Optimizing web queries through Semantic Caching

Boris Chidlovskii†, Claudia Roncancio‡ and Marie-Luise Schneider†

†Xerox Research Centre Europe, Grenoble Laboratory, France
E-mail: {chidlovskii, schneider}@xrce.xerox.com
‡ ENSIMAG / Lab. LSR IMAG, Grenoble, France
E-mail: Claudia.Roncancio@imag.fr

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 inefficient 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.

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 none search service can be universally efficient or even complete, this has triggered the appearance of a new type of 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 [23, 24, 25]. Figure 1 shows a typical 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, 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, 18].

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 save user's money.

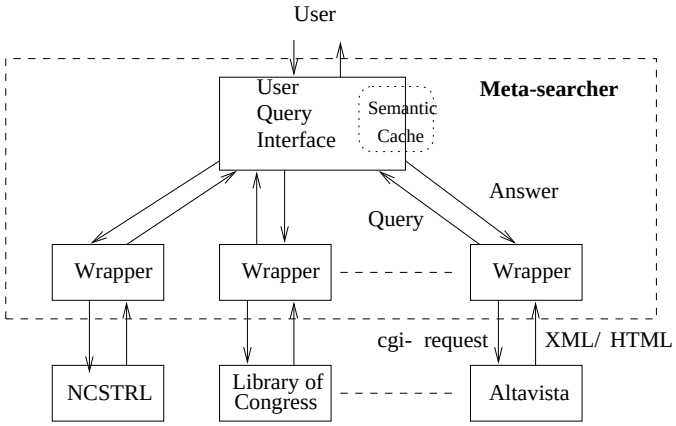

Figure 1: A meta-searcher architecture

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* [26]. 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 can not 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.

Semantic caching. To overcome the drawbacks of page and tuple caching when querying Web sources, semantic cache has been proposed by different researchers [14, 12]. 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) 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 [8].

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 the same domain 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 [20, 10, 11]. 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 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[1]) or split into a sequence of linked pages (like in Altavista[2]). Therefore, different representations of answers attribute different semantics to the cached data, thus changing the ways it is reused for new queries.

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. We analyze and identify the main characteristics of Web-based sources which influence the cache management, namely source completeness and checkability, 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 completeness and checkability of Web sources, while Section 3 proposes the corresponding algorithms. In Section 4, we study the replacement and coherence strategies 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-searcher, the semantic cache architecture and how the heterogeneity of query answers from different sources influences the cache management.

[1] Database and Logic Programming site at
`http://www.informatik.uni-trier.de/~ley/db/index.html`

[2] `http://altavista.digital.com/`

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

$$Q = \texttt{Title}\ \textit{Contains}\ \texttt{Web AND Title}\ \textit{Contains}\ \texttt{caching}.$$

Note the conjunction can contain negated terms, for example,

$$\texttt{Title}\ \textit{Contains}\ \texttt{applet AND NOT Title}\ \textit{Contains}\ \texttt{netscape}.$$

Such a query will retrieve documents containing the keyword `applet` in titles, but not `netscape`.

When a query answer is returned by a Web source, we assume that the wrapper converts the answer in a list of tuples where each tuple contains attribute-value pairs. By this assumption, we will be able to easily check whether an answer tuple satisfies query terms.

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.Tuples contained in a region are issued from a single source.

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 identifier* : beyond the region id, a binary signature can be assigned to the region formula which allows for a fast comparison among conjunction formulas (for more detail on the signature construction and use, see [12]);
- *replacement value* : when a new query arrives, the replacement values are used to detect region(s) which are removed to free room for the new query;
- a *pointer* to the region content, where the actual tuples are stored;
- *Web source name*.

The proposed structure allows for an efficient retrieval of answers to a query coming from a specific information source using the name of the originating source and the constraint formula, but we can also search for all answers in the cache to a certain query, 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 $Probe(Q)$ represents the formula describing the set of answers provided by the cache; it addresses those cache regions which contain tuples satisfying the query and thus contribute to the answer. The remainder query $Rem(Q)$ refers to data that should be shipped from the source server.

If the remainder query is empty, the probe query serves the answer from the cache content and the source 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: $Q = Probe(Q) \cup Rem(Q)$.[3] For example, assume that the cache contains the region

R= `Title` *Contains* `applet` AND `Abstract` *Contains* `java`,

and query is

Q =`Title` *Contains* `applet`.

For brevity, we denote the terms `Title` *Contains* `applet` and `Abstract` *Contains* `java` by letters `A` and `J`, respectively. Then, the probe query coincides with R : *Probe(Q)*= `A AND J`, while the remainder query is *Rem(Q)*= `A AND NOT J`. Remark that the portion of the answer obtained locally with the probe query may be showed immediately to the user.

Generally, once the cache detects some regions relevant to the query and constructs the probe query, the remainder query $Rem(Q)$ can be defined by the formula Q `AND NOT` *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 new query arrives.

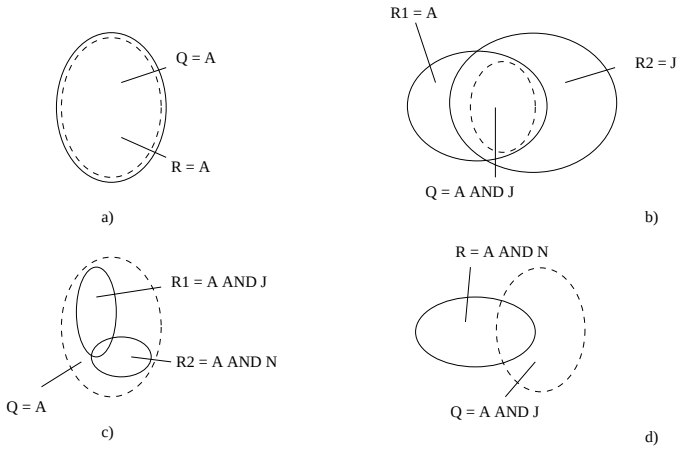

Figure 2: Four operational cases.

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

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

[3]The answers to the probe and the reminder queries may not be disjoint.

- *Region containment* : the cache contains regions $R_1, \ldots, R_m, m \geq 1$, which formulas are contained in the query formula: $R_i \subset Q$ (Figure 2.c).
- *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 Figure 2.d, region $R =$ `A AND N`, where letter `N` denotes the term `Abstract` *Contains* `netscape`,[4] has one-term difference from the query $Q =$ `A AND J` and the difference term is `N`, $d =$ `N`. Similarly, the region $R =$ `A` has a one-term difference from query $Q =$ `J` and the difference term is $d =$ `A`.

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 [12] 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, the answer to the query 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 *completeness* and *checkability* of answers from the sources.

Completeness. 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*. Completeness of an answer is considered with respect to the information the source owns. The answer completeness is crucial for the cache manipulations, such as generation of probe and remainder queries. Web sources with the boolean retrieval model often return complete answers, while those with ranking model return incomplete ones. Although the Altavista search engine 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 to detect the completeness of answers and inform the cache about that.

[4] We remind that letters `A` and `J` stand for query terms `Title` *Contains* `applet` and `Abstract` *Contains* `java`, respectively.

[5] `http://www.acm.com/dl/`

[6] `http://lcweb.loc.gov/z3950/gateway.html`

	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)

Table 1: Completeness and checkability: four possible cases.

Checkability. 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 URL's 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 answer checkability is often query-dependent. In Altavista, the answer is noncheckable 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/noncheckable answers. As we have seen above, Altavista can return answers of any type, although an incomplete-and-noncheckable answer is the most frequent. Other well-known search engines like Yahoo, 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 noncheckable 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.

Multi-source cache. Caching results from multiple Web sources implies a particular management as a same query may concern several sources. This 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, the cache has to deal with structural and semantical heterogeneity inside a region. In any case the origin source of a tuple needs to be known by the cache as some queries specify specific data repositories.

The multi-region solution corresponding to the same query but each region containing exclusively results from one information source has the following features. Regions are naturally smaller. When having too large regions, we empty the cache too much when purging it from one region. This can result in poor utilization of the available cache memory. Additionally, it is possible to perform a more selective replacement of regions as we may consider the qualities of the originating information source when calculating the replacement value of a region. In the following we accept the multi-region solution, that is, a cached region contains only results coming from one information source.

This choice is also beneficial with respect to the case if we had one cache per information source. In a one-cache-per-source solution we would have to statically fix the size of each

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

cache. Instead, in our solution the globally available cache space is divided dynamically among the different information sources depending on actual access patterns.

3 Handling heterogeneity

In this section we present caching algorithms for Web queries taking into account the completeness and checkability of their answers.

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 destination source.

Algorithm 1.
Input: cache with semantic regions and query Q.
Output: answer to Q and updated cache.

1. 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, $Probe(Q) = R$. Assign the replacement value $Rpc(R)$ 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: $Probe(Q) = Q \cap R$. Improve the replacement value $Rpc(R)$ 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, $Probe(Q) = \cup_i R_i$. Construct the query remainder as $Rem(Q) = Q - (\cup_i R_i)$ and send it to the source server. When the answer is received and reported to the user, coalesce regions R_i with $Rem(Q)$ 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; $Probe(Q) = Q \cap (\cup_i R_i)$. Construct the remainder query $Rem(Q) = Q - (\cup_i d_i)$ and send it to the source server. When the answer is received and reported to the user, add a new region with formula $Rem(Q)$ 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 is higher in the list is used. One 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 source server.

   If none of the above cases is detected, the query Q is processed in an ordinary way: it is sent to the source 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 for other cases in following subsections. 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.

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

Table 2: Caching complete-and-checkable answers

3.2 Incomplete-and-checkable answers

The answer incompleteness changes the processing of two operational cases. Consider an example given in Figure 3 where the cache contains an incomplete region R with formula `A`. 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 $Q =$ `A AND J`, the query containment holds, thus the cache can use the corresponding tuples from R^a to answer the 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.

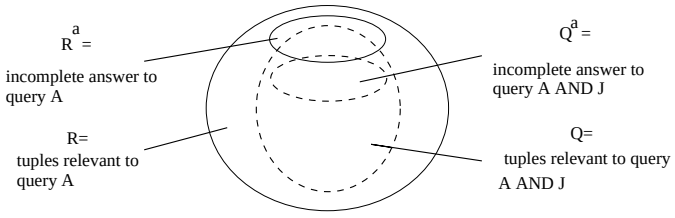

Figure 3: Caching incomplete result sets.

Generally speaking, incomplete regions can be used to retrieve the partial answer available from the cache, but they can not 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 Figure 3, the formula `A` of region R contains the query formula `J AND A`, 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 `A` and `J AND A`.

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 that 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).

Case	Equivalence	Region containment	Query containment	One-term difference
Formula	$Q \equiv R$	$R_i \subset Q$, $i = 1, \ldots, m$	$Q \subset R_i$, $R = min(R_i)$	$\|R_i - Q\| = 1$, $i = 1, ..., m$
Probe	R	$\cup_i R_i$	$Q \cap R$	$Q \cap (\cup_i R_i)$
Remainder	$\emptyset$	Q	Q	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 3: Caching incomplete-and-checkable answers

3.3 Complete-and-noncheckable 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).

The non-checkability of answers changes the coalescing strategy as follows. The region containment is more 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 $Q - \cup_i R_i$ 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).

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

Table 4: Caching complete-and-noncheckable answers

Generally, it is possible to convert noncheckable answers into checkable ones. Using the references in tuple views, wrappers can download corresponding pages to complete the cached tuples. However, the conversion is often very expensive. In the case of Altavista that provides 10 tuple views per answer page, the conversion would require accessing 11 Web pages (1 access to Altavista and 10 accesses to referred pages) instead of one. Although the users often prefer fast-and-noncheckable answers to slow-and-checkable ones, the conversion might be still useful and eventually included as an option into the query language.

3.4 Incomplete-and-noncheckable answers

In the incomplete-and-noncheckable case, we have the combination of the incomplete-and-checkable and complete-and-noncheckable cases discussed in the previous subsections. Actually, the noncheckable answers constrain the cache reuse more than the incomplete answers in all cases but the region containment. As a result, Table 5 inherits three operational cases for the complete-and-noncheckable case (Table 4) and the region containment from the incomplete-and-checkable case (Table 3).

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

Table 5: Caching incomplete-and-noncheckable answers

4 Replacement and coherence strategies

4.1 Defining a replacement strategy

The definition of a replacement strategy for our caching context is influenced by both the replacement strategies in Web caches and the suggested strategies in the literature about semantic caching. The experiences of Web caching replacement have an impact on our work, as we have also to suffer from the congestion of networks and 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 can not completely validate our proposal.

Algorithms presented in Section 3 assign replacement values to regions with a *"least-recently-used"* (LRU) approach. The replacement strategy may be based exclusively on that value or consider other informations. 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, 27]. 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 [27]. 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*, *4-8am*,..., *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 checkability. 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.

User preferences. As users know the best which information is important for them, his knowledge could be integrated in the replacement policy. The user information may concern (1) the subjective estimation of further reuse of a query result, (2) a preference for results coming from certain data repositories, (3) a fixed requirement of the staleness of the results returned by the cache(as claimed in [15]). Nevertheless, those preferences were never taken into account in caches so far, because the average user can not provide reasonable information, which will improve the percepted performance of the cache. On the other hand, it is difficult to integrate different preferences in a multi-users cache.

Expiration time of data. The "expiration time" of cached data influences its relative value. Stale data is, in principle, less interesting for the user and have therefore less priority to be kept in the cache. Nevertheless it is not so simple as it depends of the kind of data and the coherency policy, i.e, coherency of cached data with respect to data in the original server. The problem is different if we handle extremely "changing" data or very "stable" ones (see Section 4.2).

We don't consider the complexity of the query as a parameter to be integrated in the replacement function. The reason is two-fold: (1) information about evaluation and optimization capabilities of the sources is not easily available; (2) we suppose that data transfer time is dominant.

Based on this analysis we study the definition of a replacement function using a cost/benefit model. We consider the following parameters: size of regions, retrieval time, probability of further reuse, accessibility of the server and region characteristics. User preferences and expiration time are not considered at present. Our implementation of the cache presented in Section 5 allows to experiment with different functions to be validated.

4.2 Coherence strategy

General speaking, caching poses the problem of maintaining the coherence of cached data and original data in the server. A strict coherence policy implies that the cache contains only up-to-date data. It is impossible to guarantee such a strict coherency when querying several heterogeneous and autonomous web repositories.

Web data repositories as Altavista have also the problem of keeping the served data up-to-date with the current state of the information available on the Web. On the other hand, libraries accessible over the Web, as the Library of Congress, do not handle so much volatile information as Web documents. Typically, libraries add only information to the existing one but hardly modify stored data. For cached results retrieved from libraries we can suppose that our cached data is not incorrect but contains possibly not all answers currently available at the information source. In this situation, the problem of cache coherence is not a prevailing problem.

Nevertheless, let's consider the coherency mechanisms used in web caches [6, 15]. In the first general approach the cache verifies for each access, whether the cached data is up-to-date. This mechanism has advantages in Web caches (page caches), as one can use the header "If-Modified-Since: date" in a HTTP GET message. The cost of this verification is less than retrieving once again the document. We can not use this approach in our semantic cache, as the information servers that we access do not implement such a mechanism. A verification of the cached data at each access involves the same costs as retrieving the data directly at each access and even more as we have to compare the results from the cache with the results from the source. This coherence mechanism is hence not practicable in our context.

Another approach is to consider an expiration time for cached items. The cache can calculate this expiration time depending on the informations about the "Last-Modification-Date" and other information related with the probability of further changes of the data. When the information source does not deliver such information, the cache can fix a default expiration time for each cached region retrieved from that source. This approach is more appropriate to our context: we propose to fix a default expiration time for cached regions depending on the originating source. The use of the cache will allow to adjust these default values in order to balance the hit rate of the cache and the probability of stale information in the cache.

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,[9] 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 (see Figure 4), it replaces the tuple-based cache mechanism used in the

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

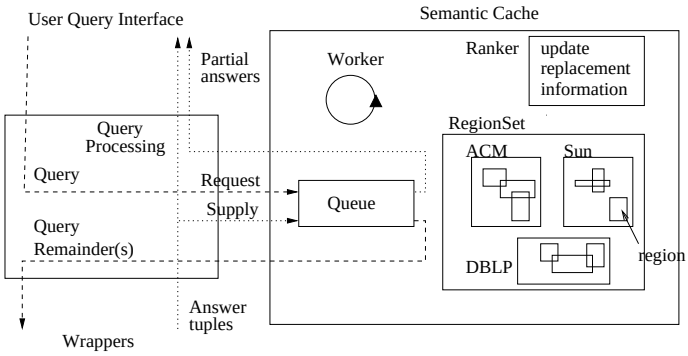

Figure 4: The semantic cache in Knowledge Broker system

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 Figure 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. 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 Sect. 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 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, 12].

Experiments. To see how the semantic cache works, we have tested two different query sets 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.[10] 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. Figure 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.

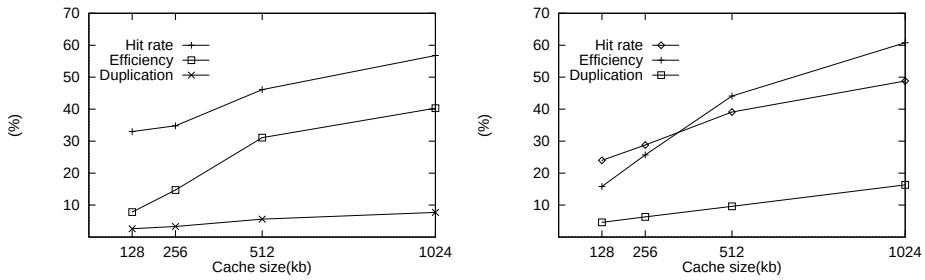

Figure 5: Semantic cache measurements: a) random query set; b) user-log query set

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 management systems [9]. In the introduction, we have argued that semantic caching have advantages 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, 27].

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 in relational databases.

Query caching in heterogeneous systems was discussed in [18], where it is reduced to a Datalog query evaluation, which, however, may by computationally 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

[10] `http://www.yahoo.com/Science/Computer_Science`

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 [12], 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 [12] is for interrogating one information source only.

7 Conclusion

We have presented a semantic cache mechanism designed for meta-searchers querying heterogeneous Web repositories. Query caching allows to reuse answers to previous queries, so reducing the deliver 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 checkability 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 noncheckable 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 and coherence strategy in the cache.

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 [10].

Acknowledgment. We thank Laurent 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, V. S. Subrahmanian. Query Caching and Optimization in Distributed Mediator Systems. In *Proc. SIGMOD'96 Conference*, pp. 137-148, 1996.

[2] R. Alonso, D. Barbara, H. Garcia-Molina. Data Caching Issues in an Information Retrieval System. *ACM TODS*, Vol. 15, No. 3, pp. 359-384, 1990.

[3] J.-M. Andreoli, U. Borghoff, R. Pareschi. Constraint-Based Knowledge Broker Model: Semantics, Implementation and Analysis. In *Journal of Symbolic Computation*, Vol. 21, No.4, pp. 635-667, 1996.

[4] J.-M. Andreoli, U. Borghoff, P.-Y. Chevalier et all. The Constraint-Based Knowledge Broker System. In *Proc. 13th IEEE Int. Conf. on Data Engineering*, April 7-11, 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, Maryland, pp. 433-438, 1994.

[6] M. Baentsch, G. Molter, P. Sturm Introducing application-level replication and naming into today's Web. In *Proc. Computer Networks and ISDN Systems*, N. 28, pp. 921 - 930, 1996.

[7] J-C. Bolot, P. Hoschka Performance engineering of the World Wide Web: Application to dimensioning and cache design. In *Proc. Computer Networks and ISDN Systems*, N. 28, pp. 1397 - 1405, 1996.

[8] U. M. Borghoff, R. Pareschi, F. Arcelli, F. Formato. Constraint-Based Protocols for Distributed Problem Solving. In *Science of Computer Programming*, vol. 30, 201-225, 1998.

[9] M. J. Carey, M. J. Franklin, M. Livny, E. J. Shekita. Data Caching Tradeoffs in Client-Server DBMS Architectures. In *Proc. 1991 SIGMOD Conference*, pp. 357-366, 1991.

[10] C.-C. K. Chang, H. Garcia-Molina, A. Paepcke. Boolean Query Mapping Across Heterogeneous Information Sources. In *IEEE Transaction on Knowledge and Data Engineering*, vol.8, N. 4, 1996.

[11] C.-C. K. Chang and H. Garcia-Molina. Evaluating the Cost of Boolean Query Mapping. In *Proc. 2nd ACM Intern. Conf. Digital Library*, 1997.

[12] B. Chidlovskii, U. Borghoff. Signature File Methods for Semantic Query Caching. In: *Proc. of the 2nd European Conf. on Digital Libraries*, LNCS 1513, September, 1998, Greece.

[13] C. Cunha, A. Bestavros, M. Crovella Characteritics of WWW client-based traces. In *Tech. Report TR-95-010*, Boston University,1995.

[14] S. Dar, M. J. Franklin, B. Jonsson, D. Srivastava, M. Tan. Semantic Data Caching and Replacement. In *Proc 22nd VLDB Conference*, Bombay, India, pp. 330-341, 1996.

[15] A. Dingle, T.Partl. Web cache coherence. In *Proc. Computer Networks and ISDN Systems*, N. 28, pp.907 - 920, 1996.

[16] C. Faloutsos. Signature files: Design and Performance Comparison of Some Signature Extraction Methods. In *Proc. SIGMOD'85 Conference*, pp. 63-82, 1985.

[17] C. Faloutsos and S. Christodoulakis. Signature Files: An Access Method for Documents and Its Analytical Performance Evaluation. In *ACM Trans. of Information Systems*, Vol. 2, n. 4, pp. 267-288, 1984.

[18] P. Godfrey and J. Gryz. Semantic Query Caching For Heterogeneous Databases. In *Proc. 4th KRDB Workshop "Intelligent Access to Heterogeneous Information"*, Athens, Greece, pp.6.1-6.6, 1997.

[19] D. L. Lee, Y. M. Kim and G. Patel. Efficient Signature File Methods for Text Retrieval. In *IEEE Transactions on Knowledge and Data Engineering*, Vol. 7, No. 3, pp. 423-435, 1995.

[20] A. Y. Levi, A. Rajaraman, J. J. Ordille. Quering Heterogeneous Information Sources Using Source Descriptions. In *Proc. 22nd VLDB Conference*, Bombay, India, 1996, pp.251-262.

[21] P. T. Martin and J. I. Russell. Data caching strategies for distributed full text retrieval systems. In *Information Systems*, Vol.16, No. 1, pp. 1-11, 1991.

[22] A. Paepcke, S. B. Cousins, H. Garcia-Molina, et al. Towards Interoperability in Digital Libraries: Overview and Selected Highlights of the Stanford Digital Library Project. In *IEEE Computer Magazine*, Vol. 29, No.5, 1996.

[23] Y. Papakonstantinou, A. Gupta, H. Garcia-Molina, J. Ullman. A Query Transaction Scheme for Rapid Implementation of Wrappers. In *Proc DOOD'95 Conference*, LNCS, 1013, pp. 161-186, 1995.

[24] Y. Papakonstantinou, H. Garcia-Molina, J. Ullman. MedMaker: A Mediation System Based on Declarative Specifications. in *Proc ICDE'96 Conference*, pp.132-141, 1996.

[25] 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*, Vol. 1202, 1997.

[26] A. Yoshida. MOWS: Distributed Web and Cache Server in Java. In *Computer Networks and ISDN Systems*, Vol. 29, No. 8-13, pp. 965-976, 1997.

[27] R.P. Wooster, M.Abrams. Proxy caching that estimates page load delays In *Proc. Computer Networks and ISDN Systems*, number 29, pp. 977 - 986, 1997.