framework |
| HeiTS | R | Y | N | N | AIX |
| [Jeffay et al. 98] | H | N | Y[3] | N[4] | FreeBSD |
| Lottery, Stride | P | N | Y | Y | Mach |
| MTR-LS | P | Y | N | Y | Eclipse |
| Rialto | H | Y | Y | Y | Rialto |
| RT Mach | R | Y | Y | Y | RT-Mach |
| RT Upcalls | R | N | N | N | NetBSD |
| SMART | P | N | Y | N | Solaris |
| SRT | H | Y | Y | N | UL scheduler |
| [Steere et al. 99] | R | Y | Y | N[5] | UL prototype[5] |
| SUMO | H | Y | N | N | Chorus |

[1] BERT merges the features of virtual clock and EDF
[2] Provides a work around
[3] Supports only monitoring of execution times
[4] Supports inheritance of weight from user to kernel processes
[5] Prototype that has been implemented in user-level (UL)

In the following subsections, we describe in more detail two distinct approaches, i.e., Rialto scheduler and SMART, that differ in all classification criteria except adaptation support.

### 5.2 RIALTO SCHEDULER

The scheduler of the Rialto OS [Jones et al. 95], [Jones et al. 96], [Jones et al. 97] is based on three fundamental abstractions:

- *Activities* are typically an executing program or application that comprises multiple threads of control. Resources are allocated to activities and their usage is charged to activities.
- *CPU reservations* are made by activities and are requested in the form: "reserve *x* units of time out of every *Y* units for activity *A*". Basically, period length and reservations for each period can be of arbitrary length.
- *Time constraints* are dynamic requests from threads to the scheduler to run a certain code segment within a specified start time and deadline to completion.

The scheduling decision, i.e., which threads to activate next, is based on a pre-computed scheduling graph. Each time a request for CPU reservation is issued, this scheduling graph is recomputed. In this scheduling graph, each node represents an activity with a CPU reservation, specified as time interval and period, or represents free computation time. For each base period, i.e., the lowest common denominator of periods from all CPU reservations, the scheduler traverses the tree in a depth first manner, but backtracks always to the root after visiting a leaf in the tree. Each node, i.e., activity that is crossed during the traversal, is scheduled for the specified amount of time. The execution time associated with the schedule graph is fixed. Free execution times are available for non-time-critical tasks. This fixed schedule graph keeps the number of context switches low and keeps the scheduling algorithm scalable.

If threads request time constraints, the scheduler analyzes their feasibility with the so-called *time interval assignment* data structure. This data structure is based on the knowledge represented in the schedule graph and checks whether enough free computation time is available between start time and deadline (including the already reserved time in the CPU reserve). Threads are not allowed to define time constraints when they might block – except for short blocking intervals for synchronization or I/O. When during the course of a scheduling graph traversal an interval assignment record for the current time is encountered, a thread with an active time constraint is selected according to EDF. Otherwise, threads of an activity are scheduled according to round-robin. Free time for non-time-critical tasks is also distributed according to round-robin.

If threads with time constraints block on a synchronization event, the thread priority (and its reservations) is passed to the holding thread.

### 5.3 SMART

The SMART scheduler [Nieh et al. 97] is designed for multimedia and real-time applications and is implemented in Solaris 2.5.1. The main idea of SMART is to differentiate between *importance* to determine the overall resource allocation for each task and *urgency* to determine when each task is given its allocation. Importance is valid for real-time and conventional tasks and is specified in the system by a tuple of *priority* and *biased virtual finishing time*. Here, the virtual finishing time, as known from fair-queuing schemes, is extended with a bias, which is a bounded offset measuring the ability of conventional tasks to tolerate longer and more varied service delays. Application developers can specify time constraints, i.e., deadlines and execution times, for a particular block of code, and they can use the system notification. The system notification is an upcall that informs the application that a deadline cannot be met and allows the application to adapt to the situation. Applications can query the scheduler for availability, which is an estimate of processor time consumption of an application relative to its processor allocation. Users of applications can specify priority and share to bias the allocation of resources for the different applications.

The SMART scheduler separates importance and urgency considerations. First, it identifies all tasks that are important enough to execute and collects them in a candidate set. Afterwards, it orders the candidate set according to urgency consideration. In more detail, the scheduler works as follows: if the tasks with the highest value-tuple is a conventional task, schedule it. The highest value-tuple is either determined by the highest priority or for equal priorities by the earliest biased virtual finishing time. If the task with the highest value-tuple is a real-time task, it creates a candidate set of all real-time tasks that have a higher value-tuple than the highest conventional task. The candidate set is scheduled according to the so-called best-effort real-time scheduling algorithm. Basically, this algorithm finds the task with the earliest deadline that can be executed without violating deadlines of tasks with higher value-tuples. SMART notifies applications if their computation cannot be completed before its deadline. This enables applications to implement downscaling.

There is no admission control implemented in SMART. Thus, SMART can only enforce real-time behavior in underload situations.

## 6 DISK MANAGEMENT

Magnetic and optical disks enable permanent storage of data.[1] The file system is the central OS abstraction to handle data on disk. However, in most commodity OSs, it is possible to by-pass the file system and to use raw disks, e.g., for database systems. The two main resources that are of importance for disk management, no matter whether the file system of an OS or raw disk is used, are:

- *Memory space on disk*: allocation of memory space at the right place on disks, i.e., appropriate data placement, can strongly influence the performance.

[1] We focus in this article only on issues related to magnetic disks.

- *Disk I/O bandwidth*: access to disk has to be multiplexed in the temporal domain. CPU scheduling algorithms cannot directly be applied for disks, because of the following reasons [Molano et al. 97]: (1) a disk access cannot be preempted, i.e., it is always necessary to read a whole block; (2) access times (which correspond to execution time for CPU scheduling) are not deterministic, because they depend on the position of the disk head and the position of data on disk; and (3) disk I/O represents the main performance bottleneck in today's systems.

Multimedia data can be managed on disk in two different ways [Steinmetz 95]: (1) the file organization on disk is not changed and the required real-time support is provided by special disk scheduling algorithms and large buffers to avoid jitter; or (2) the data placement is optimized for continuous multimedia data in distributed storage hierarchies like disk arrays [Chen et al. 94].

In the following subsections, we discuss file management, data placement, and disk scheduling issues of multimedia file systems separately.

### 6.1 File Management

The traditional access and control tasks of file systems, like storing file information in sources, objects, program libraries and executables, text, and accounting records, have to be extended for multimedia file systems with real-time characteristics, coping with larger file sizes (high disk bandwidth), and with multiple continuous and discrete data streams in parallel (real-time delivery) [Steinmetz 95]. The fact that in the last years storage devices have become only marginally faster compared to the exponentially increased performance of processors and networks, makes the effect of this discrepancy in speed for handling multimedia data by file systems even more important. This is documented by the large research activity to find new storage structures and retrieval techniques. The existing approaches can be categorized along multiple criteria, and we present a brief classification along architectural issues and data characteristics.

From the architectural perspective, multimedia file systems can be classified as [Shenoy et al. 99]:

- *Partitioned file systems* consist of multiple subfile systems, each tailored to handle data of a specific data type. An integration layer may provide transparent access to the data handled by the different subfile systems. There are multiple examples of systems using this approach, e.g., FFS [Leffler et al. 90], *Randomized I/O* (RIO) [Santos et al. 98], Shark [Haskin 93], Tiger Shark [Haskin et al. 96], and the combination of UFS and CMFS in [Ramakrishnan et al. 93].
- *Integrated file systems* multiplex all available resources (storage space, disk bandwidth, and buffer cache) among all multimedia data. Examples of integrated multimedia file systems are the file system of Nemesis [Barham 97], Fellini [Martin et al. 96], and Symphony [Shenoy et al. 98a].

Another way to classify multimedia file systems is to group the systems according to the supported multimedia data characteristics:

- General file systems capable of handling multimedia data to a certain extend, e.g., FFS [Leffler et al. 90], and *log-structured file systems* [Rosenblum 95], [Seltzer et al. 93], [Wang et al. 99a].
- Multimedia file systems optimized for continuous multimedia data (video and audio data), e.g., SBVS [Vernick et al. 96], Mitra [Ghandeharizadeh et al. 97], CMFS [Anderson et al. 92b], PFS [Lee et al. 97], Tiger [Bolosky et al. 96], [Bolosky et al. 97], Shark [Haskin 93], Tiger Shark [Haskin et al. 96], and CMSS [Lougher et al. 93].
- Multimedia file systems handling mixed-media workloads (continuous and discrete multimedia data), e.g., Fellini [Martin et al. 96], Symphony [Shenoy et al. 98a], MMFS [Niranjan et al. 97], the file system of Nemesis [Barham 97], and RIO [Santos et al. 98].

The file system of Nemesis [Barham 97] supports QoS guarantees using a device driver model. This model realizes a low-level abstraction providing separation of control and data path operations. To enable the file system layers to be executed as unprivileged code within shared libraries, data path modules supply translation and protection of I/O requests. QoS guarantees and isolation between clients are provided by scheduling low-level operations within the device drivers.

Fellini [Martin et al. 96] supports storage and retrieval of continuous and discrete multimedia data. The system provides rate guarantees for active clients by using admission control to limit the number of concurrent active clients.

Symphony [Shenoy et al. 98a] can manage heterogeneous multimedia data supporting the coexistence of multiple data type specific techniques. Symphony comprises a *QoS-aware disk scheduling* algorithm for real-time and non-real-time requests, and a storage manager supporting multiple block sizes and data type specific placement, failure recovery, and caching policies.

MMFS [Niranjan et al. 97] handles interactive multimedia applications by extending the UNIX file system. MMFS has a two-dimensional file structure for single-medium editing and multimedia playback: (1) a single-medium *strand* abstraction [Rangan et al. 91]; and (2) a multimedia file construct, which ties together multiple strands that belong logically together. MMFS uses application-specific information for performance optimization of interactive playback. This includes intelligent prefetching, state-based caching, prioritized real-time disk scheduling, and synchronized multi-stream retrieval.

RIO [Santos et al. 98] provides real-time data retrieval with statistical delay guarantees for continuous and discrete multimedia data. The system applies random data allocation on heterogeneous disks and partial data replication to achieve load balancing and high performance.

In addition to the above mentioned aspects, new mechanisms for multimedia file organization and metadata handling are needed. For instance, in MMFS [Niranjan et al. 97], each multimedia file has a unique *mnode*, and for every strand in a multimedia file exists a unique *inode*. mnodes include metadata of multimedia files and multimedia-specific metadata of each strand, e.g., recording rate, logical block size, and size of the application data unit.

Metadata management in Symphony [Shenoy et al. 98a] uses a *two-level metadata structure* (similar to inodes) allowing both data type specific structure for files and supporting the traditional byte stream interface. Like in UNIX, fixed size metadata structures are stored on a reserved area of the disk. The file metadata comprises, in addition to the traditional file metadata, information about block size used to store the file, type of data stored in the file, and a two-level index. The first index level maps logical units, e.g., frames, to byte offsets, and the second index level maps byte offsets to disk block locations.

Fellini [Martin et al. 96] uses the raw disk interface of UNIX to store data. It maintains the following information: raw disk partition headers containing free space administration information on the disks, file control blocks similar to UNIX inodes describing data layout on disk, and file data.

Minorca [Wang et al. 99b] divides the file system partition into multiple sections: super block section, cylinder group section, and extent section. Metadata such as inodes and directory blocks are allocated in the cylinder group section in order to maintain the contiguity of block allocation in the extent section.

### 6.2 DATA PLACEMENT

*Data placement* (also often referred to as *disk layout* and *data allocation*) and *disk scheduling* are responsible for the actual values of seek time, rotation time, and transfer time, which are the three major components determining disk efficiency [Garcia-Martinez et al. 2000]. There are a few general data placement strategies for multimedia applications in which read operations dominate and only few non-concurrent write operations occur:

- *Scattered placement*: blocks are allocated at arbitrary places on disk. Thus, sequential access to data will usually cause a large number of intra-file seeks and rotations resulting in high disk read times. However, in RIO [Santos et al. 98], random data placement is used to support mixed-media workloads stored on heterogeneous disk configurations. In their special scenario, the reported performance measurements show similar results as those for conventional striping schemes [Berson et al. 94].
- *Contiguous placement*: all data blocks of a file are successively stored on disk. Contiguous allocation will mostly result in better performance compared to scattered allocation. The problem of contiguous allocation is that it causes external fragmentation.
- *Locally contiguous placement* (also called *extent-based allocation*): the file is divided into multiple fragments (extents). All blocks of a fragment are stored contiguously, but fragments can be scattered over 1-n disks. The fragment size is usually determined by amount of data required for one service round. Locally contiguous placement causes less external fragmentation than contiguous placement.
- *Constrained placement:* this strategy restricts the average distance measured in tracks, between a finite sequence of blocks [Anderson et al. 92b], [Vin et al. 93]. Constrained placement represents a compromise of performance and fragmentation between scattered and contiguous placement. However, complex algorithms are needed to obey the defined constraints [Chang et al. 97]. This strategy takes into account only seek times and not rotation times.
- *VBR compressed data placement*: conventional fixed-sized clusters correspond to varying amounts of time, depending on the achieved compression [Bell et al. 95]. Alternatively, the system can store data in clusters that correspond to a fixed amount of time, with a variable cluster size. Additionally, compressed data might not correspond to an even number of disk sectors, which introduces the problem of *packing* data [Gemmell et al. 92].

To optimize write operations, *log-structured placement* has been developed to reduce disk seeks for write intensive applications [Rosenblum 95], [Seltzer et al. 93], [Wang et al. 99a]. When modifying blocks of data, log-structured systems do not store modified blocks in their original positions. Instead, all writes for all streams are performed sequentially in a large contiguous free space. Therefore, instead of requiring a seek (and possibly intra-file seeks) for each stream writing, only one seek is required prior to a number of write operations. However, this does not guarantee any improvement for read operations, and the mechanism is more complex to implement.

For systems managing multiple storage devices, there exist two possibilities of distributing data among disks [Gemmell et al. 95], [Garcia-Martinez et al. 2000]:

- *Data striping*: to realize a larger logical sector, many physical sectors from multiple disks are accessed in parallel.
- *Data interleaving*: requests of a disk are handled independent of requests from other disks. All fragments of a request can be stored on 1-n disks [Abbott 84].

Some multimedia file systems, e.g., Symphony [Shenoy et al. 98a] and Tiger Shark [Haskin et al. 96], use *striping* techniques to interleave both continuous and non-continuous multimedia data across multiple disks. There are two factors crucially determining the performance of multi-disk systems [Garcia-Martinez et al. 2000]:

- *Efficiency* in using each disk. The amount of seek and rotation times should be reduced as much as possible in order to have more time available for data transfer.
- *Fairness* in distributing the load over all disks.

These two factors largely depend on the data distribution strategy and the application characteristics. They are very important to achieve *synchronization*, which means the temporal relationship between different multimedia data streams [Steinmetz 95]. Synchronization is often achieved by storing and transmitting streams interleaved, e.g., by using a MPEG compression mechanism. Another solution is *time-stamping* of the multimedia data elements and appropriate buffering at the presentation system to enable the OS to synchronize related data elements of continuous and discrete multimedia data.

### 6.3 DISK SCHEDULING

Traditional disk scheduling algorithms focused mainly on reducing seek times [Denning 67], e.g., *Shortest Seek Time First* (SSTF) or *SCAN*. SSTF has high response time variations and may result in starvation of certain requests. SCAN reduces the response time variations and optimizes seek times by serving the requests in an elevator-like way. There exist many variations and hybrid solutions of the SSTF and SCAN algorithms that are widely used today [Geist et al. 87], [Coffman et al. 90], [Yu et al. 93].

Modern disk scheduling algorithms [Jacobson et al. 91], [Seltzer et al. 90] try to minimize the sum of seek and rotational delays by prioritizing, e.g., the request with the *Smallest Positioning Time First*.

However, disk scheduling algorithms for multimedia data requests need to optimize, beside the traditional criteria, also other criteria special for multimedia data including QoS guarantees [Steinmetz 95], [Gemmell et al. 95]. The following list represents an overview of recent multimedia data disk scheduling algorithms, which are primarily optimized for continuous data streams [Rompogiannakis et al. 98], [Garcia-Martinez et al. 2000]:

- *EDF strategy* [Liu et al. 73] serves the block request with the nearest deadline first. Strict EDF may cause low throughput and very high seek times. Thus, EDF is often adapted or combined with other disk scheduling strategies.
- *SCAN-EDF strategy* [Reddy et al. 94] combines the seek optimization of SCAN and the real-time guarantees of EDF. Requests are served according to their deadline. The request with the earliest deadline is served first like in EDF. If multiple requests have the same (or similar) deadline, SCAN is used to define the order to handle the requests. The efficiency of SCAN-EDF depends on how often the algorithm can be applied, i.e., how many requests have the same (or similar) deadline, because the SCAN optimization is only achieved for requests in the same deadline class [Reddy et al. 93].
- *Group Sweeping Strategy* (GSS) [Chen et al. 93], [Yu et al. 93] optimizes disk arm movement by using a variation of SCAN handling the requests in a round-robin fashion. GSS splits the requests of continuous media streams into multiple groups. The groups are handled in a fixed order. Within a group, SCAN is used to determine time and order of request serving. Thus, in one service round, a request may be handled first. In another service round, it may be the last request served in this group. To guarantee continuity of playout, a smoothing buffer is needed. The buffer size is depending of the service round time and the required data rate. Thus, the playout can first start at the end of the group containing the first retrieval requests when enough data is buffered. GSS represents a trade-off between optimizations of buffer space and disk arm movement. GSS is an improvement compared to SCAN, which requires a buffer for every continuous media request. However, GSS may reduce to SCAN when only one group is built, or in the other extreme case, GSS can behave like round-robin when every group contains only one request.
- *Scheduling in rounds*, e.g., [Berson et al. 94], [Gemmell et al. 95], [Özden et al. 96a], and [Triantafillou et al. 98], splits every continuous media requests into multiple *blocks* (so-called *fragments*) in a way that the playout duration of each fragment is of a certain constant time (normally 1-n seconds). The length of the round represents an upper time limit for the system to retrieve the next fragment from disk for all active requests. For each round, the amount of buffered data must not be less than the amount of consumed data avoiding that the amount of buffered data effectively decreases over the time. Disk scheduling algorithms with this property are called work-ahead-augmenting [Anderson et al. 92b] or buffer-conserving [Gemmell et al. 94]. Within a round, it is possible to use round-robin or SCAN scheduling.

However, there has only been done little work on disk scheduling algorithms for mixed multimedia data workloads, serving discrete and continuous multimedia data requests at the same time. Some examples are described in [Rompogiannakis et al. 98], [Lin et al. 91], [Nerjes et al. 98], [Reddy et al. 94], [Ramakrishnan et al. 93], and [Wijayaratne et al. 99]. These algorithms have to satisfy three performance goals: (1) display continuous media streams with minimal delay jitter; (2) serve discrete requests with small average response times; and (3) avoid starvation of discrete request and keep variation of response times low. In [Rompogiannakis et al. 98], disk scheduling algorithms for mixed-media workloads are classified by:

- *Number of separate scheduling phases per round*: one-phase algorithms produce mixed schedules, containing both discrete and continuous data requests. Two-phase algorithms have two, not timely overlapping, scheduling phases serving discrete and continuous data requests isolated in the corresponding phase.

- *Number of scheduling levels*: hierarchical scheduling algorithms for discrete data requests are based on defining *clusters*. The higher levels of the algorithms are concerned with the efficient scheduling of clusters of discrete requests. The lower levels are efficiently scheduling the requests within a cluster. The most important task to solve in this context is how to schedule discrete data requests within the rounds of continuous data requests, which are mostly served by SCAN variations.

For instance, *Cello* [Shenoy et al. 98b] uses such a two-level disk scheduling architecture. It combines a class-independent scheduler with a set of class-specific schedulers. Two time scales are considered in the two levels of the framework to allocate disk bandwidth: (1) coarse-grain allocation of bandwidth to application classes is performed by the class-independent scheduler; and (2) the fine-grain interleaving of requests is managed by the class-specific schedulers. This separation enables the co-existence of multiple disk scheduling mechanisms at a time depending on the application requirements.

## 7 MEMORY MANAGEMENT

Memory is an important resource, which has to be carefully managed. The virtual memory subsystem of commodity OSs allows processes to run in their own virtual address spaces and to use more memory than physically available. Thus, the memory manager has several complex tasks such as bookkeeping available resources and assigning physical memory to a single process [Steinmetz 95], [Tanenbaum 92]. Further key operations of the virtual memory system include [Cranor 98], [Cranor et al. 99]:

- Allocation of each process' virtual address space and mapping physical pages into a virtual address space with appropriate protection.
- The page fault handler manages unmapped and invalid memory references. Page faults happen when unmapped memory is accessed, and memory references that are inconsistent with the current protection are invalid.
- Loading data into memory and storing them back to disk.
- Duplicating an address space in case of a *fork* call.

Since virtual memory is mapped onto actual available memory, the memory manager has to do paging or swapping, but due to the real-time performance sensitiveness of multimedia applications, swapping should not be used in a multimedia OS [Steinmetz 95]. Thus, we focus on paging-based memory systems. Techniques such as demand-paging and memory-mapped files have been successfully used in commodity OSs [Schulzrinne 96], [Hand 99]. However, these techniques fail to support multimedia applications, because they introduce unpredictable memory access times, cause poor resource utilization, and reduce performance. In the following subsections, we present new approaches for memory allocation and utilization, data replacement, and prefetching using application-specific knowledge to solve these problems. Furthermore, we give a brief description of the UVM Virtual Memory System that replaces the traditional virtual memory system in NetBSD 1.4.

### 7.1 MEMORY ALLOCATION

Usually, upon process creation, a virtual address space is allocated which *contains* the data of the process. Physical memory is then allocated and assigned to a process and then mapped into the virtual address space of the process according to available resources and a global or local allocation scheme. This approach is also called *user-centered allocation*. Each process has its own share of the resources. However, traditional memory allocation on a per client (process) basis suffers from a linear increase of required memory with the number of processes.

In order to better utilize the available memory, several systems use so-called *data-centered allocation* where memory is allocated to data objects rather than to a single process. Thus, the data is seen as a resource principal. This enables more cost-effective data-sharing techniques [Garofalakis et al. 98], [Krishnan et al. 97]: (1) *batching* starts the video transmission when several clients request the same movie and allows several clients to share the same data stream; (2) *buffering* (or *bridging*) caches data between consecutive clients omitting new disk requests for the same data; (3) *stream merging* (or *adaptive piggy-backing*) displays the same video clip at different speeds to allow clients to catch up with each other and then share the same stream; (4) *content insertion* is a variation of stream merging, but rather than adjusting the display rate, new content, e.g., commercials, is inserted to align the consecutive playouts temporally; and (5) *perodic services* (or *enhanced pay-per-view*) assigns each clip a retrieval period where several clients can start at the beginning of each period to view the same movie and to share resources. These data-sharing techniques are used in several systems. For example, a per movie memory allocation scheme, i.e., a variant of the buffering scheme, for VoD applications is described in [Rotem et al. 95]. All buffers are shared among the clients watching the same movie and work like a sliding window on the continuous data. When the first client has consumed nearly all the data in the buffer, it starts to refresh the oldest buffers with new data. Periodic services are used in pyramid broadcasting [Viswanathan et al. 96]. The data is split in partitions of growing size, because the consumption rate of one partition is assumed to be lower than the downloading rate of the subsequent partition. Each partition is then broadcasted in short intervals on separate channels. A client does not send a request to the server, but instead it tunes into the channel transmitting the required data. The data is cached on the receiver side, and during the playout of a partition, the next partition is downloaded. In [Hua et al. 97] and [Gao et al. 98], the same broadcasting idea is used. However, to avoid very large partitions at the end of a movie and thus to reduce the client buffer requirement, the partitioning is changed such that not every partition increases in size, but only each $n^{th}$ partition. Performance evaluations show that the data-centered allocation schemes scale much better with the numbers of users compared to user-centered allocation. The total buffer space required is reduced, and the average response time is minimized by using a small partition size at the beginning of a movie.

The *memory reservation per storage device* mechanism [Garcia-Martinez et al. 2000] allocates a fixed, small number of memory buffers per storage device in a server-push VoD server using a cycle-based scheduler. In the simplest case, only two buffers of identical size are allocated per storage device. These buffers work co-operatively, and during each cycle, the buffers change task as data is received from disk. That is, data from one process is read into the first buffer, and when all the data is loaded into the buffer, the system starts to transmit the information to the client. At the same time, the disk starts to load data from the next client into the other buffer. In this way, the buffers change task from receiving disk data to transmitting data to the network until all clients are served. The admission control adjusts the number of concurrent users to prevent data loss when the buffers switch and ensures the maintenance of all client services.

In [Nakajima et al. 97], the traditional allocation and page-wiring mechanism in Real-Time Mach is changed. To avoid that privileged users monopolize memory usage by wiring unlimited amount of pages, only real-time threads are allowed to wire pages, though, only within their limited amount of allocated memory, i.e., if more pages are needed, a request has to be sent to the reservation system. Thus, pages may be wired in a secure way, and the reservation system controls the amount of memory allocated to each process.

### 7.2 Data Replacement

When there is need for more buffer space, and there are no available buffers, a buffer has to be replaced. How to best choose which buffer to replace depends on the application. However, due to the high data consumption rate in multimedia applications, data is often replaced before it might be reused. The gain of using a complex page replacement algorithm might be wasted and a traditional algorithm as described in [Effelsberg et al. 84] or [Tanenbaum 92] might be used. Nevertheless, in some multimedia applications where data often might be reused, proper replacement algorithms may increase performance. The *distance* [Özden et al. 96b], the *generalized interval caching* [Dan et al. 97], and the SHR [Kamath et al. 95] schemes, all replace buffers after the distance between consecutive clients playing back the same data and the amount of available buffers.

Usually, data replacement is handled by the OS kernel where most applications use the same mechanism. Thus, the OS has full control, but the used mechanism is often tuned to best overall performance and does not support application-specific requirements. In Nemesis [Hand 99], *self-paging* has been introduced as a technique to provide QoS to multimedia applications. The basic idea of self-paging is to "require every application to deal with all its own memory faults using its own concrete resources". All paging operations are removed from the kernel where the kernel is only responsible for dispatching fault notifications. This gives the application flexibility and control, which might be needed in multimedia systems, at the cost of maintaining its own virtual memory operations. However, a major problem of self-paging is to optimize the global system performance. Allocating resources directly to applications gives them more control, but that means optimizations for global performance improvement are not directly achieved.

### 7.3 Prefetching

The poor performance of demand-paging is due to the low disk access speeds. Therefore, prefetching data from disk to memory is better suited to support continuous playback of time-dependent data types. Prefetching is a mechanism to preload data from slow, high-latency storage devices such as disks to fast, low-latency storage like main memory. This reduces the response time of a data read request dramatically and increases the disk I/O bandwidth. Prefetching mechanisms in multimedia systems can take advantage of the sequential characteristics of multimedia presentations. For example, in [Anderson et al. 98], a read-ahead mechanism retrieves data before it is requested if the system determines that the accesses are sequential. In [Ng et al. 94], the utilization of buffers and disk is optimized by prefetching all the shortest database queries maximizing the number of processes that can be activated once the running process is finished. In [Tezuka et al. 96], assuming a linear playout of the continuous data stream, the data needed in the next period (determined by a tradeoff between the maximum concurrent streams and the initial delay) is prefetched into a shared buffer. Preloading data according to the loading and consuming rate and the available amount of buffers is described in [Zhang et al. 95].

In addition to the above mentioned prefetching mechanisms designed for multimedia applications, more general purpose facilities for retrieving data in advance are designed which also could be used for certain multimedia applications. The *informed prefetching and caching* strategy [Patterson et al. 95] preloads a certain amount of data where the buffers are allocated/deallocated according to a global max-min valuation. This mechanism is further developed in [Chang et al. 99] where the automatic hint generation, based on speculative pre-executions using mid-execution process states, is used to prefetch data for possible future read requests. Moreover, the *dependent-based prefetching*, described in [Roth et al. 98], captures the access patterns of linked data structures. A prefetch engine runs in parallel with the original program using these patterns to predict future data references. Finally, in [Lei et al. 97], an analytic approach to file prefetching is described. During the execution of a process a semantic data structure is built showing the file accesses. When a program is re-executed, the saved access trees are compared against the current access tree of the activity, and if a similarity is found, the stored tree is used to preload files.

Obviously, knowledge (or estimations) about application behavior might be used for both replacement and prefetching. In [Moser et al. 95], the buffer replacement and preloading strategy *least/most relevant for presentation* designed for interactive continuous data streams is presented. A multimedia object is replaced and prefetched according to its relevance value computed according to the presentation point/modus of the data playout. In [Halvorsen et al. 98], this algorithm is extended for multiple users and QoS support.

### 7.4 UVM Virtual Memory System

The UVM Virtual Memory System [Cranor 98], [Cranor et al. 99] replaces the virtual memory object, fault handling, and pager of the BSD virtual memory system; and retains only the machine dependent/independent layering and mapping structures. For example, the memory mapping is redesigned to increase efficiency and security; and the map entry fragmentation is reduced by memory wiring. In BSD, the memory object structure is a stand-alone abstraction and under control of the virtual memory system. In UVM, the memory object structure is considered as a secondary structure designed to be embedded with a handle for memory mapping resulting in better efficiency, more flexibility, and less conflicts with external kernel subsystems. The new copy-on-write mechanism avoids unnecessary page allocations and data copying, and grouping or clustering the allocation and use of resources improves performance. Finally, a virtual memory based data movement mechanism is introduced which allows data sharing with other subsystems, i.e., when combined with the I/O or IPC systems, it can reduce the data copying overhead in the kernel.

## 8 Management of Other Resources

This section takes a brief look at management aspects of OS resources that have not yet been discussed, like scheduling of system bus and cache management. Furthermore, we describe some mechanisms for speed improvements in memory access. Packet scheduling mechanisms to share network bandwidth between multiple streams at the host-network interface are not discussed here due to space considerations. All solutions for packet scheduling in OSs are adopted from packet scheduling in packet networks.

### 8.1 Bus Scheduling

The SCSI bus is a priority arbitrated bus. If multiple devices, e.g., disks, want to transfer data, the device with the highest priority will always get the bus. In systems with multiple disks, it is possible that real-time streams being supported from a low priority disk get starved from high priority disks that serve best effort requirements [Reddy 95]. DROPS [Härtig et al. 98] schedules requests to the SCSI subsystem such that the SCSI bandwidth can be fully exploited. It divides SCSI time into slots where the size of slots is determined by the worst case seek times of disk drives.

SCSI is a relatively old technology, and PCI has become the main bus technology for multimedia PCs and workstations [Nishikawa et al. 97]. However, to the best of our knowledge, no work has been reported on scheduling of PCI bus or other advanced bus technologies to support QoS. Probably, because the bus is no longer regarded as one of the most limiting performance bottlenecks, except in massive parallel I/O systems.

### 8.2 Cache Management

All real-time applications rely on predictable scheduling, but the memory cache design makes it hard to forecast and schedule the processor time [Härtig et al. 97]. Furthermore, memory bandwidth and the general OS performance has not increased at the same rate as CPU performance. Benchmarked performance can be improved by enlarging and speeding up static RAM-based cache memory, but the large amount of multimedia data that has to be handled by CPU and memory system will likely decrease cache hit ratios. If two processes use the same cache lines and are executed concurrently, there will not only be an increase in context switch overheads, but also a cache-interference cost that is more difficult to predict. Thus, the system performance may be dominated by slower main memory and I/O accesses. Furthermore, the busier a system is, the more likely it is that involuntary context switches occur, longer run queues must be searched by the scheduler, etc., flushing the caches even more frequently [Schulzrinne 96].

One approach to improve performance is to partition the second-level cache as described in [Härtig et al. 97], [Härtig et al. 98]. Working sets of real-time and time-sharing applications are allowed to be separated into different partitions of the second-level cache. The time-share applications then cannot disrupt the cached working sets of real-time applications, which leads to better worst case predictability.

Another approach is discussed in [Philbin et al. 96]. A very low overhead thread package is used letting the application specify each thread's use of data. The thread scheduler then execute in turn all threads using the same data. In this way, the data that is already in the cache is used by all threads needing it before it is flushed.

Bershad et al. [Bershad et al. 94] describe an approach using conflict detection and resolution to implement a cheap, large, and fast direct-mapped cache. The conflicts are detected by recording and summarizing a history of cache misses, and a software policy within the OS's virtual memory system removes conflicts by dynamically remapping pages. This approach nearly matches the performance of a two-way set associative cache, but with lower hardware cost and lower complexity.

### 8.3 Speed Improvements in Memory Accesses

The term dynamic RAM (DRAM), coined to indicate that any random access in memory takes the same amount of time, is slightly misleading. Most modern DRAMs provide special capabilities that make it possible to perform some accesses faster than others [McKee et al. 98]. For example, consecutive accesses to the same row in a page-mode memory are faster than random accesses, and consecutive accesses that hit different memory banks in a multi-bank system allow concurrency and are thus faster than accesses that hit the same bank. The key point is that the order of the requests strongly affects the performance of the memory devices. For certain classes of computations, like those which involve streams of data where a high degree of spatial locality is present and where we, at least in theory, have a perfect knowledge of the future references, a reordering of the memory accesses might give an improvement in memory bandwidth.

The most common method to reduce latency is to increase the cache line size, i.e., using the memory bandwidth to fill several cache locations at the same time for each access. However, if the stream has a non-unit-stride (stride is the distance between successive stream elements in memory), i.e., the presentation of successive data elements does not follow each other in memory, the cache will load data which will not be used. Thus, lengthening the cache line size increases the effective bandwidth of unit-stride streams, but decreases the cache hit rate for non-streamed accesses.

Another way of improving memory bandwidth in memory-cache data transfers for streamed access patterns is described in [McKee et al. 98]. First, since streams often have no temporal locality, they provide a separate buffer storage for streamed data. This means that streamed data elements, which often are replaced before they might be reused, do not affect the replacement of data elements that might benefit from caching. Second, to take advantage of the order sensitivity of the memory system, a memory-scheduling unit is added to reorder the accesses. During compile-time, information about addresses, strides of a stream, and number of data elements are collected enabling the memory-scheduling unit to reorder the requests during run-time.

## 9 I/O Tuning

Traditionally, there are several different possible data transfers and copy operations within an end-system as shown in Figure 5. These often involve several different components. Using the disk-to-network data path as an example, a data object is first transferred from disk to main memory (A). The data object is then managed by the many subsystems within the OS designed with different objectives, running in their own domain (either in user or kernel space), and therefore, managing their buffers differently. Due to different buffer representations and protection mechanisms, data is usually copied, at a high cost, from domain to domain ((B), (C), or (D)) to allow the different subsystems to manipulate the data. Finally, the data object is transferred to the network interface (E). In addition to all these data transfers, the data object is loaded into the cache (F) and CPU registers (G) when the data object is manipulated.

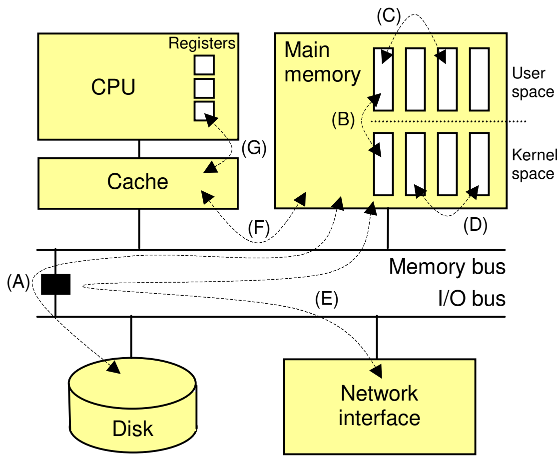


*Figure 5: Data transfers and copy operations*

Figure 5 clearly identifies the reason for the poor performance of the traditional I/O system. Data is copied several times between different memory address spaces which also causes several context switches. Both, copy operations and context switches represent the main performance bottleneck. Furthermore, different subsystems, e.g., file system and communication subsystem, are not integrated. Thus, they include redundant functionality like buffer management, and several identical copies of a data object might be stored in main memory, which in turn reduces the effective size of the physical memory. Finally, when concurrent users request the same data, the different subsystems might have to perform the same operations on the same data several times.

We distinguish three types of copy operations: memory-CPU, direct I/O (i.e., memory – I/O device), and memory-memory. Solutions for these types of copy operations have been developed for general purpose and application specific systems. The two last subsections describe a sample approach for each.

### 9.1 Memory- CPU Copy Operations

Data manipulations are time consuming and are often part of different, distinct program modules or communication protocol layers, which typically access data independently of each other. Consequently, each data manipulation may require access to uncached data resulting in loading the data from memory to a CPU register, manipulating it, and possibly storing it back to memory. Thus, these repeated memory-CPU data transfers, denoted (F) and (G) in Figure 5, can have large impacts on the achieved I/O bandwidth. To decrease the number of memory-CPU data transfers, *integrated layer processing* [Abbott et al. 93], [Clark et al. 90] performs all data manipulation steps, e.g., calculating error detection checksums, executing encryption schemes, transforming data for presentation, and moving data between address spaces, in one or two integrated processing loops instead of performing them stepwise as in most systems.

### 9.2 Memory - I/O Device Copy Operations

Data is transferred between hardware devices, such as disks and network adapters, and applications' physical memory. This is often done via an intermediate subsystem, like the file system or the communication system, adding an extra memory copy. A mechanism to transfer data without multiple copying is *direct I/O*, which in some form is available in several commodity OSs today, e.g., Solaris and Windows NT. *Direct memory access* (DMA) or *programmed I/O* (PIO) is used to transfer data directly into a frame buffer where, e.g., the file system's buffer cache is omitted in a data transfer from disk to application. Without involving the CPU in the data transfers, DMA can achieve transfer rates close to the limits of main memory and the I/O bus, but DMA increases complexity in the device adapters, and caches are often not coherent with respect to DMA [Druschel et al. 93b]. Using PIO, on the other hand, the CPU is required to transfer every word of data between memory and the I/O adapter. Thus, only a fraction of the peak I/O bandwidth is achieved. Due to high transfer rates, DMA is often used for direct I/O data transfers. However, despite the reduced bandwidth, PIO can sometimes be preferable over DMA. If data manipulations, e.g., checksum calculations, can be integrated with the PIO data transfer, it is possible to save one memory access, and after a programmed data movement, the data may still reside in the cache, reducing further memory traffic.

In case of application-disk transfers, direct I/O can often be applied since the file system usually does not touch the data itself. However, in case of application-network adapter transfers, the communication system must generate packets, calculate checksums, etc., making it harder to avoid the data transfer through the communication system. Nevertheless, there are several attempts to avoid data touching and copy operation transfers, i.e., reducing the traditional (B)(E) data path in Figure 5 to only (E). Afterburner [Dalton et al. 93] and medusa [Banks et al. 93] copy data directly onto the on-board memory using PIO, with integrated checksum and data length calculation, leaving just enough space in front of the cluster to add a packed header. Using DMA and a user-level implementation of the communication software, the *application device channel* [Druschel 96], [Druschel et al. 94] gives restricted but direct access to an ATM network adaptor removing the OS kernel from the critical network send/receive path. In [Yau et al. 96], no memory-to-memory copying is needed using shared buffers or direct media streaming by linking the device and network connection together. Finally, in [Chu 96] and [Kitamura et al. 95], zero-copy communication system architectures are reported for TCP and ATM respectively. Virtual memory page remapping (see next subsection) is used to eliminate copying between applications running in user space and the OS kernel, and DMA is used to transfer data between memory and the network buffer.

### 9.3 Memory-Memory Copy Operations

Direct I/O is typically used when transferring data between main memory and a hardware device as described above. However, data transfers between different process address spaces is done through well-defined channels, like pipes, sockets, files, and special devices, giving each process full control of its own data [McKusick et al. 96]. Nevertheless, such physical copying is slow and requires at least two system calls per transaction, i.e., one on sender and one on receiver side. One way of reducing the IPC costs is to use *virtual page (re)mapping*. That is, the data element is not physically copied byte by byte, but only the address in virtual memory to the data element in physical memory is copied into the receiver's address space. Access rights to the data object after the data transfer are determined by the used semantic:

- The *copy model* copies all data from domain to domain giving each process full control of its own data at the cost of cross domain data copying and maintaining several identical copies in memory.
- The *move model* removes the data from the source domain by virtually remapping the data into the destination domain avoiding the multiple-copies problem. However, if the source later needs to re-access the moved data, e.g., when handling a retransmission request, the data must be fetched back.
- The *share model* makes the transferred data visible and accessible to both the source and the target domain by keeping pointers in virtual memory to the same physical pages, i.e., by using *shared memory* where several processes map the same data into their address space. Thus, all the sharing processes may access the same piece of memory without any system call overhead other than the initial cost of mapping the memory.

Several general cross-domain data copy avoidance architectures are suggested trying to minimize respectively to eliminate all (C), (B), and (D) copy operations depicted in Figure 5. Tenex [Bobrow et al. 72] was one of the first systems to use virtual copying, i.e., several pointers in virtual memory refer to one physical page. Accent [Fitzgerald et al. 86], [Rashid et al. 81] generalized the concepts of Tenex by integrating virtual memory management and IPC in such a way that large data transfers could use memory mapping techniques rather than physical data copying. The V distributed system [Cheriton 88] and the DASH IPC mechanism [Tzou et al. 91] use page remapping, and the *container shipping* facility [Anderson 95], [Pasquale et al. 94] uses virtual inter-domain data transfers based on the move model where all in-memory copying is removed. Furthermore, *fast buffers* (fbufs) [Druschel 96], [Druschel et al. 93a] is a facility for I/O buffer management and data transfers across protection domain boundaries primarily designed for handling network streams using shared virtual memory is combined with virtual page remapping. In [Thadani et al. 95], fbufs is extended to a zero-copy I/O framework. Fast in-kernel data paths between I/O objects, increasing throughput and reducing context switch operations, are described in [Fall 94], [Fall et al. 94]. A new system call, `splice()`, moves data asynchronously and without user-process intervention to and from I/O objects specified by file descriptors. These descriptors specify the source and sink of I/O data respectively. This system call is extended in the `stream()` system call of the Roadrunner I/O system [Miller et al. 98a], [Miller et al. 98b] to support kernel data streaming between any pair of I/O elements without crossing any virtual memory boundaries using techniques derived from stackable file systems. The Genie I/O system [Brustoloni 99], [Brustoloni et al. 96], [Brustoloni et al. 97] inputs or outputs data to or from shared buffers in-place (i.e., directly to or from application buffers) without touching distinct intermediate system buffers. Data is shared by managing reference counters, and a page is only deallocated if there are no processes referencing this page. The universal continuous media I/O system [Cranor et al. 94], [Cranor et al. 95] combines all types of I/O into a single abstraction. The buffer management system is allowed to align data buffers on page boundaries so that data can be moved without copying which means that the kernel and the application are sharing a data buffer rather than maintaining their own separate copy. The UVM Virtual Memory System [Cranor 98], [Cranor et al. 99] data movement mechanism provides new techniques that allow processes to exchange and share data in memory without copying. The page layout and page transfer facilities give support for page loan out and reception of pages of memory, and the map entry passing enables exchange chunks of the processes' virtual address space.

### 9.4 IO-Lite

IO-Lite [Pai 97], [Pai et al. 99] is an I/O buffering and caching system for a general purpose OS inspired by the fbuf mechanism. IO-Lite unifies all buffering in a system. In particular, buffering in all subsystems are integrated, and a single physical copy of the data is shared safely and concurrently. This is achieved by storing buffered I/O data in immutable buffers whose location in memory never change. Access control and protection is ensured at the granularity of processes by maintaining access control lists to cached pools of buffers. For cross-domain data transfers, IO-Lite combines page remapping and shared memory.

All data is encapsulated in mutable buffer aggregates, which are then passed among the different subsystems and applications by reference. The sharing of read-only immutable buffers enables efficient transfers of I/O data across protection domain boundaries, i.e., all subsystems may safely refer to the same physical copy of the data without problems of synchronization, protection, consistency, etc. However, the price to pay is that data cannot be modified in-place. This is solved by the buffer aggregate abstraction. The aggregate is mutable, and a modified value is stored in a new buffer, and the modified sections are logically joined with the unchanged data through pointer manipulation.

### 9.5 Multimedia Mbuf

The *multimedia mbuf* (mmbuf) [Buddhikot et al. 98], [Buddhikot 98] is specially designed for disk-to-network data transfers. It provides a zero-copy data path for networked multimedia applications by unifying the buffering structure in file I/O and network I/O. This buffer system looks like a collection of clustered mbufs that can be dynamically allocated and chained. The mmbuf header includes references to mbuf header and buffer cache header. By manipulating the mmbuf header, the mmbuf can be transformed either into a traditional buffer, that a file system and a disk driver can handle, or an mbuf, which the network protocols and network drivers can understand.

A new interface is provided to retrieve and send data, which coexist with the old file system interface. The old buffer cache is bypassed by reading data from a file into an mmbuf chain. Both synchronous (blocking) and asynchronous (non-blocking) operations are supported, and read and send requests for multiple streams can be bunched together in a single call minimizing system call overhead. At setup time, each stream allocates a ring of buffers, each of which is an mmbuf chain. The size of each buffer element, i.e., the mmbuf chain, depends on the size of the multimedia frame it stores, and each buffer element can be in one of four states: empty, reading, full, or sending. Furthermore, to coordinate the data read and send activities, two pointers (read and send) to the ring buffer are maintained. Then, for each periodic invocation of the stream process, these pointers are used to handle data transfers. If the read pointer is pointing to a buffer element in the empty state, data is read into this chain of mmbufs, and the pointer is advanced to the next succeeding chain on which the next read is performed. If the send pointer is holding a full buffer element, the data stored in this buffer element is transmitted.

## 10 Conclusions

The aim of this article is to give an overview of recent developments in the area of OS support for multimedia applications. This is an active area, and a lot of valuable research results have been published. Thus, we have not discussed or cited *all* recent results, but tried to identify the major approaches and to present at least one representative for each.

Time-dependent multimedia data types, like audio and video, will be a natural part of future applications and integrated together with time-independent data types, like text, graphics, and images. Commodity OSs do not presently support all the requirements of multimedia systems. New OS abstractions need to be developed to support a mix of applications with real-time and best effort requirements and to provide the necessary performance. Thus, management of all system resources, including processors, main memory, network, disk space, and disk I/O, is an important issue. The management needed encompasses admission control, allocation and scheduling, accounting, and adaptation. Proposed approaches for better multimedia support include:

- New OS structures and architectures, like the library OSs Exokernel and Nemesis.
- New mechanisms that are especially tailored for QoS support, like specialized CPU and disk scheduling.
- New system abstractions, like resource principals for resource ownership, inheritance of the associated priorities, and accounting of resource utilization.
- Extended system abstractions to additionally support new requirements, like synchronization support and metadata in file systems.
- Avoiding the major system bottlenecks, like copy operations avoidance through page remapping.
- Support for user-level control over resources including user-level communication.

It is not clear how new OS architectures should look like or even if they are really needed at all. Monolithic and µ-kernel architectures can be developed further, and a careful design and implementation of such systems can provide both good performance and build on time proven approaches. When proposing new architectures, it becomes very important to demonstrate both comparable or better performance and better functionality than in existing solutions. Furthermore, it is important to implement and evaluate integrated systems and not only to study one isolated aspect. In this respect, Nemesis is probably the most advanced system.

To evaluate and compare performance and functionality of new approaches, more detailed performance measurements and analysis are necessary. This implies designing and implementing systems, and developing and using a common set of micro- and application benchmarks for evaluation of multimedia systems. The field is still very active, and much work remains to be done before it becomes known how to design and implement multimedia platforms.

## ACKNOWLEDGEMENTS

We would like to thank Frank Eliassen, Liviu Iftode, Martin Karsten, Ketil Lund, Chuanbao Wang, and Lars Wolf for reviewing earlier versions of this paper and their valuable comments and suggestions.

## REFERENCES

[Abbott 84] Abbott, C.: Efficient Editing of Digital Sound on Disk, Journal of Audio Engineering, Vol. 32, No. 6, June 1984, pp. 394-402

[Abbott et al. 93] Abbott, M.B., Peterson, L.L.: Increasing Network Throughput by Integrating Protocol Layers, IEEE/ACM Transactions on Networking, Vol. 1, No. 5, October 1993, pp. 600-610

[Anderson 95] Anderson, E.W.: Container Shipping: a Uniform Interface for Fast, Efficient, High-Bandwidth I/O, PhD Thesis, Computer Science and Engineering Department, University of California, San Diego, CA, USA, 1995

[Anderson et al. 90] Anderson, D.P., Tzou, S.Y., Wahbe, R., Govindan, R., Andrews, M.: Support for Continuous Media in the DASH System, Proc. of 10th Int. Conf. on Distributed Computing Systems (ICDCS'90), Paris, France, May 1990, pp. 54-61

[Anderson et al. 92a] Anderson, T.E., Bershad, B.N, Lazowska, E.D, Levy, H.M, Scheduler Activations: Effective Kernel Support for the User-Level Management of Parallelism, ACM Transactions on Computer Systems, Vol. 10, No. 1, February 1992, pp. 53-79

[Anderson et al. 92b] Anderson, D., Osawa, Y., Govindan, R.: A File System for Continuous Media, ACM Transactions on Computer Systems, Vol. 10, No. 4, November 1992, pp. 311-337

[Anderson et al. 98] Anderson, D.C., Chase, J.S., Gadde, S., Gallatin, A.J., Yocum, K.G., Feeley, M.J.: Cheating the I/O Bottleneck: Network Storage with Trapeze/Myrinet, Proc. of 1998 USENIX Annual Technical Conf., New Orleans, LA, USA, June 1998

[Araki et al. 98] Araki, S., Bilas, A., Dubnicki, C., Edler, J., Konishi, K., Philbin, J.: User-Space Communication: A Quantitative Study, Proc. of 10th Int. Conf. of High Performance Computing and Communications (SuperComputing'98), Orlando, FL, USA, November 1998

[Banga et al. 99] Banga, G., Drutchel, P., Mogul, J. C.: Resource Containers: A New Facility for Resource Management in Server Systems, Proc. of 3rd USENIX Symp. on Operating Systems Design and Implementation (OSDI'99), New Orleans, LA, USA, February 1999

[Banks et al. 93] Banks, D., Prudence, M.: A High-Performance Network Architecture for a PA-RISC Workstation, IEEE Journal on Selected Areas in Communications, Vol. 11, No. 2, February 1993, pp. 191-202

[Barham 97] Barham, P.R.: A Fresh Approach to File System Quality of Service, Proc. of 7th Int. Workshop on Network and Operating System Support for Digital Audio And Video (NOSSDAV'97), St. Louis, MO, USA, May 1997, pp. 119-128

[Bavier et al. 98a] Bavier, A., Peterson, L., Mosberger, D.: BERT: A Scheduler for Best-Effort and Realtime Paths, Technical Report TR 587-98, Princeton University, Princeton, NJ, USA, August 1998

[Bavier et al. 98b] Bavier, A., Montz, B. Peterson, L.: Predicting MPEG Execution Times, Proc. of 1998 ACM Int. Conf. on Measurement and Modeling of Computer Systems (SIGMETRICS'98), Madison, WI, USA, June 1998, pp. 131-140

[Bell et al. 95] Bell, T.C., Moffat, A., Witten, I.H., Zobel, J.: The MG Retrieval System: Compressing for Space and Speed, Communications of the ACM, Vol. 38, No. 4, April 1995, pp. 41-42

[Bershad et al. 94] Bershad, B.N., Lee, D., Romer , T.H., Chen, J.B.: Avoiding Conflict Misses Dynamically in Large Direct-Mapped Caches, Proc. of 6th Int. Conf. On Architectural Support for Programming Languages and Operating Systems (ASPLOS-VI), San Jose, CA, USA, October 1994, pp. 158-170

[Berson et al. 94] Berson, S., Ghandeharizadeh, S., Muntz, R.R., Ju, X.: Staggered Striping in Multimedia Information Systems, Proc. of 1994 ACM Int. Conf. on Management of Data (SIGMOD'94), Minneapolis, MN, USA, May 1994, pp. 70-90

[Bobrow et al. 72] Bobrow, D.G., Burchfiel, J.D., Murphy, D.L., Tomlinson, R.S., Beranek, B.: Tenex, A Paged Time Sharing System for the PDP-10, Communications of the ACM, Vol. 15, No. 3, March 1972, pp. 135-143

[Bolosky et al. 96] Bolosky, W., Barrera, J., Draves, R., Fitzgerald, R., Gibson, G., Jones, M., Levi, S., Myhrvold, N., Rashid, R.: The Tiger Video File Server, Proc. of 6th Int. Workshop on Network and Operating System Support for Digital Audio and Video (NOSSDAV'96), Zushi, Japan, April 1996, pp. 212-223

[Bolosky et al. 97] Bolosky, W.J., Fitzgerald, R.P., Douceur, J.R.: Distributed Schedule Management in the Tiger Video File Server, Proc. of 16th ACM Symp. on Operating System Principles (SOSP'97), St. Malo, France, October 1997, pp. 212-223

[Bringsrud et al. 93] Bringsrud, K.A., Pedersen, G.: Distributed Electronic Class Rooms with Large Electronic White Boards, Proc. of 4th Joint European Networking Conf. (JENC4), Trondheim, Norway, May 1993, pp. 132-144

[Bruno et al. 98] Bruno, J., Gabber, E., Özden, B., Silberschatz, A.: The Eclipse Operating System: Providing Quality of Service via Reservation Domains, Proc. of 1998 USENIX Annual Technical Conf., New Orleans, LA, June 1998

[Brustoloni 99] Brustoloni, J.C.: Interoperation of Copy Avoidance in Network and File I/O, Proc. of 18th IEEE Conf. on Computer Communications (INFOCOM'99), New York, NY, USA, March 1999

[Brustoloni et al. 96] Brustoloni, J.C., Steenkiste, P.: Effects of Buffering Semantics on I/O Performance, Proc. of 2nd USENIX Symp. on Operating Systems Design and Implementation (OSDI'96), Seattle, WA, USA, October 1996, pp. 227-291

[Brustoloni et al. 97] Brustoloni, J.C., Steenkiste, P.: Evaluation of Data Passing and Scheduling Avoidance, Proc. of 7th Int. Workshop on Network and Operating System Support for Digital Audio and Video (NOSSDAV'97), St. Louis, MO, USA, May 1997, pp. 101-111

[Buddhikot et al. 98] Buddhikot, M.M., Chen, X.J., Wu, D., Parulkar, G.M.: Enhancements to 4.4BSD UNIX for Efficient Networked Multimedia in Project MARS, Proceeding of IEEE Int. Conf. on Multimedia Computing and Systems (ICMCS'98), Austin, TX, USA, June/July 1998

[Buddhikot 98] Buddhikot, M.M: Project MARS: Scalable, High Performance, Web Based Multimedia-on-Demand (MOD) Services and Servers, PhD Thesis, Sever Institute of Technology, Department of Computer Science, Washington University, St. Louis, MO, USA, August 1998

[Chang et al. 97] Chang, E., Garcia-Molina, H.: Reducing Initial Latency in Media Servers, IEEE Multimedia, Vol. 4, No. 3, July-September 1997, pp. 50-61

[Chang et al. 99] Chang, F., Gibson, G.A.: Automatic I/O Hint Generation through Speculative Execution, Proc. of 3rd USENIX Symp. on Operating Systems Design and Implementation (OSDI'99), New Orleans, LA, USA, February 1999, pp. 1-14

[Chen et al. 93] Chen, M.-S., Kandlur, D.D., Yu, P.S.: Optimization of the Group Sweep Scheduling (GSS) with Heterogeneous Multimedia Streams, Proc. of 1st ACM Multimedia Conf. (ACM MM'93), Anaheim, CA, USA, August 1993, pp. 235-241

[Chen et al. 94] Chen, P.M., Lee, E.K., Gibson, G.A., Katz, R.H., Patterson, D.A.: RAID: High-Performance, Reliable, Secondary Storage, ACM Computing Surveys, Vol. 26, No. 2, June 1994, pp. 145-185

[Chen et al. 96] Chen, J.B., Endo, Y., Chan, K., Mazières, D., Dias, D., Seltzer, M.I., Smith, M.D.: The Measured Performance of Personal Computer Operating Systems, ACM Transactions on Computer Systems, Vol. 14, No. 1, February 1996, pp. 3-40

[Cheriton 88] Cheriton, D.R.: The V Distributed System, Communications of the ACM, Vol. 31, No. 3, March 1988, pp. 314-333

[Chu 96] Chu, H.-K.J.: Zero-Copy TCP in Solaris, Proc. of 1996 USENIX Annual Technical Conf., San Diego, CA, USA, January 1996, pp. 253-264

[Chu et al. 99] Chu, H.-H., Nahrstedt, K.: CPU Service Classes for Multimedia Applications, Proc. of IEEE Int. Conf. on Multimedia Computing and Systems (ICMCS'99), Florence, Italy, June 1999

[Clark et al. 90] Clark, D.D., Tennenhouse, D.L.: Architectural Considerations for a New Generation of Protocols, Proc. of ACM SIGCOMM'90, Philadelphia, PA, USA, September 1990, pp. 200-208

[Clark et al. 92] Clark, R.K., Jensen, E.D., Reynolds, F.D.: An Architectural Overview of the Alpha Real-Time Distributed Kernel, Workshop on Micro-Kernels and other Kernel Architectures, April 1992

[Coffman et al. 90] Coffman, J., Hofri, M.: Queuing Models of Secondary Storage Devices, Stochastic Analysis of Computer and Communication Systems, Takagi, H. (Ed.), North-Holland, 1990

[Coulson et al. 94] Coulson, G., Blair, G., Robin, P., Shepherd, D.: Supporting Continuous Media Applications in a Micro-Kernel Environment, in: Spaniol, O. (Ed.): Architecture and Protocols for High-Speed Networks, Kluwer Academic Publishers, 1994

[Coulson et al. 95] Coulson, G., Campbell, A., Robin, P., Blair, G., Papathomas, M. Hutchinson, D.: The Design of a QoS Controlled ATM Based Communication System in Chorus, IEEE Journal on Selected Areas of Communications, Vol. 13, No. 4, May 1995, pp. 686-699

[Cranor 98] Cranor, C.D.: The Design and Implementation of the UVM Virtual Memory System, PhD Thesis, Sever Institute of Technology, Department of Computer Science, Washington University, St. Louis, MO, USA, August 1998

[Cranor et al. 94] Cranor, C.D., Parulkar, G.M.: Universal Continuous Media I/O: Design and Implementation, Technical Report WUCS-94-34, Department of Computer Science, Washington University, St. Louis, MO, USA, 1994

[Cranor et al. 95] Cranor, C.D., Parulkar, G.M.: Design of Universal Continuous Media I/O, Proc. of 5th Int. Workshop on Network and Operating Systems Support for Digital Audio and Video (NOSSDAV'95), Durham, NH, USA, April 1995, pp. 83-86

[Cranor et al. 99] Cranor, C.D., Parulkar, G.M.: The UVM Virtual Memory System, Proc. of 1999 USENIX Annual Technical Conf., Monterey, CA, USA, June 1999

[Dalton et al. 93] Dalton, C., Watson, G., Banks, D., Calamvokis, C., Edwards, A., Lumley, J.: Afterburner, IEEE Network, Vol. 7, No. 4, July 1993, pp. 36-43

[Dan et al. 97] Dan, A., Sitaram, D.: Multimedia Caching Strategies for Heterogeneous Application and Server Environments, Multimedia Tools and Applications, Vol. 4, No. 3, May 1997, pp. 279 – 312

[Demers et al. 90] Demers, A., Keshav, S., Shenker, S.: Analysis and Simulation of a Fair Queueing Algorithm, Internetworking: Research and Experience, Vol. 1, No. 1, September 1990, pp. 3-26

[Denning 67] Denning, P.J.: Effects of Scheduling on File Memory Operations, Proc. AFIPS Conf., April 1967, pp. 9-21

[Druschel 96] Druschel, P.: Operating System Support for High-Speed Communication, Communication of the ACM, Vol. 39, No. 9, September 1996, pp. 41-51

[Druschel et al. 93a] Druschel, P., Peterson, L.L.: Fbufs: A High-Bandwidth Cross-Domain Transfer Facility, Proc. of 14th ACM Symp. on Operating Systems Principles (SOSP'93), Asheville, NC, USA, December 1993, pp. 189-202

[Druschel et al. 93b] Druschel, P., Abbot, M.B., Pagels, M.A., Peterson, L.L.: Network Subsystem Design, IEEE Network, Vol. 7, No. 4, July 1993, pp. 8-17

[Druschel et al. 94] Druschel, P., Peterson, L.L., Davie, B.S.: Experiences with a High-Speed Network Adaptor: A Software Perspective, Proc. of ACM SIGCOMM'94, London, UK, September 1994, pp. 2-13

[Effelsberg et al. 84] Effelsberg, W., Härder, T.: Principles of Database Buffer Management, ACM Transactions on Database Systems, Vol. 9, No. 4, December 1984, pp. 560-595

[Engler et al. 95] Engler, D., Gupta, S.K., Kaashoek, F.: AVM: Application-Level Virtual Memory, Proc. of 5th Workshop on Hot Topics in Operating Systems (HotOS-V), Orcas Island, WA, USA, May 1995

[Fall 94] Fall, K.R.: A Peer-to-Peer I/O System in Support of I/O Intensive Workloads, PhD Thesis, Computer Science and Engineering Department, University of California, San Diego, CA, USA, 1994

[Fall et al. 94] Fall, K., Pasquale, J.: Improving Continuous-Media Playback Performance with In-Kernel Data Paths, Proc. of IEEE Int. Conf. on Multimedia Computing and Systems (ICMCS'94), Boston, MA, USA, May 1994, pp. 100-109

[Fitzgerald et al. 86] Fitzgerald, R., Rashid, R.F: The Integration of Virtual Memory Management and Interprocess Communication in Accent, ACM Transactions on Computer Systems, Vol. 4, No. 2, May 1986, pp. 147-177

[Ford et al. 94] Ford, B., Lepreau, J.: Evolving Mach 3.0 to the Migrating Thread Model, Proc. of 1994 USENIX Winter Conf., San Francisco, CA, USA, January 1994

[Ford et al. 96] Ford, B., Susarla, S.: CPU Inheritance Scheduling, Proc. of 2nd USENIX Symp. on Operating Systems Design and Implementation (OSDI'96), Seattle, WA, USA, October 1996, pp. 91-105

[Gao et al. 98] Gao, L., Kurose, J., Towsley, D.: Efficient Schemes for Broadcasting Popular Videos, Proc. of 8th Int. Workshop on Network and Operating Systems Support for Digital Audio and Video (NOSSDAV'98), Cambridge, UK

[Garofalakis et al. 98] Garofalakis, M.N., Özden, B., Silberschatz, A.: On Periodic Resource Sheduling for Continuous-Media Databases, The VLDB Journal, Vol. 7, No. 4, 1998, pp. 206-225

[Garcia-Martinez et al. 2000] Garcia-Martinez, A., Fernadez-Conde, J., Vina, A.: Efficient Memory Management in VoD Servers, to appear in: Computer Communications, 2000

[Gecsei 97] Gecsei, J.: Adaptation in Distributed Multimedia Systems, IEEE Multimedia, Vol. 4, No. 2, April-June 1997, pp. 58-66

[Geist et al. 87] Geist, R., Daniel, S.: A Continuum of Disk Scheduling Algorithms, ACM Transactions on Computer Systems, February 1987, Vol. 5, No. 1, pp. 77-92

[Gemmell et al. 92] Gemmell, D.J., Christodoulakis, S.: Principles of Delay Sensitive Multimedia Data Storage and Retrieval, ACM Transactions on Information Systems, Vol. 10, No. 1, January 1992, pp. 51-90

[Gemmell et al. 94] Gemmell, D.J., Han, J.: Multimedia Network File Servers: Multichannel Delay Sensitive Data Retrieval, Multimedia Systems, Vol. 1, No. 6, April 1994, pp. 240-252

[Gemmell et al. 95] Gemmell, D.J., Vin, H.M., Kandlur, D.D., Rangan, P.V., Rowe, L.A.: Multimedia Storage Servers: A Tutorial, IEEE Computer, Vol. 28, No. 5, May 1995, pp. 40-49

[Ghandeharizadeh et al. 97] Ghandeharizadeh, S., Zimmermann, R., Shi, W., Rejaie, R., Ierardi, D., Li, T.-W.: Mitra: A Scalable Continuous Media Server, Multimedia Tools and Applications, Vol. 5, No. 1, July 1997, pp. 79-108

[Goel et al. 98] Goel, A., Steere, D., Pu, C., Walpole, J.: SWiFT: A Feedback Control and Dynamic Reconfiguration Toolkit, Technical Report CSE-98-009, Oregon Graduate Institute, Portland, OR, USA, June 1998

[Govindan et al. 91] Govindan, R, Anderson D. P, Scheduling and IPC Mechanisms for Continuous Media, Proc. of 13th ACM Symp. on Operating Systems Principles (SOSP'91), Pacific Grove, CA, USA, October 1991, pp. 68-80

[Goyal et al. 96a] Goyal, P., Guo, X., Vin, H.M.: A Hierarchical CPU Scheduler for Multimedia Operating Systems, Proc. of 2nd USENIX Symp. on Operating Systems Design and Implementation (OSDI'96), Seattle, WA, USA, October 1996, pp. 107-121

[Goyal et al. 96b] Goyal, P., Vin, H.M., Cheng, H.: Start-time Fair Queuing: A Scheduling Algorithm for Integrated Services Packet Switching Networks, Proc. of ACM SIGCOMM'96, San Francisco, CA, USA, August 1996, pp. 157-168

[Halvorsen et al. 98] Halvorsen, P., Goebel, V., Plagemann, T.: Q-L/MRP: A Buffer Management Mechanism for QoS Support in a Multimedia DBMS, Proc. of 1998 IEEE Int. Workshop on Multimedia Database Management Systems (IW-MMDBMS'98), Dayton, OH, USA, August 1998, pp. 162-171

[Hamilton et al. 93] Hamilton, G., Kougiouris, P.: The Spring Nucleus: A Microkernel for Objects, Proc. 1993 USENIX Summer Conf., Cincinnati, OH, USA, June 1993

[Hand 99] Hand, S.M.: Self-Paging in the Nemesis Operating System, Proc. of 3rd USENIX Symp. on Operating Systems Design and Implementation (OSDI'99), New Orleans, LA , USA, February 1999, pp. 73-86

[Härtig et al. 97] Härtig, H., Hohmuth, M., Liedtke, J., Schönberg, S., Wolter, J.: The Performance of µKernel-Based Systems, Proc. of 16th ACM Symp. on Operating System Principles (SOSP'97), October 1997, St. Malo, France, pp. 66-77

[Härtig et al. 98] Härtig, H., Baumgartl, R., Borriss, M., Hamann, C.-J., Hohmuth, M., Mehnert, F., Reuther, L., Schönberg, S., Wolter, J.: DROPS - OS Support for Distributed Multimedia Applications, Proc. of 8th ACM SIGOPS European Workshop, Sintra, Portugal, September 1998

[Haskin 93] Haskin, R.L.: The Shark Continuous-Media File Server, Proc. of 38th IEEE Int. Conf.: Technologies for the Information Superhighway (COMPCON'93), San Francisco, CA, USA, February 1993, pp. 12-15

[Haskin et al. 96] Haskin, R.L., Schmuck, F.B.: The Tiger Shark File System, Proc. of 41st IEEE Int. Conf.: Technologies for the Information Superhighway (COMPCON'96), Santa Clara, CA, USA, February 1996, pp. 226-231

[Hua et al. 97] Hua, K.A., Sheu, S.: Skyscraper Broadcasting: A New Broadcasting Scheme for Meteropolitan Video-on-Demand System, Proc. of ACM SIGCOMM'97, Cannes, France, September 1997, pp. 89-100

[Jacobson et al. 91] Jacobson, D.M., Wilkes, J.: Disk Scheduling Algorithms Based on Rotational Position, HP Laboratories Technical Report HPL-CSP-91-7, Palo Alto, CA, USA, February 1991

[Jeffay et al. 98] Jeffay, K., Smith, F.D., Moorthy, A., Anderson, A.: Proportional Share Scheduling of Operating System Services for Real-Time Applications, Proc. of 19th IEEE Real-Time System Symp. (RTSS'98), Madrid, Spain, December 1998, pp. 480-491

[Jones et al. 95] Jones, M.B., Leach, P.J., Draves, R.P., Barrera, J.S.: Modular Real-Time Resource Management in the Rialto Operating System, Proc. of 5th Workshop on Hot Topics in Operating Systems (HotOS-V), Orcas Island, WA, USA, May 1995, pp. 12-17

[Jones et al. 96] Jones, M.B., Barrera, J.S., Forin, A., Leach, P.J., Rosu, D., Rosu, M.-C.: An Overview of the Rialto Real-Time Architecture, Proc. of 7th ACM SIGOPS European Workshop, Connemara, Ireland, September 1996, pp. 249-256

[Jones et al. 97] Jones, M.B., Rosu, D., Rosu, M.-C: CPU Reservations and Time Constraints: Efficient, Predictable Scheduling of Independent Activities, Proc. of 16th ACM Symp. on Operating Systems Principles (SOSP'97), St. Malo, France, October 1997, pp. 198-211

[Kaashoek et al. 97] Kaaskoek, M.F., Engler, D.R., Ganger, G.R., Briceno, H.M., Hunt, R., Mazieres, D., Pinckney, T., Grimm, R., Jannotti, J., Mackenzie, K.: Application Performance and Flexibility on Exokernel Systems, Proc. of 16th Symp. on Operating Systems Principles (SOSP'97), St. Malo, France, October 1997, pp. 52-65

[Kamath et al. 95] Kamath, M., Ramamritham, K., Towsley, D.: Continuous Media Sharing in Multimedia Database Systems, Proc. of 4th Int. Conf. on Database Systems for Advanced Applications (DASFAA'95), Singapore, April 1995, pp. 79-86

[Kitamura et al. 95] Kitamura, H., Taniguchi, K., Sakamoto, H., Nishida T.: A New OS Architecture for High Performance Communication Over ATM Networks: Zero-Copy Architecture, Proc. of 5th Int. Workshop on Network and Operating Systems Support for Digital Audio and Video (NOSSDAV'95), Durham, NH, USA, April 1995, pp. 87-90

[Krishnan et al. 97] Krishnan, R., Venkatesh, D., Little, T.D.C.: A Failure and Overload Tolerance Mechanism for Continuous Media Servers, Proc. of 5th ACM Int. Multimedia Conf. (ACM MM'97), Seattle, WA, USA, November 1997, pp. 131-142

[Lakshman 97] Lakshman, K.: AQUA: An Adaptive Quality of Service Architecture for Distributed Multimedia Applications, PhD Thesis, Computer Science Departement, University of Kentucky, Lexington, KY, USA, 1997

[Lee et al. 97] Lee, W., Su, D., Wijesekera, D., Srivastava, J., Kenchammana-Hosekote, D.R., Foresti, M.: Experimental Evaluation of PFS Continuous Media File System, Proc. of 6th ACM Int. Conf. on Information and Knowledge Management (CIKM'97), Las Vegas, NV, USA, November 1997, pp. 246-253

[Leffler et al. 90] Leffler, S.J., McKusick, M.K., Karels, M.J., Quarterman, J.S.: The Design and Implementation of the 4.3BSD UNIX Operating System, Addison-Wesley Publishing Company, 1989

[Lei et al. 97] Lei, H., Duchamp, D.: An Analytical Approach to File Prefetching, Proc. of 1997 USENIX Annual Technical Conf., Anaheim, CA, USA, January 1997

[Leslie et al. 96] Leslie, I., McAuley, D., Black, R., Roscoe, T., Barham, P., Evers, D., Fairbairns, R., Hyden, E.: The Design and Implementation of an Operating System to Support Distributed Multimedia Applications, IEEE Journal on Selected Areas in Communications, Vol. 14, No. 7, September 1996, pp. 1280-1297

[Liedtke 95] Liedtke, J.: On Micro Kernel Construction, Proc. of 15th ACM Symp. on Operating Systems Principles (SOSP'95), Cooper Mountain, Colorado, USA, December 1995, pp. 237-250

[Liedtke 96] Liedtke, J.: Toward Real Microkernels, Communication of the ACM, Vol. 39, No. 9, September 1996, pp. 70-77

[Lin et al. 91] Lin, T.H., Tarng, W.: Scheduling Periodic and Aperiodic Tasks in Hard Real Time Computing Systems, Proc. of 1991 ACM Int. Conf. on Measurement and Modeling of Computer Systems (SIGMETRICS'91), San Diego, CA, USA, May 1991, pp. 31-38

[Liu et al. 73] Liu, C.L., Layland, J.W.: Scheduling Algorithms for Multiprogramming in a Hard Real Time Environment, Journal of the ACM, Vol. 20, No. 1, January 1973, pp. 46-61

[Lougher et al. 93] Lougher, P., Shepherd, D.: The Design of a Storage Server for Continuous Media, The Computer Journal, Vol. 36, No. 1, February 1993, pp. 32-42

[Martin et al. 96] Martin C., Narayanan, P.S., Özden, B., Rastogi, R., Silberschatz, A.: The Fellini Multimedia Storage Server, in: Chung, S.M. (Ed.): Multimedia Information and Storage Management, Kluwer Academic Publishers, 1996, pp. 117-146

[McKee et al. 98] McKee, S.A., Klenke, R.H., Wright, K.L., Wulf, W.A., Salinas, M.H., Aylor, J.H., Barson, A.P.: Smarter Memory: Improving Bandwidth for Streamed References, IEEE Computer, Vol. 31, No. 7, July 1998, pp. 54-63

[McKusick et al. 96] McKusick, M.K., Bostic, K., Karels, M.J., Quarterman, J.S.: The Design and Implementation of the 4.4 BSD Operating System, Addison Wesley, 1996

[Mercer et al. 94] Clifford, W., Mercer, J.Z., Ragunathan, R.: On Predictable Operating System Protocol Processing, Technical Report CMU-CS-94-165, School of Computer Science, Carnegie Mellon University, Pittsburgh, PA, May 1994

[Miller et al. 98a] Miller, F.W., Keleher, P., Tripathi, S.K.: General Data Streaming, Proc. of 19th IEEE Real-Time System Symp. (RTSS'98), Madrid, Spain, December 1998

[Miller et al. 98b] Miller, F.W., Tripathi, S.K.: An Integrated Input/Output System for Kernel Data Streaming, Proc. of SPIE/ACM Multimedia Computing and Networking (MMCN '98), San Jose, CA, USA, January 1998, pp. 57-68

[Molano et al. 97] Molano, A., Juvva, K., Rajkumar, R.: Real-Time Filesystems Guaranteeing Timing Constraints for Disk Accesses in RT-Mach, Proc. of 18th IEEE Real-Time Systems Symp. (RTSS'97), San Francisco, CA, USA, December 1997

[Mosberger et al. 96] Mosberger, D., Peterson, L.L.: Making Paths Explicit in the Scout Operating System, Proc. of 2nd USENIX Symp. on Operating Systems Design and Implementation (OSDI'96), Seattle, WA, USA, October 1996

[Moser et al. 95] Moser, F., Kraiss, A., Klas, W.: L/MRP: A Buffer Management Strategy for Interactive Continuous Data Flows in a Multimedia DBMS, Proc. of 21st IEEE Int. Conf. on Very Large Databases (VLDB'95), Zurich, Switzerland, 1995, pp. 275-286

[Nahrstedt et al. 95] Nahrstedt, K., Steinmetz, R.: Resource Management in Networked Multimedia Systems, IEEE Computer, Vol. 28, No. 5, May 1995, pp. 52-63

[Nahrstedt et al. 99] Nahrstedt, K., Chu, H., Narayan, S.: QoS-Aware Resource Management for Distributed Multimedia Applications, Journal on High-Speed Networking, Special Issue on Multimedia Networking, Vol. 7, No. 3/4, Spring 99, pp. 229-258

[Nakajima et al. 97] Nakajima, T., Tezuka, H.: Virtual Memory Management for Interactive Continuous Media Applications, Proc. of IEEE Int. Conf. on Multimedia Computing and Systems (ICMCS'97), Ottawa, Canada, June 1997

[Nerjes et al. 98] Nerjes, G., Rompogiannakis, Y., Muth, P., Paterakis, M., Triantafillou, P., Weikum, G.: Scheduling Strategies for Mixed Workloads in Multimedia Information Servers, Proc. of IEEE International Workshop on Research Issues in Data Engineering (RIDE'98), Orlando, FL, USA, February 1998, pp. 121-128

[Ng et al. 94] Ng, R.T., Yang, J.: Maximizing Buffer and Disk Utilization for News-On-Demand, Proc. of 20th IEEE Int.

Conf. on Very Large Databases (VLDB'94), Santiago, Chile, 1994, pp. 451-462

[Nieh et al. 93] Nieh, J., Hanko, J.G., Northcutt, J.D., Wall, G.A.: SVR4 UNIX Scheduler Unacceptable for Multimedia Applications, Proc. of 4th Int. Workshop on Network and Operating System Support for Digital Audio and Video (NOSSDAV'93), Lancaster, UK, November 1993

[Nieh et al. 97] Nieh, J., Lam, M.S.: The Design, Implementation and Evaluation of SMART: A Scheduler for Multimedia Applications, Proc. of 16th ACM Symp. on Operating System Principles (SOSP'97), St. Malo, France, October 1997, pp. 184-197

[Niranjan et al. 97] Niranjan, T.N., Chiueh, T., Schloss, G.A.: Implementation and Evaluation of a Multimedia File System, Proc. of IEEE Int. Conf. on Multimedia Computing and Systems (ICMCS'97), Ottawa, Canada, June 1997

[Nishikawa et al. 97] Nishikawa, J., Okabayashi, I., Mori, Y., Sasaki, S., Migita, M., Obayashi, Y., Furuya, S., Kaneko, K.: Design and Implementation of Video Server for Mixed-rate Streams, Proc. of 7th Int. Workshop on Network and Operating System Support for Digital Audio and Video (NOSSDAV'97), St. Louis, MO, USA, May 1997, pp. 3-11

[Noble et al. 97] Noble, B., Satyanarayanan, M., Narayanan, D., Tilton, J.E., Flinn, J., Walker, K.: Agile Application-Aware Adaptation for Mobility, Proc. of the 16th ACM Symp. on Operating System Principles (SOSP'97), St. Malo, France, October 1997, pp. 276-287

[Oparah 98] Oparah, D.: Adaptive Resource Management in a Multimedia Operating System, Proc. of 8th Int. Workshop on Network and Operating System Support for Digital Audio and Video (NOSSDAV'98), Cambridge, UK, July 1998,

[Özden et al. 96a] Özden, B., Rastogi, R., Silberschatz, A.: Disk Striping in Video Server Environments, Proc. of IEEE Int. Conf. on Multimedia Computing and Systems (ICMCS'96), Hiroshima, Japan, June 1996

[Özden et al. 96b] Özden, B., Rastogi, R., Silberschatz, A.: Buffer Replacement Algorithms for Multimedia Storage Systems, Proc. of IEEE Int. Conf. on Multimedia Computing and Systems (ICMCS'96), Hiroshima, Japan, June 1996

[Pai 97] Pai, V.S.: IO-Lite: A Copy-free UNIX I/O System, Master of Science Thesis, Rice University, Houston, TX, USA, January 1997

[Pai et al. 99] Pai, V.S., Druschel, P., Zwaenepoel, W.: IO-Lite: A Unified I/O Buffering and Caching System, Proc. of 3rd USENIX Symp. on Operating Systems Design and Implementation (OSDI'99), New Orleans, LA, USA, February 1999, pp. 15-28

[Parek et al. 93] Parek, A.K., Gallager, R.G.: A Generalized Processor Sharing Approach to Flow Control in Integrated Services Networks: The Single-Node Case, IEEE/ACM Transactions on Networking, Vol. 1, No. 3, June 1993, pp. 344-357

[Pasquale et al. 94] Pasquale, J., Anderson, E., Muller, P.K.: Container Shipping - Operating System Support for I/O-Intensive Applications, IEEE Computer, Vol. 27, No. 3, March 1994, pp. 84-93

[Patterson et al. 95] Patterson, R.H., Gibson, G.A., Ginting, E., Stodolsky, D., Zelenka, J.: Informed Prefetching and Caching, Proc. of 15th ACM Symp. on Operating System Principles (SOSP'95), Cooper Mountain, CO, USA, December 1995, pp. 79-95

[Peterson et al. 85] Peterson, J.L., Silberschatz, A.: Operating System Concepts, Addison-Wesley, 1985

[Philbin et al. 96] Philbin, J., Edler, J., Anshus, O.J., Douglas, C.C., Li, K.: Thread Scheduling for Cache Locality, Proc. of 7th Int. Conf. on Architectural Support for Programming Languages and Operating Systems (ASPLOS-VII), Cambridge, MA, USA, October 1996, pp. 60-71

[Plagemann et al. 97] Plagemann, T., Goebel, V.: INSTANCE: The Intermediate Storage Node Concept, Proc. of 3rd Asian Computing Science Conf. (ASIAN'97), Kathmandu, Nepal, December 1997, pp. 151-165

[Plagemann et al. 99] Plagemann, T., Goebel, V.: Analysis of Quality-of-Service in a Wide-Area Interactive Distance Learning System, in: Wolf, L. (Ed.): Special Issue on European Activities in Interactive Distributed Multimedia Systems and Telecommunication Services, Telecommunication Systems, Vol. 11, No. 1-2, 1999, pp. 139-160

[Rajkumar et al. 98] Rajkumar, R., Juvva, K., Molano, A., Oikawa, S.: Resource Kernels: A Resource-Centric Approach to Real-Time Systems, Proc. of SPIE/ACM Conf. on Multimedia Computing and Networking (MMCN'98), San Jose, CA, USA, January 1998

[Ramakrishnan et al. 93] Ramakrishnan, K.K., Vaitzblit, L., Gray, C., Vahalia, U., Ting, D., Tzelnic, P., Glaser, S., Duso, W.: Operating System Support for a Video-on-Demand File Service, Proc. of 4th Int. Workshop on Network and Operating System Support for Digital Audio and Video (NOSSDAV'93), Lancaster, U.K., 1993, pp. 216-227

[Rangan et al. 91] Rangan, P.V., Vin, H.: Designing File Systems for Digital Video and Audio, Proc. of the 13th Symp. on Operating Systems Principles (SOSP'91), Pacific Grove, CA, USA, October 1991, pp. 81-94

[Rashid et al. 81] Rashid, R., Robertson, G.: Accent: A Communication-Oriented Network Operating System Kernel, Proc. of 8th ACM Symp. on Operating System Principles (SOSP'81), New York, NY, USA, 1981, pp. 64-75

[Reddy 95] Reddy, A.L.N.: Scheduling in Multimedia Systems, in: Design and Applications of Multimedia Systems, Kluwer Academic Publishers, August 1995

[Reddy et al. 93] Reddy, A.L.N., Wyllie, J.: Disk Scheduling in a Multimedia I/O System, Proc. of 1st ACM Multimedia Conf. (ACM MM'93), Anaheim, CA, USA, August 1993, pp. 225-233

[Reddy et al. 94] Reddy, A.L.N., Wyllie, J.C.: I/O Issues in a Multimedia System, IEEE Computer, Vol. 27, No. 3, March 1994, pp. 69-74

[Rompogiannakis et al. 98] Rompogiannakis, Y., Nerjes, G., Muth, P., Paterakis, M., Triantafillou, P., Weikum, G.: Disk Scheduling for Mixed-Media Workloads in a Multimedia Server, Proc. of 6th ACM Multimedia Conf. (ACM MM'98), Bristol, UK, September 1998, pp. 297-302

[Rosenblum 95] Rosenblum, M.: The design and Implementation of a Log-Structured File System, Kluwer Academic Publishers, 1995

[Rotem et al. 95] Rotem, D., Zhao, J.L.: Buffer Management for Video Database Systems, Proc. of 11th Int. Conf. on Data Engineering (ICDE'95), Tapei, Taiwan, March 1995, pp. 439-448

[Roth et al. 98] Roth, A., Moshovos, A., Sohi, G.S.: Dependence Based Prefetching for Linked Data Structures, Proc. of 8th Int. Conf. on Architectural Support for Programming Languages and Operating Systems (ASPLOS-VIII), San Jose, CA, USA, October 1998, pp. 115-126

[Santos et al. 98] Santos, J.R., Muntz, R.: Performance Analysis of the RIO Multimedia Storage System with Heterogeneous Disk Configurations, Proc. of 6th ACM Multimedia Conf. (ACM MM'98), Bristol, UK, September 1998, pp. 303-308

[Schulzrinne 96] Schulzrinne, H.: Operating System Issues for Continuous Media, ACM/Springer Multimedia Systems, Vol. 4, No. 5, October 1996, pp. 269-280

[Seltzer et al. 90] Seltzer, M., Chen, P., Ousterhout, J.: Disk Scheduling Revisited, Proc. of 1990 USENIX Technical Conf., Washington, D.C., USA, January 1990, pp. 313-323

[Seltzer et al. 93] Seltzer, M., Bostic, K., McKusick, M.K., Staelin, C.: An Implementation of a Log-Structured File System for UNIX, Proc. of 1993 USENIX Winter Conf., San Diego, CA, USA, January 1993

[Shenoy et al. 99] Shenoy, P.J., Goyal, P., Vin, H.M.: Architectural Considerations for Next Generation File Systems, to be published in Proc. of 7th ACM Multimedia Conf. (ACM MM'99), Orlando, FL, USA, October 1999

[Shenoy et al. 98a] Shenoy, P.J., Goyal, P., Rao, S.S., Vin, H.M.: Symphony: An Integrated Multimedia File System, Proc. of ACM/SPIE Multimedia Computing and Networking 1998 (MMCN'98), San Jose, CA, USA, January 1998, pp. 124-138

[Shenoy et al. 98b] Shenoy, P.J., Vin, H.M.: Cello: A Disk Scheduling Framework for Next Generation Operating Systems, Proc. of 1998 ACM Int. Conf. on Measurement and Modeling of Computer Systems (SIGMETRICS'98), Madison, WI, USA, June 1998

[Spatscheck et al. 99] Spatscheck, O., Peterson, L.L.: Defending Against Denial of Service Attacks in Scout, Proc. of 3rd USENIX Symp. on Operating Systems Design and Implementation (OSDI'99), New Orleans, LA, USA, February 1999, pp. 59-72

[Steere et al. 99] Steere, D.C., Goel, A., Gruenenberg, J., McNamee, D., Pu, C., Walpole, J.: A Feedback-driven Proportion Allocator for Real-Rate Scheduling, Proc. of 3rd USENIX Symp. on Operating Systems Design and Implementation (OSDI'99), New Orleans, LA, USA, February 1999, pp. 145-158

[Steinmetz 95] Steinmetz, R.: Analyzing the Multimedia Operating System, IEEE Multimedia, Vol. 2, No. 1, Spring 1995, pp. 68-84

[Stoica et al. 97] Stoica, I., Abdel-Wahab, W., Jeffay, K.: On the Duality between Resource Reservation and Proportional Share Resource Allocation, Multimedia Computing and Networking 1997, SPIE Proc. Series, Volume 3020, San Jose, CA, USA, February 1997, pp. 207-214

[Tanenbaum 92] Tanenbaum, A.S.: Modern Operating Systems, Prentice Hall, 1992

[Tezuka et al. 96] Tezuka, H., Nakajima, T.: Simple Continuous Media Storage Server on Real-Time Mach, Proc. of 1996 USENIX Annual Technical Conf., San Diego, CA, USA, January 1996

[Thadani et al. 95] Thadani, M.N., Khalidi, Y.A.: An Efficient Zero-Copy I/O Framework for UNIX, Technical Report SMLI TR-95-39, Sun Microsystems Laboratories Inc., May 1995

[Tzou et al. 91] Tzou, S.-Y., Anderson, D.P.: The Performance of Message-passing using Restricted Virtual Memory Remapping, Software - Practice and Experience, Vol. 21, No. 3, March 1991, pp. 251-267

[Verghese et al. 98] Vergehese, B., Gupta, A., Rosenblum, M.: Performance Isolation: Sharing and Isolation in Shared Memory Multiprocessors, Proceeding of 8th Int. Conf. on Architectural Support for Programming Languages and Operating Systems (ASPLOS-VIII), San Jose, CA, USA, October 1998

[Vernick et al. 96] Vernick, M., Venkatramani, C., Chiueh, T.: Adventures in Building the Stony Brook Video Server, Proc. of 4th ACM Multimedia Conf. (ACM MM'96), Boston, MA, USA, November 1996, pp. 287-295

[Vin et al. 93] Vin, H.M., Rangan, V.: Admission Control Algorithm for Multimedia On-Demand Servers, Proc. of 4th Int. Workshop on Network and Operating System Support for Digital Audio and Video (NOSSDAV'93), La Jolla, CA, USA, 1993, pp. 56-68

[Viswanathan et al. 96] Viswanathan, S., Imielinski, T.: Metropolitan area Video-on-Demand Service Using Pyramid Broadcasting, Multimedia Systems, Vol 4., No. 4, 1996, pp. 197-208

[Waldspurger 95] Waldspurger, C.A.: Lottery and Stride Scheduling: Flexible Proportional-Share Resource Management, PhD thesis, Department of Electrical Engineering and Computer Science, Massachusetts Institute of Technology, Cambridge, MA, USA, September 1995

[Wang et al. 99a] Wang, R.Y., Anderson, T.E., Patterson, D.A.: Virtual Log Based File Systems for a Programmable Disk, Proc. of 3rd USENIX Symp. on Operating Systems Design and Implementation (OSDI'99), New Orleans, LA, USA, February 1999, pp. 29-43

[Wang et al. 99b] Wang, C., Goebel, V., Plagemann, T.: Techniques to Increase Disk Access Locality in the Minorca Multimedia File System (Short Paper), to be published in Proc. of 7th ACM Multimedia Conf. (ACM MM'99), Orlando, FL, USA, October 1999

[Wijayaratne et al. 99] Wijayaratne, R., Reddy, A.L.N.: Integrated QoS Management for Disk I/O, Proc. of IEEE Int. Conf. on Multimedia Computing and Systems (ICMCS'99), Florence, Italy, June 1999

[Wolf et al. 96] Wolf, L.C., Burke, W., Vogt, C.: Evaluation of a CPU Scheduling Mechanism for Multimedia Systems, Software - Practice and Experience, Vol. 26, No. 4, April 1996, pp. 375-398

[Yau et al. 96] Yau, D.K.Y., Lam, S.S.: Operating System Techniques for Distributed Multimedia, Technical Report TR-95-36 (revised), Department of Computer Sciences, University of Texas at Austin, Austin, TX, USA, January 1996

[Yu et al. 93] Yu, P.S., Chen, M.S., Kandlur, D.D.: Grouped Sweeping Scheduling for DASD-Based Multimedia Storage Management, ACM Multimedia Systems, Vol. 1, No. 3, 1993, pp. 99-109

[Zhang 91] Zhang, L.: Virtual Clock: A New Traffic Control Algorithm for Packet Switching Networks, ACM Transactions on Computer Systems, Vol. 9, No. 3, May 1991, pp. 101-124

[Zhang et al. 95] Zhang, A., Gollapudi, S.: QoS Management in Educational Digital Library Environments, Technical Report CS-TR-95-53, State University of New York at Buffalo, New York, NY, USA, 1995