CMSC 818S, 03 March 1998

Semantic Data Caching and Replacement

Dar, Franklin, Jonsson, Srivastava and Tan

Appears in Proceedings of the 22nd VLDB Conference

Semantic Data Caching and Replacement

Support associative access in data-shipping architectures

Associative access is traditionally supported in pure query-shipping architectures

Semantic caching & replacement can:

- Exploit client resources (disk, memory)
- Exploit semantic knowledge (constraint formula/query predicate) to manage cache

Semantic Data Caching and Replacement(cont'd)

-Data-shipping architectures (ODBMS)

+ client can cache to disk or memory

+ servers primarily service faults from disks

-Client caches are managed using techniques similar to a buffer manager

+ granularity (page or tuple-based)

+ method of access (index or full scan)

+ replacement policy (locality-based on access history: LRU/MRU)

Semantic Caching Model

Key ideas of semantic caching model:

Client maintains semantic description of cache contents (vs. page/tuple list)

-query uses semantic information to determine location (client/server)

-*remainder* queries are used to retrieve data not found in cache

Cache is dynamically managed as a set of "semantic regions"

More elaborate value functions for cache replacement can be computed

Architectures for Cache Management

Semantic caching is evaluated by comparing to traditional page- and tuple-based architectures:

Primary dimensions for comparison (associative query processing):

-Data granularity (of concern in data-shipping)

+ client-server transfer

+ consistency maintenance (updates)

+ cache management: pages (static), tuples, semantic (dynamic)

Architectures for Cache Management (cont'd)

-Remainder queries vs. faulting

+ page caching: fault-driven
+ tuple caching: similar to page, but tuples are grouped for xfer
+ semantic: uses remainder queries (small msgs/parallelism)

-Cache replacement policy

+ page-caching clusters "close" tuples - spatial locality
+ tuple-caching could be either spatial or temporal
+ semantic-caching introduces notion of "semantic locality"; dynamically adopts to query access pattern

Page caching architectures (page servers)

Queries are posed at clients

Servers: fault in missing page from disk & sends data to clients

Clients:

- -Cache in memory or on disk
- -Indexed scans: perform partial scans on indexes (faulting in index pages)
- -Non-indexed scans: scan full relation, faulting in missing pages
- -Manages cache using LRU or MRU policy

Tuple caching architecture

-Cache maintained as list of tuples (or objects)

-Allows much more flexibility - tuning to access locality of query

-Suffers from high maintenance (overhead) & small messages (on faults)

-Indexed scans: similar to page-caching

-Non-indexed scan:

+ local scan of cache & send tuple list/constraint to server

+ ignore cache; server does scan

-LRU is used to manage the cache; temporal locality is exploited

Semantic caching architecture

-Cache managed as a collection of ”semantic regions”

-Semantic regions allow dynamic grouping of tuples (not static)

-Semantic region includes:

+ contraint formula, tuple count, list of cached tuples, replacement data

-Query execution:

+ query is split into a *probe query* and a *remainder query*

+ a non-null *remainder query* is sent to & processed at server

Concepts of semantic caching model

-Exploits semantic info in associative query specifications

-Selection queries on single relations

+ selection condition: arbitrary constraint formula

-Cached portion of relation is described by constraint formula; one per relation

Concepts of semantic caching model (cont'd)

-Query execution:

probe query: defined as $P(Q, V) = Q \wedge V$ on relation R

remainder query: defined as $R(Q, V) = Q \wedge \neg V$ on relation R

where, Q: constraint formula on relation R and V: contraint formula describing cached tuples of R

-Constraint formulas can be visualized as sub-spaces in n-dim space

Managing a Semantic Cache

-Cache is managed at the granularity of a "semantic region"

-Intersecting query splits a semantic region into 2 disjoint regions

one is the intersection of query & region

the other is the difference w.r.t query

-Cached query data returned by $R(Q, V)$ also forms a semantic region

Managing a Semantic Cache

-How are these semantic regions managed? Three policies:

a) Coalesce two regions with same replacement value

b) Never coalesce: good policy for large regions, not for small

c) Heuristic: coalesce two regions as in (a), but only if either is $< 1\%$ of cache size

Semantic Caching Replacement Issues

-Replacement policy targets regions w/ lowest ”semantic value”

-Policies

1) temporal: recency of usage (LRU, MRU)

2) semantic distance (approx spatial locality?): Manhattan distance

Simulation Environment & Experiments

-Simulator (based on CSIM)

+ models a peer-to-peer database; configured for client-server

+ detailed disk model used

-Scan at server uses read-ahead (overlap I/O & communication); not for faulting

-Buffer management (cache size: 250MB)

+ for page-caching, cache holds $\approx$ 60 pages

+ for tuple-caching, cache holds $\approx$ 1085 tuples

+ for semantic-caching, overhead (per region) + data set size $<$ cache size

Workload

-Simple selections, in which query size varies (< cache size)

-Fixed # queries with semantic centerpoint (skew)

-Remaining queries are uniformly distributed

-Relation has three candidate keys adopted from Wisconsin benchmark:

+ *Unique1*: indexed, unclustered

+ *Unique2*: indexed, clustered

+ *Unique3*: unindexed, unclustered

Results for Indexed Selections

-Primary metric: response time

-Indexed selections: single attribute, double attribute

-Client memory used for caching

-Results for *Unique2*

-General: response time increases as query-size increases (cache size fixed) due to poor cache hit rates

+ tuple/LRU: high maintenance, less tuples fit in cache

+ page/LRU: good locality, lower maintenance

+ semantic/LRU : cache utilization falls due to large regions & holes

Results for Indexed Selections (cont'd)

-Results for *Unique1*

+ page/(LRU,MRU): both suffer due unclustered data;

+ tuple/LRU: can adapt better to unclustered access pattern

+ semantic: dynamic grouping offsets unclustered data; lower overhead

Results for Indexed Selections (cont'd)

-For double attribute selection, both response time & overhead are measured

-Response time for selection on *Unique1*/*Unique2*:

+ page/(LRU,MRU): same as *Unique1* results

+ tuple/semantic: can adapt to access pattern

Results for Indexed Selections (cont'd)

-Overhead (% cache):

+ page/LRU incurs smaller overhead w.r.t. tuple/LRU

(both proportional to cache size, not query size)

+ semantic is proportional to query size & dependent on coalescing strategy

-Effects of multiple dimensional attribute space:

+ small queries for double attr selections, overhead is 4x greater than single attr selection

+ claim is: coalescing heuristic minimizes overhead explosion

Results for Non-indexed Selections

-Note: page is not included since it suffers from static clustering of tuples

-Client uses disk for caching, which semantic can take advantage of:

-process probe & remainder query in parallel

-Tuple/LRU: caching must scan local disk before retrieving missing items

-Tuple/Ignore: better since avoids scan of local disk, and server can overlap I/O & communication

-Unexpected benefit: semantic caching does well for small queries

Semantic Value Function

-Using "semantic locality" for replacement value functions

-Selection on unclustered, indexed attribute *Unique1*

-Use Manhattan distance as "locality" measure

-Results:

+ Manhattan distance is more effective at keeping a "hot" region in memory

+ Because of queries uniformly distributed over cold region, such data stay on LRU chain

Mobile Navigation Application

-Contains more semantic content than selection-based workloads

-Models mobile clients accessing remotely stored map data via low BW channel

-Each tuple represents a road segment

-Updates a display of current position as user move in 2-dimensions

Application Workload

-One relation, two attributes: $x, y \in \{0, \ldots, 8191\}$ form a dense key

-Clustered on disk using Z-ordering on attributes

-Simple selections:

+ rectangular 8X16 query; 128 tuples for answer

+ location/orientation w.r.t to current position

+ query path: navigation through 2-dimensional space in Manhattan fashion

Application Workload (cont'd)

-Query profiles (paths) simulated:

+ random: equal probability in moving in any four directions

+ square: fixed sqaure in 2-dim space

+ Manhattan "lollipop": up stick, around square, down stick & repeat

Semantic Value Function

-Define: *directional Manhattan* distance function

-Based on Manhattan distance from user's current location and direction of motion

Performance Results

-Performance metric: response time

-Replacement policies used for comparison: tuple/LRU & tuple/MRU

-Query profiles included big & small loops

+ random paths produced small loops (policy: LRU)

+ larger, more regular loops (polic: MRU)

+ sematic distance performs more rebustly across a range of loops

Performance Results (cont'd)

-Random paths: semantic distance is slightly better than LRU; slightly better than MRU for 0.8/0.1/0.1/0.0 path

-Small squares: semantic is competitive with MRU

-Larger squares: similar to LRU which are both outperformed by MRU

-Lollipop: semantic distance outperforms LRU & MRU when square is traversed many times

Conclusions

-Semantic caching maintains replacement information on a per-region basis

+ dynamically adjusts to current query access patterns

-Remainder queries are used to reduce network communication

-Allows the use of more sophisticated value functions (semantic locality)