

# **An New Approach to Storage-Aware Caching in Heterogeneous Storage Systems**



**Liton Chakraborty, Ajit Singh**  
University of Waterloo, Canada

June 19, 2005  
COSET 2005, Cambridge, USA

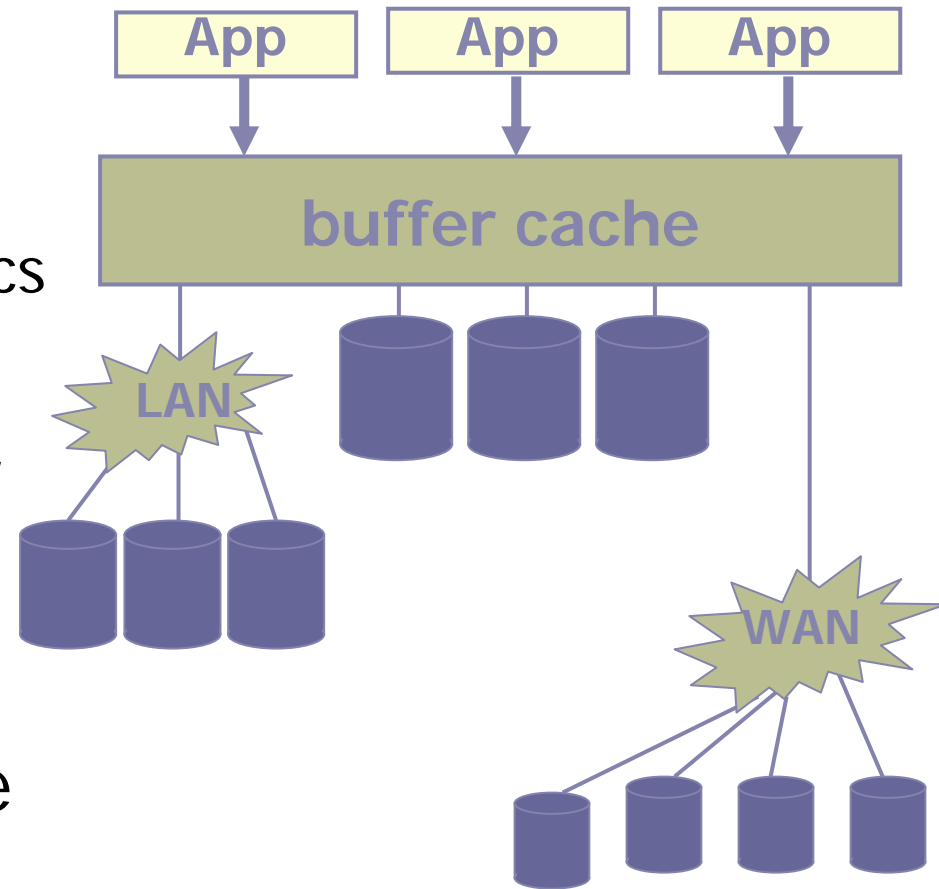
# Presentation Overview



- Introduction
- Motivation
- Solution Approach
- Experiments
- Conclusion

# Modern storage system

- Diverse set of storage devices:
  - More devices
  - Attached in various ways
  - Varying disk characteristics
- So, non-uniform access time among the devices, and non-uniform replacement cost of the cache blocks.
- Caching Policy should be cost-aware.



# General Solution



- Integrate Workload and device performance.
  - Balance work across devices (work=cumulative delay).
- Apply a dynamic partitioning algorithm.
  - **Partition** the cache, **assign** a partition to each device.
  - **Determine the proper size of each partition.**
  - Cost-oblivious caching policy (e.g., LRU) can be applied within each partition.

# Partitioning (Forney's algorithm)



- Measure cumulative wait time for each partition over the last epoch or window (i.e.,  $W$  disk requests )
- Repartition:
  - Determine page suppliers and consumers based on the relative wait time for each device and the threshold value ( $T$ ).
  - A page consumer increases its partition size by  $I$  pages ( $I$  = base correction amount).

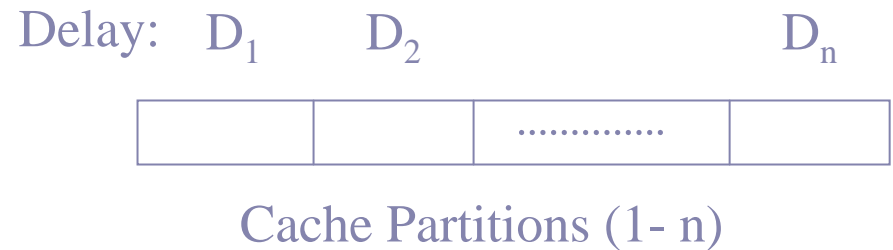
# Motivation



- In Forney's Algorithm, corrective action is taken at the end of an epoch.
  - Performance may degenerate.
- Small epoch ( or window size  $W$ ) renders smooth performance
  - But, poor feedback
  - Frequent repartitioning
- Two other parameters: threshold ( $T$ ), Base increment amount ( $I$ ).

# Solution Approach

- Balance the *work* across devices in continuous fashion (without using the notion of epoch).
- Associate cumulative delay ( $D_i$ ) with every partition.
- Upon miss, increment  $D_i$  by retrieval Delay.
- Make  $D_i$ s homogeneous or almost equal.



# Approach(Contd.)



- On cache miss, each partition decides:
  - Receive block from others, or
  - Replace block from own
- This decision is based on its  $D_i$  value

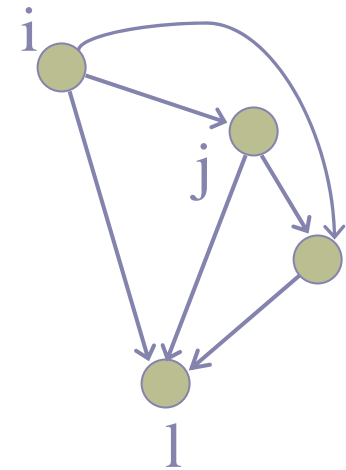
# A Simple Solution



- Partition  $i$  receives block from partition  $j$  if  $D_i > D_j$
- Can be implemented using a DAG (Directed Acyclic Graph) : with edge  $i \rightarrow j$  if  $j$  is supplier for  $i$ .
- Cons
  - Requires frequent and unnecessary block switching
    - Hence, blocks may not have good utilization
    - Block switching is not entirely free of processing overhead.

# Refinement

- Introduce a threshold or offset ( $\delta$ )
- Add an edge  $i \rightarrow j$ , if  $(D_i \geq D_j + \delta)$
- Problem
  - Partitions receiving blocks from certain partitions, may deliver blocks to some other partitions.
  - Hence, unnecessary block switching
- Don't allow a partition to be candidate to both supply and consume block at the same time



$j$  consumes from  $k$  and  $1$ ; but at the same time may supply to  $i$ .

# Further Refinement



- Previous approach bounds the variation in  $D_i$ s; but
  - it requires to update the DAG every time a miss occurs.
  - Consumes space, as each partition or node may have a number of suppliers

# Further Refinement(Contd.)

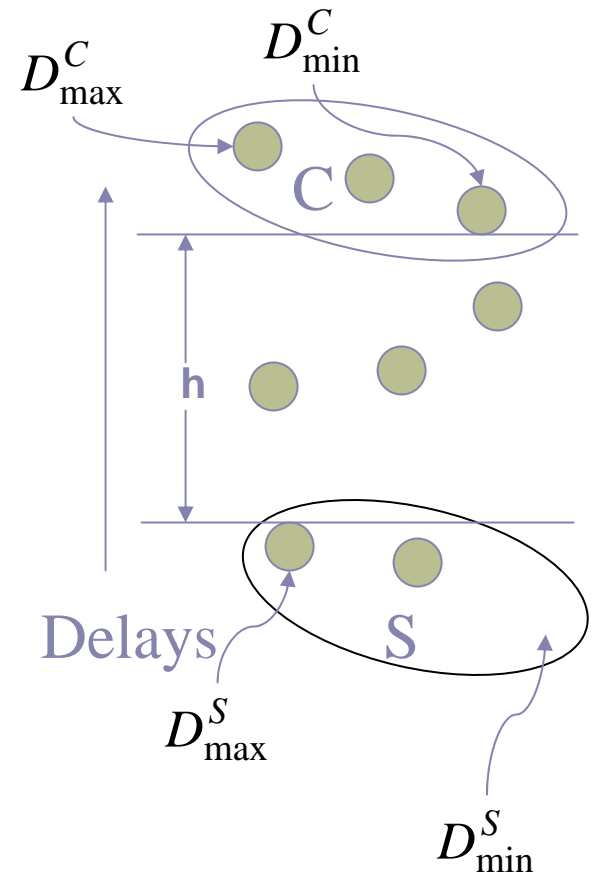
- Maintain two lists: C ( for Consumers) and S (for suppliers).
  - 4 variables (shown in figure) keep track of maximum and minimum value of both list.
  - the minimum gap,  $h \geq \delta$

- Condition (for partition i) to be in list C:

$$D_{\max}^C - D_i < \delta \quad \text{and} \quad D_i - D_{\max}^S \geq \delta$$

- Condition (for partition i) to be in list S:

$$D_i - D_{\min}^S < \delta \quad \text{and} \quad D_{\min}^C - D_i \geq \delta$$



# Evaluation methodology

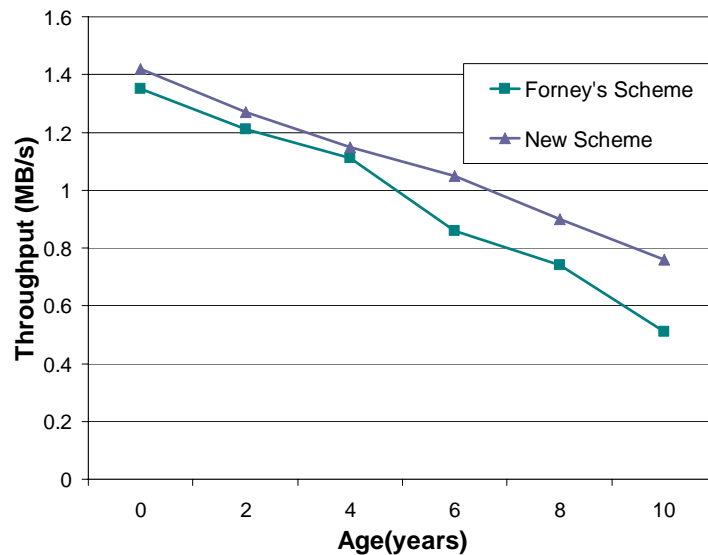


- Disks: IBM 9LZX, model the disk access time using bandwidth, seek time and rotational latency.
- Request size for the cache is equal to the block size.
- Workloads: Synthetic traces.

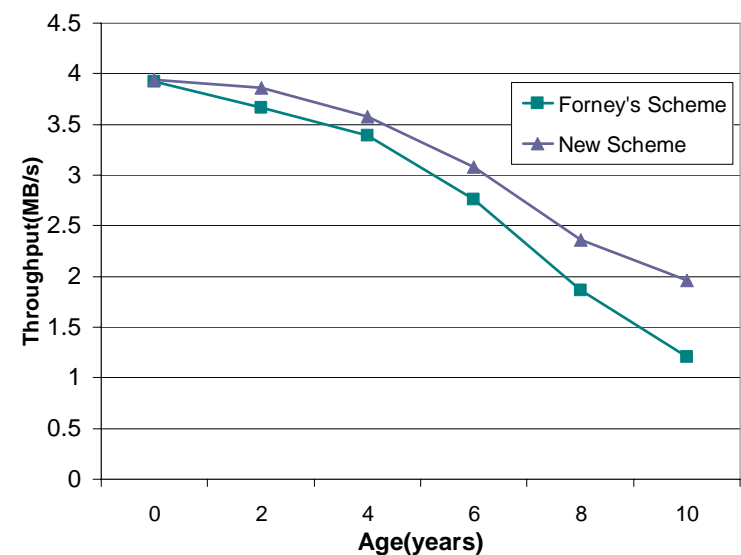
# Experiments

- Varying ages of the slow disk.

Throughput with varying ages of the slow disk (trace-1)



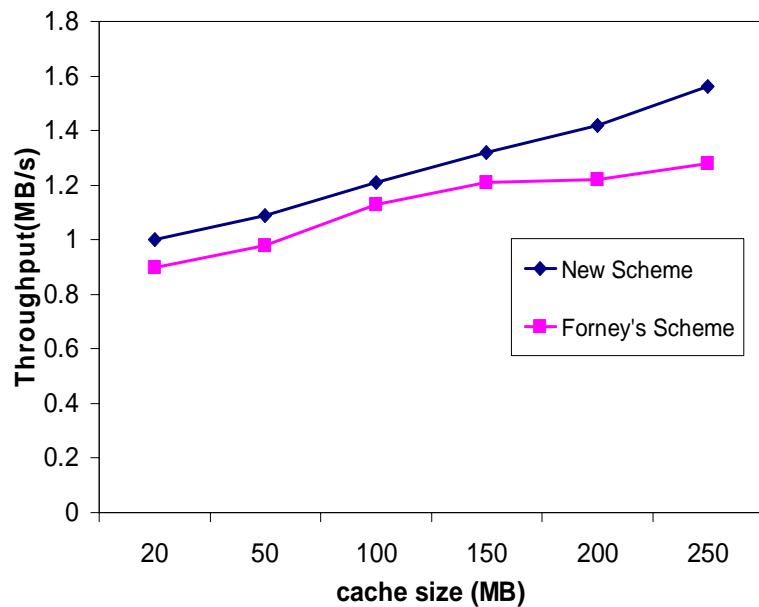
Throughput with varying ages of the slow disk (trace-2)



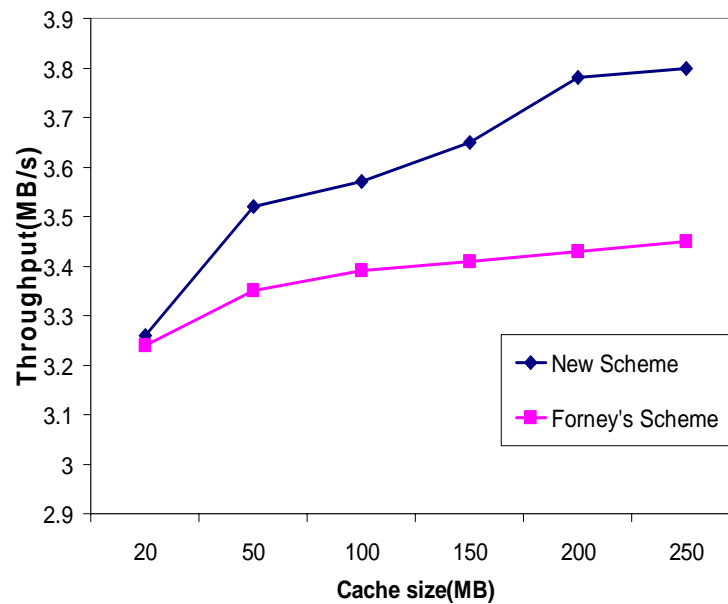
# Experiments (contd.)

## ■ Varying cache sizes

Throughput with varying cache sizes (trace-1)



Throughput with varying cache sizes (trace-2)



# Future Work



- Disks of varying age with nearly equal workload (i.e., uniform workload).
  - Blocks are continually moved to the slow disk that can no longer increase its hit rate.
  - Choking of the faster disks might occur.
  - A utility based scheme seems to be a solution to this issue.
- Integrate prefetching with the caching in the heterogeneous storage environment.

# Conclusion



- Identified the problem with the existing Algorithm.
- Attempted to adjust the partition in a continuous fashion: no periodic repartitioning.
- Up to 15% increase in throughput
- Uses only one parameter ( $\delta$ ) that can be chosen with care.