CLIC
CLient-Informed Caching for Storage Servers

Xin Liu  Ashraf Aboulnaga  Ken Salem  Xuhui Li

David R. Cheriton School of Computer Science
University of Waterloo

February 2009
Two-Tier Caching

Problems:
• cache inclusion
• poor temporal locality

One Solution:
• hinting
Two-Tier Caching

1. read(p)

Problems:
• cache inclusion
• poor temporal locality

One Solution:
• hinting
Two-Tier Caching

1. read(p)

2. read(p)

Problems:
• cache inclusion
• poor temporal locality

One Solution:
• hinting
Two-Tier Caching

1. read(p)
2. read(p)
3. fetch

Problems:
• cache inclusion
• poor temporal locality

One Solution:
• hinting
Two-Tier Caching

DBMS

1. read(p)

2. read(p) 3. fetch

storage server

4. fetch

Problems:
• cache inclusion
• poor temporal locality

One Solution:
• hinting
Two-Tier Caching

DBMS

storage server

cache

p

1. read(p)  4. fetch

2. read(p)  3. fetch

Problems:
- cache inclusion
Two-Tier Caching

Problems:
- cache inclusion
- poor temporal locality
Two-Tier Caching

Problems:
- cache inclusion
- poor temporal locality

One Solution:
- hinting
Example: Write Hints

The storage server can use TQ, an ad hoc hint-aware replacement policy, to exploit write hints.
The storage server can use TQ, an ad hoc hint-aware replacement policy, to exploit write hints.
Example: Write Hints

The storage server can use TQ, an ad hoc hint-aware replacement policy, to exploit write hints.

- This is a good candidate for caching.
The storage server can use TQ, an ad hoc hint-aware replacement policy, to exploit write hints.
Problems with Ad Hoc Hint-Aware Policies

narrowness: new hints? multiple hints?
Problems with Ad Hoc Hint-Aware Policies

narrowness: new hints? multiple hints?
brittleness: correct response to hints?
Problems with Ad Hoc Hint-Aware Policies

narrowness: new hints? multiple hints?

brittleness: correct response to hints?

single source: multiple hint generators?
The CLIC Approach

- a hint-aware caching policy for 2nd-tier caches
The CLIC Approach

- a hint-aware caching policy for 2nd-tier caches
- no hard coded response to specific hints
The CLIC Approach

- a hint-aware caching policy for 2nd-tier caches
- no hard coded response to specific hints
- instead, learn which hints signal good caching opportunities
The CLIC Approach

- a hint-aware caching policy for 2nd-tier caches
- no hard coded response to specific hints
- instead, learn which hints signal good caching opportunities
- benefits:
  - handles multiple hint types
  - handles new hint types
  - handles hints from multiple clients by treating each client’s hints as distinct
The CLIC Approach

- a hint-aware caching policy for 2nd-tier caches
- no hard coded response to specific hints
- instead, learn which hints signal good caching opportunities
- benefits:
  - handles multiple hint types
  - handles new hint types
  - handles hints from multiple clients by treating each client’s hints as distinct

CLIC Hints

CLIC separates the generation of hints (done by the storage clients) from the interpretation of those hints for caching purposes (done by the storage server).
DBMS

storage server

read(p)

I don’t know blargh or gorp but previous blargh gorp reads have been good candidates, so I will cache p

this is a blargh gorp read

cache

p
Generating Hints

- Storage client must be modified to generate one or more types of hints.
- Storage clients attach a hint set to each read or write request. A hint set includes one hint of each type generated by the client.
- A storage client may choose to generate any types of hints that might be of use to the storage server.
Generating Hints

- Storage client must be modified to generate one or more types of hints.
- Storage clients attach a hint set to each read or write request. A hint set includes one hint of each type generated by the client.
- A storage client may choose to generate any types of hints that might be of use to the storage server.

Example: Hints from DB2

- buffer pool ID
- object ID: identifies a group of related DB objects
- object type ID: distinguishes table from index
- request type: read, replacement/recovery write
- DB2 buffer priority
A CLIC-Managed Cache

- each page is associated with the hint set which it was most-recently read or written
A CLIC-Managed Cache

- each page is associated with the hint set which it was most-recently read or written
- each hint set has a priority
A CLIC-Managed Cache

- each page is associated with the hint set which it was most-recently read or written
- each hint set has a priority
- CLIC evicts pages associated with the lowest-priority hint sets
A CLIC-Managed Cache

- each page is associated with the hint set which it was most-recently read or written
- each hint set has a priority
- CLIC evicts pages associated with the lowest-priority hint sets
- CLIC chooses hint set priorities using a simple cost/benefit analysis
Cost/Benefit Analysis

- There is a benefit to caching if the next request for \( p \) is a read request.
- The cost of obtaining this benefit is that \( p \) must remain cached until the read request.
Cost/Benefit Analysis

- There is a benefit to caching if the next request for \( p \) is a read request.

\( \text{cache } p \text{ here??} \)

\( (p,H) \)

read or write request

\( \text{next request for } p \)

is this a read request?

time
Cost/Benefit Analysis

There is a benefit to caching if the next request for $p$ is a read request.

The cost of obtaining this benefit is that $p$ must remain cached until the read request.
Assigning Priorities to Hint Sets

- When request \((p, H)\) occurs, CLIC cannot know the cost and benefit of caching \(p\).
Assigning Priorities to Hint Sets

- when request \((p, H)\) occurs, CLIC cannot know the cost and benefit of caching \(p\)
- instead CLIC estimates the cost and benefit of caching \(p\) at \((p, H)\) based on previous requests with hint set \(H\)
Assigning Priorities to Hint Sets

- when request \((p, H)\) occurs, CLIC cannot know the cost and benefit of caching \(p\)
- instead CLIC estimates the cost and benefit of caching \(p\) at \((p, H)\) based on previous requests with hint set \(H\)
- CLIC assigns a priority to each hint set based on the cost and benefit of previous requests with hint set \(H\)

\[
\text{Priority}(H) = \frac{\text{Read Hit Rate}(H)}{\text{Mean Time Until Read Hit}(H)}
\]
DB2 Hint Analysis Example

The chart illustrates the relationship between the frequency and priority of database operations. The x-axis represents the frequency (log scale), and the y-axis represents the priority (log scale). The chart shows two distinct clusters:

1. **STOCK table replacement writes**: These operations have higher priority and occur less frequently.
2. **ORDERLINE table reads**: These operations have lower priority but occur more frequently.

The chart helps in understanding the performance implications of different operations within the DB2 environment.
Efficient Hint Analysis

- To analyze the cost and benefit of hint sets, CLIC must
  - track the most recent request and hint set for each page
  - track the mean read hit rate and read hit distance for each hint set

We have also investigated the use of generalization to reduce the number of distinct hint sets.
Efficient Hint Analysis

- To analyze the cost and benefit of hint sets, CLIC must
  - track the most recent request and hint set for each page
  - track the mean read hit rate and read hit distance for each hint set
- To reduce space requirements, CLIC
  - tracks the most recent request only for cached pages and a fixed number of additional, uncached pages
Efficient Hint Analysis

- To analyze the cost and benefit of hint sets, CLIC must
  - track the most recent request and hint set for each page
  - track the mean read hit rate and read hit distance for each hint set
- To reduce space requirements, CLIC
  - tracks the most recent request only for cached pages and a fixed number of additional, uncached paged
  - tracks read hit statistics only for frequently occurring hint sets

We have also investigated the use of generalization to reduce the number of distinct hint sets.
Efficient Hint Analysis

To analyze the cost and benefit of hint sets, CLIC must
- track the most recent request and hint set for each page
- track the mean read hit rate and read hit distance for each hint set

To reduce space requirements, CLIC
- tracks the most recent request only for cached pages and a fixed number of additional, uncached pages
- tracks read hit statistics only for frequently occurring hint sets

We have also investigated the use of generalization to reduce the number of distinct hint sets.
we have used trace-driven simulation of the storage server buffer cache to compare CLIC to other replacement policies

methodology

1. modify DB2 and MySQL to generate hints and produce I/O traces
2. run TPC-C (on-line transaction processing) and TPC-H (decision support) workloads on the database systems and collect I/O traces
3. feed the traces to a simulation of second-tier cache, which implements CLIC, LRU, ARC, TQ and OPT
4. measure the hit ratio achieved by different policies.
DB2 TPC-C - Medium DB2 Buffer Cache

![Graph showing Server Cache Read Hit Ratio for different cache sizes and cache policies.]

- OPT
- TQ
- LRU
- ARC
- CLIC
DB2 TPC-H - Medium DB2 Buffer Cache

Server Cache Size (pages)

Server Cache Read Hit Ratio

DB2_H400

OPT
TQ
LRU
ARC
CLIC
DB2 TPC-C - Small DB2 Buffer Cache

![Graph showing Server Cache Read Hit Ratio vs. Server Cache Size (pages) for different cache management policies: OPT, TQ, LRU, ARC, CLIC. The graph illustrates how the hit ratio increases with cache size and varies among the different policies.]
DB2 TPC-C - Large DB2 Buffer Cache

![Graph showing Server Cache Read Hit Ratio for DB2_C540 across different cache sizes and hinting algorithms.](image)
Summary and Conclusions

- CLIC learns to identify I/O hints that signal good caching opportunities by tracking the request stream observed by the second-tier cache.
- Because CLIC’s responses to specific hints are not predefined, it naturally accommodates new hint types and hints from multiple storage clients.
- For our traces:
  - CLIC’s performance usually dominates ARC’s and LRU’s, sometimes by a factor of 2 or more.
  - CLIC dominates the ad hoc, hint-aware TQ algorithm.
  - CLIC’s space overhead can be kept low (1% of storage server cache size in our experiments).