SDM Center

Berkeley Lab

File Caching in Data Intensive Scientific Applications on Data-Grids

E. Otoo, D. Rotem, A. Romosan

Lawrence Berkeley National Lab.
1 Cyclotron Road
University of California
Berkeley, CA 94720

S. Seshadri

Leonard Stern School of Business
New York University,
44 W. 4th Street
New York, NY 10012

Outline

- Traditional File Caching – Storage Hierarchy
- Problem Statement
- Motivation
- Caching by File Bundles
- File-Bundle Replacement Algorithm
- Land-Lord (Greedy-Dual Size) Algorithm
- Simulations and Performance Comparison
- Conclusion and Future Work

Caching in Storage System Hierarchy

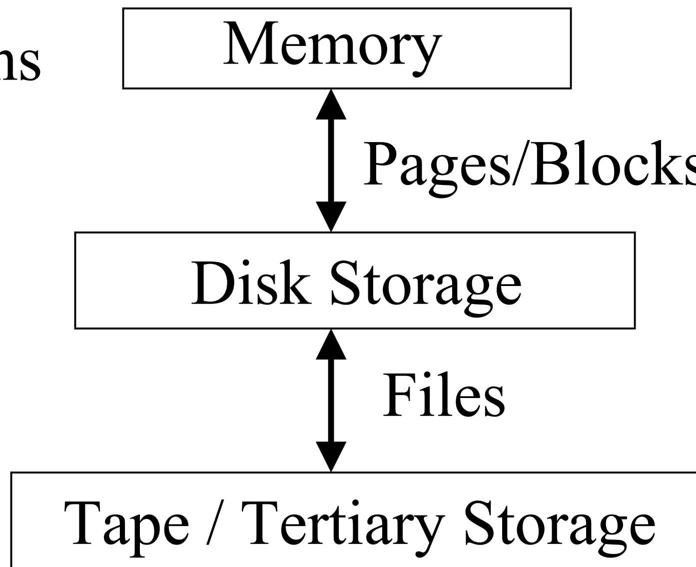
Key Concept: Keep frequently accessed data in faster but more expensive and limited storage

Capacity & Access Time

512KB – 4GB: ~ 100ns

4 – 256GB: ~ 5ms

100GB: ~ 10s



Access Fastest — Slow

Capacity Small — Largest

The Traditional File Caching Problem

- File requests arrive at a queue
  - Service is on a first come first serve (FCFS) basis.
  - Each request is for one file at a time.
- A request is serviced only if its file is in the cache, otherwise a “fault” occurs
- A replacement decision is made whenever a "fault" occurs (Cache Replacement Policy)
- A replacement decision involves choosing one or more files for eviction to make room for the needed file and then loading the file into the cache.
- Even if we have multiple files per request, the decision is still made on per file basis.

Problem Statement

- **Consider a disk-cache for staging files of jobs at some compute element of a data grid.**
- **The set of files (*file-bundle*), requested must all be in the disk-cache for the job to be processed.**
- **What is an optimal file-bundle replacement policy?**
- **Optimal in the sense of:**
  - **Minimizing the volume of in cache data replaced**
  - **Maximizing the throughput of jobs**
  - **Others: makespan, etc.**

Main Results

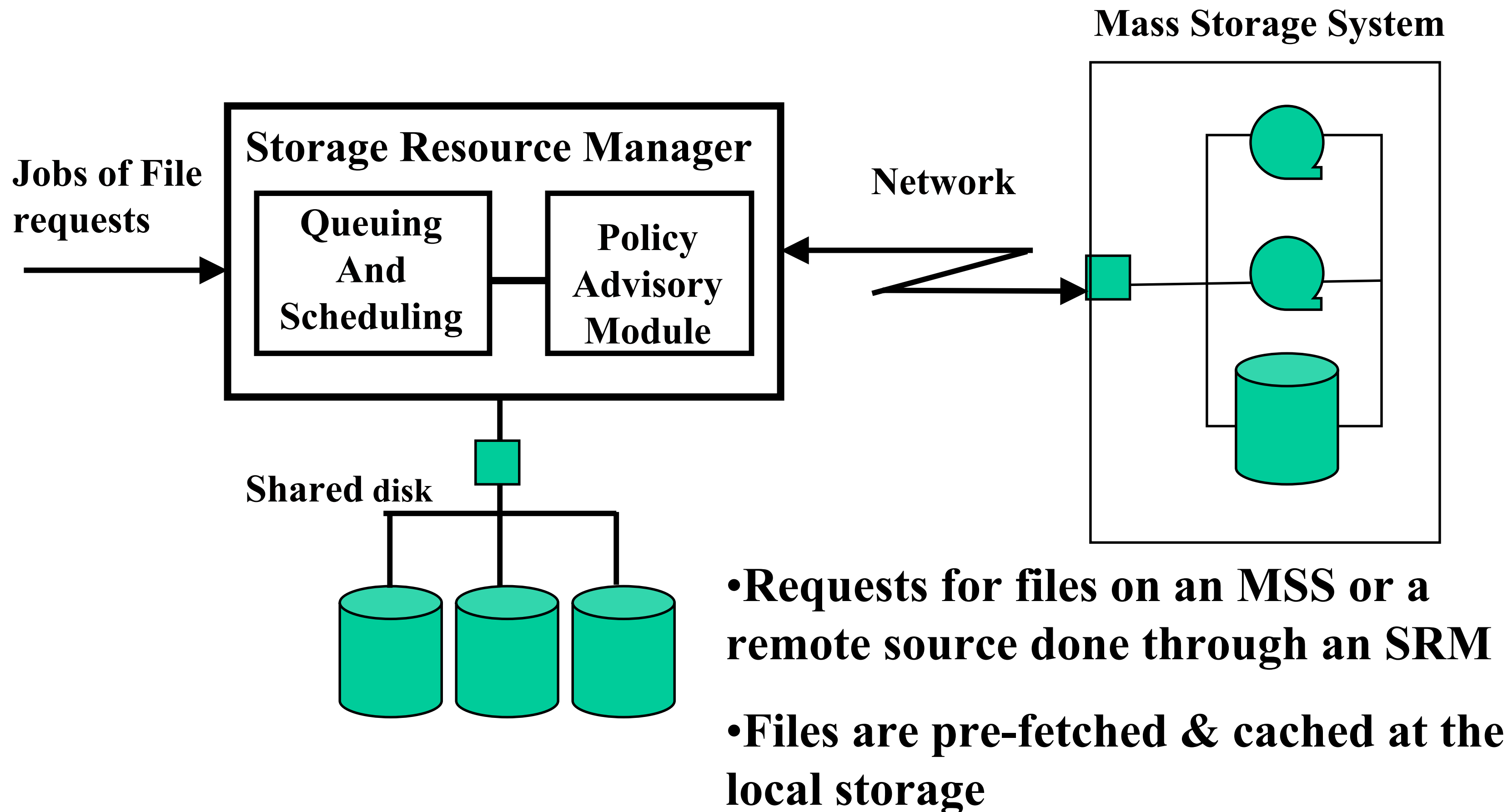
- **Heuristic Algorithm for File_Bundle_Caching**
  - **A cache-replacement algorithm that tracks the history of requests to determine combinations of files to retain in cache**
- **Optimality Condition**
  - **The solution is within a factor of ($2d^*$) from the optimal decision for each replacement where $d^* \leq \max_i (f_i)$, is the maximum number of requests for the same file.**
- **Comparison with Greedy-Dual-Size (GDS)**
  - **For both Uniform and Zipf request distributions FBC out-performs GDS in minimizing byte miss-ratio.**

Motivation

- **Domain**
  - **Data-intensive scientific applications in data-grid environment**
  - **Middleware services such as Storage Resource Managers (SRM), for efficient data access**
    - **Scheduling of file requests**
    - **Caching or pre-staging of files**
    - **Replica replacement, etc.**
  - **Different job service models:**
    - **One file at a time**
    - **Sets of files (File-Bundles), at a time**

Motivation – Cont.

Using an SRM to Access a MSS/Remote Storage



- Requests for files on an MSS or a remote source done through an SRM
- Files are pre-fetched & cached at the local storage

Motivation - Cont

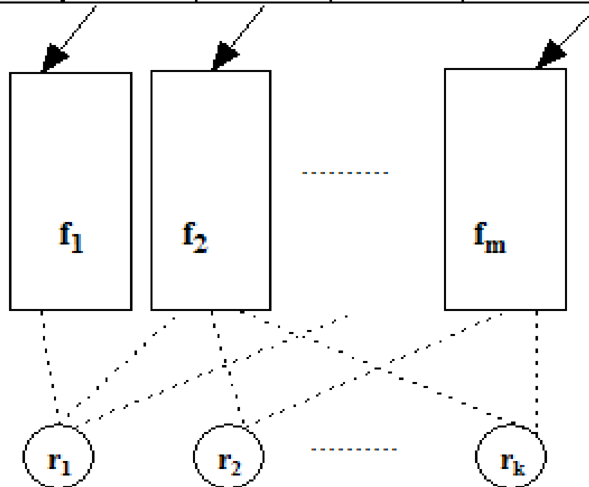
- Application areas:
  - High Energy and Nuclear Physics (HENP) data analysis
  - Climate Modeling
  - Bitmap indexing

Caching Based on File Bundles

- **Why File Bundles**

Time step	A_1	A_2	.....	A_m
T1				
T2				
T3				
T4				
T5				
.....				
Tn				

multiple time step climate simulation output



vertical partitioning into multiple files

analysis jobs requesting subsets of the files

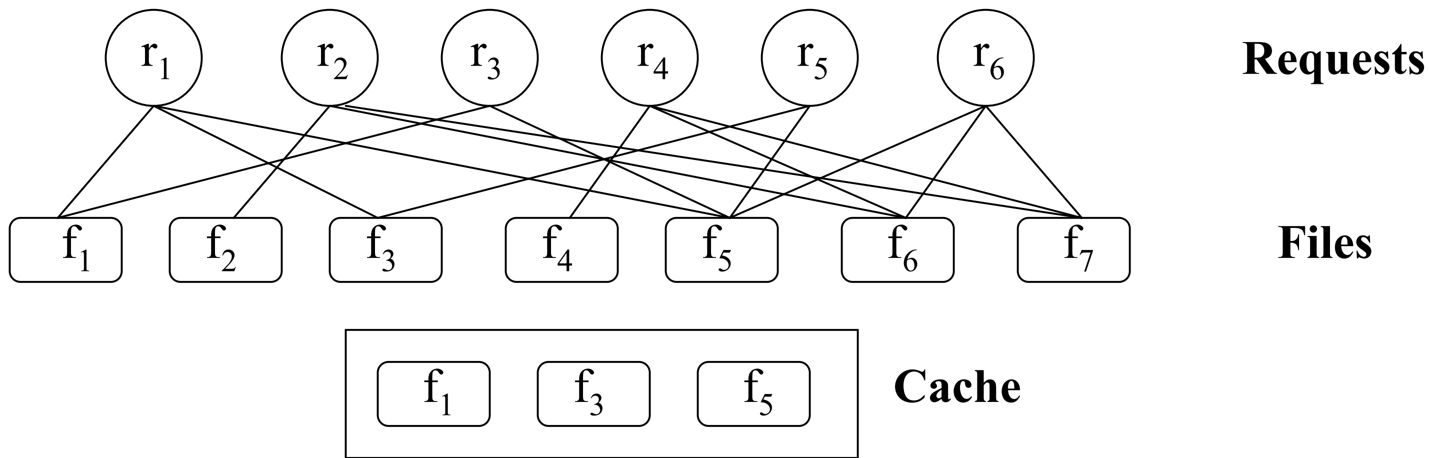
Caching Based on File Bundles - II

- **Main Idea**
  - **Load the disk-cache with files such that the probability that a request finds all its files in the cache is maximized.**

File Bundle Caching as an Optimization Problem

- **The File-Bundle Caching (FBC) problem is defined as follows:**
  - **Given a collection of requests $R = \{r_1, r_2, \ldots r_n\}$, a set of files $F = \{f_1, f_2, \ldots, f_m\}$, and a disk cache of constant size $s(C)$.**
  - **Each request r_i is associated with a value $v(r_i)$, defined over the files F with each file f_j being of size $s(f_i)$.**
  - **Find a subset R' of the requests R such that the requests in R' have maximum total value and the total size of the files needed by R' is at most $s(C)$.**

Caching by File Bundles - Example



File	# Requests	Req. Prob
f_1	2	1/3
f_2	1	1/6
f_3	2	1/3
f_4	1	1/3
f_5	4	2/3
f_6	3	1/2
f_7	3	1/2

Cache Content	Requests	Hit Prob.
f_5, f_6, f_7	r_6	1/6
f_1, f_3, f_5	r_1, r_3, r_5	1/2
f_1, f_5, f_6	r_3	1/6
f_3, f_5, f_6	r_5	1/6
f_1, f_2, f_3	-	0

Complexity of the Problem (The Bad News)

- NP-hard even if no file sharing between requests
  - Reduction to knapsack problem
    - Knapsack: N items each has a cost and a value
    - FBC:
      - Requests are items
      - Total size of files needed by a request corresponds to the item cost
      - Value of a request corresponds to item value
    - Knapsack: Maximize value of items loaded to knapsack subject to total cost constraint C
    - FBC: Cache corresponds to knapsack,
    - Maximize value of requests satisfied subject to cache size constraint C

Greedy Heuristic - Main idea

For each file compute adjusted size

$$s'(f_i) = \frac{s(f_i)}{d(f_i)}$$

- $s(f_i)$: File size
- $d(f_i)$: Number of requests using file

For each request compute adjusted value

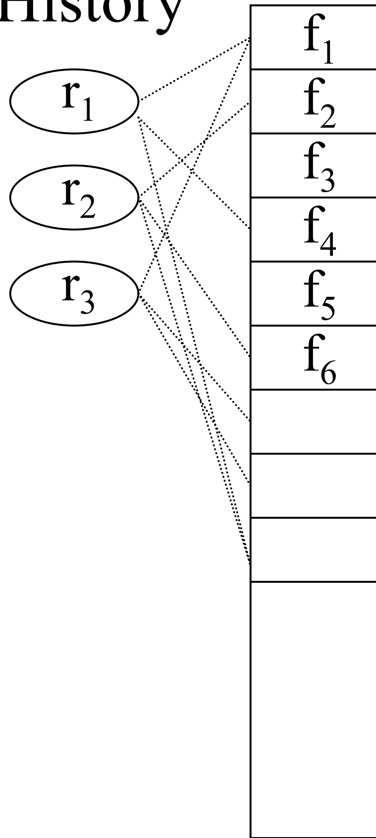
$$v'(r_j) = \frac{v(r_j)}{\sum_{f_i \in F(r_j)} s'(f_i)}$$

- $v(r_j)$: Value of request
- $f_i \in F(r_j)$: Files of request r_j
- $\sum_{f_i \in F(r_j)} s'(f_i)$: Total adjusted file sizes per request

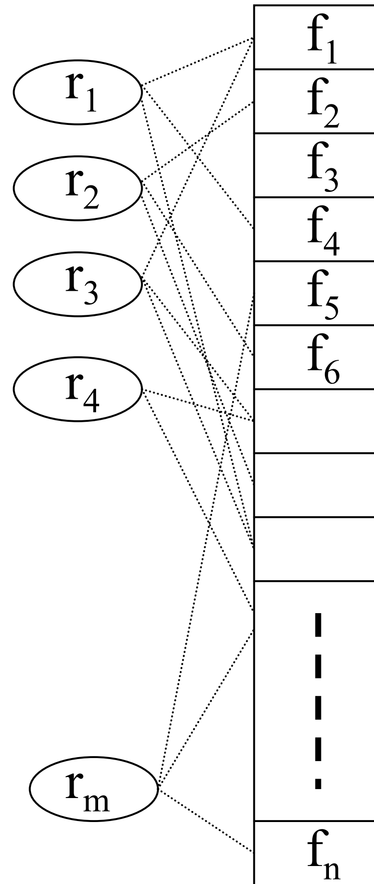
File-Bundle Heuristic Algorithm

- Maintain a history of previous requests
  - Frequency, files needed, etc.
  - Length of history can be adjusted to fit available resources
- Current request must be satisfied by placing its files in the cache
- For the remaining cache size
  - Sort requests (in the history), in decreasing order of their adjusted value and
  - Load (missing) files associated with them into the cache until cache is filled

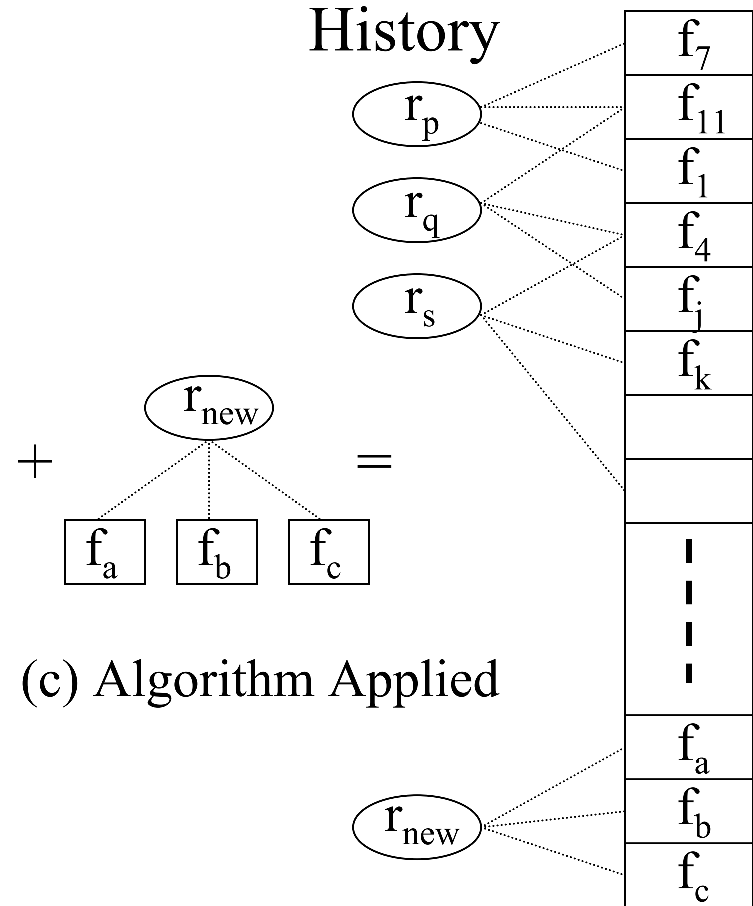
File-Bundle Algorithm - Illustration



(a) Cache Filling Up



(b) Cache Full



(c) Algorithm Applied

(d) Resulting Cache

LandLord / Greedy-Dual Size

Greedy Dual Algorithm[Cao and Irani, 1997]

Initialize: Set L ← 0

Consider request for a file *p*

if *p* is already in cache *then*

set $\phi(p) \leftarrow L + c(p)/s(p)$

else

while there is not enough room for *p* *do*

set $L \leftarrow \min_{q \in cache} \phi(q)$

evict *q* such that $\phi(q) = L$

endwhile

Bring *p* into cache and set $\phi(p) \leftarrow L + c(p)/s(p)$

endif

Simulations and Performance Measurements

- Performance Metric
  - Miss Ratio
    - Ratio of files requested but not found in the cache to total number of files
  - Byte miss ratio sensitive to file sizes
    - Number of bytes not found divided by total number of bytes requested
  - System throughput
    - Number of jobs processed by the system in some unit of time
  - Average wait time in queue before being serviced
  - Maximum wait time in queue
    - Measure of system fairness
    - Possible starvation due to cache admission

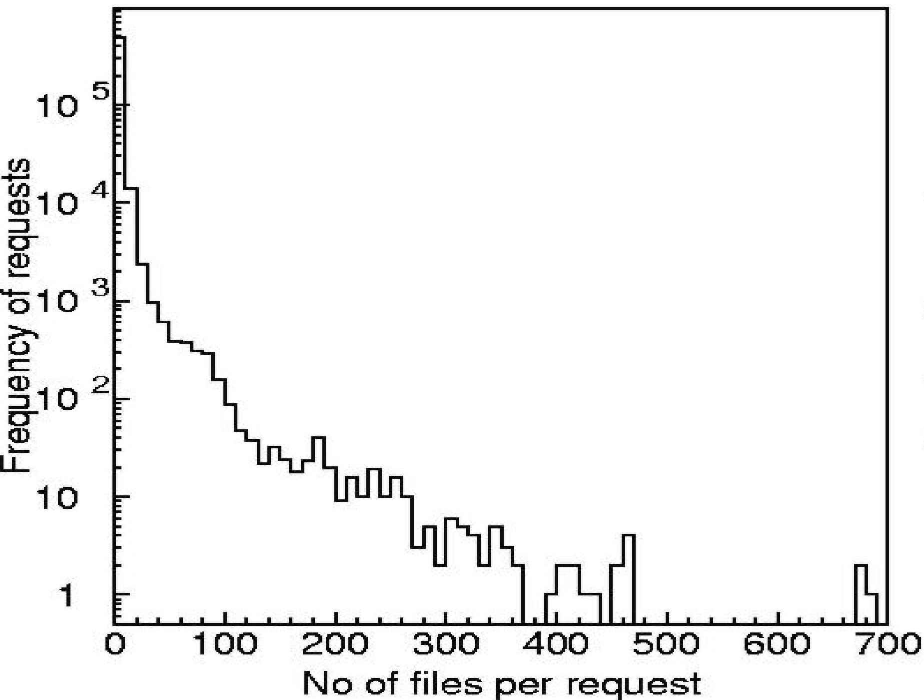
Simulations and Performance
Environment and Parameters

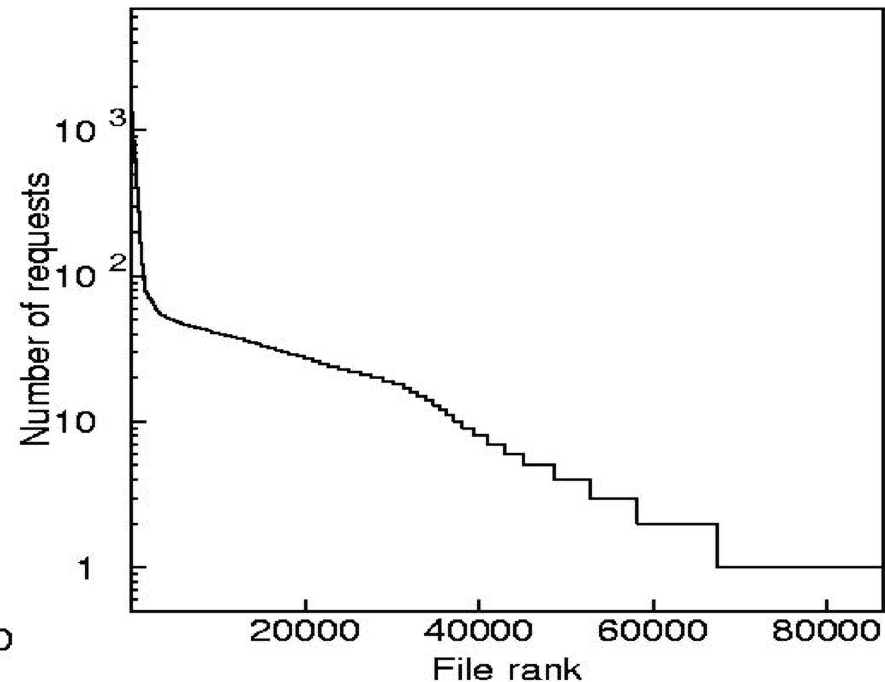
- **Workload**
  - **Trace data available on one file per request basis**
  - **Synthetic data generated based on request distributions of real logs from BaBar Experiments at SLAC**
- **Parameters Varied**
  - **Request sizes**
  - **Distribution of file sizes**
  - **Distribution of request frequencies**
  - **Degree of file sharing among requests**
  - **Length of request history retained**

SDM Center

Sample Workload from Babar Experiment Runs

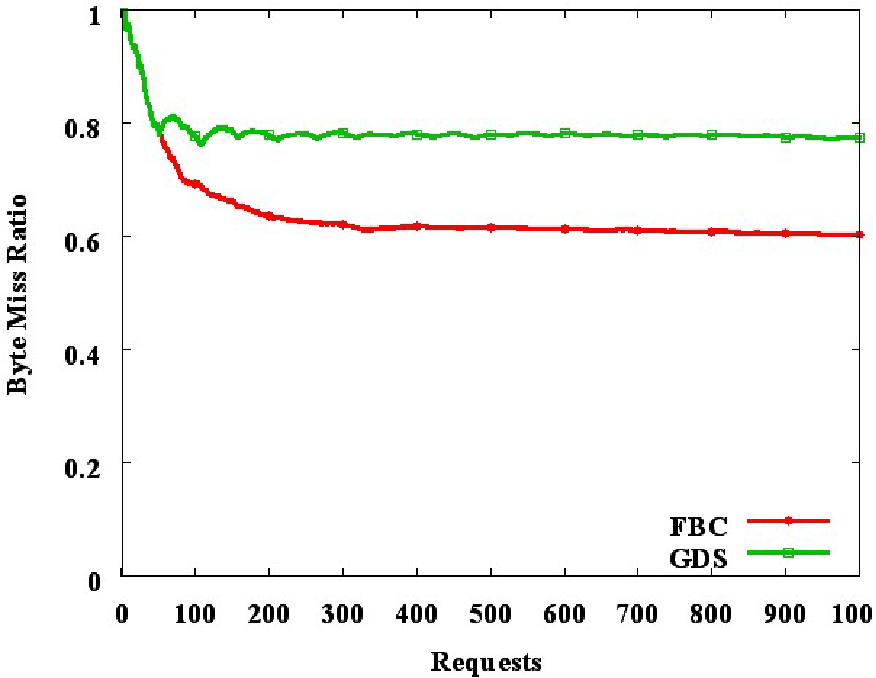
Berkeley Lab

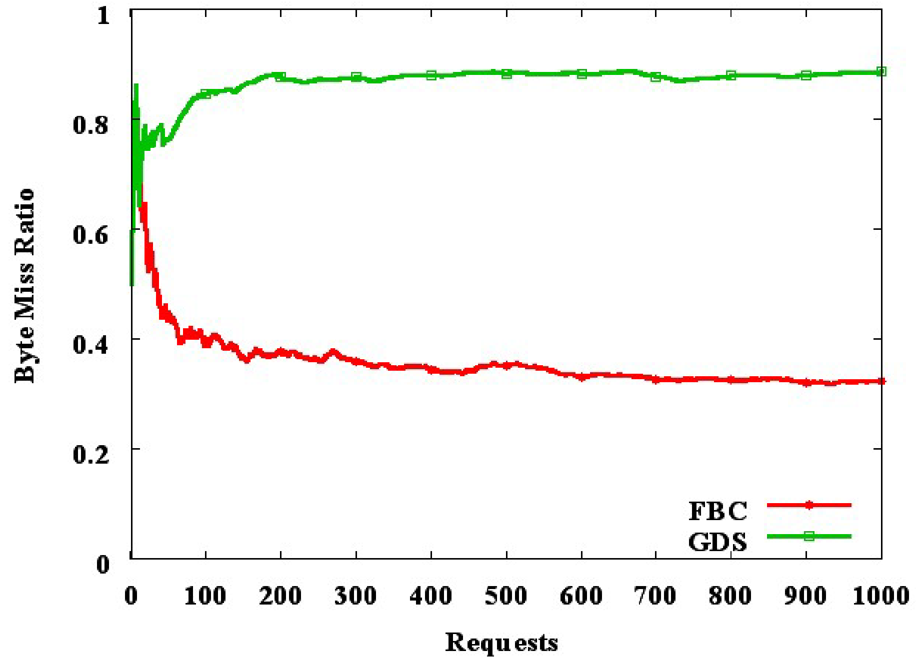




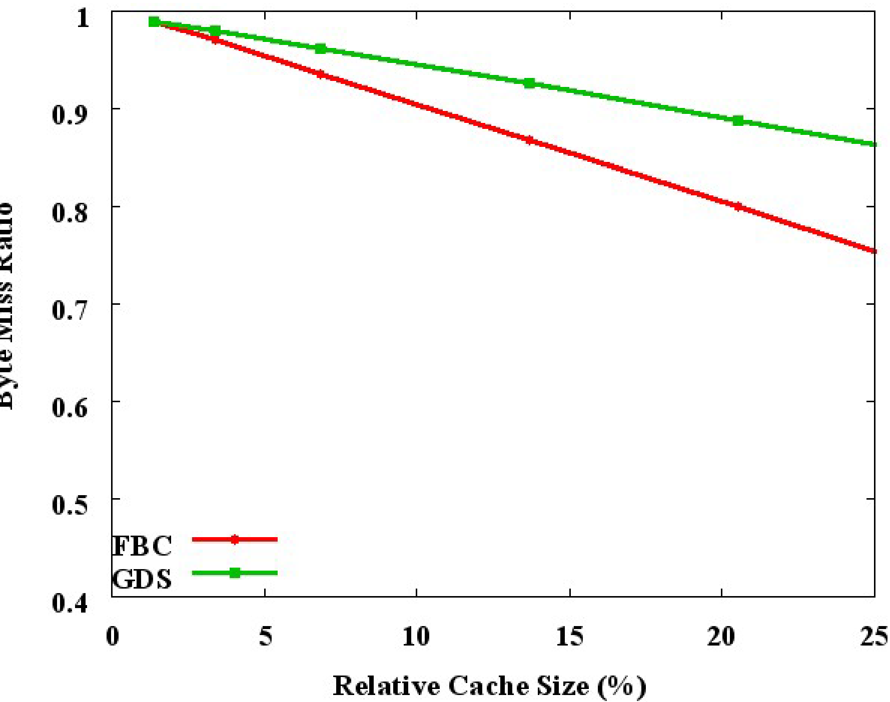
- No of requests that ask for some no of files
- Files ranked by no of requests

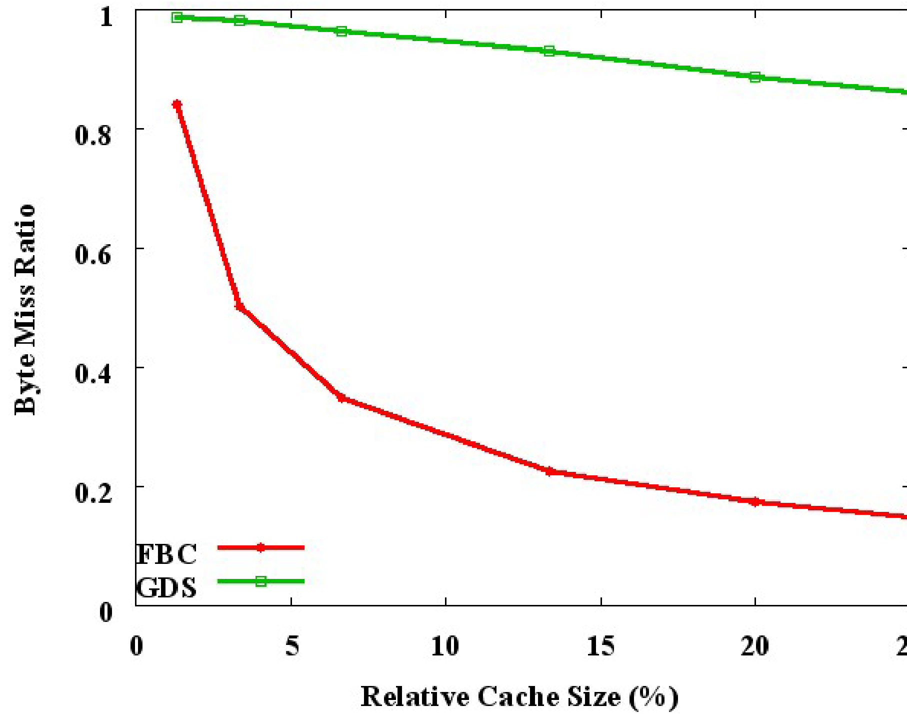
Byte-Miss Ratio vs #Request in Cache





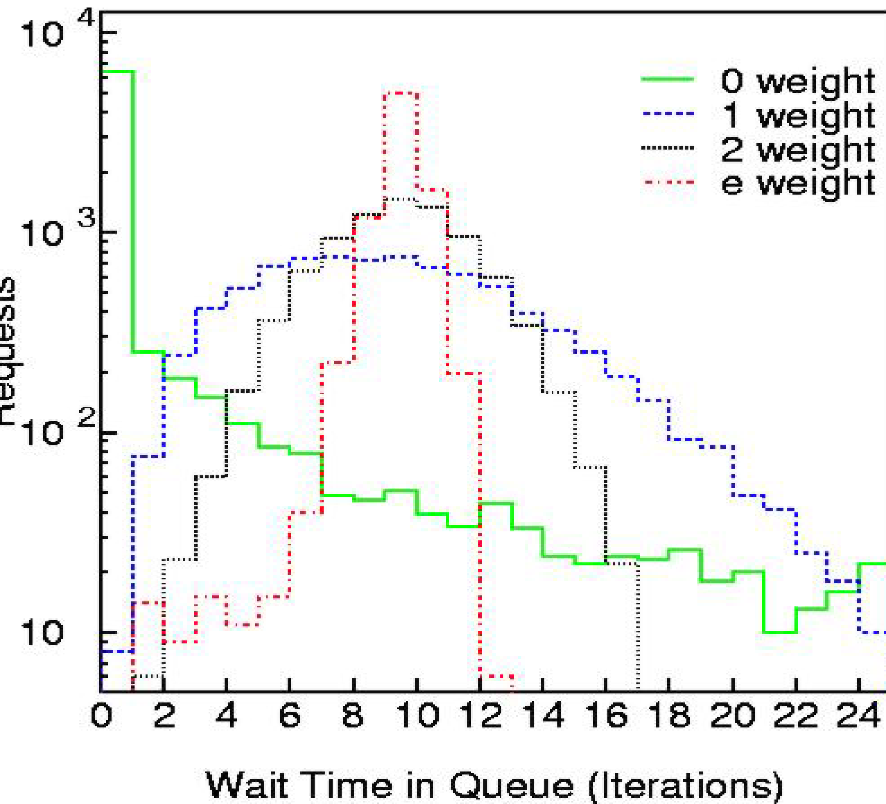
Byte-Miss Ratio vs Cache Size as % of Data Request

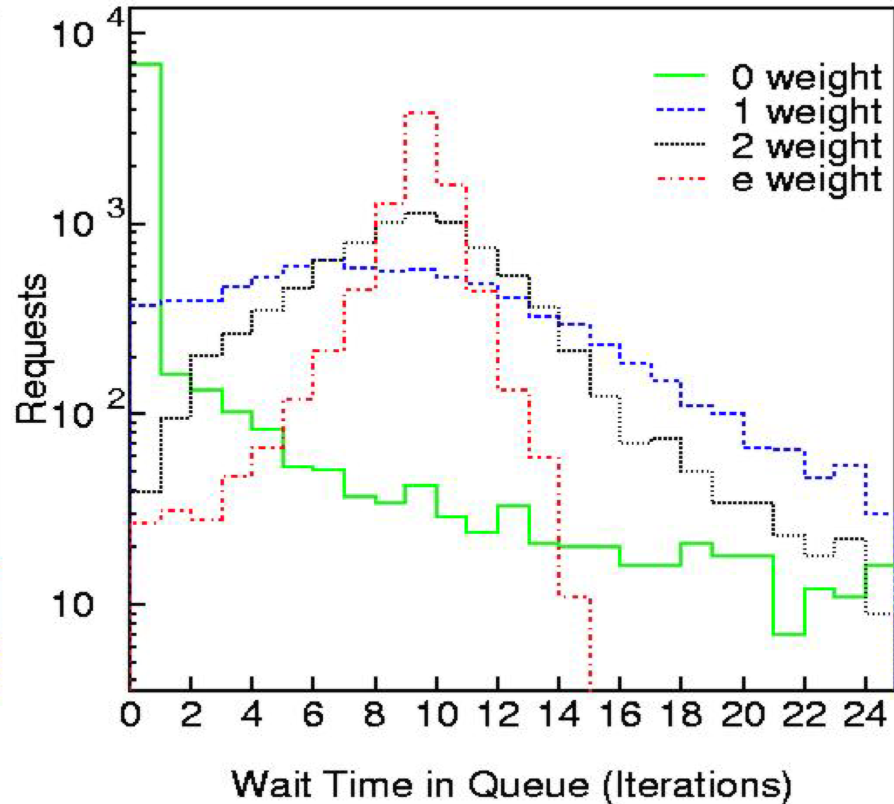




SDM Center

Requsts Waiting vs Time in Units of Iterations





Weight is i^1, i^2, e^i

Conclusion

- Addressed a new caching problem, i.e., File-Bundle Caching, that arises in data intensive applications in data-grids.
- Derivation of a heuristic algorithm that is simple to implement and takes into account the dependencies among the files in requests.
- Analysis of the heuristics and derivation of bounds from the optimal solution
- Simulation results comparing the new algorithm with one of the best known algorithm – Greedy-Dual-Size

Future work

- **Proposed algorithms may have similar implications in web caching**
- **Hybrid model of service:**
  - **some jobs need all files before processing**
  - **some can start processing with a subset of the requested files**
  - **algorithm will need some adjustment**
- **Use as a policy module of a production system of the Storage Resource Manager (SRM) developed at Lawrence Berkeley Laboratory.**

The End

- **Thank You**