Optimizing Information Mediators by Selectively Materializing Data

Naveen Ashish
Information Sciences Institute, Integrated Media Systems Center and Department of Computer Science

University of Southern California
Information Mediators

Map Servers

Geocoders

Example: Restaurant and Theatre Info on the Web

Ariadne Mediator

Movies

Zagat

Health Ratings
Talk Outline

- Performance - speed of application dependent on sources
- Approach to performance optimization by local materialization
- Materialization framework for mediators
- Design of materialization system
- Selecting data to materialize
  - Distribution of user queries
  - Structure of sources
  - Updates
- Admission and replacement
- The integrated materialization system
- Experimental results
- Related work, applicability to other mediator systems
- Conclusion and future directions
Performance Issue in Information Mediators

- Speed of the application is heavily dependent on sources
- Query response time is high despite having high quality query plans
- Dominant cost is retrieving data from remote sources
  - May have to retrieve a large number of Web pages
  - Source is structured such that retrieving data is time consuming
    - Source may be slow
- Typical Query: “Find all chinese restaurants in Santa Monica with an excellent food rating”
- Takes several minutes to return an answer
Solution: Materialize Data Locally

- Materialize data locally
- Materializing all the data is impractical
  - Mediator degenerates into data warehouse
- Significant performance gain can be achieved by materializing small fraction of data
  - Hypotheses that some portions of data queried more frequently
  - Materializing certain portions of data speeds up response time for expensive queries
- Data has to be *selectively* materialized
- Primary Issues
  - How is materialized data represented and used
  - How do we automatically identify what to materialize
Overall Approach: Define Materialized Data as Another Information Source

- Existing mediator infrastructure to address two issues
  - Providing semantic description of materialized data contents
  - Query planner can reason with contents of materialized data
Selecting Data to Materialize

Distribution of User Queries
(Identify frequently accessed classes)

Structure of Sources
(Prefetch data to speed up expensive queries)

Updates
(Have to consider maintenance cost)

SELECTING CLASSES

Classes of Data to Materialize
Materialization System: Architecture

- Update Specifications
- Refresh Frequency
- Maintenance Cost
- Classes to Materialize
- Classes Proposed to Prefetch
- Classes Proposed by Query Distribution Analysis
- Query Distribution
- Axioms
- GUI Spec

Update Specifications:
- Less Frequently Updated Classes

Source Structure Analysis:
- Classes Proposed by Query Distribution Analysis

Query Distribution Analysis:
- Classes Proposed to Prefetch

Optimizer:
- LOCAL DB

Admission and Replacement:
- Classes to Materialize
SELECT name, tel
FROM restaurant
WHERE cuisine=“Chinese”

SELECT name, review, address
FROM restaurant
WHERE city=“Los Angeles”

SELECT name, address
FROM restaurant
WHERE cuisine=“Mexican”

SELECT name, tel, address
FROM restaurant
WHERE cuisine=“Chinese”

SELECT name, review
FROM restaurant
WHERE cuisine=“Italian”

SELECT name, address
FROM restaurant
WHERE city=“Santa Monica”

SELECT name, tel, review
CM Algorithm for Extracting Patterns

- Too many classes i.e., new information sources create performance problems for query planner
  - Compact description of patterns extracted
- Analyze each query in query distribution
- Create subclasses of interest by analyzing constraints
- For each subclass cluster attribute groups
- Merge across class coverings
- Outputs compact description
Ontology of Subclasses of Interest

- Analyze constraints in each query
- Identify subclasses of information of interest
- Maintain ontology in KR system LOOM
- Record attribute groups queried for each subclass
Clustering Attribute Groups

- Cluster by attribute group similarity and hits
- 2D clustering - optimal clustering NP complete, approximate
Clustering Attribute Groups

(name, address, showtimes) 13
(movieurl, tel) 12
(tel, reviews, name) 5
(name, showtimes) 2
(name, address) 2
(movieurl, tel, reviews) 10
(tel, reviews) 7
(name, showtimes, trailers) 10
(name, showtimes) 8
...

Santa Monica

(~)

(movieurl, tel, reviews) 11
...

(name, address, showtimes, trailers) 10
(name, address, showtimes) 2
(tel, reviews, name) 6
(movieurl, tel, reviews) 11
...
Merging Across Coverings

- Covering: (chinese, mexican, italian) --> Restaurant
- (chinese,{A}) U (mexican,{A}) U (italian,{A}) -->(Restaurant,{A})
Merging Across Coverings

RESTAURANT

(name, address, tel)

Chinese
(name, cuisine)
(rating, service)
(décor, service, tel)

Mexican
(tel, address, décor)

Italian
(name, decor)
(name, rating)
Effectiveness, Complexity

- Measured ‘precision’ and ‘recall’ in extracting patterns
- Pattern P in query distribution
  - Precision is % of patterns extracted that is in P
  - Recall is % of P that is in patterns extracted
- High precision and recall for q=0.2
- Complexity = \(O(M^2N^2)\)
  - \(M\) = number of queries, \(N\) = Number of attributes in a class
Source Structure Analysis

- **Problem:** Certain kinds of queries are expensive as wrapped Web sources not originally designed for database-like querying.
- **Solution:** Prefetch and materialize data to improve response time.
- **Such data cannot be identified by analyzing user queries.**

Diagram showing:
- User Interface
- Cost Estimator
- Axioms
- (name, latitude, longitude) of restaurant
- (name, cuisine) of restaurant
GUI Specification

- Mediator GUI is typically more restrictive
- Formal specification language
- Data items that can be retrieved
- Details of selection conditions that can be specified

```
SELECT {name, tel, address, cuisine, review, city, rating, map}
FROM ent
WHERE [city,1,(LA, NYC, Santa Monica ....)] {cuisine,1,(chinese,... )}
```
Query Processing Axioms

- Precompiled axioms for query processing

restaurant(name, cuisine, address, tel) = zagats(z.name, z.cuisine, z.address, z.tel)

restaurant(name, cuisine, address, tel, lat, long) = zagats(z.name, z.cuisine, z.address, z.tel) and ent_geocoder($z.address, g.lat, g.long)

- Axioms tell what data operations will be performed on what sources
- Can be used to determine data to prefetch
- Cost Estimator: Costs of queries
- Process of Source Structure Analysis
  - Use GUI specification and axioms to identify queries
  - Use cost estimator to determine expensive queries
  - Use axioms and knowledge of type of query to determine data to prefetch
Source Structure Analysis

Example:

GUI specification: selection queries on “cuisine” of restaurant
Cost estimator: Expensive query
Query processing axioms:

restaurant(name, cuisine, address, tel) = zagats(z.name, z.cuisine, z.address, z.tel)

Heuristic: Prefetch key (name) and selection attribute (cuisine)
Optimization: selection can now be done locally, thus faster

Examples of heuristics
1. selection query - materialize key and selection attribute
2. join query - materialize join attributes and keys
3. ordered join - materialize result of ordered join
Updates

- Data materialized can change at original sources
- Strategy
  - Do not materialize very frequently updated data
  - Refresh materialized data at appropriate intervals
- Specifying update characteristics, frequency
- Need not assume that user always absolutely requires the latest data
- Also specify user’s requirements for freshness of data
# Update Specifications

<table>
<thead>
<tr>
<th>CLASS</th>
<th>MEMBERSHIP</th>
<th>CHANGE</th>
<th>TIME_PERIOD</th>
<th>TIME</th>
</tr>
</thead>
<tbody>
<tr>
<td>MOVIE_SRC</td>
<td>A</td>
<td>Y</td>
<td>1 week</td>
<td>week : friday</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ATTRIBUTE</th>
<th>CHANGE</th>
<th>TIME_PERIOD</th>
<th>TIME</th>
</tr>
</thead>
<tbody>
<tr>
<td>actors</td>
<td>N</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>CLASS</th>
<th>TOLERANCE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MOVIE</td>
<td>0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ATTRIBUTE</th>
<th>TOLERANCE</th>
</tr>
</thead>
<tbody>
<tr>
<td>theatre</td>
<td>0</td>
</tr>
<tr>
<td>actors</td>
<td>0</td>
</tr>
<tr>
<td>review</td>
<td>6 weeks</td>
</tr>
</tbody>
</table>
Maintenance Frequency and Cost

For each attribute:

\[
\text{maintenance period} = \max(\text{update period}, \text{tolerance})
\]
\[
= \text{tolerance}, \text{ if update period unknown}
\]

Example

\[
\text{review} = \max(1 \text{ week, 6 weeks}) = 6 \text{ weeks}
\]
\[
\text{theatre} = \max(1 \text{ week, 0}) = 1 \text{ week}
\]

Maintenance frequency and Query cost - \(\rightarrow\) Maintenance cost

Do not materialize an attribute if maintenance cost too high

Total maintenance cost must be within limit
Admission and Replacement

- Choose optimal set of classes to materialize under 2 constraints
  - Space and Maintenance Cost
- Same as fractional knapsack in 2 dimensions
  - Greedy algorithm for multi-dimensional fractional knapsack

\[ w_i \text{ (query response time savings)}, \ s_i \text{ (space)}, \ m_i \text{ (maintenance cost)} \]

- Store in order of \( w_i / \max(s_i, m_i) \)
- Guarantees optimal selection, generalizes to \( n \) dimensions
Optimizer Module

- Gets set of classes of data to materialize
- Retrieves data from original sources
- Populates local database with data
- Makes appropriate changes to domain and source models
- Recompiles axioms
- Periodically refreshes materialized classes
- Periodically reevaluates materialized data classes
## Experimental Results

<table>
<thead>
<tr>
<th>Query Set</th>
<th>Response Time (No optimization)</th>
<th>Response Time (with Optimizer)</th>
<th>%improvement</th>
</tr>
</thead>
<tbody>
<tr>
<td>Q1</td>
<td>38115</td>
<td>9301</td>
<td>75 %</td>
</tr>
<tr>
<td>Q2</td>
<td>44186</td>
<td>3775</td>
<td>91 %</td>
</tr>
</tbody>
</table>

(a) Source structure analysis (Countries)

<table>
<thead>
<tr>
<th>Query Set</th>
<th>Response Time (No optimization)</th>
<th>Response Time (with Optimizer)</th>
<th>Response Time (Page level)</th>
<th>%improvement (Optimizer)</th>
<th>%improvement (Page level)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Q1</td>
<td>38115</td>
<td>1549</td>
<td>34320</td>
<td>96 %</td>
<td>10 %</td>
</tr>
<tr>
<td>Q2</td>
<td>44186</td>
<td>2174</td>
<td>37993</td>
<td>95 %</td>
<td>13 %</td>
</tr>
</tbody>
</table>

(b) All factors, comparison with page level (Countries)

<table>
<thead>
<tr>
<th>Query Set</th>
<th>Response Time (No optimization)</th>
<th>Response Time (with Optimizer)</th>
<th>Response Time (Page level)</th>
<th>%improvement (Optimizer)</th>
<th>%improvement (Page level)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Q1</td>
<td>22013</td>
<td>1644</td>
<td>17832</td>
<td>93 %</td>
<td>19 %</td>
</tr>
</tbody>
</table>

(c) All factors, comparison with page level (TheaterLoc)
Experimental Results

- Total work done with and without materialization
- Varied update frequency
Related Work

- Semantic Caching [Dar 1996, Keller and Basu 1996]
  - Compact description is solution to containment problem
- Caching for Web Proxy Servers [Chankunthod et al., 1995]
  - Significant improvement over page level
- Caching for Federated Databases [Goni et al., 1996]
  - Only fixed classes
- View Selection for materialization in Data Warehousing [Gupta and Mumick 1998]
  - Fixed set of views, minimizing cost of data operations
- Extracting patterns from queries similar to mining association rules [Agrawal, Imielinski & Swami 1993]
  - Differences in language learnt, also we have dynamic ontology
Related Work ....

- Studied applicability of materialization approach in Information Manifold, InfoMaster, TSIMMIS, InfoSleuth, DISCO and Garlic
- Different modeling and query processing approaches
  - views, object-based, datalog based, KR based etc.
- Materialization framework applicable
- Compact description needed in all
- Can be extended for source structure analysis
- Approach for updating data will also apply
Conclusion

◆ Contributions
  – Approach for performance optimization by materialization
  – Identifying what to materialize
    ◆ Query distribution analysis
    ◆ Source structure analysis
    ◆ Update issue

◆ Future Work
  – Support for multimedia data
  – Tighter integration with query planner
  – Automating collection of statistics about sources

◆ Extending contributions to other data management areas
  – Query mining
  – Semantic caching in databases
  – Semi-structured data management