Query Result Caching in Data Warehouses and Data Marts

Krithi Ramamritham
Prasan Roy
S. Seshadri
S. Sudarshan
Jinesh Vora
Hoshi Mistry
Amit Gupta

Ongoing project at
Computer Science and Engineering Dept, I.I.T. Bombay

Introduction

- Problem Definition
  - Given a sequence of queries, dynamically select query results to be cached, so as to reduce overall execution time

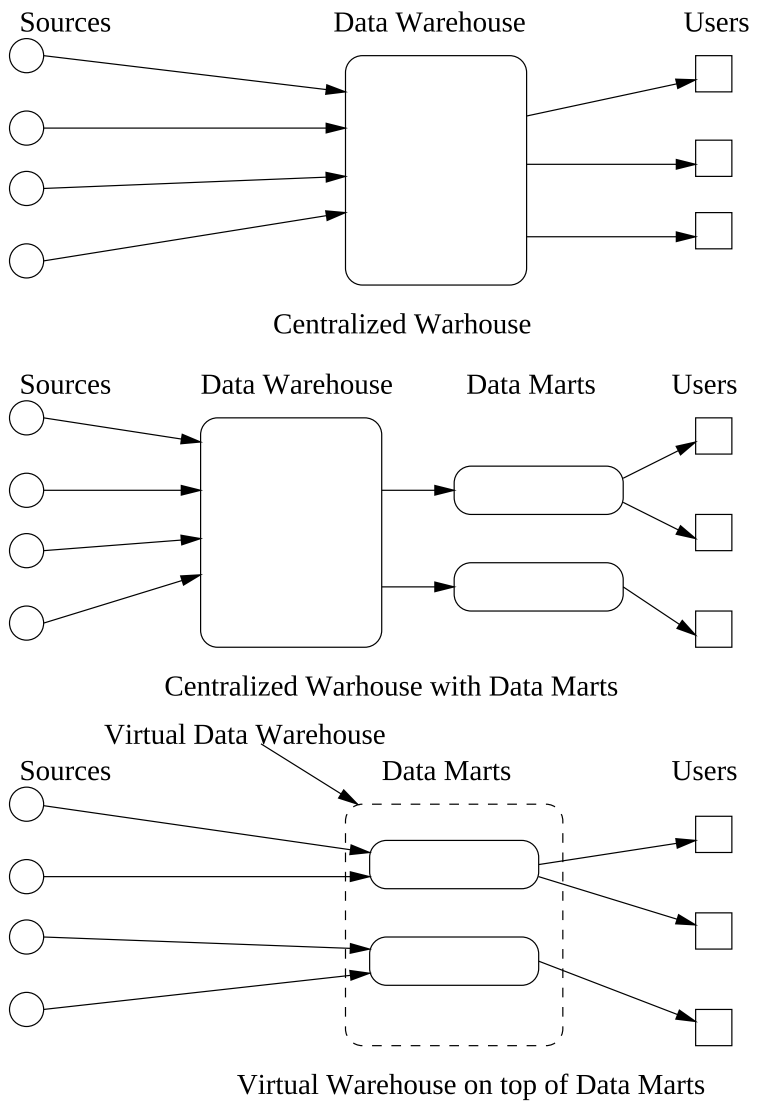

Figure 1: Data Warehouse Architectures

Motivation

- Centralized data warehouse can be heavily loaded
- Data Marts help to distribute load on central data warehouse
- Caching query results at the data marts
  - reduces back end database access
  - improves system performance
  - reduces network traffic

Page caching and logical caching

- Example
  - consider a query $A \bowtie B \bowtie C$, and A is in the cache.
- Differences
  - unit of caching
  - size
  - access cost
  - cache maintenance
    - factors considered for replacement
    - notion of cost benefit

System Model

- Generic architecture
  - multiple servers, cache middleware
- Cache manager integrated with optimizer
- Caching of evaluation plans also
- Deferred and incremental update propagation
- Benefit cost based cache replacement and admission
- Assumptions
  - single query processed at a time
  - no concurrency control, i.e., updates and query processing do not happen concurrently

Associative Caching

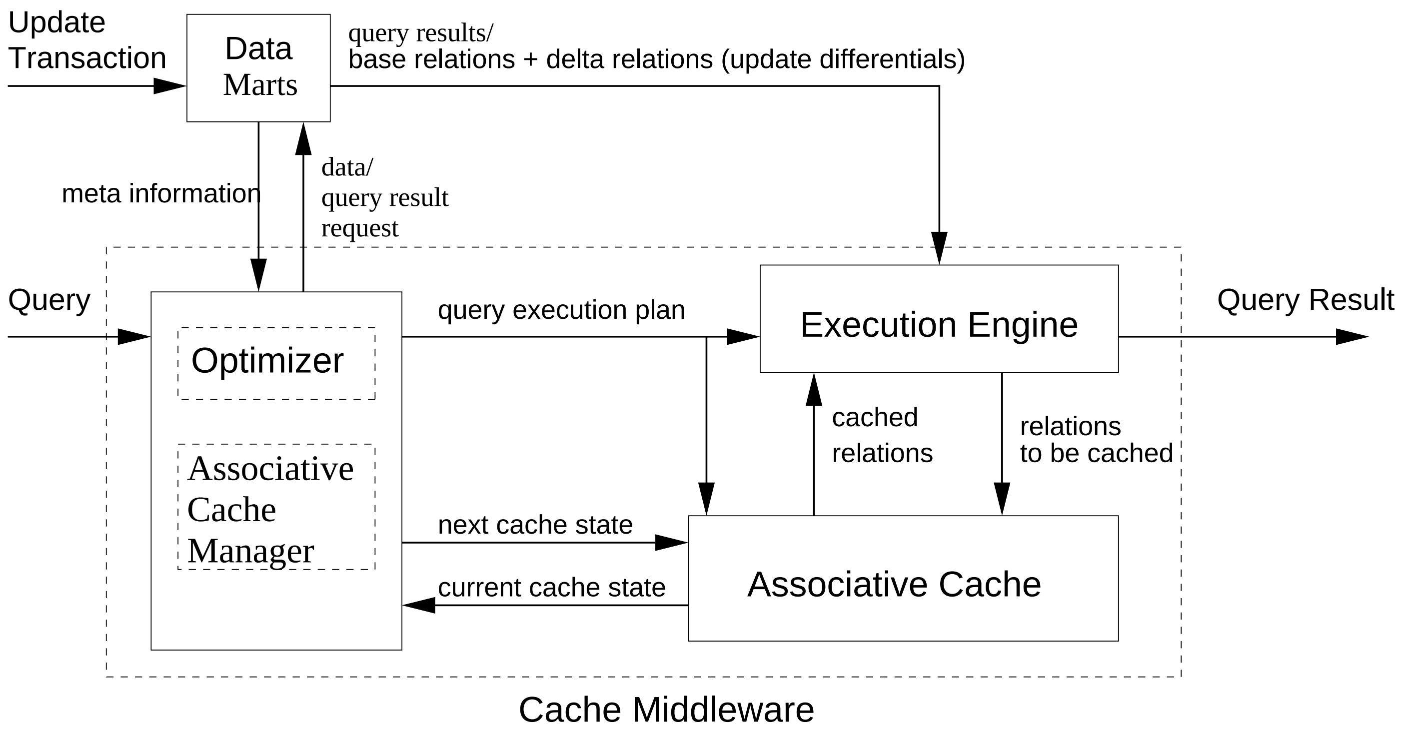

Figure 2: Architecture of Caching Middleware

Associative Cache Management

- Representative set of queries
  - set of queries that represent current workload
- Monitoring workload
  - consider query frequency and recency of usage
  - keep intermediate results that are frequently reused

Associative Cache Management(cont'd)

- Cost metric at a given instant t

$$W(S,t) = \sum_{Q \in R(t)} wt(Q,t)C(Q,S)$$

  - where S is the set of cached results
  - $wt(Q,t)$ is the weight of Q at the instant t in the representative set,
  - $R(t)$ is representative set at t, and
  - $C(Q,S)$ is the best cost of evaluating the query Q, given that all results are contained in S.
- Cost metric should be minimized

Associative Cache Management(cont'd)

- Candidate set
  - set of newly computed results
  - set of cached result
- Cache Admission/Replacement
  - some of the newly computed results may replace existing cache contents if they reduce the cost metric
- Greedy heuristic

Update Propagation

- To manage updated results, we have following options:
  - recomputation of the cached results
  - incremental refresh of the cached results
  - discard the updated cached results
- Decision is made on a cost based manner by the optimizer
- Deferred updates lead to less wasted effort
- Immediate updates can be performed during free CPU/disk cycles

Incremental Refresh

- Delta relations
  - consider result $A \bowtie B$ is in cache and A is updated
  - $(\text{delta}(A) \bowtie B) \bigcup (A \bowtie B)$
- Timestamps determine tuples to be added
- *Refresh operator*
  - defined only on cached relations
  - takes delta relation and outdated cached results as input
  - produces refreshed cache result

Extensions

- Discarding parts of query results
  - Selects
    - consider $A \bowtie B$ in cache, and tuples in range $A.a > 50$ are accessed more frequently, so discard the rest
  - Aggregates
    - cached query results can be broken into various aggregates
    - most beneficial aggregates can be retained in cache
- Sharing query processing between data marts/warehouse and caching middleware
- Concurrency control
  - updates and query processing occur concurrently

Project Implementation

- Optimizer is implemented
- Working on cache manager
- Using Oracle as back end database with ODBC interface
- Using Oracle for storing cached data and for execution engine

Conclusion

- A novel approach to adaptively cache query/intermediate results according to current workload
- Integrating cache manager with optimizer is beneficial
- Extendible approach
  - not specific to any class of queries or any class of operators
- Handles on-line updates
- Updates are performed according to decision made by the optimizer
- Integrating incremental refresh with query processing has minimal overheads