

Selectively Materializing Data in Mediators by Analyzing User Queries

Naveen Ashish, Craig A. Knoblock and Cyrus Shahabi
Information Sciences Institute, Integrated Media Systems Center and
Department of Computer Science
University of Southern California
4676 Admiralty Way, Marina del Rey, CA 90292
{nashish,knoblock,cshahabi}@cs.usc.edu

February 25, 1999

Abstract

There is currently great interest in building information mediators that can integrate information from multiple data sources such as database systems or Web sources. An issue in building information mediators is how to make them responsive given the high response time for retrieving data from the data sources being integrated, which are databases or Web servers residing remotely. We present an approach for optimizing the performance of information mediators by selectively materializing data.

We first present our overall framework for materialization in a mediator environment. We then stress the need to materialize data *selectively* and outline the factors that must be considered to select data to materialize. We present an algorithm for identifying classes of data to materialize by analyzing one of the factors which is the distribution of user queries. We present results with an implemented version of our optimization system for the Ariadne information mediator which show significant performance improvement over no materialization and existing schemes such as page level caching.

1 Introduction

There are several projects focusing on building information mediators, the representative systems include TSIMMIS [Hammer *et al.*, 1995], Information Manifold [Kirk *et al.*, 1995], The Internet Softbot [Etzioni & Weld, 1994], InfoSleuth [Bayardo *et al.*, 1996], Infomaster [Genesereth, Keller, & Duschka, 1997], DISCO [Tomasic, Raschid, & Valduriez, 1997], HERMES [Adali *et al.*, 1997], SIMS [Arens, Knoblock, & Shen, 1996] and Ariadne [Knoblock *et al.*, 1998]. Many of these systems allow database like querying of semi-structured Web sources through *wrappers* around the Web sources, and they also provide integrated access to multiple data sources. The query response time for information mediators, particularly Web based mediators is often very high, mainly because to answer most queries a large number of Web pages must be fetched over the network. For instance consider a mediator that provides integrated access to Web sources of information about countries in the world (We have made available online an implemented mediator on countries information at <http://www.isi.edu/ariadne/demo/mapping-factbook/index.html>). Some useful Web sources about countries information are:

- The CIA World Factbook¹ which provides interesting information about the geography, people, government, economy etc. of each country in the world.
- The NATO homepage² from which we can get a list of NATO member countries.
- The InfoNation³ source which provides statistical data about UN member countries.

¹<http://www.odci.gov/cia/publications/factbook/country-frame.html>

²<http://www.nato.int/family/countries.htm>

³http://www.un.org/Pubs/CyberSchoolBus/infonation/e_infonation.htm

Without any kind of optimization i.e., assuming that all data must be fetched from the Web sources at real time, a typical query to this mediator such as “*Find the defense expenditure and spending on education of all countries that have a national product greater than \$500 billion*” can take as much as around 5 minutes to return an answer. This is because for this particular query the mediator must retrieve the pages of all countries in the CIA World Factbook to determine which ones have a national product greater than \$500 billion, which takes a large amount of time. The query response time can be greatly improved if frequently accessed data is materialized at the mediator side.

We present a performance optimization approach for information mediators based on selectively materializing data. There are two primary issues that must be addressed. First we must design the overall framework for materialization i.e., how do we represent and use the materialized data. Then there is the issue of what data should be materialized. We describe our overall approach in Section 2. In section 3 we describe an approach for selecting data to materialize based on user queries. We present experimental results in Section 4, followed by related work in Section 5 and finally a conclusion.

2 Overall Approach

We now present a description of our approach to performance optimization by materialization. We first provide a brief overview of the SIMS information mediator, in fact the SIMS architecture is typical of many of the other information mediator systems we mentioned. The Ariadne architecture also borrows heavily from that of SIMS. SIMS is used to integrate information from mainly database systems whereas Ariadne integrates information from semi-structured Web sources. We then provide a description of our framework for materializing data in mediators. Finally we outline the factors in selecting data to materialize and the portion focused in this paper.

2.1 SIMS Architecture

In the SIMS system we use the LOOM [MacGregor, 1988] knowledge representation language (we can also view this as a data modeling language) for modeling data. The user is presented with an integrated view of the information in several different sources, which is known as the *domain model*. We describe the contents of the individual information sources in terms of the domain model. A simple example is shown in Figure 1(a). The white circle labeled COUNTRY represents a domain *concept* (equivalent of a class in an object-oriented model) and the shaded circles represent sources. The arrows on the circles represent attributes of the concepts. In this example the domain concept COUNTRY provides an integrated view over two sources of information about countries – FACTBOOK-COUNTRY and UN-COUNTRY. The user queries the integrated view i.e., concepts in the domain model and the query planner in the mediator generates plans to retrieve the requested information from one or more sources. Please refer to [Arens, Knoblock, & Shen, 1996] for a more detailed description of SIMS.

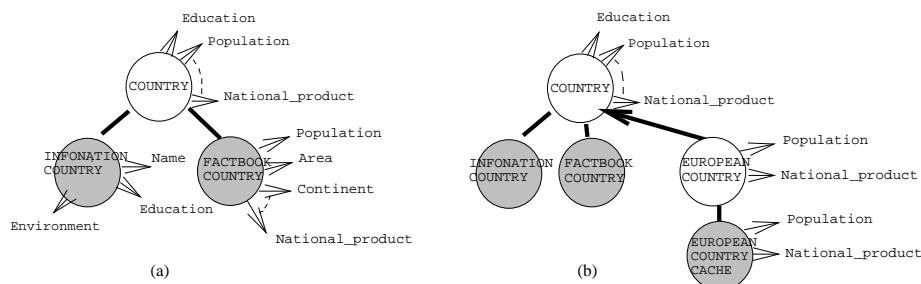


Figure 1: Information modeling in SIMS

2.2 Materializing Data in Mediators

Our approach to optimization is based on an idea described in [Arens & Knoblock, 1994] where we identify useful classes of information to materialize, materialize the data in these classes in a database local to the mediator and define these classes as auxiliary information sources that the mediator can access. For instance in the countries application⁴ suppose we determined that the class of information - *the population and national product of all European countries* was frequently queried and thus useful to materialize. We materialize this data and define it as an additional information source as shown in Figure 1(b). Given a query the mediator prefers to use the materialized data instead of the original Web source(s) to answer the query.

Defining the materialized data as another information source has two advantages. First, we can provide a semantic description of the contents of the materialized data (in LOOM). Second, the query planner in the mediator considers the materialized data source when generating plans to retrieve data to answer a user query. The SIMS query planner is designed to generate high quality plans and will generate plans that attempt to use the materialized data sources whenever they reduce the cost of processing a query. We use the materialized data in essentially the same manner as in a *semantic* caching [Dar *et al.*, 1996] system. Our approach of defining the materialized data as another information source allows us to use the information mediator's knowledge representation system and query planner to address two important problems that arise in any semantic caching system, namely providing a description of the materialized or cached data and doing containment checking i.e., determining if all or some portion of data in a query has already been cached.

2.3 Selecting Data to Materialize

The key problem that remains to be addressed is how to determine what classes of information are useful to materialize. Note that we wish to only selectively materialize data that is useful. The brute force approach of materializing all the data in all the Web sources being integrated is impractical for the following reasons:

- The sheer amount of space needed to store all the data could be very large.
- Data gets updated at the original Web sources and the maintenance cost of keeping the materialized data consistent could be very high.

It is clear that data must be materialized *selectively*. This leaves us with the question of how do we automatically identify the portion of data that is most useful to materialize. We propose an approach where we consider several factors in combination to identify data to materialize. The factors considered are:

- User Query Distribution: By analyzing the user query distribution we can identify the classes of data that are queried most frequently by users and consider materializing such classes. In addition to the materialization framework, an algorithm to identify such frequently accessed classes is the other major contribution of this paper.
- Application and Source Structure: Since we are gathering and integrating data from Web sources that often do not support database like querying, certain kinds of queries can be very expensive. For many such queries we can materialize data that will improve response time for those queries. For instance consider again the information about countries mediator. Assume we have an interface that allows us to ask queries with a selection condition on the GDP. Now such a query is expensive as the source for this information - the CIA World Factbook source does not support selections by GDP and we have to retrieve pages of all the 270 countries in the CIA World Factbook to determine which ones have GDP > \$500 billion. We can prefetch and materialize the primary key (country name) and the selection attribute (GDP) which will improve the response time for selection queries on GDP as the selection can now be done locally. We are developing an approach where we start with a specification of the query interface to a mediator application that defines exactly the kinds of queries a user could ever ask of that mediator application. We then estimate the costs of all different classes of queries using

⁴The model showing the attributes of the COUNTRY concept in the countries application is given in the appendix. We will be using this model in examples throughout the paper

a cost estimator which is part of the mediator. The purpose is to identify in advance the expensive kinds of queries that could be asked in a particular mediator application. Then using heuristics based on the kind of query, source or sources used to answer the query and also the different data processing operations that are performed to answer the query we prefetch and materialize data that can improve the response time for the expensive query.

- Update Characteristics and Frequency: We integrate data from Web sources which may get updated. The materialized data must be kept consistent with the original sources and also the maintenance cost for the materialized data must be taken into account. We are working on an approach to automatically estimating the maintenance cost for each proposed materialized class of data from specifications about the update characteristics of the sources and the user's requirements for freshness of data. The maintenance cost will also be considered when deciding what classes of data to materialize.

In this paper our focus is on the first factor i.e., the distribution of user queries to decide what classes of data to materialize. In the following section we present an algorithm that determines the frequently accessed classes of data by extracting patterns in user queries. In the first version of a local materialization system that we have developed for the Ariadne mediator, we materialize data based on just the output of this algorithm.

3 Analyzing the Distribution of User Queries

One of the hypotheses of our approach is that there will be *patterns* present in user queries i.e., some classes of data would be queried more frequently than others. It would be very useful if we could extract such patterns by analyzing previous user queries as we could consider materializing those. We provide a description of the CM (Cluster and Merge) algorithm which identifies useful classes of information to materialize, by extracting patterns in user queries. The algorithm takes as input a distribution of user queries and outputs a *compact* description of patterns that it can extract from the query distribution in the form of classes of data. A compact description of frequently accessed classes is necessary from a performance point of view. For each class of data we materialize we define a new information source for the mediator. The general problem of query planning to gather information from many sources is combinatorially hard and having a large number of sources will create performance problems for the query planner.

3.1 CM Algorithm

The pseudo code for the CM() algorithm is given in Figure 2. There are three main steps in the algorithm:

- Classifying queries (CLASSIFY_QUERIES()). This is to determine what classes of data the user is interested in.
- Clustering attribute groups (CLUSTER_ATTRIBUTE_GROUPS()). To determine attribute groups of interest for each class.
- Merging classes (MERGE_CLASSES()). This is to try and merge classes of data to make the description more compact.

We now describe the steps in the algorithm in more detail.

3.1.1 Classifying Queries

We first analyze queries to determine what classes of information users are interested in (procedure CLASSIFY_QUERIES()). For instance queries of the form:

```
SELECT A
FROM COUNTRY
WHERE region= "Europe"
```

indicate that the user is interested in the class of European countries. We maintain an ontology in LOOM of

```

CM (QS) { /* QS is set of user queries */
  O=CLASSIFY_QUERIES(QS) /* build ontology O of classes of interest */
  CLUSTER_ATTRIBUTE_GROUPS(O) /* cluster attribute groups in each class */
  FOR (all coverings (S, C) in O) {
    MERGE_CLASSES(S, C) /* merge classes */
  }
}

CLASSIFY_QUERIES(QS)
WHILE ( more_queries(QS) ) {
  Q = get_next_query(QS)
  SP = get_query_subclass(Q)
  { SPi } = get_interest_subclasses(Q)
  update_ontology(SP)
  IF P =  $\emptyset$  {
    update_attribute_count(S, A)
  } ELSE {
    n=count({ SPi })
    FOR i = 1 to n DO update_attribute_count( SPi, A )
    update_ontology( SP )
  }
}

CLUSTER_ATTRIBUTE_GROUPS(O) {
/* cluster attribute groups for each class in O with similar frequency */
}

MERGE_CLASSES(S, C)
i=1
n=number_of_elements(C) /* size of set C */
totalsize = 0 /* space occupied by matching groups */
seed = get_seed(Ci) /* pick an attribute group from Ci */
WHILE (seed  $\neq$   $\emptyset$ ) {
  seedsize=get_size(seed) /* space occupied by seed */
  totalseedsize= seedsize*n /* space occupied by (S,seed) */
  FOR i = 1 to n DO {
    temp = find_match(seed, Ci) /* find best matching group for seed in Ci */
    size = get_size(temp)
    totalsize += size /* total size of matching groups */
  }
  IF (totalsize/totalseedsize)  $\geq$  MERGETHRESHOLD { /* merging criteria satisfied */
    remove(seed, C) /* remove attributes in seed from all classes */
    add_group(S,seed) /* form group in superclass */
  }
  ELSE {
    mark_down(seed, Ci) /* so that the same seed is not chosen again */
  }
}
i=(i+1)mod(n) /* chose next class in C cyclically */
seed = get_seed(Ci) /* get seed from next class */
}

```

Figure 2: The CM algorithm for extracting patterns in queries

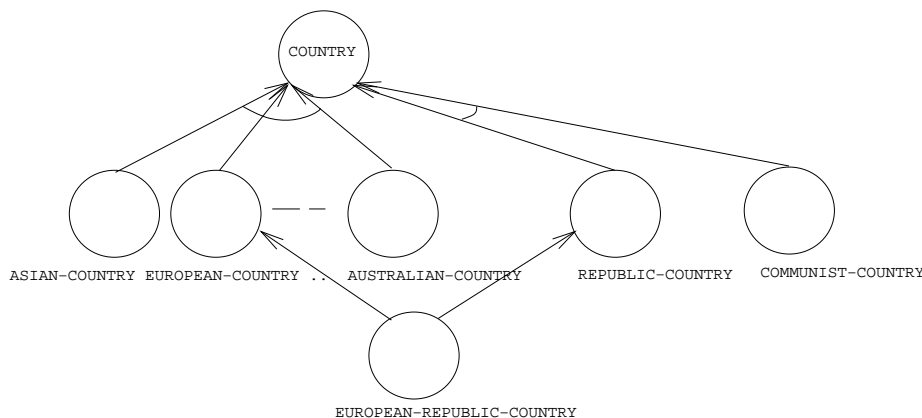
classes of information that are queried by users. Initially the ontology contains only the classes in the domain model. We then add sub-classes of these existing classes to the ontology, the sub-classes are generated by analyzing constraints in the user queries. Assuming an SQL syntax for the queries, a query to a domain class has the following general form:

```
SELECT A
FROM S
WHERE P
```

where A is the set of attributes queried for the domain class S and $P = P_1$ and P_2 and ... P_n are predicates specifying the query constraints (we restrict ourselves to conjunctive queries). We denote as S_P the “query sub-class” which is the sub-class of S satisfying P . We denote as $\{S_{P1}, S_{P2}, \dots, S_{Pn}\}$ the set of “sub-classes of interest” where the P_i s are the individual predicates comprising P and S_{Pi} is the sub-class of S satisfying P_i . For instance consider a query such as:

```
SELECT population, area
FROM COUNTRY
WHERE (region = “Europe”) AND (government = “Republic”);
```

In the above query the query sub-class is that of European Republic Countries and the sub-classes of interest are European Country and Republic Country. In the `CLASSIFY_QUERIES()` procedure for each query we first determine the query sub-classes and set of sub-classes of interest and insert them into the ontology if they are not already present. For instance for a set of queries on the concept `COUNTRY` in which the `WHERE` clauses have constraints on the attributes *region* (bound to a value such as Europe, Asia etc.) or *government* (bound to a value such as Republic, Monarchy, Communist etc.) or both, we would create an ontology such as shown in Figure 3. (The arcs in the figure represent coverings of groups of subclasses for the superclass `COUNTRY`).



3.1.3 Merging Classes

We mentioned earlier that it is important to keep the number of classes of information materialized small from a query processing perspective. Consider the following classes of information, each of which is essentially a group of attributes in a class. (EUROPEAN-COUNTRY, {population, area}), (ASIAN-COUNTRY, {population, area}), (AFRICAN-COUNTRY, {population, area}), (N.AMERICAN-COUNTRY, {population, area}), (S.AMERICAN-COUNTRY, {population, area}) and (AUSTRALIAN-COUNTRY, {population, area}). We could replace the above six classes by just one class (COUNTRY, {population, area}) which represents exactly the same data. In general thus a group of classes of information of the form $(C_1, A), (C_2, A), \dots, (C_n, A)$ ⁵ may be replaced by one class i.e., (S, A) if C_1, C_2, \dots, C_n are direct subclasses of S and form a covering of S. As the ontology of classes is maintained in LOOM, we use LOOM to determine groups of classes we can merge based on class/subclass relationships.

set of subclasses C with attribute groups	matching groups	size
(EUROPEAN-COUNTRY, { *imports, exports }, { area, gdp, economy })	{ imports, exports }	2
(ASIAN-COUNTRY, { imports, exports, climate }, { debt, economy })	{ imports, exports }	2
(AFRICAN-COUNTRY, { imports }, { population, languages })	{ imports }	1
(N.AMERICAN-COUNTRY, { climate, terrain }, { government }, { literacy })	{ }	0
(S.AMERICAN-COUNTRY, { area, coastline }, { imports, exports })	{ imports, exports }	2
(AUSTRALIAN-COUNTRY, { imports, exports, debt }, { gdp, defense })	{ imports, exports }	2

Table 1: Merging across classes

set of subclasses C with attribute groups
(EUROPEAN-COUNTRY, { area, gdp, economy })
(ASIAN-COUNTRY, { *climate }, { debt, economy })
(AFRICAN-COUNTRY, { population, languages })
(N.AMERICAN-COUNTRY, { climate, terrain }, { government }, { literacy })
(S.AMERICAN-COUNTRY, { area, coastline })
(AUSTRALIAN-COUNTRY, { debt }, { gdp, defense })

Table 2: Classes after one merging step

In fact we also allow for a kind of ‘relaxed’ merge where we may merge a set of classes such as $(C_1, A_1), \dots, (C_n, A_n)$ to (S, A) where the C_i s are direct subclasses of S as above. However A_1, \dots, A_n need not be exactly equal groups rather they just need to overlap well, and A is the union of A_1, \dots, A_n . The disadvantage in this case is that the merged class of information will contain some extra data i.e., data not present in any of the classes of information merged to form the merged class. There is a tradeoff between space wasted to store the extra data and the time gained (in query planning) in reducing the number of classes materialized. The amount of space that can be wasted by extra data is limited by a parameter known as the MERGE-THRESHOLD.

The procedure MERGE_CLASSES() shows how we do exact or relaxed merging of classes. The procedure takes as input a superclass S and a set of subclasses C of S that form a covering of S. For each class in C we also have the set of groups of attributes queried. The basic idea is to take an attribute group A in a class C_i in C and see if we can merge with other groups in other classes in C to the group A in the superclass S . We describe the steps in the procedure MERGE_CLASSES() by stepping through the procedure with an example as shown in Table 1. The first column shows the the various classes in C along with their attribute groups. The asterisk(*) next to the {imports, exports} group of the first class i.e., EUROPEAN-COUNTRY indicates that we will choose that group as a *seed* for merging with other classes. The next step is to find matching groups for the seed in all other classes. This is done by the find_match() procedure which given a seed and a class returns the largest subset of attributes of the seed that it can find in any group in the class. The results of find_match() for each of the classes in C are shown in the third column. The next step is to find the ratio of the space occupied by the matching groups in the classes of C to the space needed to store the group A for

⁵ C_i s are classes and A is an attribute group

the superclass S . The ratio should be higher than the MERGE-THRESHOLD to allow merging the matching clusters to the cluster A in S . Intuitively this is to ensure that the attributes in A occur sufficiently through the classes in C to justify merging the matching groups to the group A in S . In this example *totalsize* i.e., the space occupied by the matching groups is $(2+2+1+0+2+2) = 9$ units. The *totalseedsize* i.e., the space that should be occupied in case of an exact merge is $2*6 = 12$. Thus the ratio is $9/12=0.75$ and we do merge to the group {imports,exports} for the superclass COUNTRY (assume that MERGE-THRESHOLD is 0.7). Table 2 shows the same set of classes after the merging step when the attributes in $A = \{imports,exports\}$ have been removed from the classes in C . In case we do not merge the groups we do not remove the attributes from the classes. However we mark the seed group as 'down' so that it is not picked again as a seed. We then pick a seed from the next class ASIAN-COUNTRY and repeat the above procedure.

The main motivation behind the steps of merging attribute groups for a single class and also merging classes based on class/subclass relationships is keep the description of the classes of data extracted as patterns compact. Finally we also keep count of how many queries each class of data supports i.e., how many queries can be answered using that class. From this we can calculate the ratio of supported queries by each class to the total number of queries in the distribution. A class is finally output as a pattern by the CM algorithm only if this ratio is greater than a threshold known as the *query ratio threshold* (q).

4 Experimental Evaluation

The experimental evaluation consists of two parts. First we evaluate how effective the CM algorithm is in extracting patterns from a query distribution. Next we evaluate the effectiveness of a performance optimization system that we built for Ariadne which materializes data based on just the user query distribution.

4.1 Evaluating the CM Algorithm

We set up an experiment to evaluate the effectiveness of the CM algorithm in extracting patterns from a query distribution. The experiment is based on standard precision and recall measurements for evaluating information retrieval systems. This is because we are trying to estimate how effective CM is in extracting patterns that are present and also to what extent it extracts extraneous data as patterns.

We first defined a schema for an imaginary mediator application against which we can pose queries. The schema consists of a class S with 50 attributes $A1, A2, \dots, A50$. The class S is further partitioned into 5 disjoint subclasses $S1, S2, S3, S4$ and $S5$. Each subclass has 10 instances, $S1$ has instances $E1, E2, \dots, E10$, $S2$ has instances $E11, \dots, E20$ etc. We then defined a "pattern" P which is the class $S3$ with attributes $A25, \dots, A30$. We then generated different query distributions against this schema varying the percentage of queries that fall within the pattern. We input each distribution to the CM algorithm to see what patterns it would extract from the distribution. We use standard precision and recall measurements from information retrieval to measure the effectiveness of CM in extracting the predefined pattern P . The precision is the percentage of data extracted that is relevant whereas recall is the percentage of relevant data extracted. In our experiment the predefined pattern P is the *relevant* data, while for each time we run the CM algorithm over a query distribution the patterns extracted from the distribution is the data *retrieved*. Finally since the query ratio threshold (q) affects what patterns are ultimately output by the CM algorithm we present precision and recall measures for varying threshold values.

Figures 4(a) and (b) show the precision and recall values respectively against varying percentages of queries that fall within the predefined pattern P in a query distribution. For each we present precision and recall values for different query ratio thresholds (q). The CM algorithm does indeed prove to be efficient in extracting the predefined pattern P as the recall values are very high (100%) for most of the threshold range for moderate or high percentages of queries in the pattern P . For extraneous data extracted along with P we must analyze the precision values graph. For high threshold values ($q=0.4 - 0.5$) the precision is very high when a high percentage of queries are in P ($> 50\%$) but very low for lower percentages. This is because even if queries in the pattern P are present, they need to be in a very high proportion for the CM algorithm to extract them at all. For lower threshold ($q=0 - 0.1$) the precision is quite low even when a high percentage of queries is in P . This is because of a low threshold the CM algorithm extracts a lot of random classes as patterns in addition to the pattern P . It is best to keep the threshold at an intermediate value (0.2 - 0.3)

where the precision is high for moderate or high percentage of queries in P. The recall remains quite high (100%) for most of the threshold range for moderate or high percentages of queries in the pattern P.

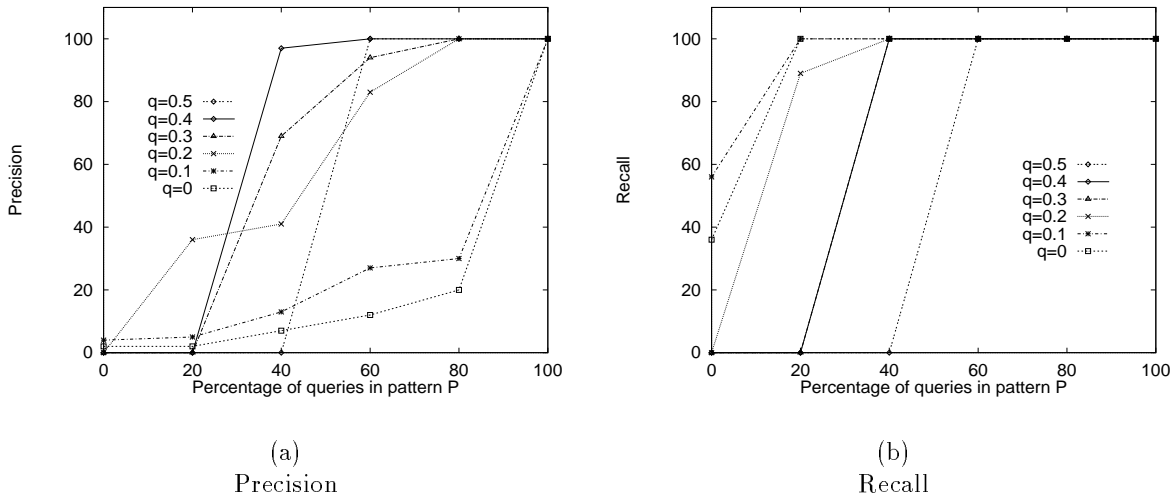


Figure 4: Effectiveness of CM Algorithm

4.2 Evaluating the Performance Improvement

We have implemented an optimization system for the Ariadne information mediator where we materialize data based on the user query distribution i.e., we materialize classes of data output by running the CM algorithm over the query distribution. To compare the effectiveness of our approach with existing approaches we also implemented a page level caching system for the Ariadne mediator based on an LFU (Least Frequently Used) replacement scheme. The aim of the performance evaluation experiments is to compare the performance improvement in Ariadne achieved with our scheme with Ariadne without any optimization based on materialization and also with an optimization system based on an existing scheme such as page level caching.

We measured the total query response time against two query sets to the mediator without any optimization system, with our optimization system and with an optimization system based on page level caching. The CM algorithm in the system was run using a query ratio threshold of 0.3. The first query set Q1 is one that we generated and in which we introduced some distinct query patterns. The details of the distribution are given in the appendix. The second query set Q2 is the set of actual user queries to the countries mediator that we have made available online at <http://www.isi.edu/ariadne/demo/mapping-factbook/index.html>. We logged queries to this mediator over the past several months. Table 3 shows the response time (total time for the entire query set) without any optimization, with page level caching and with our system for both query sets. Our system appears to be effective in improving performance. For Q1 (200 queries) with a limited space for materialized data our system not only provides significant improvement in query response time but also the optimization is an order of magnitude better than with page level caching with the same local space. We also show query response against the set of actual user queries Q2 (182 queries). Again with our optimization system the performance is much better than no optimization or with page level caching.

Query set	Response Time (No optimization)	Response Time (Page level)	Response Time (Our system)	%improvement (Page level)	%improvement (Our system)
Q1	9661 sec	8695 sec	1536 sec	10 %	84 %
Q2	3742 sec	3255sec	2245 sec	13 %	40 %

Table 3: Query response times

5 Related Work

We materialize data in a manner similar to a semantic caching [Dar *et al.*, 1996] or *predicate based* [Keller & Basu, 1996] caching system. A problem with the semantic caching approach is that the containment checking problem is hard and having a large number of semantic regions creates performance problems. A solution proposed in [Keller & Basu, 1996] is to reduce the number of semantic regions by merging them whenever possible. This is in fact an idea we have built on. In the CM algorithm we have presented an approach for systematically creating new semantic regions to consider for materializing and merging them when possible. We have also proposed a relaxed merging of semantic regions in addition to exact merging.

For the mediator environment an approach to caching is described in [Adali *et al.*, 1997]. The focus of their work however is caching for a mediator environment where we integrate information from sources that may not be traditional database systems. Their contribution is a caching approach based on first estimating the cost of accessing various sources based on statistics of costs of actually fetching data from the sources. In their approach reasoning about cache contents is done through the notion of *invariants* which are basically expressions that show possible substitutions or rewritings of queries. This approach provides very limited semantic reasoning capabilities about the contents of the cached data as compared to our approach in which we are able to perform more powerful reasoning of the materialized data contents through LOOM.

Another approach to caching for federated databases is described in [Goni *et al.*, 1997]. There is also a semantic caching approach where the data cached is described by queries. They also define some criteria for choosing an optimal set of queries to cache. Finding the optimal set is an NP-complete problem and they use an A* algorithm to obtain a near optimal solution. A limitation of their approach is that the cached classes can only be in terms of classes in a predefined hierarchy of classes of information for a particular application. Our approach is much more flexible in that we dynamically construct classes of information to materialize.

There is some research on materializing the Web described in [Rosa *et al.*, 1998]. The focus of their work however is to build a fully materialized view over Web data for data in a user specified generic domain of interest. Our goal is different in that we wish to just partially materialize data to improve performance for mediators that would otherwise access data from remote sources.

6 Future Work and Conclusion

We have described an approach for optimizing the performance of information mediators. The major contributions of this paper can be summarized as follows:

- We outlined a proposal for selecting data to materialize based on a combination of user query distribution, source structure analysis and update cost.
- We presented the CM algorithm for compactly identifying data to materialize by analyzing the user query distribution.
- We demonstrated the effectiveness of our approach with an initial implementation of the system

Although the initial version of our materialization system uses only the user query distribution for materializing data our ultimate goal is to build a system that uses all of query distribution analysis, source structure analysis and update costs in combination to materialize data in an optimal fashion. We are working on the design and implementation issues of incorporating the other factors of source structure analysis and update cost when selectively materializing data and plan to develop a comprehensive solution to the performance issue in information mediators.

References

[Adali *et al.*, 1997] Adali, S.; Candan, K. S.; Papakonstantinou, Y.; and V.S.Subrahmanian. 1997. Query caching and optimization in distributed mediator systems. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*.

- [Arens & Knoblock, 1994] Arens, Y., and Knoblock, C. A. 1994. Intelligent caching: Selecting, representing, and reusing data in an information server. In *Proceedings of the Third International Conference on Information and Knowledge Management*.
- [Arens, Knoblock, & Shen, 1996] Arens, Y.; Knoblock, C. A.; and Shen, W.-M. 1996. Query reformulation for dynamic information integration. *Journal of Intelligent Information Systems, Special Issue on Intelligent Information Integration* 6(2/3):99–130.
- [Bayardo *et al.*, 1996] Bayardo, R.; Bohrer, W.; Brice, R.; Cichocki, A.; Fowler, G.; Helal, A.; Kashyap, V.; Ksiezyk, T.; Martin, G.; Nodine, M.; Rashid, M.; Rusinkiewicz, M.; Shea, R.; Unnikrishnan, C.; Unruh, A.; and Woelk, D. 1996. Semantic integration of information in open and dynamic environments. Technical Report MCC-INSL-088-96, MCC, Austin, Texas.
- [Dar *et al.*, 1996] Dar, S.; Franklin, M. J.; Jonsson, B. T.; Srivastava, D.; and Tan, M. 1996. Semantic data caching and replacement. In *Proceedings of the 22nd VLDB Conference*.
- [Etzioni & Weld, 1994] Etzioni, O., and Weld, D. S. 1994. A softbot-based interface to the Internet. *Communications of the ACM* 37(7).
- [Genesereth, Keller, & Duschka, 1997] Genesereth, M.; Keller, A.; and Duschka, O. 1997. Infomaster: An information integration system. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*.
- [Goni *et al.*, 1997] Goni, A.; Illarramendi, A.; Mena, E.; and Blanco, J. M. 1997. An optimal cache for a federated database system. *Journal of Intelligent Information Systems* 1(34).
- [Hammer *et al.*, 1995] Hammer, J.; Garcia-Molina, H.; Ireland, K.; Papakonstantinou, Y.; Ullman, J.; and Widom, J. 1995. Information translation, mediation, and mosaic-based browsing in the tsimmis system. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*.
- [Keller & Basu, 1996] Keller, A. M., and Basu, J. 1996. A predicate-based caching scheme for client-server database architectures. *The VLDB Journal* 5(2):35–47.
- [Kirk *et al.*, 1995] Kirk, T.; Levy, A. Y.; Sagiv, Y.; and Srivastava, D. 1995. The information manifold. In *Working Notes of the AAAI Spring Symposium on Information Gathering in Heterogeneous, Distributed Environments*.
- [Knoblock *et al.*, 1998] Knoblock, C. A.; Minton, S.; Ambite, J.-L.; Ashish, N.; Modi, P. J.; Muslea, I.; Philpot, A.; and Tejada, S. 1998. Modeling web sources for information integration. In *Proceedings of the Fifteenth National Conference on Artificial Intelligence (AAAI)*.
- [MacGregor, 1988] MacGregor, R. 1988. A deductive pattern matcher. In *Proceedings of AAAI-88, The National Conference on Artificial Intelligence*.
- [Rosa *et al.*, 1998] Rosa, M. D.; Catarci, T.; Iocchi, L.; Nardi, D.; and Santucci, G. 1998. Materializing the web. In *Proceedings of the Third IFCIS International Conference on Cooperative Information Systems (CoopIS '98)*.
- [Tomasic, Raschid, & Valduriez, 1997] Tomasic, A.; Raschid, L.; and Valduriez, P. 1997. A data model and query processing techniques for scaling access to distributed heterogeneous databases in disco. In *Invited paper in the IEEE Transactions on Computers, special issue on Distributed Computing Systems*.

A COUNTRY Relation

Relation: COUNTRY

Attributes: (geography location map_references region area total_area land_area comparative_area land_boundaries coastline maritime_claims international_disputes climate terrain natural_resources land_use irrigated_land environment note people population age_structure population_growth_rate birth_rate death_rate net_migration_rate

infant_mortality_rate life_expectancy_at_birth total_fertility_rate nationality ethnic_divisions religions languages literacy labor_force government_names digraph_type capital administrative_divisions independence national_holiday constitution legal_system suffrage executive_branch legislative_branch judicial_branch political_parties_and_leaders other_political_or_pressure_groups diplomatic_representation_in_US us_diplomatic_representation organization flag economy overview national_product national_product_real_growth_rate national_product_per_capita inflation_rate_consumer_prices unemployment_rate budget exports imports external_debt industrial_production electricity industries agriculture illicit_drugs economic_aid currency exchange_rates fiscal_year transportation railroads highways inland_waterways pipelines ports merchant_marine airports communications telephone_system radio television defense_Forces branches manpower_availability defense_expenditures)

B Query Set Q1

1. 30 % of the queries are of the form:

```
SELECT A
FROM COUNTRY
WHERE name= C;
```

where A is a set of attributes and C is a country name. A and C are generated for each query of this form. In 80% of the instances of A all the attributes in set A lie within a set of 3 predefined “favourite” attributes. The remainder 20% contain randomly selected attributes. We chose the favourite set to be {imports, exports, economy}. For C we randomly selected a country name from the set of all country names. The names were picked with approximately equal probabilities.

2. 20 % of the queries are of the form:

```
SELECT A
FROM COUNTRY
WHERE region = “ASIA” ;
```

80% of the instances of A are within a set of 4 favourite attributes that we chose to be {location,map_references, area,climate} .

3. Another 20 % of the queries are of the form:

```
SELECT A
FROM COUNTRY
WHERE region = “ASIA” and organization = “NATO” ;
```

As above 80% of the instances of A are within a set of 2 favourite attributes that in this case we chose to be {defense_expenditure,external_debt} .

4. Finally the remaining 30 % queries are of the form

```
SELECT A
FROM COUNTRY
WHERE region = X;
```

where X is a region such as ASIA, EUROPE etc. The queries are uniformly distributed amongst the (six) possible values of region. Again, 80% of the instances of A are within a set of 4 favourite attributes that in this case we chose to be {economy,currency,religions,literacy} .