CS 414 – Multimedia Systems Design

Lecture 35 – Media Server (Part 4)

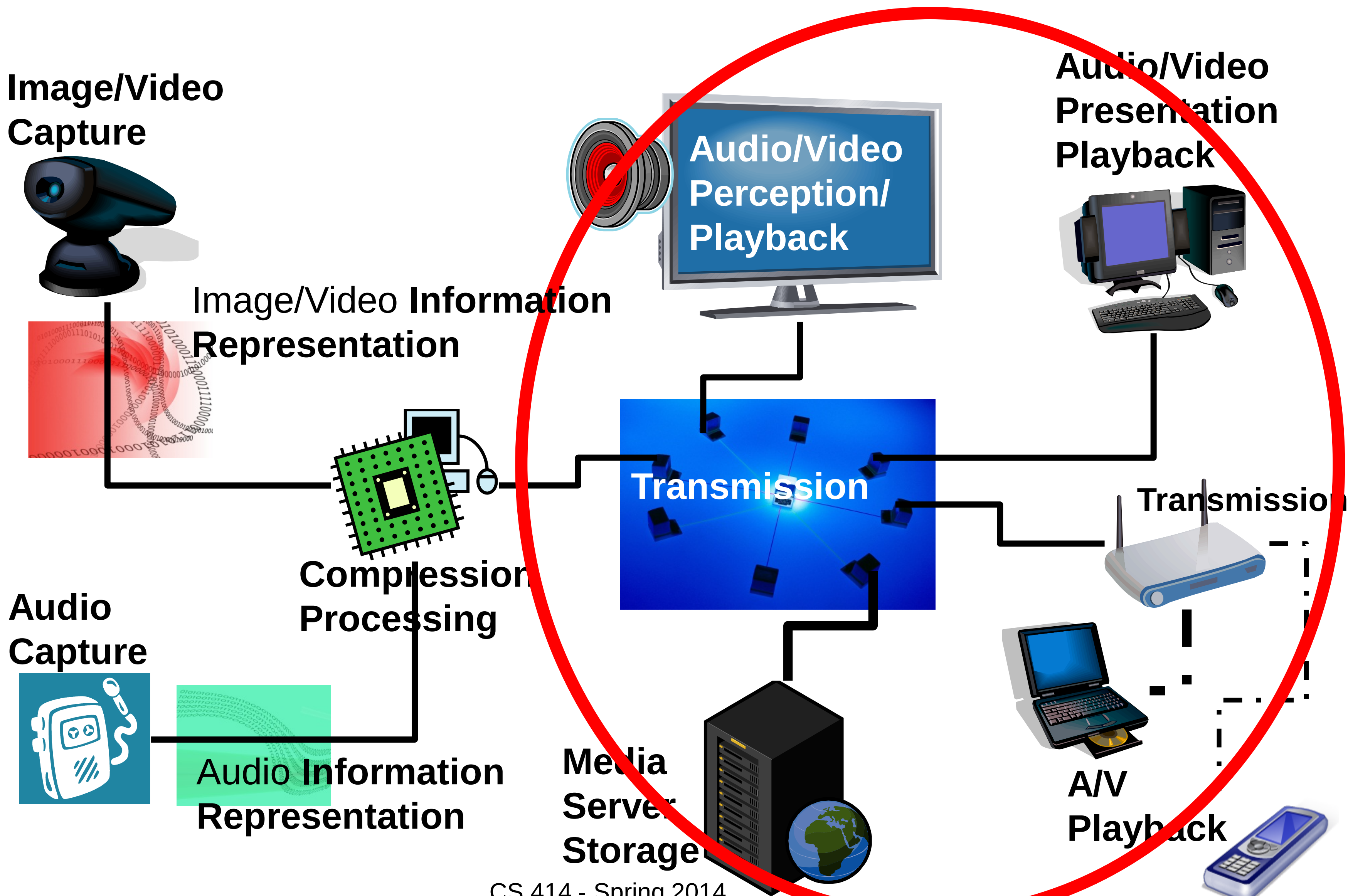
Klara Nahrstedt

Spring 2014

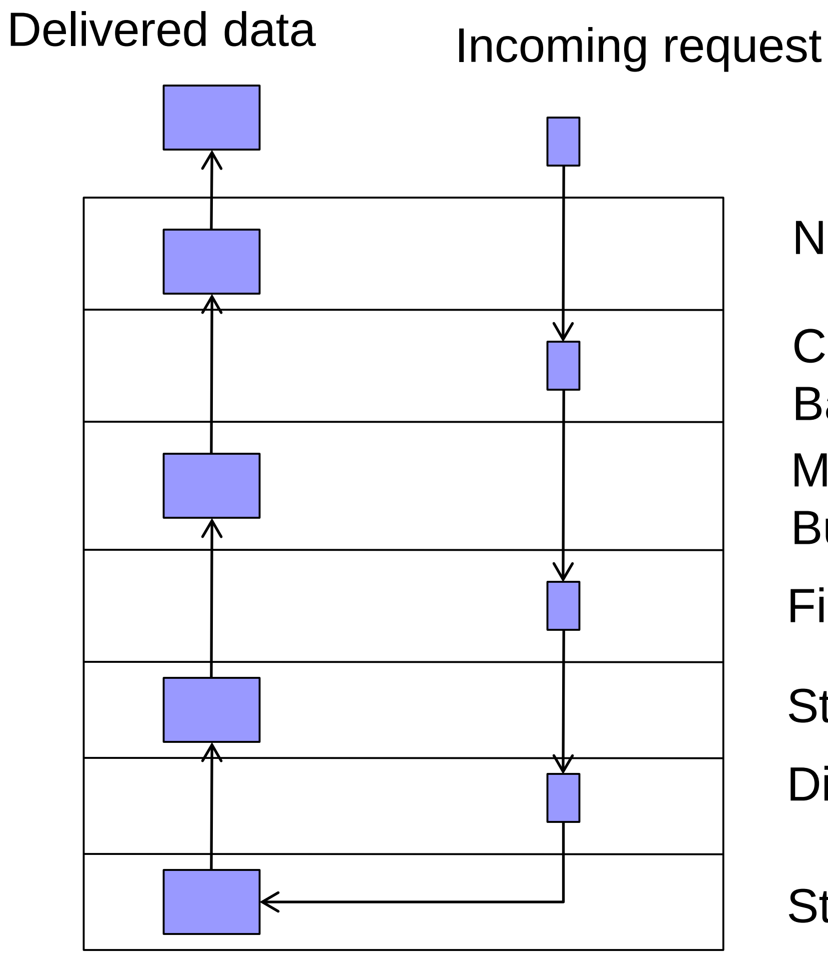
Administrative

- MP3 going on

Covered Aspects of Multimedia



Media Server Architecture



Network Attachment (RTP/RTCP, ….)

Content Distribution (Caching, Patching, Batching)

Memory Management (MaxBuf, MinBuf Policy Buffering)

File System

Storage management

Disk controller

Storage device

Outline

- Multimedia File Server/System Organization
- Example of Early Media Server – **Medusa**
- Example of Multimedia File System – **Symphony**
- Example of Industrial Multimedia File System – **Google File System**

Constant and Real-time Retrieval of MM Data

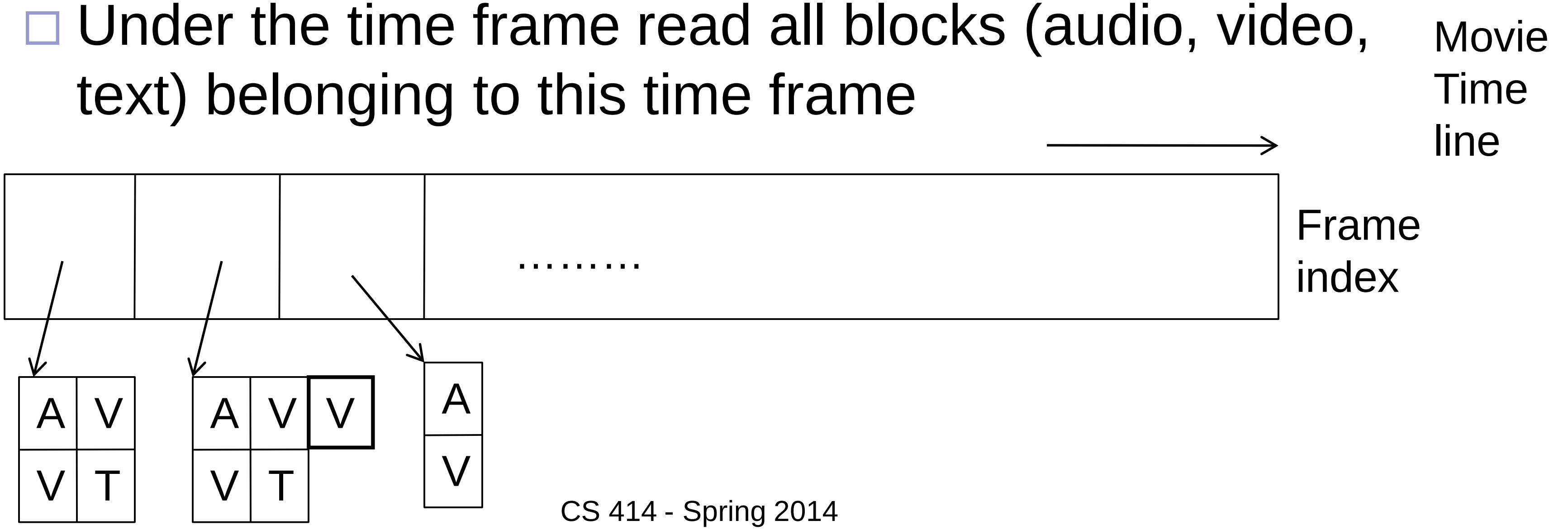
- Retrieve index in real-time
- Retrieve block information from FAT
- Retrieve data from disk in real-time
- Real-time playback
  - Implement linked list
- Random seek (Fast Forward, Rewind)
  - Implement indexing
- MM File Maps
  - include metadata about MM objects: creator of video, sync info

Fast Forward and Rewind (Implementation)

- Play back media at higher rate
  - Not practical solution
- Continue playback at normal rate, but skip frames
  - Define skip steps, e.g. skip every 3rd, or 5th frame
  - Be careful about interdependencies within MPEG frames
- Approaches for FF:
  - Create a separate and highly compressed file
  - Categorize each frame as relevant or irrelevant
  - Intelligent arrangement of blocks for FF

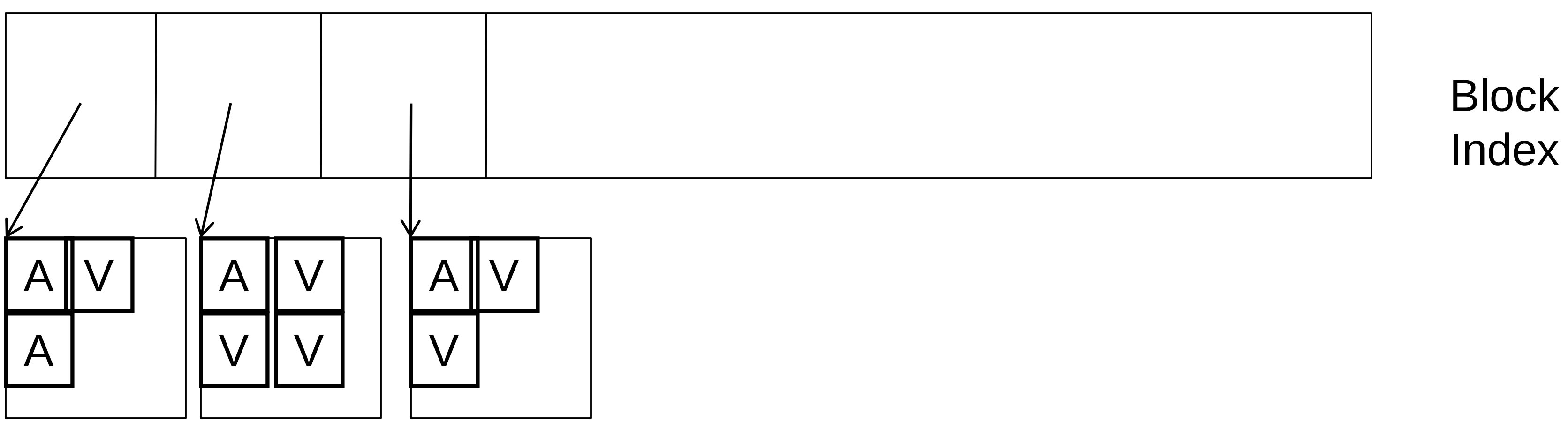
Block Size Issues in File Organization

- Small Block Sizes
  - Use smaller block sizes, smaller than average frame size
- Organization Strategy: Constant Time Length
- Need Metadata structure, called Frame Index
  - Frame means a time frame within a movie
  - Under the time frame read all blocks (audio, video, text) belonging to this time frame



Block Size Issues

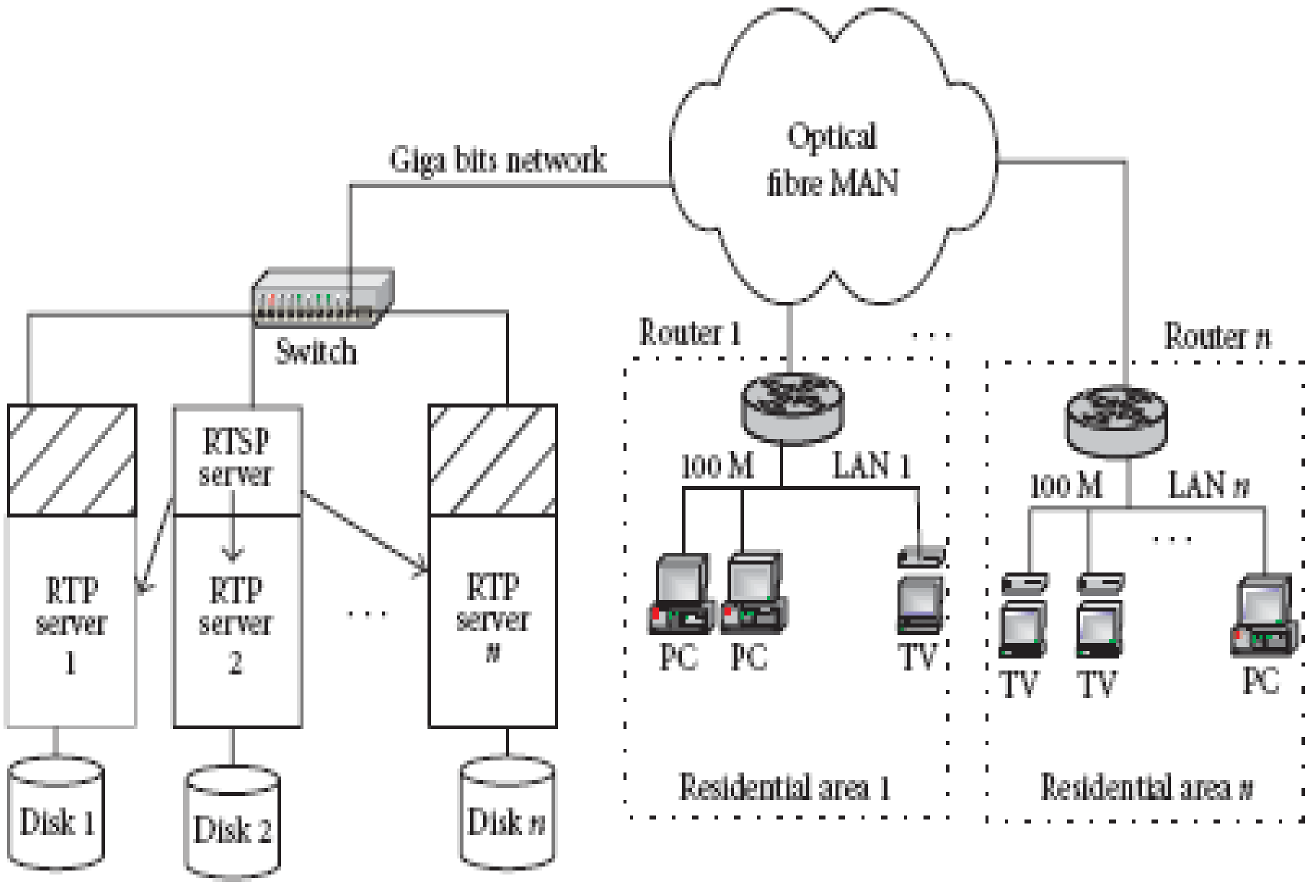
- Large Block Size
  - Use large blocks (e.g., 256 KB) which include multiple audio/video/text frames
- Organization Strategy: Constant Data Length
- Need Metadata structure, called Block Index
  - Each block contains multiple movie frames



Tradeoffs

- Frame index : needs large RAM usage while movie is playing, however little disk wastage
- Block index (if frames are not split across blocks): need low RAM usage, but major disk wastage – internal disk fragmentation
- Block index(if frames are split across blocks): need low Ram usage, no disk wastage, extra seek times

Example of Media Server Architecture

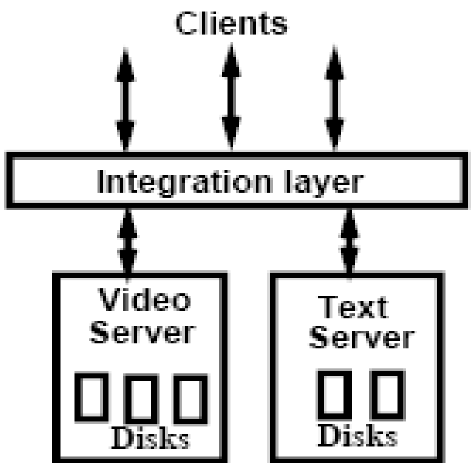


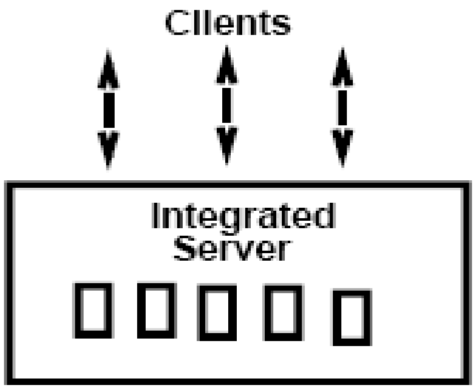
Source: Medusa (Parallel Video Servers), Hai Jin, 2004

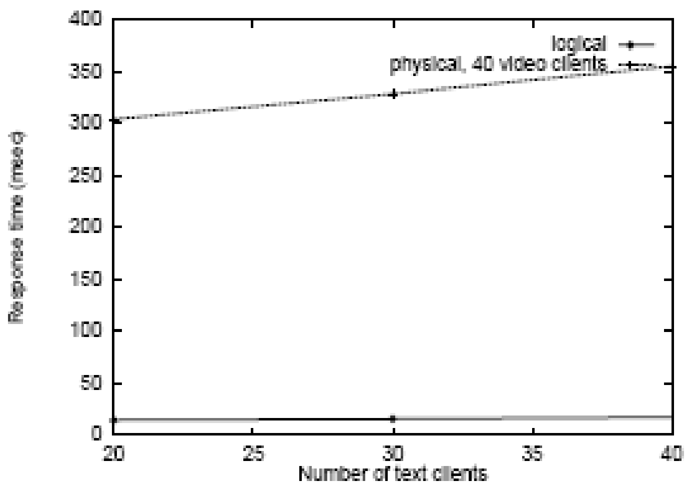
Example Multimedia File System (**Symphony**)

- *Source: P. Shenoy et al, "Symphony: An Integrated Multimedia File System", SPIE/ACM MMCN 1998*
- System out of UT Austin
- **Symphony's Goals:**
  - Support real-time and non-real time request
  - Support multiple block sizes and control over their placement
  - Support variety of fault-tolerance techniques
  - Provide two level metadata structure that all type-specific information can be supported

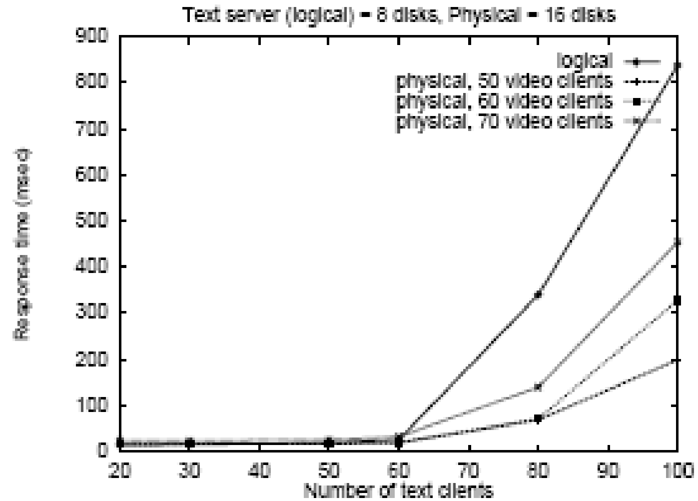
Design Decisions





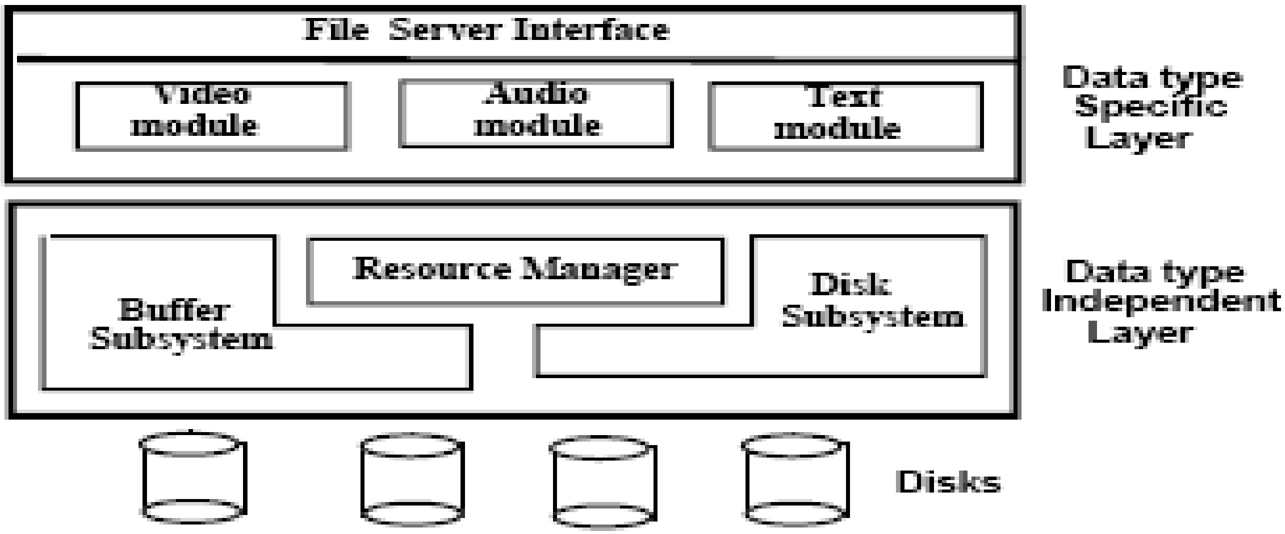


(a) SCAN



(b) Symphony disk scheduling algorithm

Two Level Symphony Architecture

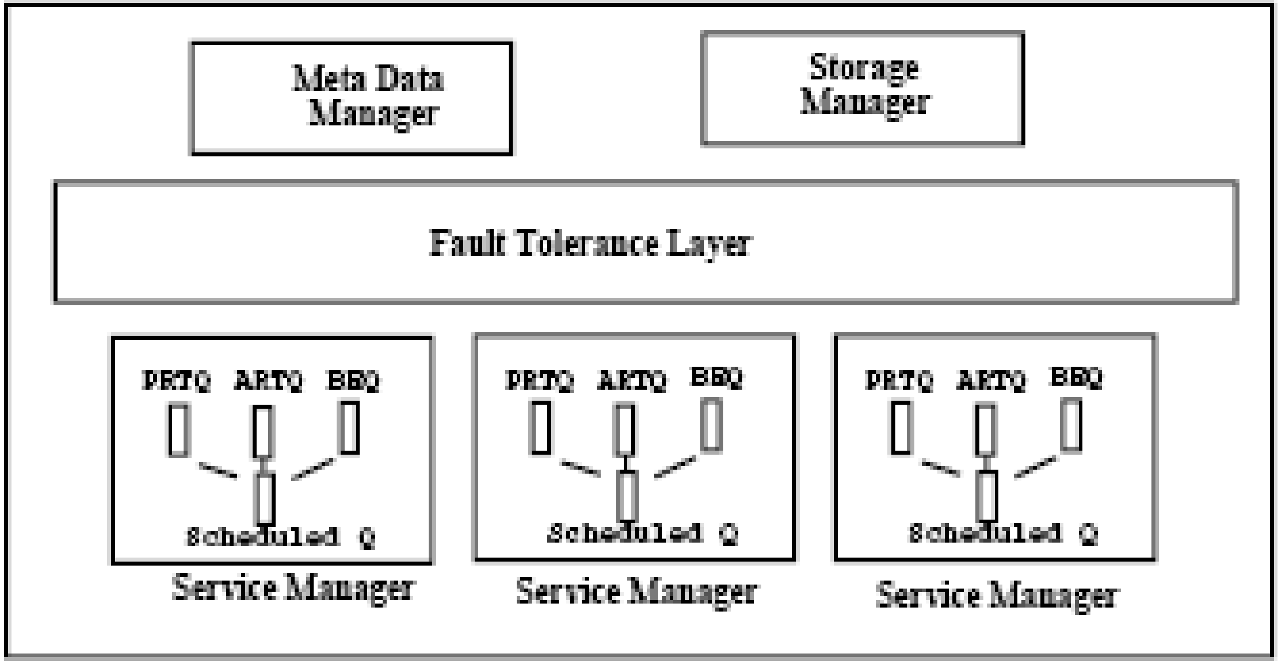


Resource Manager:

• Disk Schedule System (called Cello) that uses modified **SCAN-EDF** for RT Requests and **C-SCAN** for non-RT requests as long as deadlines are not violated

• Admission Control and Resource Reservation for scheduling

Disk Subsystem Architecture



Service Manager : supports mechanisms for efficient scheduling of best-effort, aperiodic real-time and periodic real-time requests

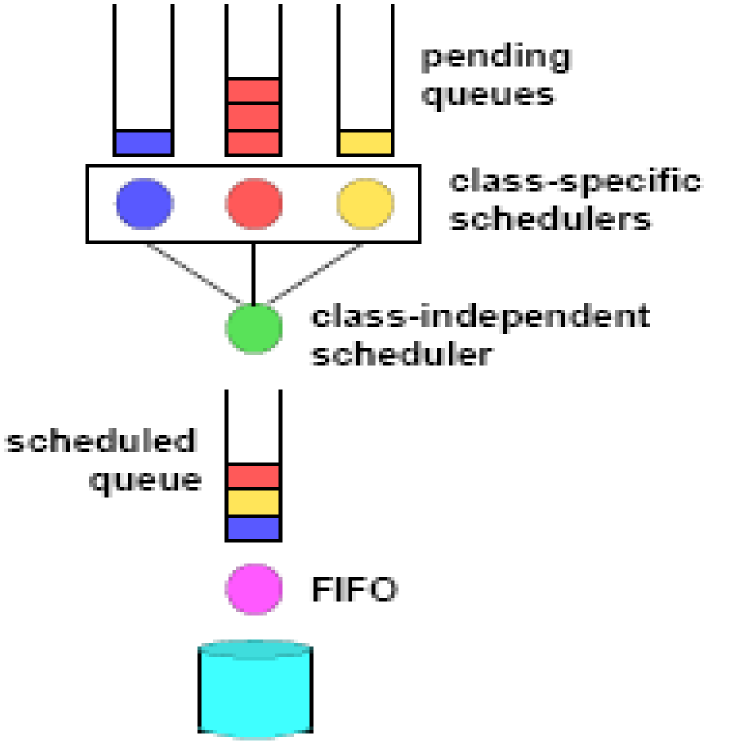
Storage Manager: supports mechanisms for allocation and de-allocation of blocks Of different sizes and controlling data placement on the disk

Fault Tolerance layer: enables multiple data type specific failure recovery techniques

Metadata Manager: enables data types specific structure to be assigned to files

Cello Disk Scheduling Framework

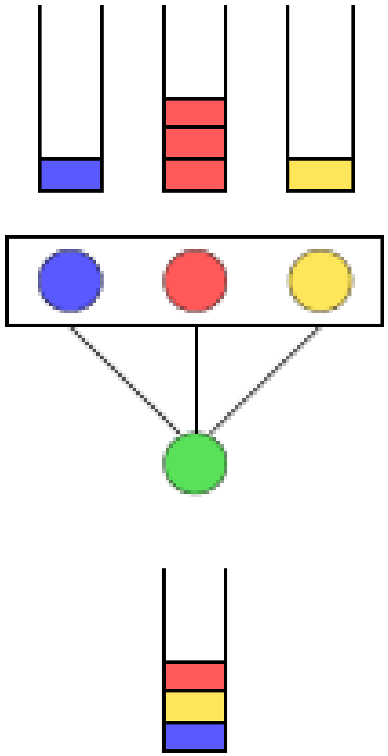
- Class-independent scheduler
  - Coarse grain bandwidth allocation to classes
  - Determines *when* and *how many* requests to insert
- Class-specific schedulers
  - Fine grain interleaving of requests
  - Create a schedule that meets request needs
  - Determine *where* to insert requests



Source: Prashant Shenoy, 2001

Class-Independent Scheduler

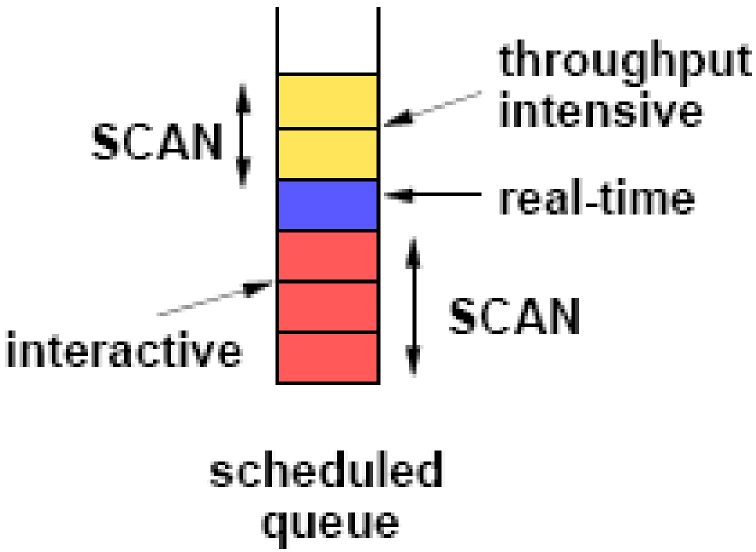
- Assign weights to each class
  - Allocate bandwidth in proportion to its weight
  - Time allocation versus byte allocation
- Algorithm:
  - Invoke a class specific scheduler for a request
  - Insert request at specified position if
    - class has sufficient unused allocation
    - total used allocation $\leq$ interval length
  - Update used allocation
  - Reallocate unused allocation to classes with pending requests



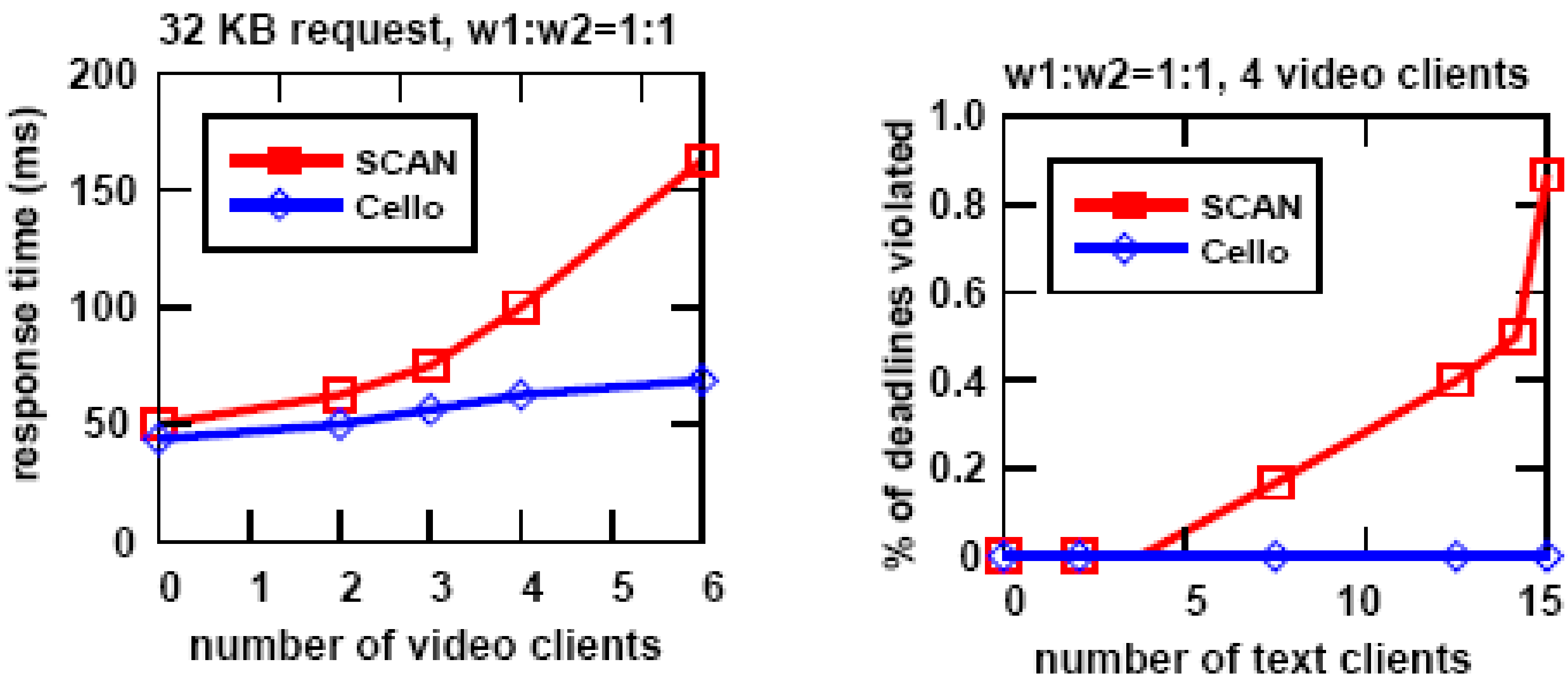
Source: Prashant Shenoy, 2001

Class-Specific Schedulers

- Determine insert position based on
  - Requirements of requests (e.g., deadlines)
  - State of the scheduled queue
- Interactive best-effort
  - Insert using *slack-stealing* [Lehoczky92]
- Throughput-intensive best-effort
  - Insert at tail in SCAN order
- Real-time
  - Insert in SCAN-EDF order

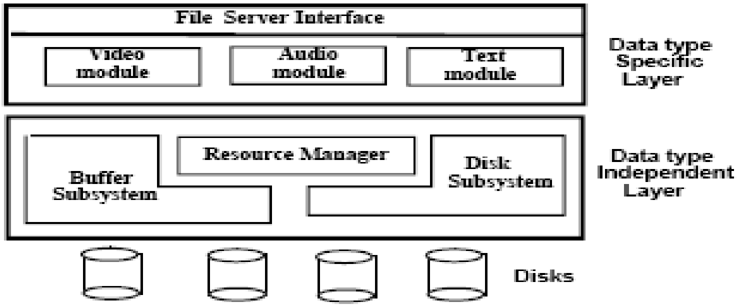


Validation: Symphony's scheduling system (Cello)



Source: Shenoy Prashant, 2001

Buffer Subsystem

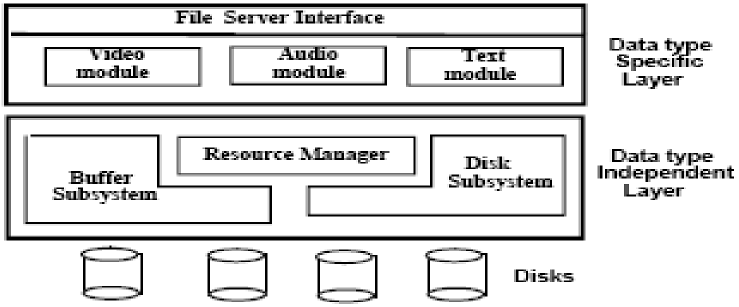


- Enable multiple data types specific caching policies to coexist
- Partition cache among various data types and allow each caching policy to independently manage its partition
- Maintain two buffer pools:
  - a pool of de-allocated buffers
  - pool of cached buffers.
    - Cache pool is further partitioned among various caching policies
    - Examples of caching policies for each cache buffer: LRU, MRU.

Buffer Subsystem (Protocol)

- Receive buffer allocation request
- Check if the requested block is cached.
  - If yes, it returns the requested block
  - If cache miss, allocate buffer from the pool of de-allocated buffers and insert this buffer into the appropriate cache partition
- Determine (Caching policy that manages individual cache) position in the buffer cache
  - If pool of de-allocated buffers falls below low watermark, buffers are evicted from cache and returned to de-allocated pool
  - Use TTR (Time-To- Reaccess) values to determine victims
    - TTR – estimate of next time at which the buffer is likely to be accessed

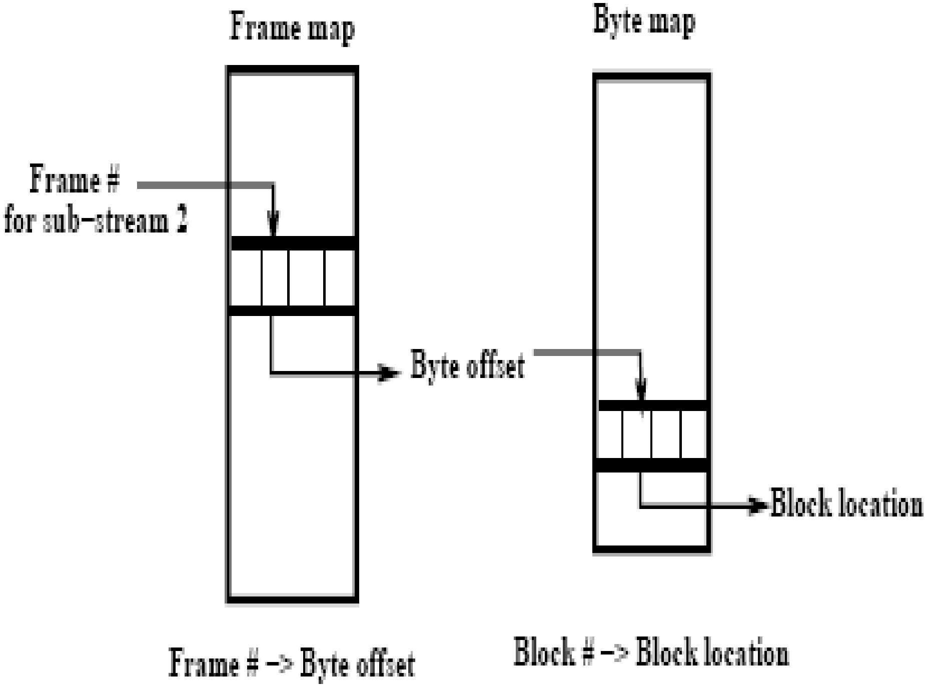
Video Module



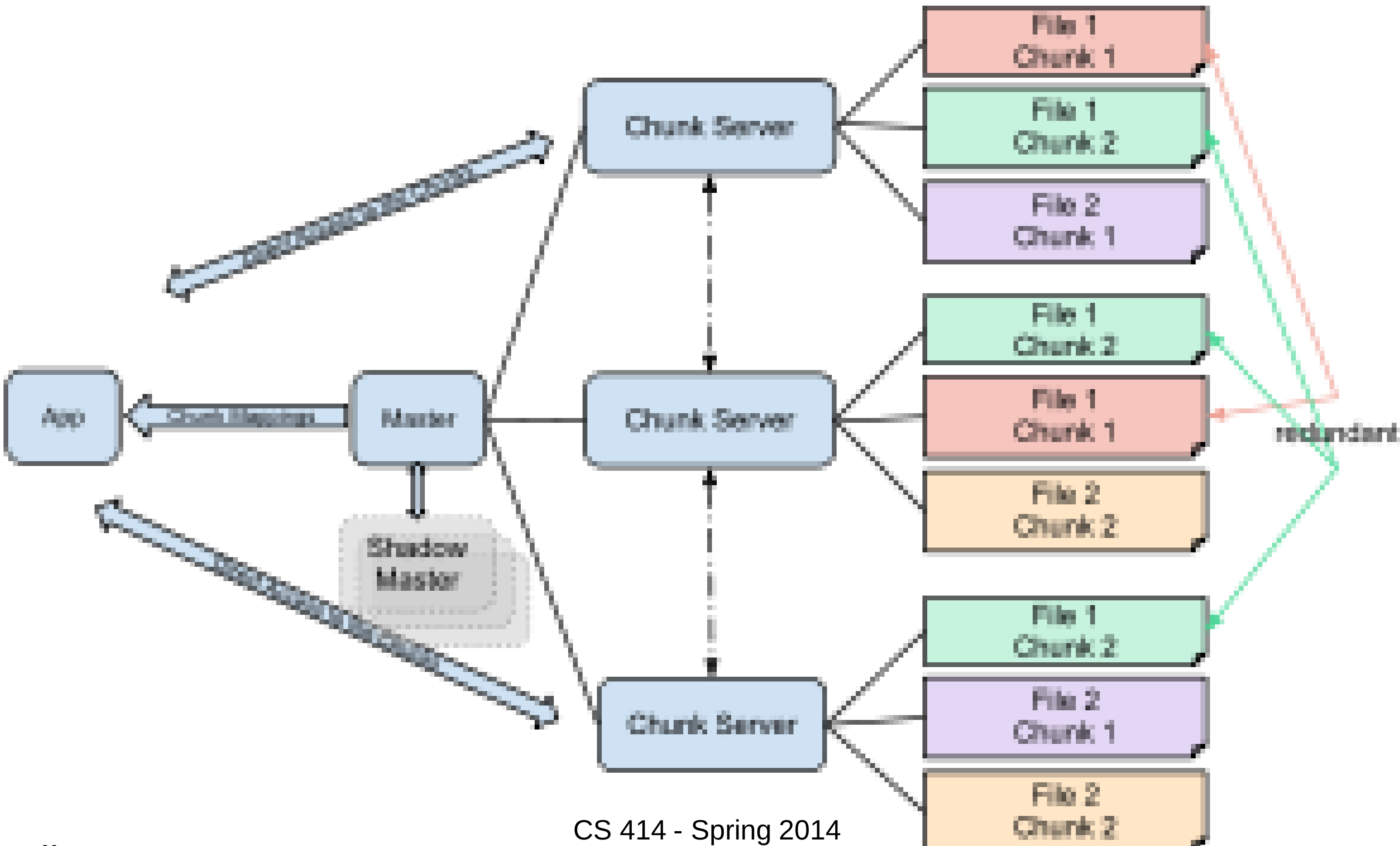
- Implements policies for placement, retrieval, metadata management and caching of video data
- Placement of video files on disk arrays is governed by two parameters: block size and striping policy.
  - supports both fixed size blocks (fixed number of bytes) and variable size blocks (fixed number of frames)
  - uses location hints so as to minimize seek and rotational latency overheads
- Retrieval Policy:
  - supports periodic RT requests (server push mode) and aperiodic RT requests (client pull mode)

Video Module (Metadata Management)

- To allow efficient random access at byte level and frame level, video module maintains two-level index structure
  - First level of index , referred to as frame map, maps frame offset to byte offset
  - Second level, referred to as byte map, maps byte offset to disk block locations



Google File System (Big Table)



Wikipedia:GFS

Google File System

- Files divided into fixed size big chunks of 64 Mbytes
- Chunk servers
- Files are usually appended to or read
- Run on cheap commodity computers
- High failure rate
- High data throughputs and cost of latency
- Two types of nodes
  - Master node and large number of chunk servers.
- Designed for system-to-system interaction

The Google File System: Sanjay Ghemawat, Howard Gobioff, and Shun-Tak Leung, Google, SOSP '03

Big Table

- High-performance data storage system
- Built on top of
  - Google File System
  - Chubby Lock Service
  - SSTable (log-structured storage)
- Supports systems such as
  - MapReduce
  - YouTube
  - Google Earth
  - Google Maps
  - Gmail

Spanner

- Successor to Big Table
- Globally distributed relational database management system (RDBMS)
- Google F1 – built on top of Spanner (replaces MySQL)

Corbett, James C; Dean, Jeffrey; Epstein, Michael; Fikes, Andrew; Frost, Christopher; Furman, JJ; Ghemawat, Sanjay; Gubarev, Andrey; Heiser, Christopher; Hochschild, Peter; Hsieh, Wilson; Kanthak, Sebastian; Kogan, Eugene; Li, Hongyi; Lloyd, Alexander; Melnik, Sergey; Mwaura, David; Nagle, David; Quinlan, Sean; Rao, Rajesh; Rolig, Lindsay; Saito, Yasushi; Szymaniak, Michal; Taylor, Christopher; Wang, Ruth; Woodford, Dale, "Spanner: Google's Globally-Distributed Database", *Proceedings of OSDI 2012* (Google), retrieved 18 September 2012.

Conclusion

- The data placement, scheduling, block size decisions, caching, concurrent clients support, buffering, are very important for any media server design and implementation.
- Huge explosion in media storage-cloud storage
  - Similar software – Apache Cassandra, Hadoop, Hypertable, Apache Accumulo, Apache Hbase, …
- Next Lecture – we discuss **P2P Streaming**

Symphony Caching Policy

- Interval-based caching for video module
- LRU caching for text module