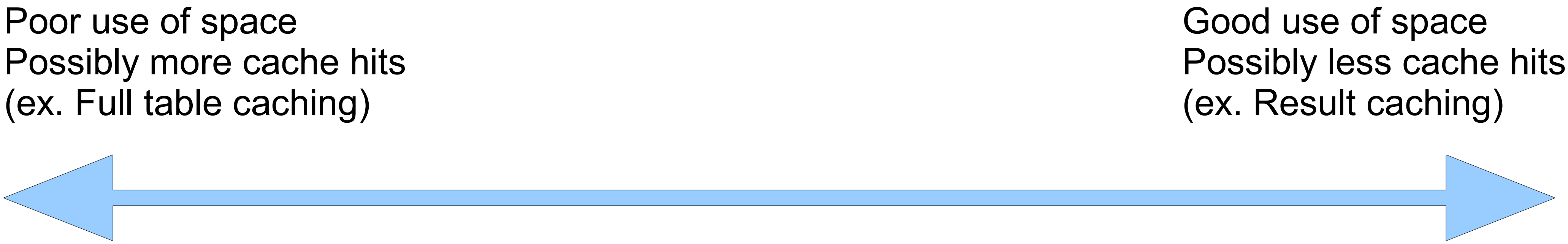
A Cache Management Strategy

CS 848: Course Project
Fall, 2006
University of Waterloo
Jeff Pound

Caching trade-off



Poor use of space
Possibly more cache hits
(ex. Full table caching)

Good use of space
Possibly less cache hits
(ex. Result caching)

- Storing too much data can be a waste of cache space, but can yield a high cache hit rate
- Not storing enough data can mean a low cache hit rate, but can be a better use of limited space
- We also have to address the problem of ***when*** a query can be answered from the cache

Proposed Strategy

- *Weaken* the query sent to the back-end DB to cache a ***superset*** of the results for the given query

Proposed Strategy

- *Weaken* the query sent to the back-end DB to cache a ***superset*** of the results for the given query
  - Do this in such a way that: we can avoid computing query containment, yet still have an easy way to determine when to use the cache and,
  - The extra cached data is semantically related to the query that caused the caching

Proposed Strategy

- Re-write queries (cache misses) with conjunctive predicates in the *where* clause as disjunctions (excluding join-conditions) and cache the results
  - This makes every predicate disjoint from every other predicate that built the cache (a notion of “domain completeness” on every key/value pair)
  - (The results specified by any one predicate are not further restricted by any other predicate)

Proposed Strategy

- Approximating query containment is now a matter of *n* constant time table look-ups (where *n* is the number of disjoint predicates in the *where* clause of the incoming query)
  - (we need to satisfy one predicate in each conjunctive clause, and every predicate in each disjunctive clause)
- No need to process *probe queries* at the cache
- Cache eviction can be done on a per-predicate basis (Evict predicate *p* by deleting tuples matching $\forall q \left(q \in predicateList , p \neq q , p \wedge \neg q \right)$

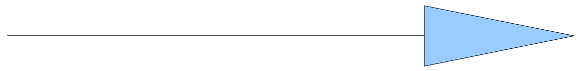
Proposed Project

- Implement and evaluate the specified cache management strategy
  - Compare against baseline strategies of full table caching, and query result caching
  - Measure comparative performance in various caching situations (cache miss, cache hit, eviction, etc...)
  - Cache hit rate is only meaning full for real workloads, of which I don't have :(

Example

$$\pi_{a,b}(\sigma_{A \wedge B \wedge C}(R))$$

$$\sigma_{A \vee B \vee C}(R)$$



Cache

R:{A,B,C}

Example

$$\pi_{a,b}(\sigma_{A\wedge B\wedge C}(R))$$

$$\sigma_{A\vee B\vee C}(R) \longrightarrow$$

Cache
R:{A,B,C}

$$\pi_{\ldots}(\sigma_{A}(R))$$

Hit

Example

$$\pi_{a,b}(\sigma_{A\wedge B\wedge C}(R))$$

$$\sigma_{A\vee B\vee C}(R) \longrightarrow$$

Cache
R:{A,B,C}

$\pi_{\dots}(\sigma_{A}(R))$ Hit

$\pi_{\dots}(\sigma_{A\wedge D}(R))$ Hit

Example

$$\pi_{a,b}(\sigma_{A\wedge B\wedge C}(R))$$

$\sigma_{A\vee B\vee C}(R)$ → Cache
R:{A,B,C}

$\pi_{\dots}(\sigma_{A}(R))$ Hit

$\pi_{\dots}(\sigma_{A\wedge D}(R))$ Hit

$\pi_{\dots}(\sigma_{A\wedge D\wedge \dots \wedge \dots}(R))$ Hit

Example

$\pi_{a,b}(\sigma_{A\wedge B\wedge C}(R))$

$\sigma_{A\vee B\vee C}(R)$ $\longrightarrow$ <u>Cache</u>
R:{A,B,C}

$\pi_{\dots}(\sigma_{A}(R))$ Hit

$\pi_{\dots}(\sigma_{A\wedge D}(R))$ Hit

$\pi_{\dots}(\sigma_{A\wedge D\wedge \dots \wedge \dots}(R))$ Hit

$\pi_{\dots}(\sigma_{A\vee D}(R))$ Miss

$\sigma_{A\vee D}(R)$ $\longrightarrow$ <u>Cache</u>
R:{A,B,C,D}

Questions/Comments?

?