CMPSCI 754 Multimedia Systems **Spring 2001**

Lecture 12: March 15, 2001

Lecturer: Prashant Shenoy *Scribe: Jiang Lan*

12.1 Designing Integrated Multimedia File Systems

12.1.1 Why do we need integrated file systems?

An integrated file system is one that can efficiently manage the storage and retrieval of multiple data types. Different data types have significantly different characteristics, most existing file systems are only optimized for a single data type thus are not suitable for this heterogeneous environment.

Characteristics of conventional file systems:

- Oblivious of semantics of data, i.e., data are simply treated as byte streams.
- Optimized for small files.
- Block sizes are optimized for text files.
- Provide only a best-effort service to user requests.

Characteristics of continuous media servers:

- Exploit the periodic and sequential nature of continuous media accesses to improve server throughput.
- Optimized for audio and video files.
- Use resource reservation algorithms to guarantee real-time performance.

Conclusion: need an integrated file system that efficiently supports multiple data types.

12.1.2 How to achieve integrated file system?

12.1.2.1 Logically integrated file systems

In this scheme separate file systems are used for each data type, and they are glued together by an integration layer that provides a uniform way of accessing files stored on different file servers. Figure 12.1 illustrates this architecture.

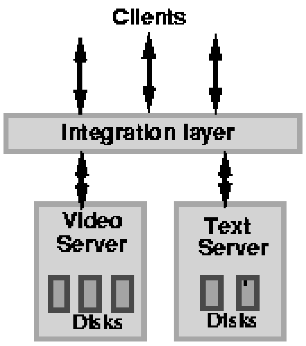

Figure 12.1: Logically integrated file systems

Advantages and disadvantages of this approach:

- Easy to implement since techniques for each component file server are well known.
- No interference between user requests, e.g., servicing best-effort text requests does not violate deadlines of real-time continuous media requests.
- Static allocation of resources (disks). This in one aspect leads to no interference of requests, but in the other aspect, it causes trouble if the file system needs to be repartitioned (e.g., when the work load changes dramatically).
- Does not support relationship between files. For example, some video file is associated with some text file, this relationship can not be well supported because the two files are in different partitions.
- Worse synchronization. This is similar to the previous problem. I.e., the support is poor when a text file and a continuous media file need synchronization to work together.
- Duplication of services. Some of the file system components (such as directory service) have to be duplicated among the component file servers.
- The storage space and bandwidth requirements for a component file server can be very different from each other, allocation of the disks is governed by the maximum of these two values. This can lead to under-utilization of either resource.

12.1.2.2 Physically integrated file systems

In this approach, the file system consists of a single file server that stores all data types. Figure 12.2 illustrates this architecture.

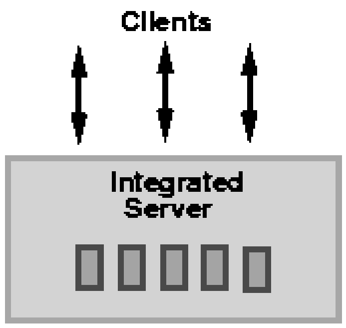

Figure 12.2: Physically integrated file systems

Advantages of physical integration:

- Co-locating of objects with different bandwidth-space requirements. By co-locating a set of files with large storage space but small bandwidth requirements with another set of files with small storage space but large bandwidth requirements, this architecture yields better resource utilization.
- On-demand resource allocation makes it easy to accommodate dynamic changes in access patterns.
- All the resources are available to service each request, this improves performance.

Disadvantages of physical integration:

- Increased complexity due to the need for different service requirements.
- Inteference among the requests for different data types.

12.1.2.3 Which approach should we take?

When the workload of a system is known beforehand and rarely change dynamically, we can choose the logical integration. One thing need to note here is that the system needs to accomodate for its peak load. Since the sum of individual components' peak loads is greater than the peak load of a physically integrated system, the latter will save significant resources, also gives better performance (due to better parallelism).

When the workload of a system is bursty, the physical integration is preferred due to better performance and higer utilization of resources.

12.1.3 Integrated File System Design Fundamentals

File system manages three resources:

1. Storage space: placement of files, failure recovery, etc.
2. Disk bandwidth: disk scheduling algorithms. Note that although disk scheduling is normally implemented in the disk driver (Figure 12.3), it's still considered under the file system's control since it is dictated by the file system's needs.

3. Buffer cache (in memory): caching techniques

Physical integration involves integration in each of the above components.

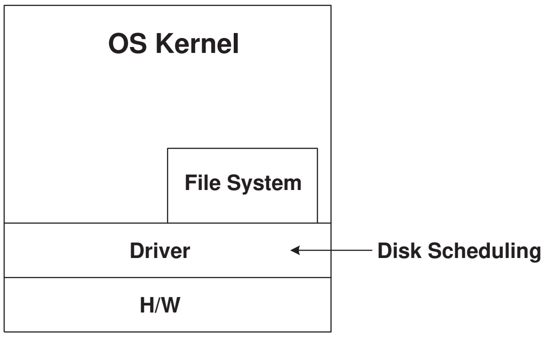

Figure 12.3: System architecture

12.1.3.1 Managing Disk Bandwidth

Different applications have different service requirements

- Interactive applications require low average response times
- Throughput-intensive applications (e.g., FTP) require high average throughput
- Real-time applications use deadline based algorithms and the deadlines must be guaranteed

Existing disk scheduling algorithms are optimized for single class of service. For exmple, text-based file servers normally use best-effort scheduling algorithms (e.g., SCAN, SATF, etc.) that does not take into account real time requirements, real-time requests may miss their deadlines under these algorithms. On the other hand, real-time scheduling algorithms (e.g., SSEDO, SCAN-EDF, FD-SCAN, etc.) are usually deadline based, when requests with deadlines and requests without deadlines are mixed together, a sophisticated algorithm need to be invented to guarantee service requirements for both. Simply assign those non-real-time requests' deadlines to infinity won't solve the problem since non-real-time jobs may starve, or their average response times get longer and untolerable to the users.

An integrated file system need to export multiple classes of services (real-time, interactive, etc.). Furthermore the disk schedulers should align the services provided with application needs. Neither of the above mentioned algorithm classes meet these requirements, thus new algorithms need to be invented to meet all these service requirements.

12.1.3.2 Managing Storage Space

Placement policies are governed both by data characteristics and application requirements. Use a single block size for all the applications is not appropriate. For example, large block size is suitable for continuous media files, but for small files, this approach wastes space and causes stripping problems; on the other hand, small block size is suitable for conventional text-based applications, but it is not suitable for large media files since throughput is degraded. In summary, a single placement policy will not be suitable for all data types, hence the integrated system need to support multiple datatype specific placement policies, and mechanisms need to be emploied to enable their coexistence. The cost of doing these is increased complexity.

12.1.3.3 Managing Buffers

Cache policies are governed by access characteristics. For example: continuous media applications usually have sequential access, thus LRU is not appropriate here, instead algorithms that utilize this sequential access pattern (e.g., interval caching) will give much better performance. In contrast, conventional applications have locality of reference, thus LRU is a good choice here. In summary, a single policy will not be suitable for all classes, the integrated system needs a policy that supports multiple data type specific caching policies, and mechanisms need to be emploied to enable them to share the buffer cache.

12.1.4 Architectural Requirements

As the complexity and diversity of applications keep increaing, the next generation file systems need to be extensible and customizable such that they will not only support today's but the future's applications. Extensible is in the sense that new functionalities could be easily added, customizable is in the sense that users can tailure the file system to his own needs.

12.1.5 Requirement Summary

Different data types have significantly different characteristics and service requirements, thus they adopt different policies to meet these requirements. The integrated file system need to support these data type specific policies in all of its sub-components, i.e., the storage management, the bandwidth management, and the buffer management. In addition, a file system must be extensible and customizable.