

Cooperative Caching in Linux Clusters



Ying Xu

Brett D. Fleisch

Department of Computer Science and Engineering

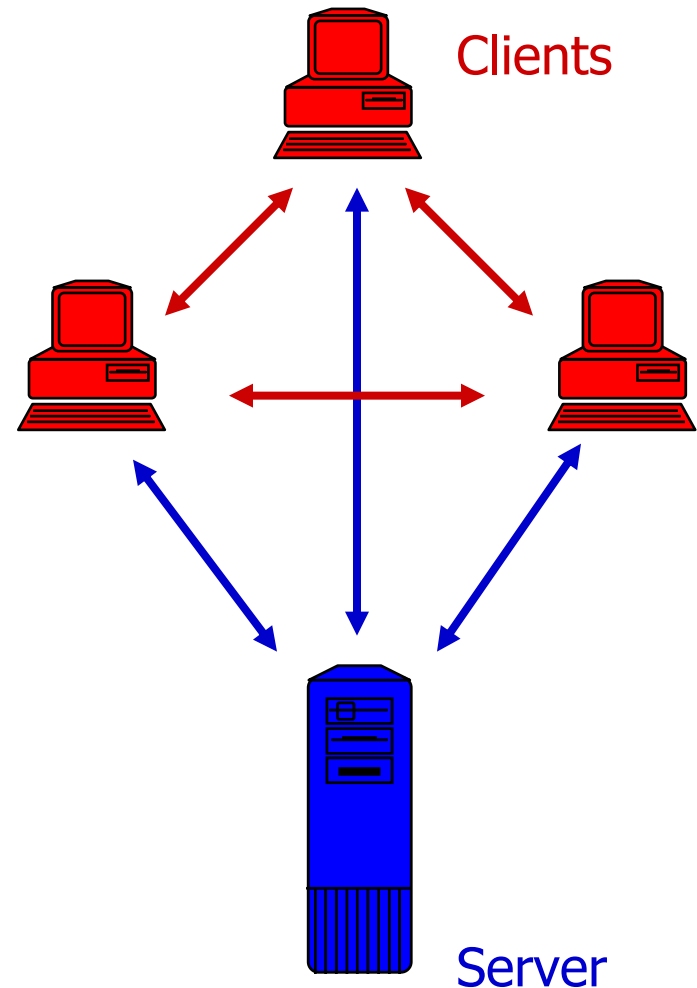
University of California, Riverside

{yxu,brett}@cs.ucr.edu

June 25, 2003

Cooperative Caching

- Cooperative caching enables clients to read file blocks from the caches of peers
- The key idea is to exploit aggregate resources of clients
 - Memory
 - Bandwidth





Motivation

- High-bandwidth and low latency networks
 - Reading data from remote memory is an order of magnitude faster than reading data from local disk
- Large Memories
 - Memory prices continue to decline
- Server Bottleneck
 - Clients can easily saturate the bandwidth of server in cluster environments



Comparison with Related Work

- [Dahlin94] Cooperative Caching: Using Remote Client Memory to Improve File System Performance
- [Feeley95] Implementing Global Memory Management in a Workstation Cluster
- [Sarkar00] Hint-Based Cooperative Caching
- Oracle's Cache Fusion



Outline

- Design Decisions
- Cooperative Caching Scheme
- Prototype and Performance Results
- Future work



Remote Memory Properties

- Speed
 - Faster than local disk
- Dynamic Size
 - Size and distribution are changing
- Unreliability
 - Built with commodity components



Design Decisions (1)

- File caching
 - Avoid expensive disk accesses
 - Improve system throughput
- Read
 - Read-only caching, only caches clean data
 - We extend read policy of file caching
- Write
 - All modified data blocks are still sent to the server as they otherwise without cooperative caching



Design Decisions (2)

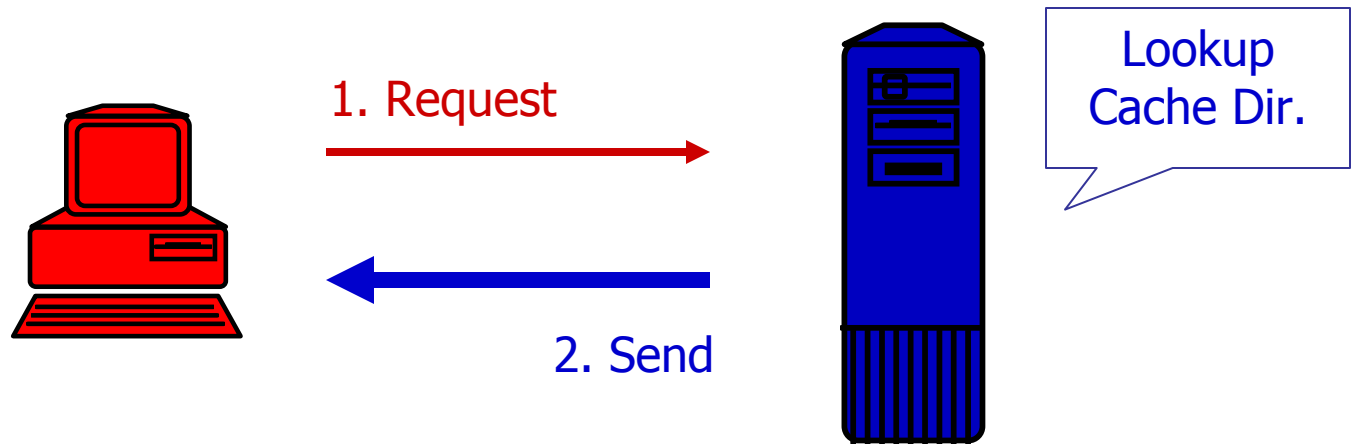
- Cache Directory
 - A data structure maintained by the server to track the data blocks cached by each client
- Hints
 - Clients maintain some hints to further reduce the number of messages sent to the server.



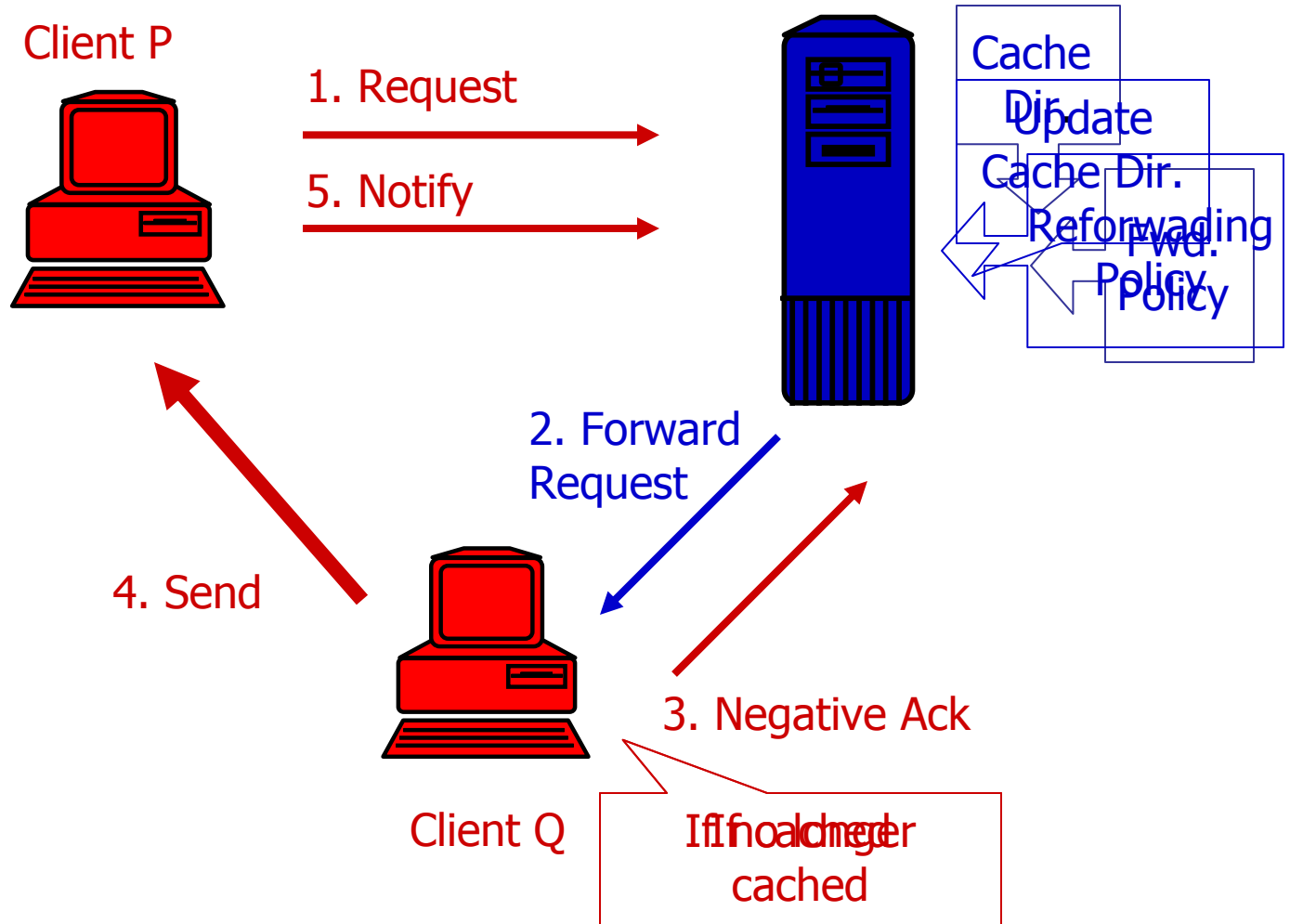
Consistency Model

- Maintaining consistency is the responsibility of network file systems
- Strict Consistency — guarantees a consistent view of data
 - Sprite
- Time-based Consistency — allows inconsistency within a small time window
 - NFS

Cooperative Caching Scheme (1)

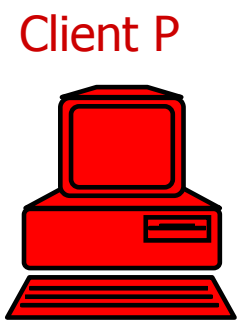


Cooperative Caching Scheme (2)



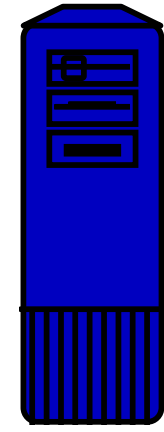
Cooperative Caching Scheme (3)

If hints indicate Q caches the data block



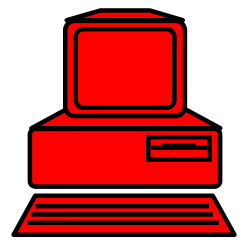
Client P

3. Notify
3. Request



1. Request

Send Data Block and Attach Negative Ack
2. Send



Client Q

If no cache
cached



The Server

- Maintains global information in its cache directory
- By coordinating the file caches, clients read large file blocks from their peers in most cases while the server only sends and receives small messages.
- The extra work added to the server is maintaining and searching its cache directory



Clients

- Added clients can be leveraged to serve the requests of other clients
- Maintain hints information to further reduce the messages sent to the server

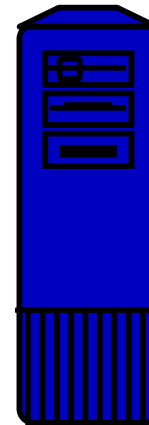
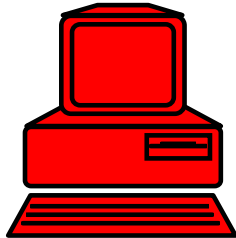


The Accuracy of Cache Directory

- Problem: A request has been forwarded but the client no longer caches the data block due to its local page replacement
- Clients notify the server when they discard data blocks
 - Don't send a message for each discarded data block. Do it in batches
 - On-demand update
- Tradeoff: the accuracy of information vs. the overhead of distributing and maintaining that information

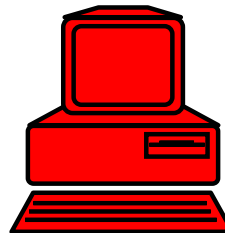
Page Discard Notification

Client P



Update
Cache Dir.

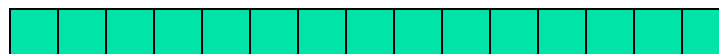
Notify



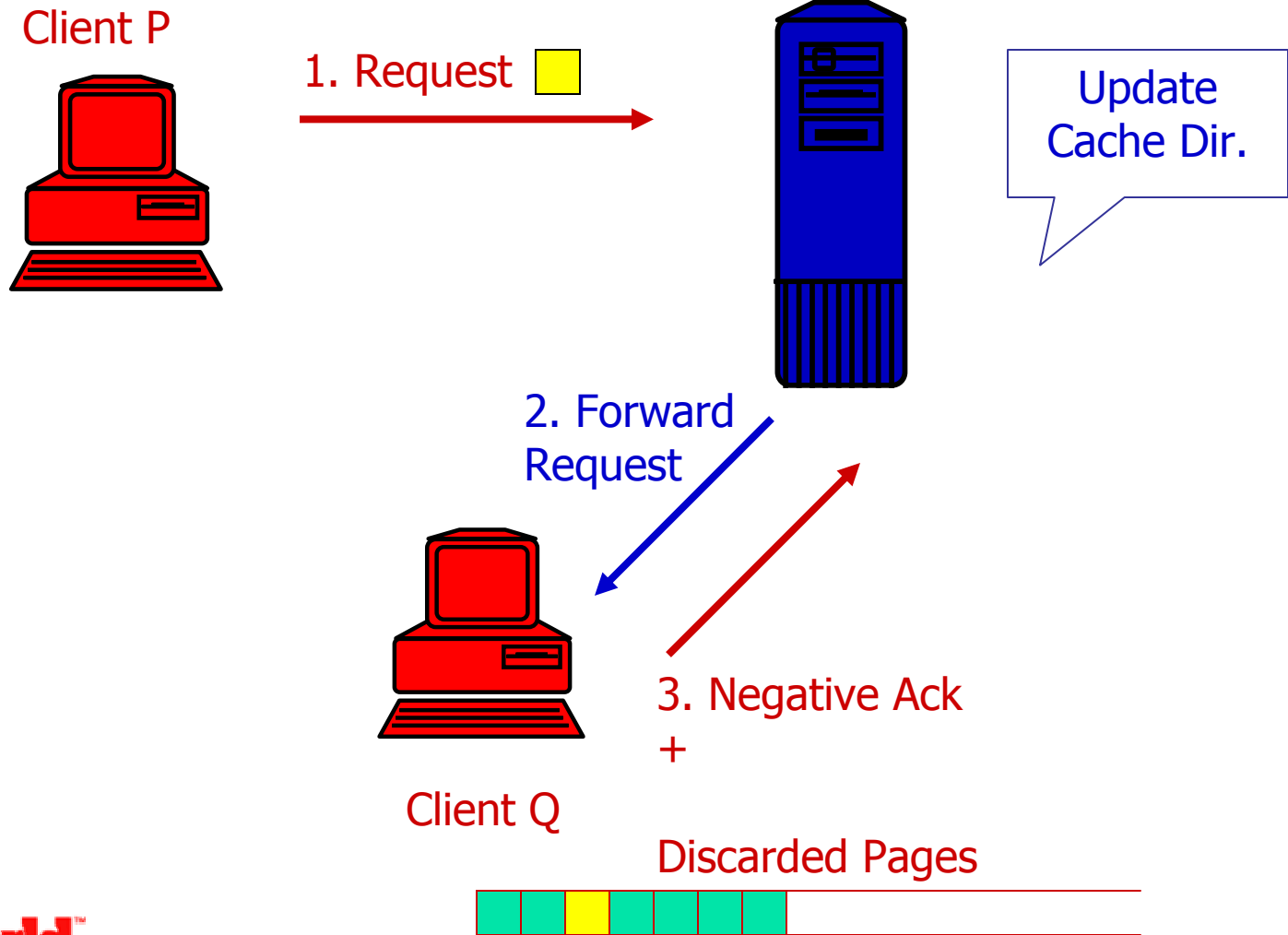
Client Q

When the
queue becomes
full

Discarded Pages



On-Demand Update





Prototype — NFS-cc

- NFS-cc is implemented in Linux kernel 2.4.20
- We extend RPC protocol to support message forwarding
- NFS Daemons
 - Server-side daemon
 - Client-side daemon
- Cache Directory
 - A hash table indexed by a pair of inode number and block number
 - Each hash table entry includes a bitmap indicating which clients cache the data block



Experiment Setup

- 12 Clients
 - 400 MHz Celeron with 128M RAM
- 1 Server
 - Dell Dimension 4500, 2 GHz P4 with 640M RAM
- Connection
 - 100Mb Fast Ethernet
 - 16-Port Linksys switch, full duplex

Overhead

We use a modified version of Andrew benchmark to measure the overhead of adding cooperative caching to NFS

	NFS 1 (ms)	NFS-cc (ms)	NFS 2 (ms)
Phase 1: Create Directory	86	89	86
Phase 2: Copy Data	2628	2663	2764
Phase 3: Intensive Stat()ing	4880	4871	4843
Phase 4: Grep	6255	6269	6520
Phase 5: Compile	140218	141087	141073

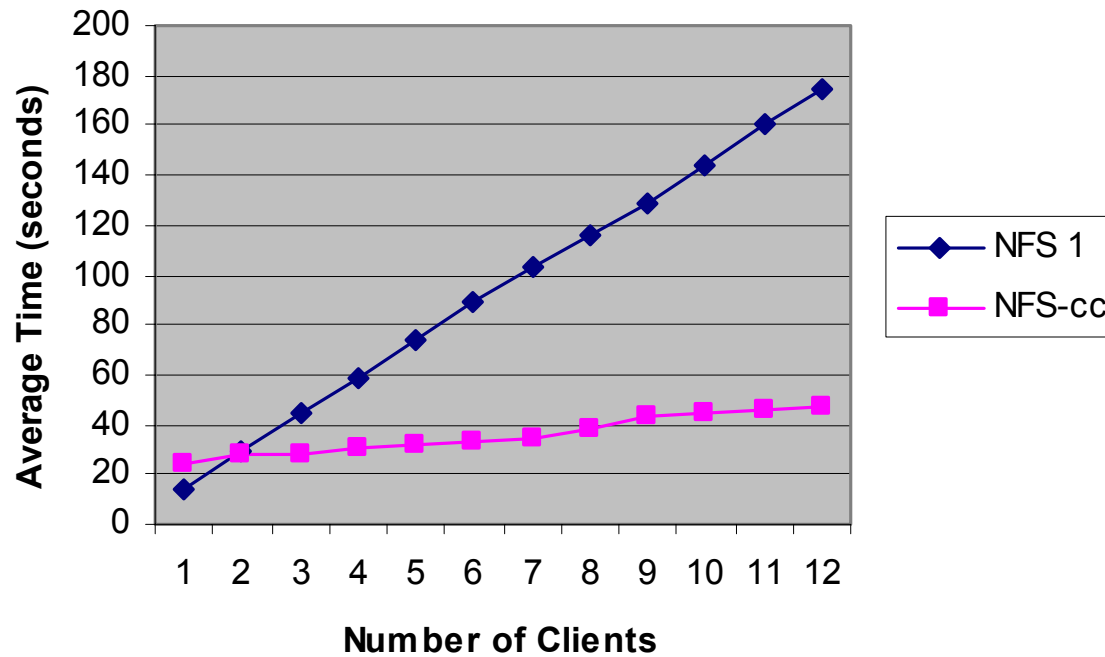
- NFS 1: Linux kernel 2.4.20
- NFS 2: Red Hat Linux kernel 2.4.18
- NFS-cc: Our prototype

The overhead of NFS-cc is less than 4% and even less than the difference between two versions of Linux kernel in some phrases

Concurrent Copy

Clients copy the same file concurrently

The average time of copying a large file (166.9MB)



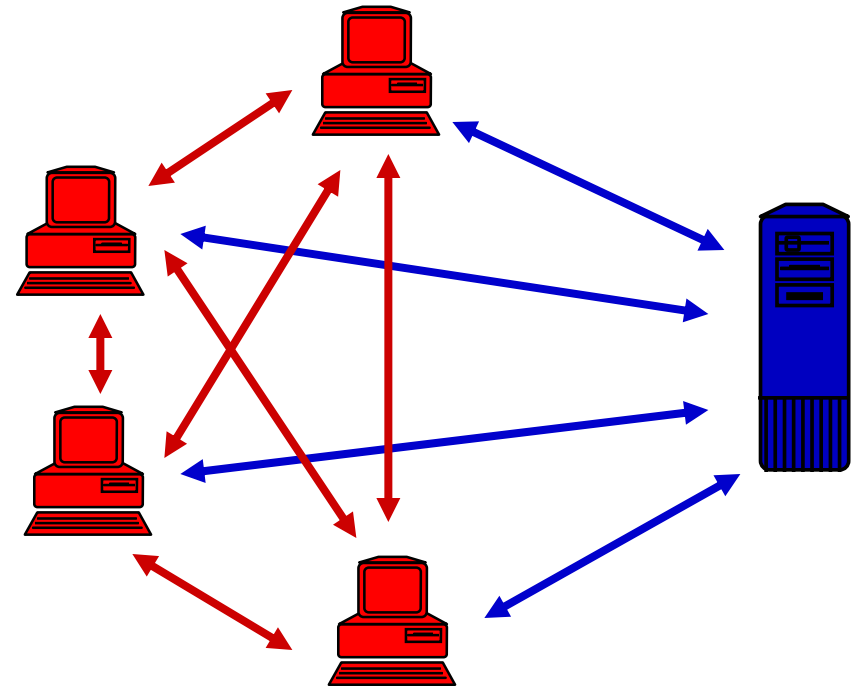


Application

- Not every application will benefit
- Parallel applications processing the same data concurrently will benefit greatly
 - Sharing of Data
 - Freq. of WRITES low
 - Freq. of READs high

Potential Applications

- Data intensive clusters
 - Data Mining
 - Multimedia apps
- We are looking for real world apps





Forwarding Policies

- The server always tries to forward requests to sites that the cache directory suggests has the data block
- Round Robin
 - Server simply forwards requests to clients one by one according to its cache directory
- Balanced
 - Server tries to forward requests to clients evenly
- Adaptive
 - Server forwards requests to clients according to their current status, such as CPU usage, unused bandwidth, idle memory



Remote Paging

- Efficient use of cluster-wide memory
 - Active clients can use the idle memory of other clients
- N-Chance forwarding [[Dahlin94](#)]
 - Avoids discarding singlet
 - Forwards singlet to random peers
 - Assigns every singlet a recirculation count N
- Global Memory Service [[Feeley95](#)]
 - A global aging algorithm
- Our Remote Page Replacement Algorithm
 - Use the information in cache directory