CMPSCI 754 Multimedia Systems **Spring 2001**

Lecture 14: March 29

Lecturer: Prashant Shenoy *Scribe: Yan Mu*

14.1 Storage Manager

Enable the coexistence of multiple stripe unit sizes, degrees of striping, and striping policies.

- Construct a stripe unit using a sequence of continuous base blocks, so as to get multiple stripe unit sizes.
- Provide control over placement of blocks using location hints, e.g. (disk#, block#)
- Fault tolerance: separate off-line rebuild from on-line reconstruction. Off-line rebuild uses parity information. On-line reconstruction by default uses parity-based reconstruction.

14.2 Buffer Manager

Support multiple data type specific caching policies.

- Each partition is managed by different policies.
- Allow each caching policy to manage its cache partition independently.
- Caching policies are implemented by the data type specific layer.

Data Type Specific Layer consists of a set of modules that implement data type specific policies, including video module (retrieve placement), audio module and text module.

- **Video Module:** For retrieval policy, server-push requests are serviced as periodic real-time requests, e.g. video streaming; client-pull requests are serviced as aperiodic real-time requests, e.g. multimedia authoring and editing. In placement policy, it uses location hints so as to minimize seek and rotational latency overheads. For example, store two substreams (base and enhancement) closely on the disk. Meta data management allows byte-level indexing as well as frame-level indexing. For streaming media, it makes more sense to have frame location information.
- **Text Module:** Similiar to conventional UNIX file systems. Its placement policy is designed fro small disk array, using round-robin striping of fixed-size blocks. Its retrieval policy supports client-pull, best-effort requests. Its meta data management uses byte map.

14.3 I/O Subsystem

If an application needs to interact with I/O devices, control and data information must be moved between the OS kernel and user space. See the following chart:

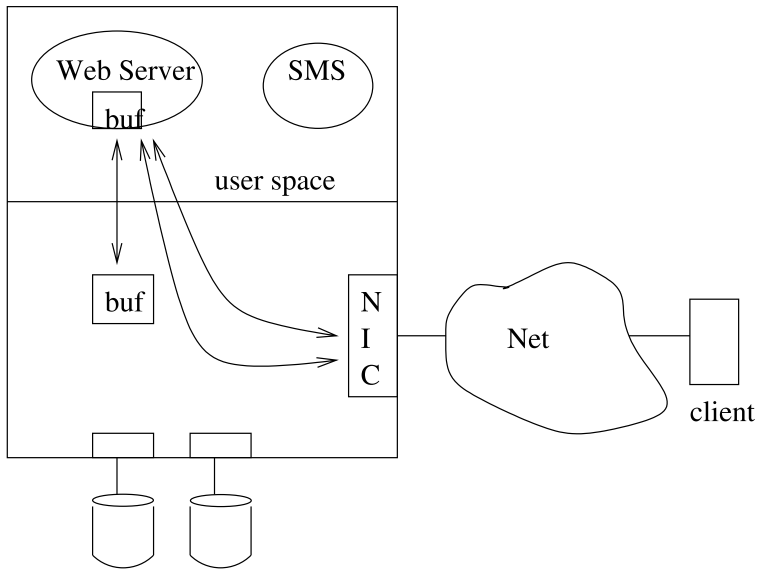

Figure 14.1: How traditional I/O subsystem works

One of the technology trends is that network and processor speeds are increasing at rapid pace. But memory speeds are not keeping pace which makes it a usual bottleneck in improving system performance. For example, effective DRAM copy rate (copy a big chunk of data) is around 400Mb/s, while CPUs and network adapters can operate at Giga b/s speed.

To deal with slower memory speeds, the conventional approach is to improve cache performance – exploit the spatial and temporal locality inherent in program. But I/O data buffers generated by OS or applications don't have sufficient locality, so cache is not effective.

For example, Windows NT provides a system call like *read(fd,sd)* which reads data from disk and send to socket directly without copying data to user space and sending back to kernel.

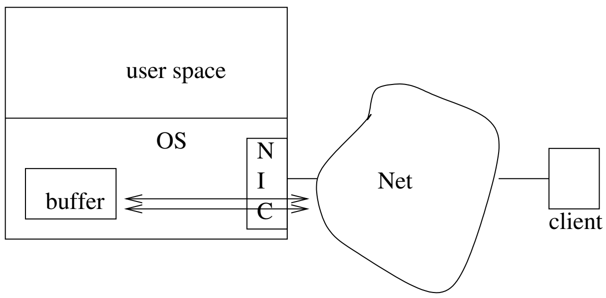

Figure 14.2: A approach dealing with slower memory speeds

14.4 Network I/O Fundamentals

There are two basic techniques:

- Direct Memory Access (DMA):

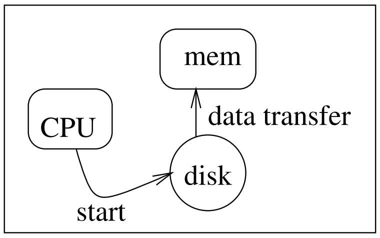

Figure 14.3: A DMA scenario

CPU lets the disk transfer a chunk of data to the memory and CPU does not bother to control the data transfer process.

- Programmed I/O (PIO)

14.5 Cross Domain Transfer

Multiplexing necessitates transfer of data across protection domains, e.g. kernel and user space. But the problem is the data copying aggravates memory bottleneck.

The remedies are:

- Virtual memory page re-mapping: move rather than copy. See the following chart (the chunck of data is assigned to application. It doesn't belong to OS):

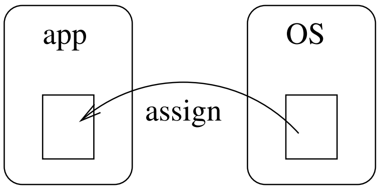

Figure 14.4: Virtual-memory page re-mapping

- Emulated copy: copy only if and when necessary(e.g. someone wants to modify the data. so the data belongs to both.)
- Shared memory: physical page mapped into two or more virtual address spaces.