Semantic cache mechanism for heterogeneous Web querying

Boris Chidlovskii [a,*,1], Claudia Roncancio [b,2], Marie-Luise Schneider [a,1]

[a] *Xerox Research Centre Europe, Grenoble Laboratory, Grenoble, France*
[b] *ENSIMAG/Lab. LSR IMAG, Grenoble, France*

Abstract

In Web-based searching systems that access distributed information providers, efficient query processing requires an advanced caching mechanism to reduce the query response time. The keyword-based querying is often the only way to retrieve data from Web providers, and therefore standard page-based and tuple-based caching mechanisms turn out to be improper for such a task. In this work, we develop a mechanism for efficient caching of Web queries and the answers received from heterogeneous Web providers. We also report results of experiments and show how the caching mechanism is implemented in the Knowledge Broker system. © Published by Elsevier Science B.V. All rights reserved.

Keywords: Semantic cache; Web querying; Source heterogeneity; Meta-searcher

1. Introduction

Nowadays, we are witnessing the rapid growth of information available over the World Wide Web. As the Web grows, more and more new general-purpose and domain-specific information services assist the user in searching for the relevant data. As no search service can be universally efficient or even complete, this has triggered the appearance of a new type of the Web-oriented software, so called meta-searchers. A *meta-searcher* is a Web information retrieval system which searches for answers to user queries not in a local index, but in various Web information providers. When a meta-searcher receives the providers' responses (in the form of XML/HTML files), special components, hereafter called *wrappers*, process the responses in order to extract data relevant to the original query [16,19,20]. Fig. 1 shows a typical meta-searcher architecture.

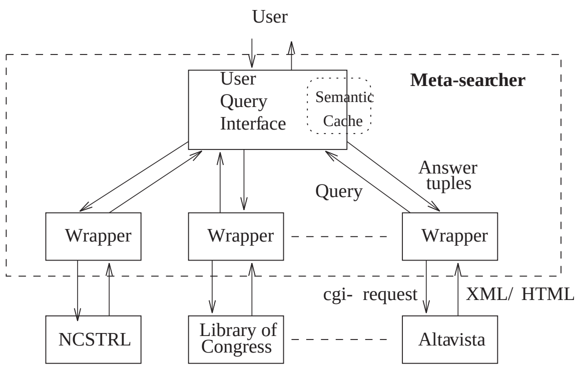


Fig. 1. A meta-searcher architecture.

As in any client–server system, high performance in a networked information retrieval system is often reached by efficient utilization of client storage resources. In the networked environment, data from remote servers are brought to clients on-demand,

* Corresponding author.
[1] E-mail: {chidlovskii,schneider}@xrce.xerox.com
[2] E-mail: Claudia.Roncancio@imag.fr

and client memory is largely used to cache data and minimize future interaction with the servers. This data caching has got a particular importance in the Web-based information systems, as the network traffic and slow remote servers can lead to long delays in the answer delivery. Unfortunately, standard caching techniques work poorly on the Web. The page caching which is widely used in operative and database management systems is improper on Web retrieval systems and tuple-caching has certain limitations. Thus much effort has been spent to cache user queries with the corresponding answers (instead of pages or tuples) to allow their future reuse [5,14,15].

Query caching takes a particular advantage when the user often refines a query, for example, by adding or removing a query term. In this case, many of the answers may already be cached and can be delivered to the user right away. Importantly, when accessing the payment sites, the query caching allows for avoiding some repeated queries and thus saves the user some money.

1.1. Page and tuple caching

A Web-based information retrieval system differs from a standard client–server architecture where the transfer units between servers and clients are pages or tuple sets. *Page caching* mechanisms are widely used in operative and database management systems; they assume that each query posed at the client can be processed locally and be broken down to the level of requests for individual pages. Then, if a requested page is not present in the client cache, a request for the entire page is sent to the server. Such a query processing is improper in a Web-based retrieval system, where the keyword-based querying is often the only way to retrieve data and where the data organization at the servers is completely hidden from the clients.

With *tuple caching*, the cache is maintained in terms of individual tuples, allowing a higher level of flexibility than pure page caching. On the Web, the tuple caching is feasible as Web documents can be referred and accessed by using their Universal Resource Locators (URL's). Moreover, this mechanism is used for caching Web pages at *proxy servers* [21]. A proxy cache maintains the set of recently accessed Web pages and reuse a page from cache each time its URL is asked by a client.

However, when Web information services are interrogated with Boolean queries, the use of tuple caching is much less attractive because of two main disadvantages. First, the user request with a query does not contain the URL of the answer page; instead it contains a filled search form. Once the service receives the request with the form, the URL of the answer is dynamically calculated by the cgi-script; this makes the proxy cache helpless for prefetching the answer items. Second, there is no way to inform the information services about qualified tuples in the client cache and thus reduce the answer size. Similarly, clients cannot detect if their local caches provide a complete answer to the queries. As a result, clients are forced to ignore the cached tuples while performing the query. Once the query is sent to the server and all qualifying tuples are returned, the clients detect and discard the duplications.

1.2. Semantic caching

To overcome the drawbacks of page and tuple caching when querying Web sources, semantic cache has been proposed by different researchers [14,11]. This approach manages the client cache as a collection of semantic regions. A semantic region groups together semantically related data covered, for example, by a user query. Moreover, the access information and cache replacement are managed at a unit of semantic regions [14].

When a query is posed at a client with a semantic cache, the query is split into two pieces:

(1) a probe query, which retrieves the portion of the answer available in the local cache, and
(2) a remainder query, which retrieves the missing data from the server.

If the remainder query is not null (i.e., the query asks for data that is not cached), the remainder query is sent to the server and processed there [14].

1.3. Heterogeneity

A meta-searcher rarely uses a single source to answer a user query; much often the query is sent to numerous sources. Therefore, a semantic cache mechanism should properly adopt the heterogeneous environment typical for the Web. Generally, a wide heterogeneity of Web services has become the main problem for meta-searchers, whose main goal is indeed to hide this heterogeneity from the user. Usually we distinguish between *structural* and *semantical* heterogeneity. When the Web sources are semantically heterogeneous, the answers to the same query are not necessarily compatible. As an example, answers to the query "`Java`" in the context of computer science and geography are indeed incompatible.

In this paper we cope rather with the structural heterogeneity, when the sources are grouped on the domain basis, but sources in a group may differ in providing the same information. With the structural heterogeneity, there are two main issues where the Web sources expose their diversity: different search facilities for formulating user queries and different representations of answers.

To cope with the heterogeneity of query languages, some techniques have been proposed in [17,9,10]. They all follow the *subsumption* strategy when an original user query is translated into one or several queries in the native query languages in a way that the answer set will comprise the answer to original query; to discard then the irrelevant extra-answers, the methods invoke the post-filtering step.

Different representation of answers by information sources hardly influences the wrappers' structure, but it also unveils some important source characteristics. Some of them are highly relevant to the cache management. For example, the answer can be complete (containing all relevant answers on the site) or not (only k top-ranked items satisfying the query); the answer items can be represented by unique page (**DBLP site**[3]) or split into a sequence of linked pages (**Altavista**[4]). Therefore, different representations of answers attribute different semantics the cached data thus changing the ways it is reused for new queries.

1.4. Our contribution

In this paper, we develop a semantic cache mechanism for querying heterogeneous Web providers. We describe a general framework for storing the Web queries along with their answers in the semantic cache. It includes the organization of regions in the cache and their replacement. The regions are assigned with the region signatures which allow for an efficient processing of the equivalence, inclusion and intersection cases between user queries and cache regions. We analyze and identify the main characteristics of Web-based sources which influence the cache management, namely source completeness and check-ability, and develop proper algorithms. We also describe how the semantic cache proposed in this paper is integrated into the Knowledge Broker system developed at the Xerox Research Centre Europe.

The rest of the paper is organized as follows. Section 2 introduces the general semantic cache mechanism and the problem of Web source heterogeneity. It describes the complete and checkable answers of Web sources and Section 3 proposes the corresponding algorithms. Section 4 studies the replacement strategy in the semantic cache. Section 5 describes the implementation of the semantic cache. Finally, Section 6 reviews the related work and Section 7 concludes the work.

2. Semantic cache mechanism

In this section, we introduce the main components of a semantic cache mechanism aimed at the optimization of distributed information retrieval from Web-based information sources. We describe a uniform query language offered to the user by a meta-searchers, the semantic cache architecture and how the heterogeneity of query answers from different sources influences the cache management.

2.1. Query language

In meta-searchers, the query language is often attribute-oriented, that is, queries can be asked against certain document attributes. A query is a conjunction of terms where each term has a form *Attribute Op Value*, where *Attribute* is an attribute name specific for a particular domain (like `Title`, `Author` and `Abstract` for the bibliographic search or `Patent_Number` for patent databases), *Op* is *Contains* or *Equals* operations, and *Value* is a keyword or phrase. A query example is $Q =$ `Title`

[3] Database and Logic Programming site at http://www.informatik.uni-trier.de/~ley/db/index.html
[4] http://altavista.digital.com/

Contains “`Web`” `AND Title` *Contains* “`caching`”. For the simplicity, in the following sections we will assume some default values for attributes and operations (for example, `Full-text` attribute and *Contains* operation); so queries will appear as simple conjunctions of keywords and phrases, for example, “`Web AND caching`”. Note the conjunction can contain negated terms, for example, “`applet AND NOT netscape`”. Such a query will retrieve the document containing the keyword `applet`, but not `netscape`.

2.2. Semantic cache architecture

2.2.1. Semantic regions

The client cache manages a collection of semantic regions grouping together semantically related data, such as the answer to a user query. In any semantic region, we distinguish between the *region descriptor* and *region content*. For the efficient processing, the cache keeps region descriptors and region contents separately. It uses the descriptors to detect regions relevant to the query; contents of those regions will be only accessed to retrieve answer tuples afterwards. A region descriptor includes the following elements:

- *region formula*: it describes the region content. It is also called constraint formula. Like a user query, any region formula is a conjunction of terms;
- *region signature*: a binary code assigned to the region formula which allows for a fast comparison among conjunction formulas [11];
- *replacement value*: when a new query arrives, the replacement values are used to detect region(s) which are removed to free a room for the new query;
- a *pointer* to the region content, where the actual tuples are stored;
- *Web source identifier*.

The proposed structure allows for an efficient retrieval of answers to a query to a specific source but one can also search for all answers in the cache independently of the source.

2.2.2. Operational model

The semantic cache plays an important role in the query evaluation. When a user formulates a query Q, the meta-searcher checks it first against the content of the semantic cache. The cache splits the user query into two portions, *probe query* and *remainder query*. The probe query $\mathit{Probe}(Q)$ represents a partial answer to the query provided by the cache; it addresses those cache regions which contain tuples satisfying the query and thus contribute to the answer. The remainder query $\mathit{Rem}(Q)$ refers to data that should be shipped from the server.

If the remainder query is empty, the probe query serves the answer from the cache content and the server is not contacted. If the remainder query is not empty, its evaluation proceeds in a regular way. The answer to the user query is build up from the answers to the probe and remainder queries: $Q = \mathit{Probe}(Q) \cup \mathit{Rem}(Q)$. For example, if the cache contains the region $R =$ “`applet AND java`”, and query is $Q =$ “*applet*”, the probe query coincides with R : $\mathit{Probe}(Q) =$ “`applet AND java`”, while the remainder query is $\mathit{Rem}(Q) =$ “`applet AND NOT java`”.

Once the cache detects some regions relevant to the query and constructs the probe query $\mathit{Probe}(Q)$, the formula Q `AND NOT` $\mathit{Probe}(Q)$ defines the optimal remainder query $\mathit{Rem}_{\mathrm{opt}}(Q)$. However, as we will see in Section 3, the optimal $\mathit{Rem}_{\mathrm{opt}}(Q)$ is not always reachable. In this case, $\mathit{Rem}(Q)$ may ship from the server more than required by $\mathit{Rem}_{\mathrm{opt}}(Q)$; therefore, the following containment holds: $\mathit{Rem}(Q) \supseteq Q$ `AND NOT` $\mathit{Probe}(Q)$.

The support of the cache implies several issues concerning the management of semantic regions. In addition to the region organization, it is necessary to define a coalescing and replacement strategy. The *coalescing strategy* determines how to merge/split the regions to provide the optimal granularity of the cached items; the *replacement strategy* (see Section 4) specifies a policy how to discard some cached regions when a room for a new query is needed.

In what follows we consider four *operational cases* processed by the semantic cache; each case refers to a particular relationship between a user query Q and regions in the cache. Below we list and describe all the cases:

- *Equivalence*: the cache contains a region R which formula is equivalent to the query formula: $Q \equiv R$ (Fig. 2a).
- *Query containment*: the cache contains one or more regions $R_1, \ldots, R_m, m \geq 1$, which formulas

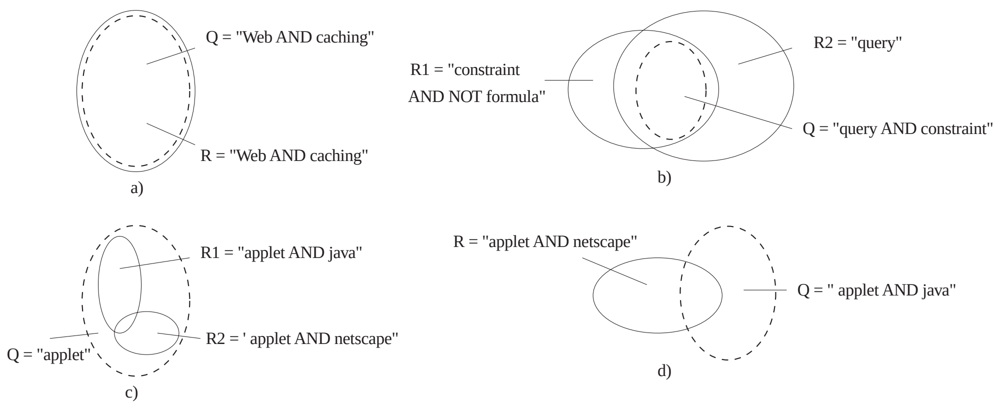


Fig. 2. Four operational cases.

contain the query formula: $Q \subset R_i$ (Fig. 2b); if $m \geq 2$ the regions are assumed to be ordered by the increasing number of tuples satisfying the query. Without loss of generality, we assume that region R_1 is a region with the minimal number of tuples.

- *Region containment*: the cache contains regions $R_1, \ldots, R_m, m \geq 1$, which formulas are contained in the query formula: $R_i \subset Q$ (Fig. 2c).
- *One-term difference* : this is a particular case of the intersection between cache region(s) and the query; region R has a one-term difference from query Q if exactly one term in the region formula is different from terms in the query: $|R_i - Q| = 1$. The difference term is denoted d. In Fig. 2d, region R = "`applet AND netscape`" has one-term difference from the query Q = "`applet AND java`" and the difference term is d = "`netscape`" Similarly, the region R = "`applet`" has a one-term difference from query Q = "`java`" with the difference term being $d = R$ = "applet".

While the three first cases are standard for any semantic cache, the one-term distance case is specific for the Web querying; it represents a particular intersection relationship between a query formula and formulas of cache regions. As the analysis shows, the one-term difference plays an important role and provides the best contribution to the partial answer, as compared to two-and-more term differences. For more detail, we refer the reader to [11] where signature files are exploited as a mean to detect in an efficient way all operational cases listed above.

2.3. Source heterogeneity

A user query is often sent to several destination Web sources; therefore, its answer is composed of the different source answers. In the context of heterogeneous information retrieval, we consider those features of the Web sources which influence the organization and processing of data in the semantic cache. Here we identify two features particularly relevant: the *complete* and *checkable* sources' answers.

2.3.1. Complete answers

To retrieve data, most Web sources implement either the boolean model or ranking model or their combination. With the *Boolean model*, the source returns all tuples satisfying the boolean query. With the *ranking model*, the source ranks the documents relevant to the query and returns first k top-ranked. The choice of the model is often domain- and application-dependent. If the amount of data stored in the Web repository is moderated (**ACM Digital Library**[5]) or the keyword selectivity is high (**Library of Congress**[6]), the boolean model can be successfully used. If instead the repository contains thousands of relevant documents (Altavista search engine), the ranking model becomes indispensable.

A source answer is considered as *complete* if it contains all relevant tuples, otherwise it is *incomplete*. The answer completeness is crucial for the cache manipulations, such as generation of probe

[5] http://www.acm.com/dl/
[6] http://lcweb.loc.gov/z3950/gateway.html

Table 1
Complete and checkable answers: four possible cases

	Checkable	Non-checkable
Complete	NCSTRL (`Title`/`Author` search), DBLP (`Author` search)	Yahoo (category search), IEEE (`Author` search)
Incomplete	Altavista (`Title` search), ACM (answer size $>$ 100 items)	Altavista (`Full-text` search), Microsoft (answer size $>$ 50 items)

and remainder queries. Web sources with the boolean retrieval model often return complete answers, while those with ranking model return incomplete ones. Although Altavista allows one to retrieve *all* answers to the query by following the *next-page* link, the complete answer often requires the retrieval of hundreds of pages, which is not acceptable in most on-line meta-searchers. Therefore, for typical queries, the most relevant (top-ranked) tuples are only considered and the answer is therefore incomplete. On the other side, Altavista asked with a very selective keyword (like `Lethoto`) returns a few items which compose a complete answer. Therefore, for the efficient cache utilization, the completeness of answers should be detected dynamically. We assume the wrappers are capable of detecting the completeness of answers and informing the cache about that.

2.3.2. Checkable answers

Answers returned by a source often represent rather *tuple views* than complete tuples. In other words, they do not necessarily contain the whole information used by the source to evaluate the query. For example, Altavista returns titles of the relevant documents, their URLs and two first lines from the body. When searching Altavista for documents about "`caching`", it is not guaranteed that the word "caching" appears in the returned tuples. Such an answer is considered as *non checkable*. If the source provides the whole information, the answer is *checkable*. The type of a source answer may depend on the query. In Altavista, the answer is non checkable if the query is a full-text search. Instead, the Altavista's answer to a query against a document title, like $Q =$ "`title:caching`", is checkable.

There exist all four possible combinations of complete/incomplete and checkable/non-checkable answers. As we have seen above, Altavista can return answers of any type, although an incomplete-and-non-checkable answer is the most frequent. Other well-known search engines like Yahoo and Infoseek do the same. However, not all Web sources are so poly-valent. The DBLP site always returns checkable results (and often complete), while **NCSTRL**[7] returns complete results (checkable when querying attributes `Author` and `Title` and non-checkable for `Abstract`).

Some Web sources have a fixed maximal number of returned answers. The ACM digital library returns at most 100 documents, while **Microsoft**[8] returns at most 50. Therefore, if ACM digital library has more than 100 documents relevant to a query, then the answer is incomplete, otherwise it is complete. Table 1 cites some typical examples of querying Web sources which yield different answer types.

2.3.3. Multi-source cache

Caching results from multiple Web sources implies a particular management as a user query may concern several sources. Mainly, it raises a choice between storing all results to one query in one semantic region and in as many regions as the sources are.

The one-region solution offers the advantage of keeping all answers to a given query in the same place. However, in this case, the cache has to deal with structural and semantical heterogeneity inside a region, it makes the region reuse difficult. Then, as regions are fairly large, purging a region from the cache can empty it too much; this can result in poor utilization of the available cache memory.

With the multi-region solution, one cache region contains exclusively query answers from one information source and a new query answer can result in creation of multiple new regions in the cache. However, such a solution allows for a much better

[7] http://www.ncstrl.org/Dienst/UI/2.0/Search
[8] http://www.microsoft.com/

region reuse, because of a lower granularity of regions. Additionally, it is possible to perform a more selective replacement of regions as one may consider the qualities of sources when calculating the replacement value of a region. Because of the better overall performance, we accept the multi-region solution in the implementation, and in the following we assume a cached region to contain only results coming from one information source.

3. Handling heterogeneity

In this section we present caching algorithms for Web queries taking into account the classification of source answers given in the previous section.

3.1. Complete-and-checkable answers

We start with the case when the semantic cache yields most advantages, namely, when the answer is complete and checkable. The algorithm proposed below refers to a single source.

Algorithm 1
Input: cache with semantic regions and query Q.
Output: answer to Q and updated cache.
Verify the query Q against all region descriptors in the cache.

- *Equivalence*: $Q \equiv R$: If the query is equivalent to a region R, then return the region content as the query answer, $\mathit{Probe}(Q) = R$. Assign the replacement value $\mathit{Rpc}(R)$ of the region with a *"most-recently-used"* value (which is used by the replacement function when a new query arrives).
- *Query containment*: $Q \subset R_i,\ i = 1, \ldots, m$: If one or more regions contain the query, choose the region R with the minimal cardinality, $R = \min(R_i)$. Check tuples in the region content and return ones matching the query: $\mathit{Probe}(Q) = Q \cap R$. Improve the replacement value $\mathit{Rpc}(R)$ in the proportion to the number of matching tuples.
- *Region containment*: $R_i \subset Q,\ i = 1, \ldots, m$: Return the tuples matching the query in the semantic regions R_i, discarding duplications, $\mathit{Probe}(Q) = \bigcup R_i$. Construct the query remainder as $\mathit{Rem}(Q) = Q - (\bigcup R_i)$ and send it to the server. When the answer is received and reported to the user, coalesce regions R_i with $\mathit{Rem}(Q)$ into one region with formula Q and *"most-recently-used"* replacement value.
- *One-term difference*: $|R_i - Q| = 1,\ i = 1, \ldots, m$: detect term differences d_i for regions $R_i,\ i = 1, \ldots, m$. Return the tuples matching the query in the contents of regions $R_0, \ldots, R_m$, discarding duplications; $\mathit{Probe}(Q) = Q \cap (\bigcup R_i)$. Construct the remainder query $\mathit{Rem}(Q) = Q - (\bigcup d_i)$ and send it to the server. When the answer is received and reported to the user, add a new region with formula $\mathit{Rem}(Q)$ and improve the replacement values of regions R_i in the proportion to the number of matching tuples.

The case order is also a priority order: if two cases are detected, one which appears higher in the list is used. The only exception is when both region containment and one-term difference cases are detected. As they coincide in the evaluation of remainder query, the cache processes them together, that is, a joint remainder query is constructed and sent to the server.

If none of the above cases is detected, the query Q is processed in an ordinary way: it is sent to the server, and when the answer is received and reported to the user, a new region for the query is created in the cache.

Below we represent the cases of Algorithm 1 in a tabular way in order to reuse it later for other cases. Four columns for the operational cases (equivalence, query and region containment and one-term difference) are reported in the order of decreasing priority from left to right. For each operational case, Table 2 gives the formula relationships, probe and remainder queries, changes in cache regions, and replacement values and possible coalescing.

3.2. Incomplete-and-checkable answers

The answer incompleteness changes the processing of two operational cases. Consider an example given in Fig. 3 where the cache contains an incomplete region R with formula "`caching`". The region holds the answer set R^a, which is a proper subset of the tuples at the Web source relevant to the query. For the new query Q = "`Web AND caching`", the query containment holds, thus the cache can use the corresponding tuples in the region R to answer the

Table 2
Caching complete-and-checkable answers

Case	Equivalence	Query containment	Region containment	One-term difference
Formula	$Q \equiv R$	$Q \subset R_i,\ R = \min(R_i)$	$R_i \subset Q,\ i = 1, \ldots, m$	$\lvert R_i - Q\rvert = 1,\ i = 1, \ldots, m$
Probe	R	$Q \cap R$	$\bigcup R_i$	$Q \cap (\bigcup R_i)$
Remainder	$\emptyset$	$\emptyset$	$Q - \bigcup R_i$	$Q - \bigcup d_i$
Cache regions	no change	no change	coalesce Q with R_i	new region $Q - \bigcup d_i$
Replacement	best $Rpc(R)$	improve $Rpc(R)$	best $Rpc(Q)$	improve $Rpc(R_i)$

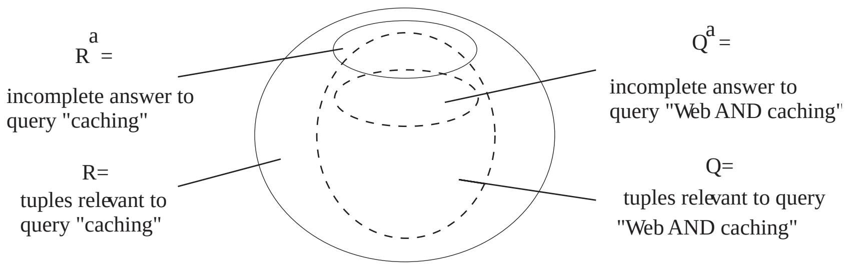

R^a =
incomplete answer to query "caching"
R=
tuples relevant to query "caching"
Q^a =
incomplete answer to query "Web AND caching"
Q=
tuples relevant to query "Web AND caching"

Fig. 3. Caching incomplete result sets.

query. However, there is no guarantee that those tuples form the complete answer to the query. In other words, the query containment $Q \subset R$ does not lead to $Q^a \subset R^a$. Hence, in the query containment case, the initial query should be submitted to the source to retrieve *all* relevant tuples.

Generally speaking, incomplete regions can be used to retrieve the partial answer available from the cache, but they cannot be used in the remainder query.

The answer incompleteness also affects the coalescing strategy. In the query containment case, as $Q \subset R$ does not necessarily lead to $Q^a \subset R^a$, region R and new region for Q can contain different tuples in their contents. In Fig. 3, the region R formula "`caching`" contains the query formula "`Web AND caching`", but the incomplete answer Q^a can contain tuples not present in R^a. Therefore, we do not merge them and prefer to keep two distinct regions "`caching`" and "`Web AND caching`".

Instead, in the region containment case, we still coalesce the remainder query $Rem(Q)$ and regions $R_1, \ldots, R_m$. It can result that the number of tuples in the content of the new region Q will be larger than the source returns, but such behavior only improves the cache performance and relax the limit imposed by the source.

We again use the tabular representation for all operational cases (see Table 3). One important change concerns the priority order (appearance of cases from left to right). In the complete-and-checkable case, the query containment was more preferable than the region containment because it allows for the local query processing. Instead, in the incomplete-and-checkable case, the region containment is verified first as more preferable (it allows to constrain the remainder query) [9].

3.3. *Complete-and-non-checkable answers*

When a *non-checkable* answer is received from a source, the attributes are not sufficiently represented to verify the query satisfiability. In other words, non-checkable answers can be reused for new queries only if no query matching is required. Such a limitation reduces the cache efficiency and allows to reuse cached data only in the equivalence and region containment cases (see Table 4, differences with respect to Table 3 are highlighted).

The non-checkable answers change the coalescing strategy as follows. The region containment is more

[9] In Table 4, we highlighted the difference items with respect to Tabel 3.

Table 3
Caching incomplete-and-checkable answers

Case	Equivalence	**Region containment**	**Query containment**	One-term difference
Formula	$Q \equiv R$	$R_i \subset Q, i = 1, \dots, m$	$Q \subset R_i,\ R = \min(R_i)$	$\lvert R_i - Q\rvert = 1, i = 1, \dots, m$
Probe	R	$\bigcup R_i$	$Q \cap R$	$Q \cap (\bigcup R_i)$
Remainder	$\emptyset$	Q	$\mathbf{Q}$	$\mathbf{Q}$
Cache regions	no change	coalesce Q with R_i	**new region** Q	**new region** Q
Replacement	best $Rpc(R)$	best $Rpc(Q)$	improve $Rpc(R)$	improve $Rpc(R_i)$

Table 4
Caching complete-and-non-checkable answers

Case	Equivalence	Region containment	Query containment	One-term difference
Formula	$Q \equiv R$	$R_i \subset Q, i = 1, \dots, m$	$Q \subset R,\ R = \min(R_i)$	$\lvert R_i - Q\rvert = 1, i = 1, \dots, m$
Probe	R	$\bigcup R_i$	$\emptyset$	$\emptyset$
Remainder	$\emptyset$	$Q - \bigcup R_i$	Q	Q
Cache regions	no change	**new region** $Q - \bigcup R_i$ **no coalesce**	**replace** R **with** Q	new region Q
Replacement	best $Rpc(R)$	**best** $Rpc(R_i)$	**best** $Rpc(Q)$	**no change**

preferable than the query containment, and therefore the cache prefers storing smaller regions to larger ones. It implies that the cache (1) does not merge new region $Q - \bigcup R_i$ with regions R_i in the region containment case and (2) replace a larger region R with a smaller region Q in the query containment case (see Table 4).

It is often possible to convert non-checkable answers into checkable ones. Using the references in tuple views, wrappers can download corresponding pages to complete the cached tuples. However, such a conversion is often very expensive. In the Altavista case, the conversion would require accessing eleven Web pages instead of one. Although the users often prefer fast-and-incomplete answers to slow-and-complete ones, the conversion might be still useful and eventually included into the query language.

3.4. Incomplete-and-non-checkable answers

In the incomplete-and-non-checkable case, we have the combination of the incomplete-and-checkable and complete-and-non-checkable cases discussed in the previous subsections. Actually, as the non-checkable answers constrain the cache reuse more than the incomplete answers, all operational cases for the incomplete-and-non-checkable case are the same as in Table 4.

4. Replacement strategies

The definition of a replacement strategy for the query caching mechanism is influenced by strategies designed for the Web and semantic caches. The experience in caching Web data has an direct impact on our work, as any meta-searcher suffers from the congestion of networks and remote servers, which results in long access latencies for the results transferred over the network. On the other hand, our cache is not structured along pages specified by their URL. Most probably, the characteristics of user traces recording query sessions of information retrieval on Web-based data repositories are different of the characteristics of document access in a Web cache. The analysis of those characteristics and the evaluation of the performance results of a replacement strategy are indispensable to establish a valuable replacement strategy maximizing the hit rate or other performance criteria of a cache. As we don't have this information today we cannot completely validate our proposal.

Algorithms presented in Section 3 assign replacement values to regions with a *"most-recently-used"* approach. The replacement strategy may be based exclusively on that value or consider other information. We identified parameters that may influence the benefit of caching a region and are therefore candidates of parameters of the replacement function.

The size of cached regions. All new replacement algorithms on the Web take into account the size parameter and show that it has a strong influence on the measured hit rate [6,7,22]. Its importance in the Web caching context comes from two facts: (1) storing small items allows to store more items; (2) on the Web, small pages are referenced more frequently [13]. In our case only (1) counts, since in the context of semantic caching we have no experience and valid data about the relation of the size of a result set and the frequency of its use.

Probability of further reuse of region. The benefit of the cache depends much on its hit rate. This is strongly related to the probability of further use of cached regions. When a user refines queries by adding or removing keywords, the regions that were used the most recently have the highest probability to be reused in future. The least-recently-used strategy seems to be the most advantageous. This holds in a single user cache, but not necessarily in a multi-user cache.

In the meta-searcher context, we have to integrate the fact that a cached region may only be partially reused. So the replacement value toward "the most recently used" depends on how large the reused part is. Also if we merge two regions R_1 and R_2, their replacement value should be used to calculate the value of the newly created region (see Section 3).

Retrieval time. Generally, comparing two regions with different retrieval times and the same characteristics otherwise, we should prefer to keep in the cache the region with the longer retrieval time. Instead of storing for each region the time it took to retrieve it, it could be more 'economic' to store once the connection establishment latency and the bandwidth for each information source. This two variables can be measured as proposed in [22]. We propose to keep different values for the two variables depending on the time of the day. For example, we keep 6 values for the bandwidth : *12pm–4am, ..., 8pm–12pm*. This allows for a more fine-grained description of the effectively available bandwidth. This distinction is particularly interesting on the Web, where the traffic and hence also the available bandwidth and connection latency varies dramatically during a day. Those variables can be refreshed in regular intervals using tests effected especially for this purpose or they can be adjusted dynamically, when running the cache.

Region characteristics. We propose to integrate in the replacement value of a region, the knowledge of its completeness and check-ability. As shown in Section 3, the definition of the query remainder for incomplete and non-checkable regions is harder and the benefit of reusing these regions appears to be less interesting than the one obtained by reusing complete-and-checkable regions.

Accessibility of the server. The inaccessibility of a server makes impossible to (re)fetch data from it. Regions containing data issued from temporary inaccessible servers should therefore have higher priority to be kept in the cache. Inaccessibility may be caused by failures or because of fixed opening hours of the server (e.g., Library of Congress); in both cases caching of their data is important.

Based on this analysis we study the definition of a replacement function using a cost/benefit model. Our cache implementation allows to experiment with different functions to be validated. User preferences and expiration time may also be interesting parameters but are not considered at present. The complexity of the query is not considered mainly because we suppose that data transfer time is dominant.

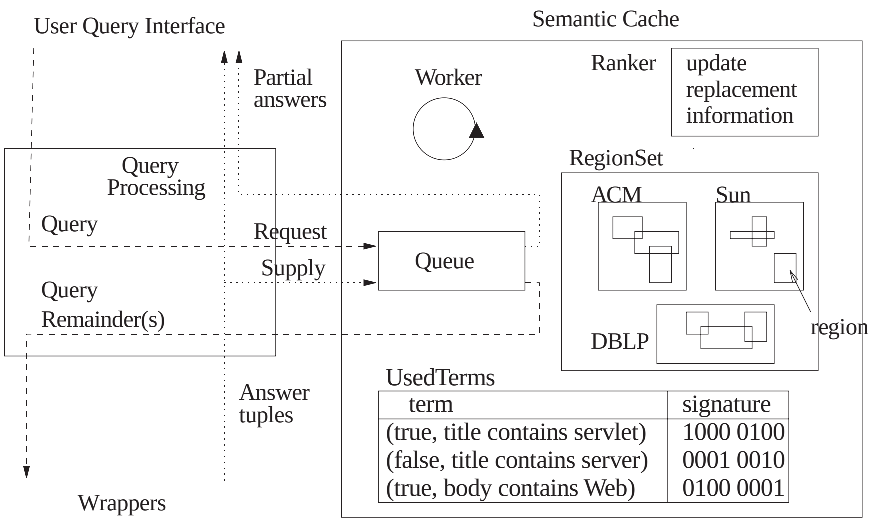
5. Implementation of the semantic cache

The proposed semantic cache mechanism has been implemented in the Knowledge Broker (KB) system [3,4]. This section describes the software architecture of the semantic cache and its integration into the KB system.

Knowledge Broker is a Web meta-searcher developed at the **Xerox Research Centre Europe** [10], it provides a uniform interface and query language for interrogating multiple, heterogeneous data repositories including standard databases (Oracle, Sybase, Informix etc.) and Web providers in different domains (newspapers, digital libraries, patents, medicine, etc.).

The semantic cache has been implemented in Java as an external service to the meta-searcher. In KB

[10] http://www.rxrc.xerox.com/research/ct/prototypes/cbkb



term	signature
(true, title contains servlet)	1000 0100
(false, title contains server)	0001 0010
(true, body contains Web)	0100 0001

Fig. 4. The semantic cache in Knowledge Broker system

(see Fig. 4), it replaces the tuple-based cache mechanism used in the previous versions. Service-like integration of the cache gives the main advantage of a clear separation between the query processing and cache processing. There are two main issues:

(1) it does not require any modification in the existing architecture of the system. The system is simply enhanced with the *SemCacheManager* module which integrates the semantic cache into the system.
(2) The Web query system can decide whether to use the cache and which one (tuple-based or semantic one).

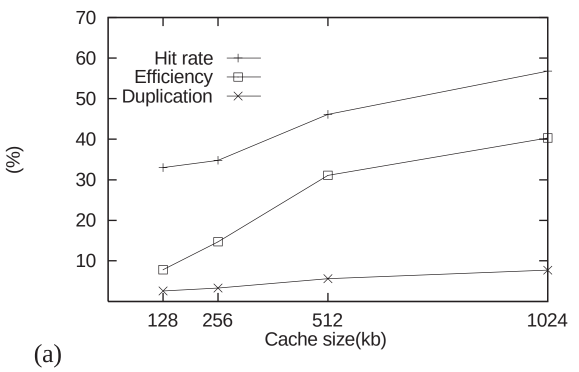
The semantic cache service is provided through an API with two methods, `Request` and `Supply`. The `Request` method takes a query with a set of destination sources and returns a remainder query for each source and the partial answer from the cache. The `Supply` method adds to the cache the answer tuples received from sources. The jobs in the cache (`Request` and `Supply` calls) are queued; a thread picks up the first one and processes it (see Fig. 4).

A KB query is a conjunction of terms which are objects of the `Constraint` class. The method `equals` allows to detect the equivalence of two constraints; to decide whether an answer matches the query, the cache invokes method `filter(Query)` provided by the class `Result`.

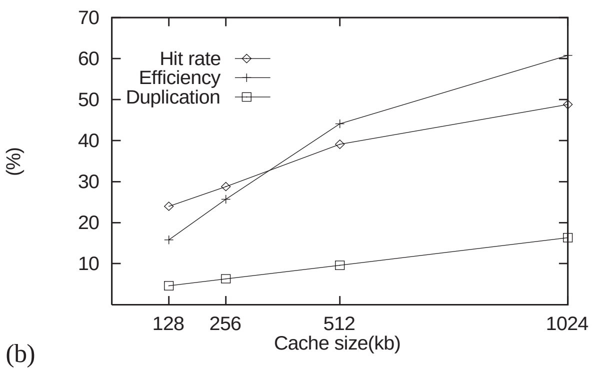
The main data structure in the cache is the `RegionSet` object which contains all regions currently stored in the cache. All terms used in the cached queries are maintained in the `UsedTerms` object; it controls the uniqueness of signatures assigned to query terms, as required for the comparison of constraint formulas in all operational cases. The object `RegionSet` contains separated sets of regions; one set for each source. A query with several destination sources is split into queries with a single destination source. For each of those queries, the cache applies the algorithms from Section 3 and considers the region set for the corresponding source.

The cache provides a highly adaptable support for the replacement strategy. The object `RepInfo` associated to each cached region and containing replacement information is used by the object `Ranker` that calculates the effective replacement value of all cached regions. Those replacement information contained in `RepInfo` can be based on parameters as cited in Section 4. Depending on the specific usage of the meta-searcher integrating the cache the replacement function implemented in the `Ranker` can be adjusted accordingly to reach optimal hit rate and performance. The current cache implementation integrated in the KB system is mainly based on a LRU-like strategy presented in [14,11].

Experiments. To see how the semantic cache works, we have tested two different query sets



(a)



(b)

Fig. 5. Semantic cache measurements: (a) random query set; (b) user-log query set.

against real Web sources. In the first, *random* set, a series of S queries is randomly generated, and each query contains one to three terms with the equal probability for each case. If more than one term is included, the last term is negated or not with the equal probability. The query keywords are applied to the attribute `Title` and chosen randomly from a dictionary of about 90 terms in computer science; the keywords have been mainly taken from the **Yahoo Classifier**[11]. In the second, *user-log* set, real user queries have been collected over 500 user sessions on the KB meta-searcher installed at XRCE and cut into series of S queries.

As a Web source, we have used the ACM Digital Library. In the tests we measure the hit rate, cache efficiency and duplication ratio of the cache when varying the cache size. The *hit rate* is the proportion of queries answered (completely or partially) by the cache to all processed queries. The *cache efficiency* is the proportion of answer tuples coming from the cache to all answers returned to the user. The *duplication ratio* is the percentage of duplicated tuples in the cache. All measures are evaluated on average over a series of $S = 200$ queries. Fig. 5 shows all three measures for both query sets against the ACM server.

In the first set, the randomness of queries results in a higher hit rate coupled with a lower efficiency, as compared to the second query test. This phenomenon can be explained as follows. The queries in the random set are rather uniformly distributed in a virtual query space, and a new query hits many regions but each region hit by the query contains a few relevant tuples. Instead, real user queries appear to be semantically rich and tend to create clusters. That is, in the user-log test, a new query hits less cache regions than in the random query set, but the probability to get a fruitful answer from the cache is considerably higher.

The time overhead for the KB meta-searcher when using the cache for processing a query with a single destination source is evaluated to 30 ms on average, for both query sets. The tests have been done on a SPARCstation-10 running SunOs 5.6.

6. Related work

Data caching have been intensively studied in the context of database and information retrieval systems [2,8,18]. In the introduction, we have argued that semantic caching have over page and tuple caching in the case of querying Web sources. On the other side, our analysis on the replacement and coherence strategy origins from page caching methods used on the Web [6,7,22].

A semantic model for the query caching in a client–server architecture was discussed in [14]. It introduced the semantic query framework and its main principles and components (regions, probe and remainder queries etc.). However, [14] considers semantic cache mainly for data stored relational databases.

Query caching in heterogeneous systems was discussed in [15], where it is reduced to a Datalog query evaluation, which, however, may by computationally

[11] http://www.yahoo.com/Science/Computer_Science

hard. Intelligent query caching is also used in the SIMS project [5], where some important principles for any intelligent caching mechanism were developed. These principles are the following: (1) a query cache should process both containment and intersection cases; (2) a cache item should not be large; (3) a cache item should have a simple formula to avoid too complex reasoning on the query remainders.

Query caching in the HERMES distributed mediator system has been studied in [1]. It is based on the invariant mechanism and uses query rewriting techniques and semantic information about sources to collect some source statistics and build optimal query plans. However, the mechanism assumes the subquery equivalence and does not consider the containment and intersection cases.

In [11], a semantic cache mechanism for Web queries based on *signature files* has been proposed. The method uses signature-based region descriptions to efficiently manage both containment and intersection cases. However, the cache structure proposed in [11] is for interrogating one information source only.

The problem of reusing cached entries in the LDAP (Lightweighted Directory Access Protocol) network directories have been recently studied in [12]. The LDAP entries are organized into a hierarchical name-space and accessed by using positive conjunctive queries. In [12], a sound and complete algorithm is proposed for determining whether a conjunctive LDAP query can be answered using cached queries. Unlike [12], a query language considered in our work is typical for Web search engines; it allows us to reduce Web queries to conjunctive formulae in prepositional logic, thus considering a larger set of operational cases and better reusing the cached queries and constraining the remainder queries.

7. Conclusion

We have presented a semantic cache mechanism designed for meta-searchers querying heterogeneous Web repositories. Query caching allows reuse of answers to previous queries, so reducing the delivery time of answers and the traffic on the net.

Semantic caching is based on the representation of cached data as semantic regions and the processing of queries by construction of probe queries for retrieving cached data and remainder queries for fetching data from remote servers. We have identified two problems particularly related to the multi-source Web querying, namely, the completeness and check-ability of answers from Web sources. We proposed a cache architecture for caching multi-source queries and consider all operational cases (equivalence, containment and intersection) when working with incomplete and non-checkable answers. For all types of answers we have developed algorithms for query evaluation against the cache content and the region management. We have also analyzed the parameters relevant to the replacement strategy in the cache. Due to lack of space the coherence strategy was not discussed.

We have described the implementation of the semantic cache in the Knowledge Broker system. We have conducted a set of tests to measure the cache performance and obtained promising results.

Our future work includes a larger testing of the semantic cache mechanism in the presence of real user queries. We also plan to experiment with different replacement and coherence strategies in order to tune them better to the real user profiles. One important research issue is related to the extension of our semantic cache to a more powerful query language which includes complete boolean expressions, word proximity operators and stemming [9].

Acknowledgements

We thank L. Julliard for useful discussions and large support of the semantic cache implementation in the Knowledge Broker meta-searcher.

References

[1] S. Adali, K.S. Candan, Y. Papakonstantinou and V.S. Subrahmanian, Query caching and optimization in distributed mediator systems, in: Proc. SIGMOD'96 Conference, 1996, pp. 137–148.

[2] R. Alonso, D. Barbara and H. Garcia-Molina, Data caching issues in an information retrieval system, ACM TODS 15 (3) (1990) 359–384.

[3] J.-M. Andreoli, U. Borghoff and R. Pareschi, Constraint-based knowledge broker model: semantics, implementation and analysis, Journal of Symbolic Computation 21 (4) (1996) 635–667.

[4] J.-M. Andreoli, U. Borghoff, P.-Y. Chevalier et al., The constraint-based knowledge broker system, in: Proc. 13th IEEE Int. Conf. on Data Engineering, 1997.
[5] Y. Arens and C.A. Knoblock, Intelligent caching: selecting, representing, and reusing data in an information server, in: Proc. CIKM'94 Conference, Gaithersburg, MA, 1994, pp. 433–438.
[6] M. Baentsch, G. Molter and P. Sturm, Introducing application-level replication and naming into today's Web, Computer Networks and ISDN Systems 28 (1996) 921–930.
[7] J.-C. Bolot and P. Hoschka, Performance engineering of the World Wide Web: application to dimensioning and cache design, Computer Networks and ISDN Systems 28 (1996) 1397–1405.
[8] M.J. Carey, M.J. Franklin, M. Livny and E.J. Shekita, Data caching tradeoffs in client–server DBMS architectures, in: Proc. 1991 SIGMOD Conference, 1991, pp. 357–366.
[9] C.-C.K. Chang, H. Garcia-Molina and A. Paepcke, Boolean query mapping across heterogeneous information sources, IEEE Trans. on Knowledge and Data Engineering 8 (4) (1996).
[10] C.-C.K. Chang and H. Garcia-Molina, Evaluating the cost of Boolean query mapping, in: Proc. 2nd ACM Int. Conf. on Digital Library, 1997.
[11] B. Chidlovskii and U. Borghoff, Signature file methods for semantic query caching, in: Proc. 2nd European Conf. on Digital Libraries, LNCS 1513, Crete, Greece, September 1998.
[12] S. Cluet, O. Kapitskaja and D. Srivastava, Using LDAP directory caches, in: ICDT Workshop on Query Processing for Semistructured Data and Non-Standard Data Formats, Jerusalem, Israel, January 1999.
[13] C. Cunha, A. Bestavros and M. Crovella, Characteritics of WWW client-based traces, Tech. Report TR-95-010, Boston University, 1995.
[14] S. Dar, M.J. Franklin, B. Jonsson, D. Srivastava and M. Tan, Semantic data caching and replacement, in: Proc 22nd VLDB Conference, Bombay, India, 1996, pp. 330–341.
[15] P. Godfrey and J. Gryz, Semantic query caching for heterogeneous databases, in: Proc. 4th KRDB Workshop Intelligent Access to Heterogeneous Information, Athens, Greece, 1997.
[16] L. Gravano, and Y. Papakonstantinou, Mediating and metaserching on the Internet, Bulletin IEEE Computer Society on Data Engineering (1998).
[17] A.Y. Levi and A. Rajaraman and J.J. Ordille, Quering heterogeneous information sources using source descriptions, in: Proc. 22nd VLDB Conference, Bombay, India, 1996, pp. 251–262.
[18] P.T. Martin and J.I. Russell, Data caching strategies for distributed full text retrieval systems, Information Systems (16) (1) (1991) 1–11.
[19] Y. Papakonstantinou, H. Garcia-Molina and J. Ullman, MedMaker: a mediation system based on declarative specifications, in: Proc ICDE '96 Conference, 1996, pp. 132–141.
[20] Ch. Reck and B. König-Ries, An architecture for transparent access to semantically heterogeneous information sources, in: Proc. Cooperative Information Agents, Lect. Note Comp. Science 1202 (1997).
[21] A. Yoshida, MOWS: Distributed Web and cache server in Java, Computer Networks and ISDN Systems 29 (8–13) (1997) 965–976.
[22] R.P. Wooster and M. Abrams, Proxy caching that estimates page load delays, Computer Networks and ISDN Systems 29 (1997) 977–986.

Boris Chidlovskii received the MSc and PhD degrees in computer science from Kiev University, Ukraine. He was with the Department of Computer Science at Kiev University and with the Department of Computer Engineering at Salerno University, Italy. He co-authored the book *Indexing Techniques for Advanced Database Systems* (Kluwer, 1997). He is now a member of the Scientific Stuff with Xerox Research Centre Europe, Grenoble, France.

Claudia Lucia Roncancio is Assistant Professor at INPG (Institut National Polytechnique de Grenoble), France. She is a member of LSR-IMAG laboratory. She received the PhD degree in Computer Science from University J. Fourier, Grenoble, France. She was a member of technical staff at Bull. Her research interests include active systems, databases and Internet-based information retrieval systems.

Marie-Luise Schneider is software engineer at Xerox Research Centre Europe, Grenoble Lab, France. She received a double diploma in computer science from ENSIMAG (École Nationale Supérieure d'Informatique et Mathématiques Appliquées Grenoble), France and University of Karlsruhe, Germany.