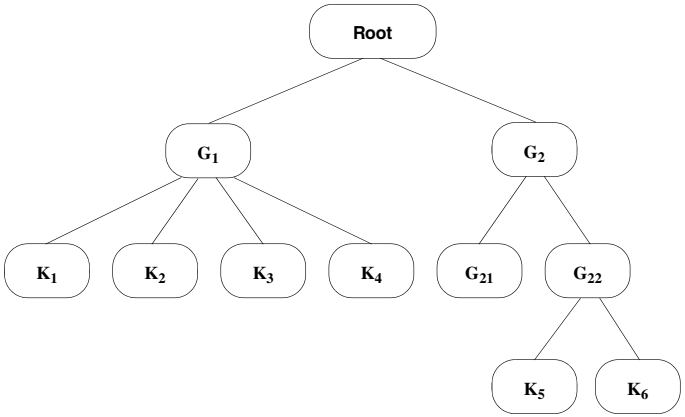
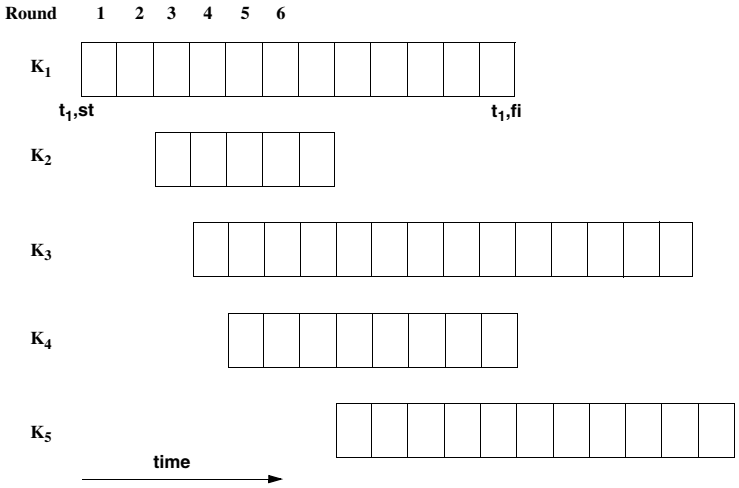
Hierarchical Scheduling



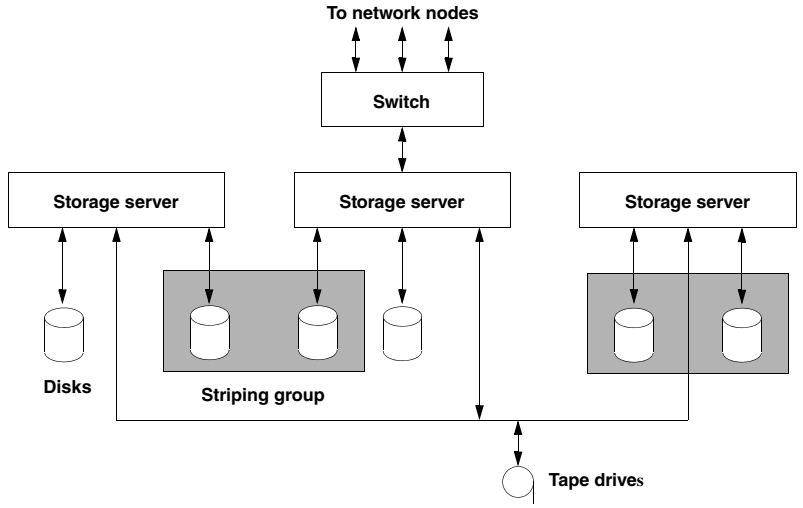
Hierarchical Scheduling

- an ideal policy would be the perfect fair-share policy that services the various groups round-robin using an infinitely small quantum of CPU-time (unrealizable in practice)
- deviation of this measure can be used for comparison
- achievable: approximations of fair-share policy
  - weighted fair queuing (WFQ)
    disadvantages: computational complexity and information on the execution times of tasks
  - start-time fair queuing

Weighted Fair Queuing



The Storage Subsystem



Block Placement

seek overhead is one of the major overheads incurring during disk block retrieval; block placement policies attempt to reduce this overhead:

- traditional block placement policies—use of information on file access patterns:
  - contiguous allocation
  - organ-pipe placement
- constrained-placement policies

Block Placement/Contiguous Allocation

- all disk blocks for a file are placed contiguously on disk; eliminates intrafile seeks; no interfile seeks nor seeks due to random access
- very suitable for read-only media
- under read/write workload policy has to deal with fragmentation of storage device space
- compaction of media files on disks
  - time consuming; file may be unavailable
  - large overhead
- space allocation in sizes of multiples of the block size for I/O operations

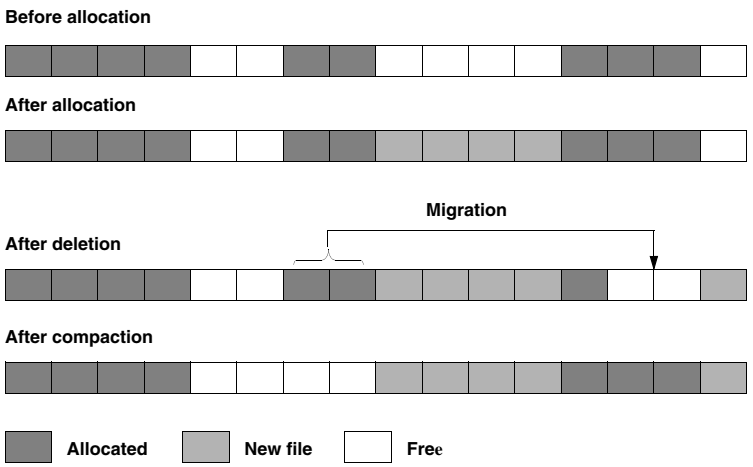
Everest Contiguous Allocation

- Everest policy—adaptation of contiguous allocation policy attempting to maintain contiguous allocation while minimizing the compaction overhead
- blocks are allocated to files only in contiguous segments of size ϖ^i blocks, where ϖ is a tunable parameter and i an integer.
- to allocate a file of L_f blocks, the policy represents L_f as a base ϖ number, $d_k d_{k-1}...$, and allocates d_i corresponding segments of length ϖ^i
- example: file length 5 blocks
  ϖ =2, allocation 1 segment of 4 blocks and 1 segm. of 1 block;
  ϖ =3, allocation 1 segment of 3 blocks and 2 segm. of 1 block;

Everest Contiguous Allocation

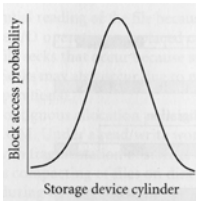
- free blocks are organized into free segments that are maintained in free segment lists (one list for each possible segment size ϖ^i
- the policy maintains a maximum of ϖ -1 segments in any list
- if file deletion results in more than ϖ -1 free segments of size ϖ^i, the policy migrates allocated segments until there are at least ϖ contiguous free segments of size ϖ^i
- contiguous segments are then collapsed into a free segment of size ϖ^{i+1}

Everest Contiguous Allocation



Organ-Pipe Placement

- attempts to minimize interfile seek overhead
- studies show that optimal placement of blocks follows an “organ-pipe” arrangement with the most frequently referenced blocks at the center of the disk
- less frequently referenced blocks are placed alternately around the hottest blocks



Constrained-Placement Policies

- policies discussed are adaptations of conventional block placement policies
- policies taking advantage of the sequential access of media streams to bound seek overhead:
  - REBECA—region-based block allocation method; attempts to trade off the seek overhead against start-up latency for new streams
  - strand-based allocation

REBECA

- storage device is divided into a fixed number of contiguous regions
- accesses to various blocks within a single region are served by a single scan step
- increasing the number of regions decreases the seek overhead while increasing initial latency
- successive blocks of a media object are allocated to different regions in the order they are visited

REBECA

Region number

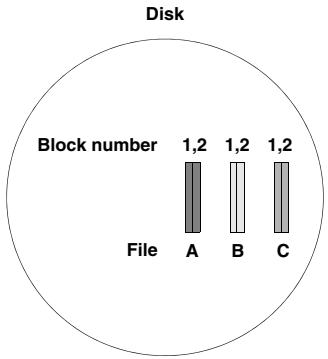
0	1	2	3	4	5
block 0	block 1	block 2	block 3	block 4	block 5
block 11	block 10	block 9	block 8	block 7	block 6
block 12	block 13	block 14	block 15		

Center of disk Disk periphery

Retrieval

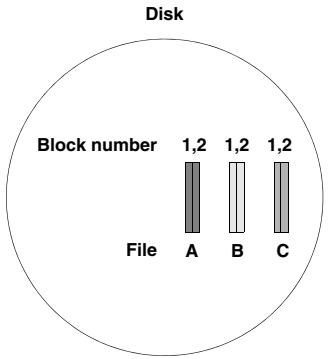
- data needed by applications have to be retrieved on time with no interruptions in delivery
- disk scheduling policies achieve this by scheduling requests for blocks such that streams retrieve their blocks before needed
- two categories
  - round-based policies—I/O requests are scheduled only during periodic scheduling rounds; assume playback rate is constant
  - fixed block size policies

Round-Robin and Scan Policies



Round-robin: A1, C1, B1, A2, C2, B2

round-robin policy



Scan: A1, B1, C1, C2, B2, A2

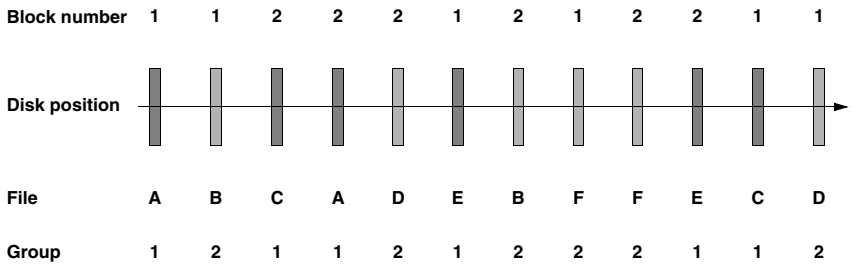
Outward Inward

scan policy

Group Sweeping Scheduling (GSS)

- combines both scan and round-robin, resulting in a policy with low seek overhead (scan) and with buffer requirements of one round
- policy:
  1. partition the V active streams into M_{gr} groups
  2. service the M_{gr} groups in round-robin order
  3. service the streams within each group using scan policy

Group Sweeping Scheduling



Schedule: {A1, E1, C1}, {D1, F1, B1}, {C2, A2, E2}, {F2, B2, D2}

File Placement

operational issues

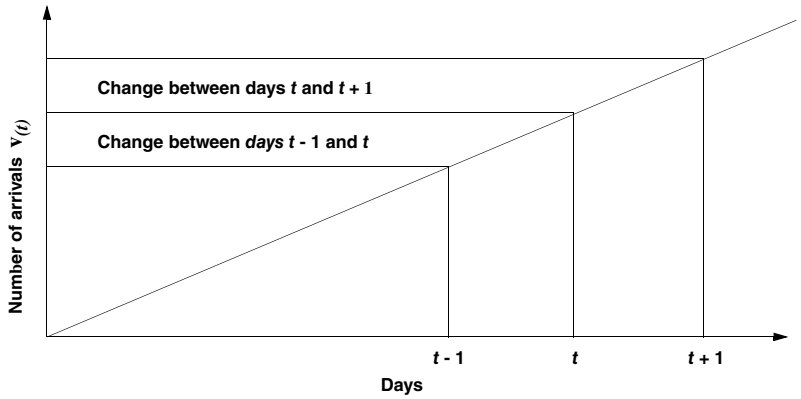
- efficient usage of storage devices
- placement based on anticipated workload
- expected workload may change in time (periodic update; on-line reconfiguration)
- all placement of media objects be carried out in coordination with the placement policy
- placement policy be integrated with policies for managing the storage hierarchy

File Placement Policies

- popularity-based assignment—used to place media objects on a set of identical disks
- policy attempts to minimize the average number of customers rejected due to storage device overload
- policy uses access probabilities to each file i on day t (popularities)
- assumption that access probabilities are fixed over 24 hours
- estimation of probabilities by linear interpolation and daily reconfiguration

File Placement Policies

popularity-based assignment



$$\Lambda_{(t+1)} = \Lambda_{(t)} + (\Lambda_{(t)} - \Lambda_{(t-1)})$$

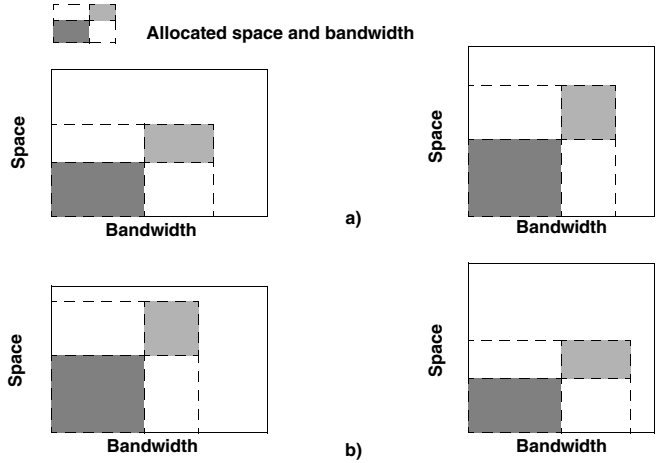
File Placement Policies

BSR (bandwidth-to-space) policy

- attempts to mix hot and cold as well as large and small media objects on heterogeneous storage devices (striping groups)
- policy characterizes each device by its bandwidth-to-space ratio
- similarly, each media object is characterized by the expected bandwidth requirement on that replica and the required storage space
- policy dynamically creates and deletes replicas

File Placement Policies

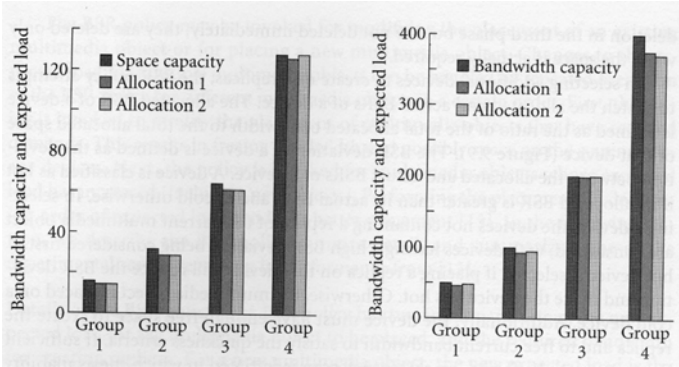
BSR policy



BSR Policy Simulation

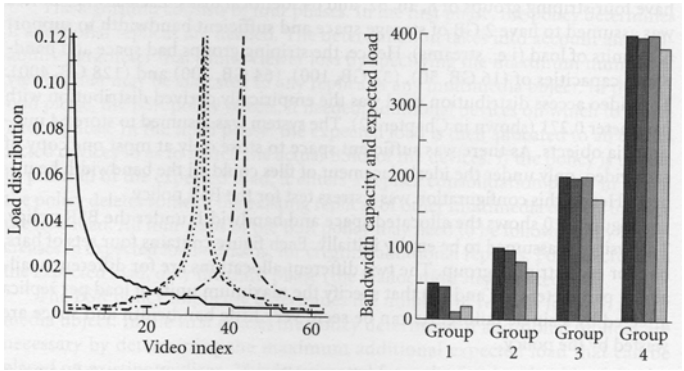
- 4 striping groups of 8, 16, 32 and 64 identical disks; each device has 2 GB and supports 6.25 units of load;
- striping groups have space and bandwidth capacities of (16, 50), (32, 100), (64, 200) and (128, 400);
- 64 media objects stored; system is initially empty
- 2 allocations for different availability parameters (30 and 50) that specify the maximum units of load per replica affected by a single failure

BSR Policy



device allocation after initial placement showing allocated space and bandwidth

BSR Policy



impact of load change on access distribution and bandwidth allocation

Strand-Based Allocation

definitions

- strand—a sequence of continuously recorded audio samples or video frames
- media block—a sequence of consecutive sectors storing media units; size of a media block of strand s_i is denoted by M_i
- media gap—the separation between successive blocks of a strand; size of a media gap of s_i is denoted by G_i
- each media block M_i and gap G_i spans a number of sectors; (M_i, G_i) is referred to as storage pattern of strand s_i

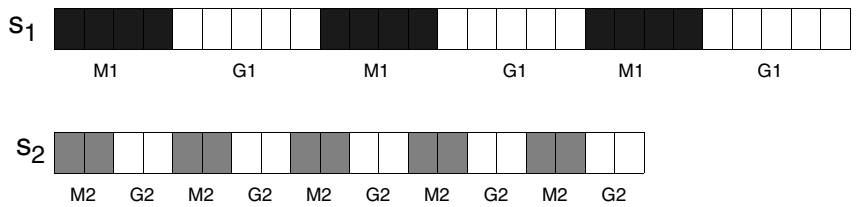
Storage Patterns

- continuity in retrieval can be guaranteed if each media block is available at the display device at or before the scheduled playback time
- random location vs. contiguous location; continuous playback may not be possible vs. copying overhead; therefore
  - block size bounded from below
  - gap size bounded from above
- continuity requirement

$$\frac{Mi + Gi}{rdt} \leq Tdis$$

Storage Pattern

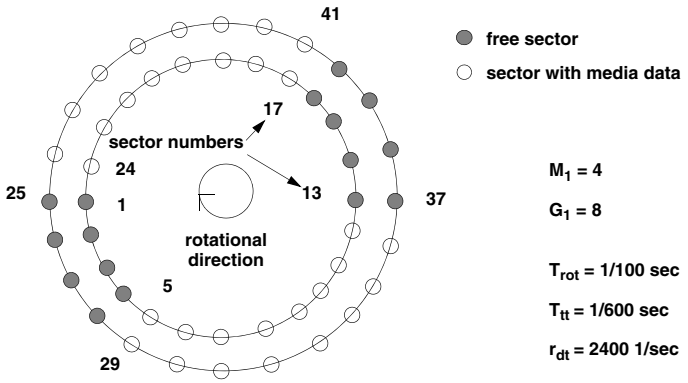
merging 2 strands



Merging Two Strands

- to use storage efficiently, blocks of s_2 have to be stored in the gaps of the pattern for s1; continuity must not be violated
- track-to-track seek time T_{tt} for s_2 can be "hidden" if T_{rot} x r_{dt} is a multiple of M_1+G_1 and $M_1>=T_{tt}$ x r_{dt}
- example: $M_1=4$, $G_1=8$, $T_{rot}=1/100$sec, $T_{tt}=1/600$sec, $r_{dt}=2400$ 1/sec

Storage Patterns



Merge Condition

- a strand s2 can be merged into s1 if the fraction of space occupied by media in s2's pattern does not exceed the fraction of empty space available in strand s1

$$\frac{M2}{M2+G2} \leq \frac{G1}{M1+G1}$$

- merge condition

$$\frac{G1}{M2} \geq \frac{M1}{G2}$$

Layout of Media Blocks

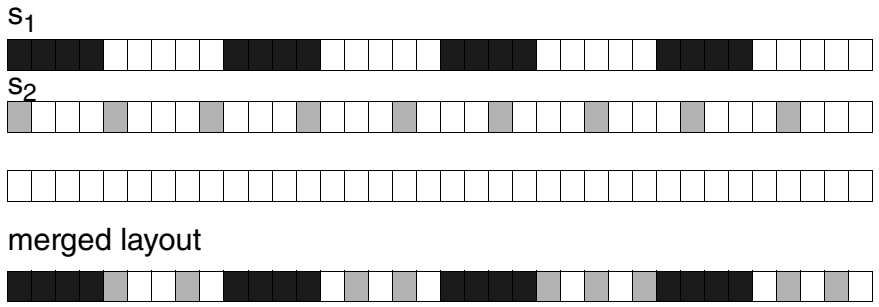
- if s_2's pattern is sparse compared to the gaps in s1, media blocks of s_2 are read earlier than their time of display leading to peaks in buffering requirements
- let k_i patterns of strand s_i span over a merge cycle L (i=1,2); we define pseudo block size of s_2 with respect to s_1 as the smallest integer p such that

$$k2 \times p \geq k1 \times G1$$

Layout of Media Blocks

- if $p=M_2$, then the merge condition is exactly satisfied
- if $p>M_2$, k_2 blocks of s_2 are distributed such that there are either $p\text{-}1\text{-}M_2$ or $p\text{-}M_2$ free sectors between consecutive media blocks of s_2
- the $(k_2+1)^{th}$ block of s_2 will be stored exactly L sectors away from the first media block of s_2, resulting in a pattern of filled and free sectors that repeats after every L sectors

Layout of Media Blocks

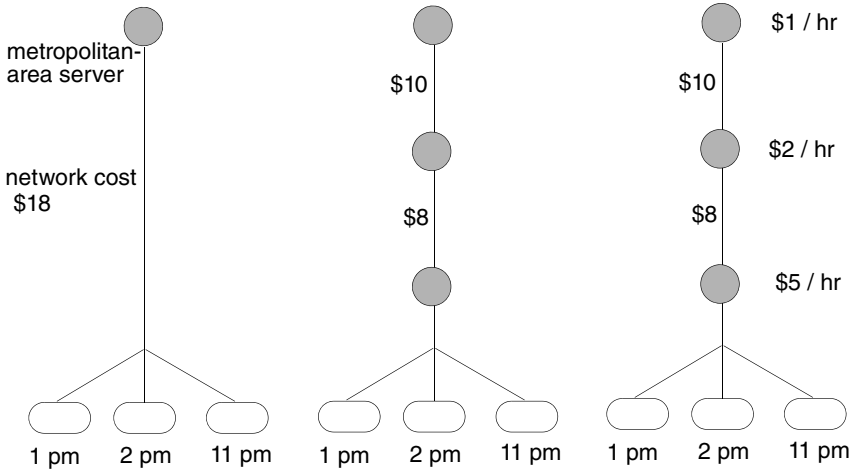


M1=4 G1=5 k1=4 L=36

p*k2-k1*G1=7

M2=1 G2=3 k2=9 p=3

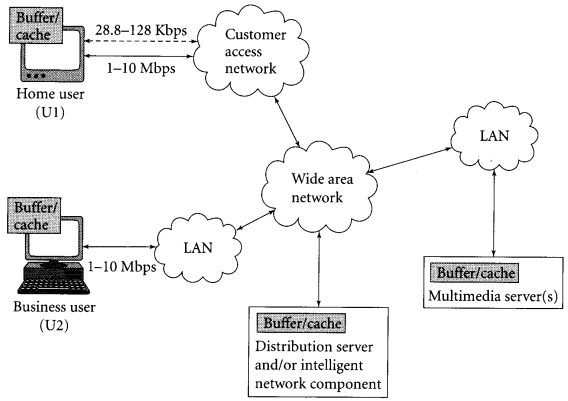
MM Information Caching



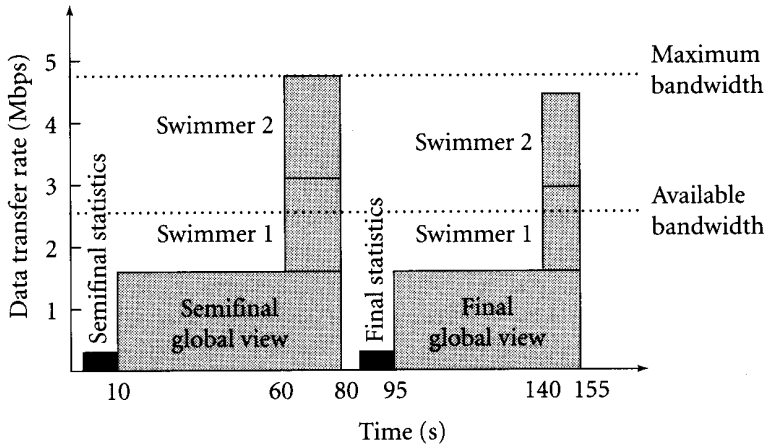
MM Information Caching

- development and deployment of large-scale media servers hindered by the lack of server and network bandwidth available
- caching of mm documents in local storage can reduce the required retrieval bandwidth
- e.g., storage of frequently used documents in client or intermediate distribution server nodes (see p. 231)
- different requirements of various applications (MOD vs mmdb)
- composite documents out of many small media segments result in wide bw. fluctuations (see p.232)

End-to-End Application Environment



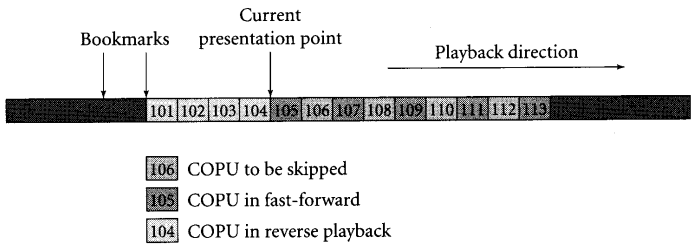
Composite MM Document



MM Information Caching

- bandwidth requirement profile can be smoothed by prefetching the required data blocks in advance
- access pattern within a single mm document may not be uniform and/or playback sequence may not always be linear => caching (and/or prefetching) of related data blocks
- data blocks retrieved continuously are referred to as a *continuous object presentation unit (COPU)*

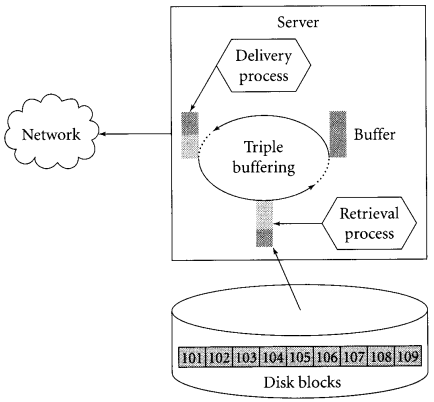
Document Access Patterns



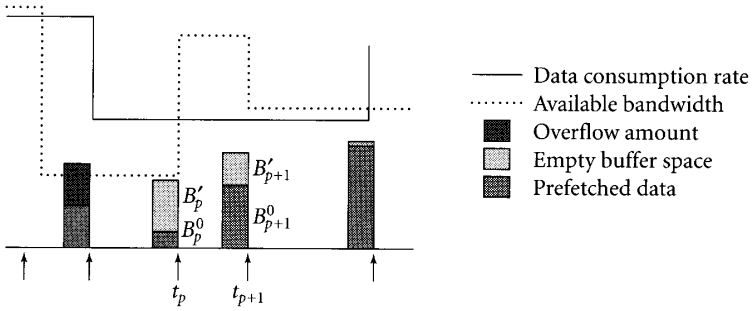
Prefetching, Buffering, Caching

- both buffering and caching concern the use of primary storage to avoid delay and/or overhead in accessing secondary storage
- distinction between buffering and caching lies in the performance objectives, application requirements and primary storage usage:
  - buffering is used to avoid access delay
  - caching is used to avoid access overhead and/or delay

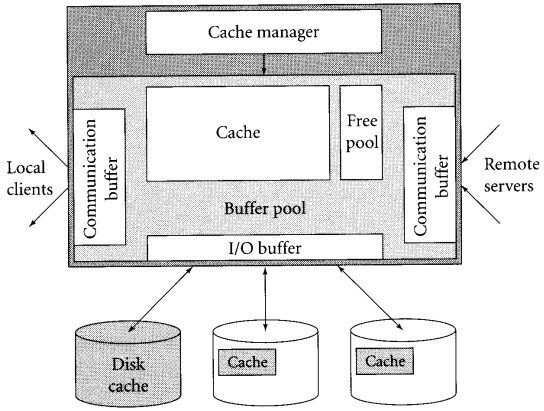
Buffering



Prefetching



Caching



Caching Objectives

- increasing server capacity—storing all or parts of frequently accessed mm objects in the server memory
- reducing access latency—access time changes with the location of data in the storage hierarchy; e.g. data stored in memory can be instantly delivered to clients
  - VOD, VCR control operations
- reducing network bandwidth requirements—a large-scale server consist of a local server (delivers data to clients) and remote storage servers (stores original copies of data); local server caches data on local disks and memory in order to avoid communication overhead

Caching Objectives

- balancing load across storage devices—depending upon initial placement of data and the current popularities of mm objects, one or more sets of servers or storage devices can be the bottleneck;
  - by employing a shared cache on the common part of the delivery paths and by caching selectively the data from the bottleneck servers, the load can be better balanced across all system resources
- supporting data migration in storage organizations—caching can be used to naturally replicate the popular data files and/or balance the load across system components; turning cached data as permanent copies and removing old copies of data that are in frequently accessed

Cache Management Policies

Characterization

- stream-dependent caching vs. block-level caching
  - traditional (e.g. block-level) caching policies cannot guarantee continuous delivery of streams
  - large mm objects need to be cached in their entirety
  - traditional concepts are based on the hot data concept (smaller in size)
  - bringing newly hot objects into cache may require a long time
  - *but*: relationships across playback streams can be established to support a following stream from the cached data

Cache Management Policies

Characterization

- memory vs. storage caching
  - objectives of memory-caching policies are improvement of cache space utilization (as many streams as possible)
  - storage caching policies need to address both, space and bandwidth
- caching vs. prefetching
  - related aspect of caching: prefetching of blocks accessed by a single stream, rather than sharing of blocks across streams; prefetching can mask variances in response time and hence, avoid jitters

Cache Management Policies

common issues

- adaptive workload
  - policies should cope with dynamic changes in workload and heterogeneity arising from small and large mm files
  - cope with large-scale distributed mm environments consisting of many heterogeneous servers
  - ensuring good cache hits in each server
  - balancing of loads across servers

Cache Management Policies

common issues

- support for VCR control
  - problem arises as well in other aspects of operations (e.g., batching, load balancing)
  - contingency policies (see GIC)
- integration with other resource optimization policies
  - blocks retrieved by one client can be reused by others closely following
  - batching
  - stream chasing

Memory Caching Policies

stream-dependent caching

- viewer enrollment window
- BASIC
- DISTANCE
- interval caching
- generalized interval caching (GIC)
- split and merge (SAM)
- least/most relevant for presentation (L/MRP)
- hypergraph access hints

Viewer Enrollment Window

single buffer

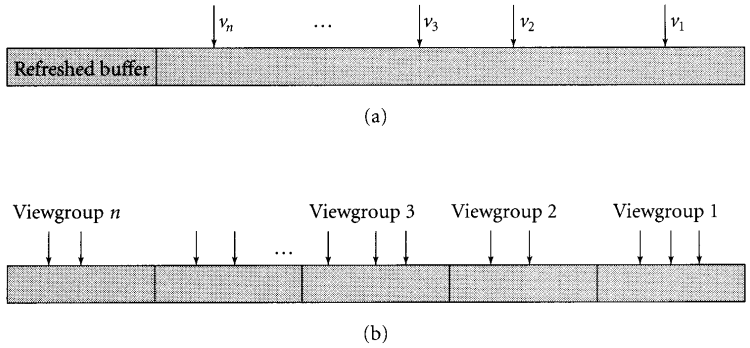
- when multiple streams access the same media object and if a large buffer is allocated to the first stream, subsequent streams can be served from that buffer
- page 248 a): v1 to vn are accessing the same object, v2 to vn are served from the buffer without prefetching
- buffers are refreshed on behalf of v1 periodically
- number of viewers is controlled such that buffer refreshment for v1 does not overwrite any blocks to be consumed by others
- viewers for the same object are accepted during an enrollment window

Viewer Enrollment Window

partitioned buffer

- policy improved by partitioning the buffer and allocating each buffer partition to a viewgroup
- multiple enrollment windows
- policy further improved
  - by partitioning the buffer adaptively
  - upon arrival of a new viewer after all enrollment windows are closed, a new viewgroup is created by reclaiming some unused buffer space

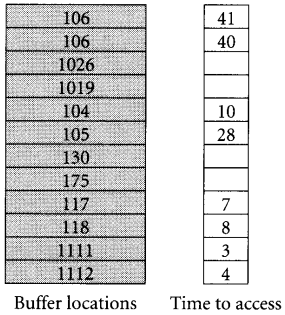
Viewer Enrollment Window



BASIC

- termed due to the simplicity of approach
- when a buffer is to be allocated, BASIC selects the buffer which contains the block that would not be accessed for the longest period of time by existing clients if each client consumed data at the specified rate
- high implementation overhead, because policy reestimates access patterns and relationships across streams on every cycle and block replacement

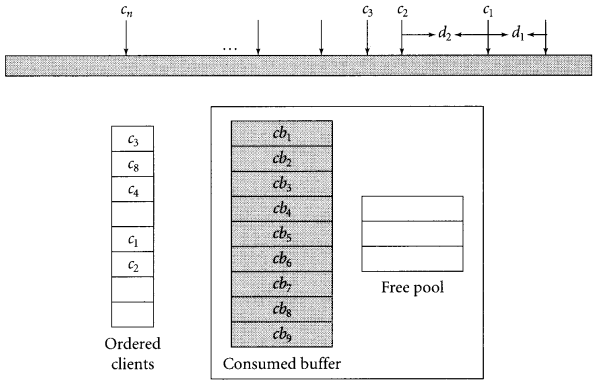
BASIC



DISTANCE

- makes explicit use of distance relationships across streams
- the distance, di, of a client, ci, from an immediately following client, ci+1, is measured by the offset in the number of blocks
- for a client with no immediately following clients, the distance is assumed to be very large
- policy sorts all clients in an increasing order of their distances
- in each service cycle, it first frees consumed buffer in the previous cycle, cbi, in the increasing order of di
- lower implementation overhead than BASIC because distance calculation changes only if a client pauses or stops or at a new arrival

DISTANCE



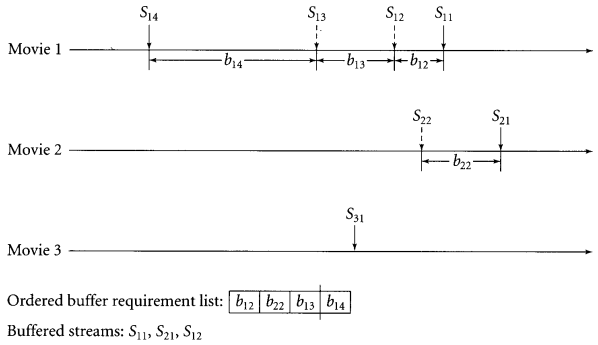
Interval Caching

- data blocks between a pair of consecutive streams, S_{ij} and $S_{i(j+1)}$, is referred to as an *interval*
- two streams associated are referred to as *preceding* and *following* streams
- by caching a running interval, the following stream can be served by using the blocks brought in by the preceding stream
- the size of an interval is estimated as the time difference between the two streams in reading the same block

Interval Caching

- number of blocks needed to store an interval—cache requirement of interval
- to maximize the cache hit ratio and to minimize I/O, the policy orders the intervals in terms of interval sizes and caches only the shortest intervals
- low implementation overhead, changes occur only at times of arrival or termination

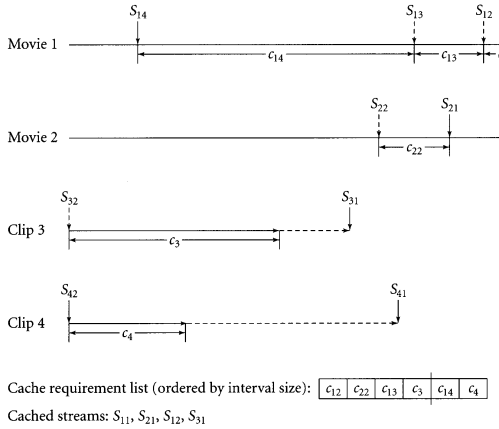
Interval Caching



General Interval Caching

- IC not optimal for short media objects (no running intervals can be performed)
- policy integrates the stream-aware cache management as well as temporal locality
- for small objects: time interval between two successive accesses of the same object
- however, the cache requirement equals the size of the object; in this case the entire object is cached
- size of the last interval is assumed to be the anticipated interval size

General Interval Caching



GIC Policy Evaluation

Interactive Workload

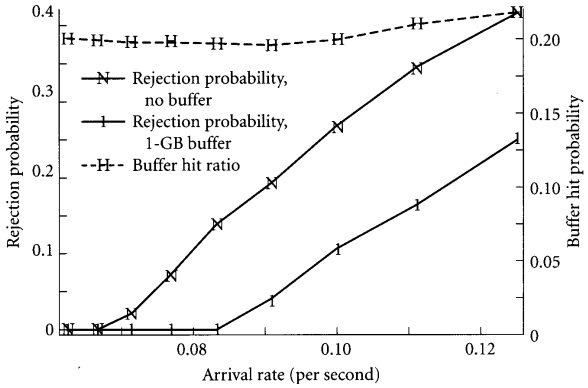
- video object access skew—80% of accesses are to 20% of video objects
- video object length distribution—Lmin to Lmax
- viewing time per individual clip—random viewing time Tv, 5 sec minimum
- client session duration—length Ts, clients repeatingly accessing clips

GIC Policy Evaluation

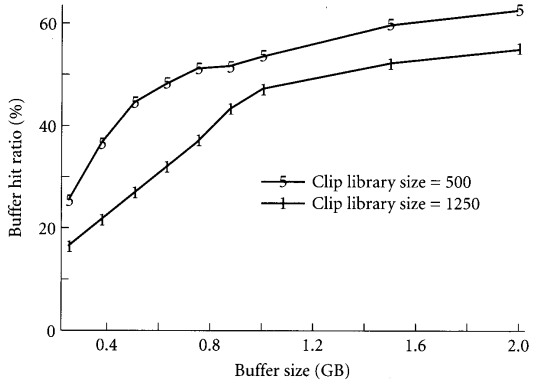
Workload Parameters

Parameter	Default value
disk capacity	400 streams
number of clips	500
length of clips	1 to 30 s
average viewing time	30 s
interactive session length	30 min
number of long videos	92
length of long videos	90 min
request interarrival time	6.25 s
interactive probability	0.8

Rejection Probability



Overall Cache Hit Ratio



Interactive and Movie Cache Hits