Cooperative Caching

Ameet Kini

February 25, 2004

1 Summary

Big Idea: coordinate file caches of machines in the cluster distributed on a LAN to form a more effective overall file cache

1.1 Motivations

1. Processor performance is progressing at a higher rate than disk performance. Therefore, reduce disk I/O as far as possible
2. Networks are getting really fast. Therefore, it may be faster to access a block across such a hi-speed network than fetch it from disk.

1.2 Goals of Study

1. Evaluate coop-caching by simulation using traces of real workloads
2. Find a practical coop-caching algorithm from a range of algorithms

2 Four Algorithms

2.1 Direct Client Cooperation

Extend client’s cache by using an idle client’s memory. Active client can use remote client’s memory to satisfy read requests until remote machine becomes active.

Pros - simple to implement, server can be kept out of the loop
Cons - No coordination between clients - if neither I nor my peer has this block, I will go to server even if one of the other clients has it in cache.
2.2 Greedy Forwarding

Simple extension to above algorithm: If client does not have this block, ask the server. If server has it in its memory, fine, but if not, it checks to see if some other client has it cached. If so, the block is forwarded directly to the requesting client without server intermission. Its greedy in the sense that each client manages its local cache greedily without regard to potential needs of other clients.

Pros - Being greedy, its simple to implement; autonomy in local cache management
Cons - Lack of coordination may lead to unnecessary duplication in cache contents

2.3 Centrally Coordinated Caching

Static partitioning of client cache into client-managed and server-managed. Server-managed portion is managed centrally (duh?). This scheme is like moving physical memory from clients to servers. Evicted blocks from server cache lands up in one of these client caches.

Pros - high global hit rate; simple to implement due to central management
Cons - clients suffer due to a smaller local cache resulting in lower local hit rate; increased load on server due to central management

2.4 N-Chance Forwarding

Last algo is usually the best, so here it is: if a block is stored in only one client cache (a.k.a singlet), then when its time to evict it, simply forward it to a random peer instead of discarding it. Blocks float around a maximum of n times before being discarded. Of course, if its referenced, its recirculation count gets resetted to n.

Pros - Dynamic tradeoff between local data and global data. Active clients have more local data in their cache whereas idle clients accumulate more global data over time. No time is spent looking for an idle client since its found randomly.
Cons - Slightly more complicated to keep track of usage of blocks due to server intervention in answering queries such as 'is this block the last cached copy in the system?'
3 Experiments and Results

Traces are from the Sprite workload - 42 client machines and 1 server over a two day period. Over 700,000 read and write block accesses. Experiments are basically comparison of above four algos plus a Base and Best algo. In Base there's no coop-caching between client and server caches. In Best, global hit rate as high as if all client memory managed as global cache yet local hit rate still remains high as if all client memory managed as private local cache. This is a really neat way to characterize the best and worst cases.

Summary of results -
1. N-chance comes within 10% of Best case (also closest to it) in average response time for a block read.
2. N-chance also pays a very small price for recirculation by having its local miss rate at 23% vs. Central Coordination at 36%.
3. As n gets larger (more than 2), response time plateaus out

Experiments conducted using another trace, Berkeley Auspex, confims result 1.

4 Comments

1. Memory is cheap. According to Jim Gray’s 'Rules of Thumb in Data Engineering', in 1999, RAM costs $1 / MB and cost decreases 100x in a decade. Why not invest in larger memory than deal with complicated caching algorithms?
   Possible reasons: While memory cost reduces, datasets increase in size. Scientific datasets have now reached the petabyte mark. Also, data is commonly distributed as its simply not possible to replicate entire datasets on every node in the cluster. Therefore, some type of sharing is needed, and coop caching is a step in the right direction. Finally, its much more expensive to buy 1 TB of server RAM than upgrade all 1000 clients to 1 GB of RAM.

2. Why hasn’t coop caching found its way into commercial systems?
   Other forms of caching has been found to be as effective if not more (e.g. LARD). DB researchers (couldnt resist this one :) advocate partitioning the DB row-wise or column-wise onto multiple servers, a form of LARD.