REVISTA COLOMBIANA DE COMPUTACIÓN
Volumen 9, número 1
Págs. 33 - 57

Semantic caching in large scale querying systems

Laurent d'Orazio[*] Claudia Roncancio[*] Cyril Labbé[*]
Fabrice Jouanot[*]

Fecha de Recibido: 08/01/2008; Fecha de Aprobación: 15/03/2008

Abstract

Caching is crucial to improve performances in many computing systems. This work proposes a caching approach for improving query evaluation. The proposed solution follows a semantic oriented approach and combines query and object caching. Separating query and objects provides high flexibility, by making possible to use several cooperations between caches and by supplying different query management tools. Our proposition has been experimented in a grid data management middleware and seems promising as showed by the results obtained with a bioinformatics data bank.

Keywords: *semantic, cooperation, grid, data querying.*

1 Introduction

Important work has been done in several areas to provide effective solutions for data sharing in large distributed systems. Nevertheless querying a data grid remains a challenging task. Neither brand new cluster technology, nor network bandwidth increase can ensure a good query execution time in many scientific domains where many clients search for large amount of data distributed on many cluster nodes, and moreover, on distant grid clusters. These data are generally used in heavy processing tasks and therefore accessing data should not be time consuming. It is important to let grid users (scientists or not) focusing on their job tasks by providing short time for data access. It means grid middleware has to optimize the use of the grid by distributing the load to the different available and relevant resources. This paper describes a contribution to improve data transfer and processing in query evaluation on data grid using an advanced caching approach. The Gedeon middleware is used to experiment our caching solution. It offers a flexible architecture which is easily deployed with DB query

[*] Laboratoire d'Informatique de Grenoble 681 rue de la Passerelle BP. 72 38402 Saint Martin d'Hères Cedex, France, *firstname.lastname@imag.fr*. LIG is a join laboratory to INP Grenoble, CNRS, UJF, INRIA and UPMF

features. The main ideas of the caching proposition relies 1) on the use of partially pre-calculated queries handled by cache services and 2) on the definition of a logical network of cache services which can cooperate together to enhance data access into the grid. These semantic caches may be deployed according to the grid infrastructure and user requirements. Queries naturally concern one or several sources distributed across the grid.

The semantic cache we propose is composed of a cache service, named *dual cache*, and a *query manager*. A dual cache is defined as a cooperation between a query cache and an object cache. The query cache keeps the calculated queries together with the identifiers of objects answering such queries. The object cache keeps accessed objects. Such a solution optimizes storage resource management, avoiding replication when objects are shared by several cached queries. In addition it saves computation, by making possible to cache a large number of query evaluations, even if corresponding objects do not all fit in the object cache. As a consequence, dual cache is particularly promising in a grid context, where resources (data sources and bandwidth) are shared among a large number of clients. The query manager provides two main services: (1) *query capabilities* to allow local evaluation of queries on the object cache and (2) *query matching* to compare received queries with pre-calculated ones in the query cache. These functions reduce server load and enhance local query evaluation. This proposal is particularly interesting in contexts where related queries are submitted in succession and their results overlap.

If a query cannot be evaluated by using the local cache service then several query evaluation strategies are possible. Among them, this paper proposes strategies using cache cooperation over the grid. Strategies rely on logical networks of cache services. A cache service may cooperate with several other caches. Cooperation may evolve and be adapted to improve query evaluation according to the current state of the grid and users affinity. Users working on a same cluster are "physically close" whereas users working on a same subject are "semantically close". Mapping this to cache services, this paper considers two main approaches to cooperation: query caches may cooperate according to a semantic logical network whereas object caches cooperation will be dominated by the characteristics of the underlying infrastructure. Experiments show the benefits of this approach in querying large scale distributed data.

This paper is organized as follows. Section 2 presents our proposition and its experimental application domains whereas section 3 presents a performance analysis in a middleware for data management on grids.

Related work is described in section 4. Section concludes this paper and gives research perspectives.

2 Dual cache and query management

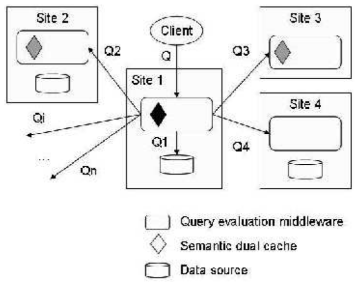


Figure 1 . Evaluation on grids

We propose a semantic cache [12, 19] solution integrating light weight query management capabilities and cooperative facilities. The figure 1 presents a simple example of the grid infrastructure we consider. Different sites are grouped in a cluster and different clusters are distributed in different places. A site is composed of clients, a query manager and eventually a data source. The query manager is part of the grid middleware and is in charge of query management, i.e. query decomposition and results building. A query manager is viewed as a flexible service built from different components. The cache service is one of these components and works together with the query manager. When `site 1` receives a client request `Q`, its query manager searches for results in the local cache. If part of the answer is not present in the cache, a cache miss occurs, generating sub queries to be posed to relevant other sites. Query decomposition is more accurate and figure 2 gives a better description of the cooperation between dual cache and query manager. The different cases of cache miss and resolution strategies will be studied in the next section. This proposal attempts to maximize advantages of semantic caching which are the reduction of both data transfers and query computation. It clearly distinguishes these two goals by managing a couple composed of a query cache and an object cache. Such a solution provides high flexibility to establish intra and inter site cooperation among the semantic caches deployed across the grid.

We illustrate our proposal with bioinformatics data management issues in grid. In this field, many software applications (BLAST and PattInProt for examples) need access to large and distributed data bank for processing protein sequences. This matches with semantic cache proposal which is particularly interesting in contexts where series of related queries will be issued in succession, with the results being at least partially overlapping. We focus on Swiss-Prot [3] a biological bank of protein sequences. This bank contains protein descriptions which consist in protein sequences, playing the role of objects, and related meta-data. Scientists search for relevant sequences to work with using meta-data to query the bank. Considering data volume, there are much more meta-data information than protein objects, therefore this is a complex and interesting use of semantic cache.

In the following, section 2.1 presents dual cache, section 2.2 introduces query management aspects and section 2.3 discusses cooperation potentiality.

2.1 Dual cache

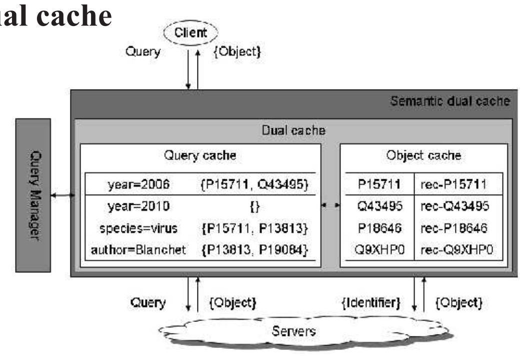


Figure 2. Semantic dual cache

A dual cache is composed by a query cache and an object cache as illustrated in figure 2. Dual caches are intended to be deployed on user or proxy [20] sites. They will rely on the resources of these machines to improve query evaluation. The query cache manages query results. It is inspired of the concept of views in DBMS. It is used to store (pre-calculated) queries. The identifier of an entry of the query cache `Q` corresponds to a query (`year=2006`, `year=2010`, `species=virus` and `author=Blanchet` in the figure 2), whereas its value is the set of identifiers of the relevant objects `objIdList` (for example `{P15711, Q43495}`). Like in view caching [26], objects are not stored in the query cache.

Example 1 *Figure 2 illustrates the content of the query cache for the considered application. In such a context, identifiers are strings. The first entry corresponds to the evaluation of a query enabling to select objects which the `year` attribute value is `2006`. For this evaluation, objects identified by `P15711` and `Q43495` are the only answers.*

Example 2 *With the same content, it can be noted that the answer of the query `year=2010` enabling to select entries created or modified in 2010 is empty. In other words, no object answers such a criteria.*

Dual cache aims to be used in largely distributed systems, where accessing data on sources is expensive. This is the opposite of a classical view caching approach, whereas objects are directly accessed via data sources. Such a solution, usually proposed in centralized systems, involves quite low access costs (in comparison to evaluation costs). Therefore, in a dual cache, cache accesses are based on an object cache. The identifier of an entry of an object cache, `idObj` (`P15711`, `Q43495`, `P18646` and `Q9XHP0` in the figure 2) corresponds to the identifier of a given object, whereas the value `obj` (`rec-P15711`, `rec-Q43495`, `rec-P18646` and `rec-Q9XHP0` in the figure 2) is the object itself. It has to be noted that none synchronization is required between content of object and query cache: this depends of application needs and infrastructure specificities. As a consequence, pre-calculated queries or views may be full (every corresponding objects are stored in the cache) or partially (some objects are absent from the cache) materialized.

```
ID      104K_THEPA          STANDARD; PRT;          924 AA.
AC      P15711;
DT      01-APR-1990 (Rel. 14, Created)
DT      01-APR-1990 (Rel. 14, Last sequence update)
DT      10-MAY-2005 (Rel. 47, Last annotation update)
DE      104 kDa microneme-rhoptry antigen.
OS      Theileria parva.
OC      Eukaryota; Alveolata; Apicomplexa; Piroplasmida; Theileriidae;
OC      Theileria.
OX      NCBI_TaxID=5875;
RN      [1]
RP      NUCLEOTIDE SEQUENCE.
RC      STRAIN=Muguga;
RX      MEDLINE=90158697; PubMed=1689460; DOI=10.1016/0166-6851(90)90007-9;
RA      Iams K.P., Young J.R., Nene V., Desai J., Webster P., Ole-Moiyoi O.K.,
RA      Musoke A.J.;
RT      "Characterisation of the gene encoding a 104-kilodalton microneme-
RT      rhoptry protein of Theileria parva.";
RL      Mol. Biochem. Parasitol. 39:47-60(1990).
CC      -!- SUBCELLULAR LOCATION: In microneme/rhoptry complexes.
CC      -!- DEVELOPMENTAL STAGE: Sporozoite antigen.
DR      EMBL; M29954; AAA18217.1; -; Unassigned_DNA.
DR      PIR; A44945; A44945.
DR      InterPro; IPR007480; DUF529.
DR      Pfam; PF04385; FAINT; 4.
KW      Antigen; Repeat; Sporozoite.
FT      REGION          1           19          Hydrophobic.
FT      REGION          905         924         Hydrophobic.
SH      SEQUENCE        924 AA;     103626 MW;  289B4B554A61870E CRC64;
SQ
MKFLILLFNILCLFPVLAADNHGVGPQGASGVDPITFDINSNQTGPAFLTAVEMAGVKYLQVQHGSNVNIH
RLVEGNVVIWENASTPLYTGA
IVTNNDGPYMAYVEVLGDPNLQFFIKSGDAWVTLSEHEYLAKLQEIRQAVHIESVFSLNMAFQLENNKYE
VETHAKNGANMVTFIPRNGHICKMVYHKNV
```

Figure 3. Example of a Swiss-Prot record

Example 3 *Figure 2 gives an example of content of the object cache. For this application context, the entry's identifier is a string and its value is a string formed as a set of (attribute, value) pairs, corresponding to* *data and meta-data associated to proteins' sequences. Figure 3 shows an example of record for the Swiss-Prot data bank. A Swiss-Prot bank is a simple flat file composed of records. Each record represents a protein sequence with its related meta-data. Each line of record is a meta-data which is introduced by a meta-data category (two characters) followed by meta-data values. The first line of a record is always `ID`, a key meta-data to identify the record (or the sequence) and the last line `SQ` is the sequence itself. Meta-data format are from semi-structured to unstructured one. This snapshot shows clearly that meta-data information is much more large than the sequence object. In order to simplify the representation, records stored in the object cache will not be detailed, but simply represented by a string `rec-id`.*

When a query is submitted to dual cache, it is first forwarded to the query cache. It may result in a hit or a miss. There is a query hit if entries of the query cache can be used to answer the query. In that case, an identifiers list `idObjList` is retrieved. If this list is not empty, it is used to load the list of the corresponding objects `objList` via the object cache.

Example 4 *The content of the query cache illustrated by figure 2 and the query `year=2006` produces the success list `{P15711, Q43495}` which is submitted to the object cache to retrieve the corresponding elements.*

If a cache miss occurs, the resolution process is initiated. Classical resolution consists in sending the query to the appropriate data sources. Another way to resolve a cache miss, is to contact other caches, using cooperative policies (see section 2.3).

Example 5 *With the content of the query cache illustrated by figure 2 and the query `year=2004` submitted, there is a miss and the query `year=2004` is forwarded to data sources.*

Dual cache can be accurately configured. In fact, the query cache and the object cache can use their own strategies. In particular, the replacement or admission strategies of each cache can be independent. As a consequence, consistency between the query cache and the object cache may be strong or weak, according to the chosen cooperation strategy. It has to be noted that weak consistency may imply miss resolution for the object cache and the query cache. As a consequence servers or data sources must provide access via queries or identifiers lists.

The dual cache approach ensures some advantages in large distributed systems sharing large amounts of data. First, as cached objects may be shared by several cached queries, memory use is optimal. In fact, an object is present at most one time in the object cache, even if it is referenced by several entries in the query cache. The independent management of query and object caches saves in many cases computation time and servers loads. In particular, if data sources provide relevant indexation mechanisms, retrieving objects via their identifiers is more efficient than evaluating the corresponding query. As a consequence, load on data sources can be greatly decreased. Finally the communication between query cache and object cache and the configuration of each of them are very flexible: (1) the retained size of each cache will determine whether data caching or computational caching is emphasized; (2) the synchronization between both query and object caches depends of the level of coherence between pre-calculated queries and objects required by the context (application, user, infrastructure); (3) the replacement strategy may be different for each cache. A fine setup of the whole intrinsic parameters of a dual cache is a challenging task for defining the best cache configuration in a specific context.

2.2 Light weight query manager

The *query manager* of the semantic cache isolates two distinct, but cooperative components to offer query capabilities and query matching. By query matching, we consider the semantic process of comparing a submitted query with the query cache content to deduce semantic overlap or semantic mismatch. By query capabilities we mean operators to locally evaluate queries on objects in the cache.

Query matching process analyzes queries to identify what cache entries are involved in answering a query `q`. When it is submitted, four types of hit may arise.

- *Exact hit*: `q` is already pre-calculated in the cache. This is the best situation where the query was already submitted. In this case, the query cache contacts the object cache to retrieve associated objects. The object cache initiates cache miss resolution if necessary. The complete answer is returned through the query cache.

Example 6 *With the content of the query cache illustrated by figure 2 and the query `year=2006` submitted, there is an exact hit using the entry which identifier is `year=2006`.*

- *Extended hit*: `q` can be obtained from the content of the query cache. Two situations may arise. In the first case, the identifier of an entry `e1` of the query cache is equivalent to `q`. In that case all objects referenced by `e1` answer `q`. In the second case, an identifier of an entry `e2` of the query cache subsumes `q`. Thus, the result of `q` can be obtained from the answer to `e2`. However some kind of filtering process is required, for example a selection or a projection. It can be achieved locally by the query manager.

Example 7 *With the content of the query cache illustrated by figure 2 and the query* `year>2005 ∧ year<2007` *submitted, there is an extended hit using the entry which identifier is* `year=2006`*. In that case, no filtering process is necessary.*

Example 8 *With the content of the query cache illustrated by figure 2 and the query* `year=2006 ∧ species=bacteria` *submitted, there is an extended hit using the entry which identifier is* `year=2006`*. In that case, the answer may be calculated evaluating the query submitted on the objects which identifiers are* `P15711` *and* `Q43495`*.*

- *Partial hit*: Two cases arise whether (1) an entry's identifier `e` subsumes `q` or (2) `q` overlaps `e` and `q` $\not\subset$ `e`. The answer of `q` is a part of the global answer of `e`. In this situation `q` is split in a probe query, which is the part known by the query cache and a remainder query corresponding to the missing part [12]. Objects in the answer to the probe query are retrieved as in the case of an exact hit. Remainder queries or miss resolution events can be solved by data servers or cooperative sites.

Example 9 *With the content of the query cache illustrated by figure 2 and the query* `year>2005` *submitted, there is a partial hit using the entry which identifier is* `year=2006`*. In that case, the answer includes the objects which identifiers are* `P15711` *and* `Q43495`*. However, some objects may be absent (objects created or modified since 2005) and has to be retrieved from the server thanks to the remainder query* `year>2005 ∧ ¬(year=2006)`.

Example 10 *Suppose a query submitted* `year=2006 ∧ author=Blanchet` *and a query cache containing a single entry which identifier is* `year=2006 ∧ species=virus` *and which value is* `{P15711, Q43495}`*. In that case, one part of the answer is obtained via an evaluation on objects which identifiers are* `P15711` *and* `Q43495` *whereas the other part is retrieved submitting the remainder query* `year=2006 ∧ author=Blanchet ∧ ¬(species=virus)` *to data sources.*

- There is a *query miss* if {\tt q} is totally disconnected from all entries of the query cache.

As query matching can be a very complex process. It is crucial to be able to judge when it is more effective to analyze query matching rather than consider a query as a miss. The complexity of this process is strongly related to querying capabilities of the query manager.

Querying capabilities are defined by operators (selections, projections, ordering, grouping, etc.) that can be evaluated by the dual cache on objects present in the object cache. As a matter of fact, when relevant object identifiers of a submitted query q are present in pre-calculated query the dual cache uses its own query capabilities to process results. As it can be assumed that the cache will work with a small amount of data compared to the data managed by the server, it could be argued that the maximum of operator should be present in the cache. But, this would be interesting only if query capabilities are handled efficiently. Two questions appear: (1) what are reliable situations for enabling these query capabilities? (2) what are the relevant places for operators between cache side and server side? As an example, if it is known that a special sorting algorithm fits better to an identified access pattern than the general one used by a server, then a sorting operator could be added to the cache query capabilities. Moreover, reducing the number of operators in the cache is also fruitful for the process of query matching.

Features built in a query manager provide high flexibility for deploying a cache architecture, enabling and configuring this component is the results of a trade off which is context dependent. It is important to take into account the complexity of typical query, resources allocated to the semantic cache, server and network loads. It is also important to know the main purpose of the cache. If its final goal is to save servers resources then an enhanced query manager is required. All this knowledge has to be taken into account to choose the appropriate level of functionality for the query manager. The instantiation of a query manager may rely on works in the style of [31]. Our semantic cache approach provides a flexible architecture and modular components to enhance data access in various middleware. We are currently working to provide guide-line for well-configured semantic cache and to apply this one in autonomous caches.

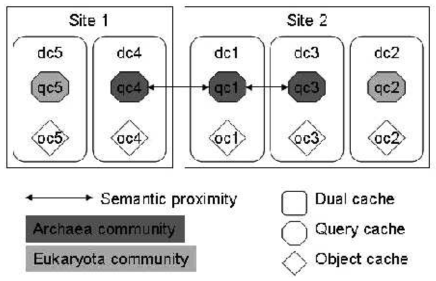
2.3 Cache cooperation

Dual cache allows better results in some grid contexts when cooperation capability is enable. This section gives an overview of cache cooperation strategies and their management by a dual cache architecture. An enhanced description can be found in [13].

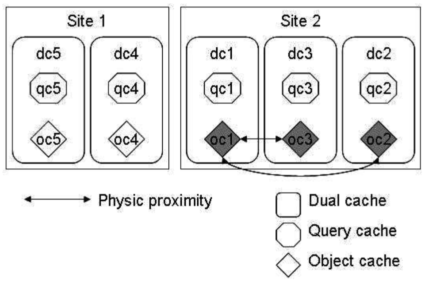
A *resolution protocol* [9] defines the process to retrieve data when a cache miss occurs. Choosing the right protocol is crucial in large distributed context. For example, to avoid data sources to become a bottleneck, one may prefer to search for data in other available and efficient resources. An usual strategy consists in deploying caches in the grid and defining cache cooperation at local level to enhance the global grid management.

When the number of caches raises up, it is obvious that cooperations between caches will be useful for miss resolution. Such cooperations may aim to reduce query evaluations or data transfers. If these aspects are sometimes antagonist for a classic cache, dual cache is a perfect support to mix up different configurations.

We introduce the notion of *proximity* to drive cooperation strategies. Proximity is a very generic concept based on distance functions to evaluate the reliability of a cooperation, therefore we focus only on two main characteristics for grid caching: physic and semantic proximity. *Physic proximity*, like the concepts proposed by of Rabinovich et al. [25] and Gadde et al. [18], prefers the cooperation between closed caches, i.e. based on a distance function which estimates physical parameters linking different caches as network type, network quality, cluster membership, etc. As a consequence, data transfers will be given in priority to caches in the same area, avoiding congestion in the global network. *Semantic proximity*, similarly to group of interest [28] or virtual community [6] notions, aims to define cooperation between caches with the same domain of interests, i.e. based on a semantic distance function which estimates similarities of query profiles in different caches. The probability of having a cache hit increases when the part of similar queries increases. As a consequence the cost of contacting caches far away is often inferior to the gain of sibling hits.



(a) Query caches and semantic proximity



(b) Object caches and physic proximity

Figure 4. Proximity based resolution in dual cache

Choosing a relevant proximity depends on the application and grid context. Nevertheless dual cache allows applying distinct proximities for the object and the query cache. In this paper, we will focus on physic proximity resolution for object caches and semantic proximity resolution for query caches, as illustrated in figures 4(a) and 4(b). On the one hand object making cooperate object caches in a same location avoids some high length data transfers. On the other hand, making query caches having the same interests cooperate raises the probability of having cache hits and reduces evaluation load on servers.

In addition to propose different groups for query and objects caches, it is also possible to choose distinct resolution protocols. While one may use a flooding resolution protocol as adaptations of Internet Cache Protocol (ICP) [34] or Hyper Text Caching Protocol (HTCP) [33], a catalogue based approach as in a Cache Digests (CD) [27] or summaries [15] can be chosen for the other one.

3 Performance analysis

This section reports our experimentations using semantic caching to improve querying in a middleware for data management in grids. Our main purpose is to compare existing semantic caching solutions to dual cache. First experiments analyze environment impact on caching, whereas the third one focuses on performances of different semantic caches in data querying in a grid context. Finally, the last experiment analyzes the impact of dual cache cooperations.

3.1 Testbed configuration

Experiments have been done in a grid data management middleware, with bioinformatics data. This section presents the testbed configuration: the experimental data set and query server, caches under test, query management in the different caches, workload generation and performance metrics.

Experimental data set and query server

Experiments have been done using the Swiss-Prot [3] biological database. It consists of a large ASCII file (750Mb) composed of about 210,000 sequence entries, each of then identified by an identifier. Gedeon middleware [32] provides a direct access to an entry through its identifier. It also provides query evaluation capabilities. Queries are composed of conjunctions and disjunctions of selection terms of the form `Attribute_name op value`. In the particular case of Swiss-Prot, `op` is often the `contain` operator and `value` is often a string. Evaluations result in a set of entries matching the query.

Caches under test

Predicate-based caching [19] presents a representative example of a semantic cache based on strict consistency between queries and objects without allowing replication of objects in the cache. This approach, used in distributed DBMS [19] and web oriented databases [21], leads to an optimal use of memory space but, consistency between queries and objects in the cache has a cost.

On the other hand query result caching [12] allows object replication in the cache. Such a replication prevents from focusing on consistency between queries and objects. When a region (corresponding to the answer of a query) is replaced, all corresponding objects are discarded. However a high level of replication and many redundant evaluations can occur.

Finally, elementary cache presents a representative example of a simple cache for a querying system. Such a cache does not consider query capabilities inside the cache. That is why it can be seen as a query result cache without query manager.

A Java and Fractal [4] version of ACS [14] has been used to instantiate the four caches under test: dual cache, predicate based cache, query result cache and elementary cache. All caches under test use LRU replacement strategy. In presented experiments, the cache size goes from 0.1 to 0.9 Gb and in all cases dual cache uses a size of 10 Mb for its query cache. When we studied the impact of workload, a 0.5 Gb size caches has been used. This is large enough to be efficient but not so large to prevent the deployment of other applications.

Query manager

Due to the server querying capabilities, the instantiated query manager for the three semantic caches only cares of query containment using query signature [8]. Query matching only cares on detecting if a query is included in a cache entry and query capabilities are reduced to selection and conjunction operators. In this configuration partial hits never occur.

Workload generation

Classical workloads used in benchmarks (TPC [29], proxy [1] and database [11] Wisconsin, and Polygraph [24] for instance) do not consider semantic locality, whereas we consider it as an important behavior for semantic caching. We use R_x, a synthetic semantic workload [23]. Queries correspond to progressive refinements.

The first query is general and following ones are more and more precise and thus reduce the set of matching elements. In an R_x workload, x is the ratio of subsumed queries. For example, with R_{50}, half of queries will be issued by constraining former queries. R_0 is equivalent to a uniform workload used in [8]. It is not the most suitable for semantic caching since it assumes queries do not present semantic locality. However, if a semantic cache is efficient in this context, it ensures this cache to be interesting for other access patterns. In presented experiments, workload is composed of queries corresponding to a single selection term, or to conjunctions of between two to four selection terms. In presented experiments, the chosen workload goes from R_0 to R_{90}. The impact of the cache size is studied with R_{60} since it seems to be representative of our application context.

Performance metrics

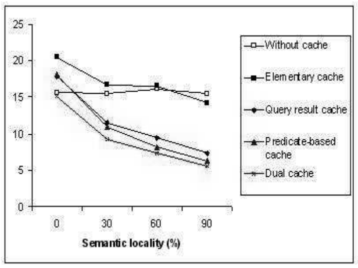
One of the most important metrics to study is the mean response time which is strongly related to the hit ratio. But the server's load and the amount of data transferred from servers to clients are also important metrics to be taken into account. As a matter of fact using a cache saves servers and network resources. As a consequence selected performance metrics are: mean response time, (exact and extended) hit ratio and the amount of data transferred.

3.2 Impact of the experimental context

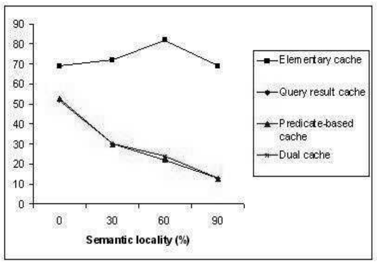
First experiments have been done with a single server and a single client having the same characteristics (dual-Xeon 3 Ghz, 2 Gb memory, SCSI disk). Such performance analysis aims to understand the comportment of different caches with respect to their size and to semantic locality. The client and the server are connected via an internal 1 Gb network. Obviously, communications can be neglected in such experiments, as well as load on servers. However, most interpretations are valid in distributed and loaded contexts. In all experiments presented in this section, each workload corresponds to 100 queries (according to a given semantic locality).

3.2.1 Impact of semantic locality

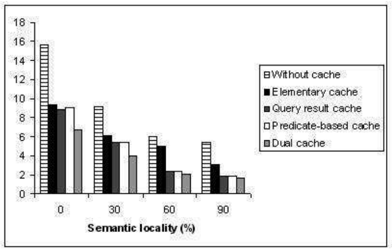
Figures 5 present the impact of semantic locality on the response time (5(a)), the ratio of queries resolved by caches (5(b)) and the amount of data transferred between the client/cache and the server (5(c)) for a system without cache, with an elementary cache, and with the different semantic caches. The size of the cache is fixed to 0.5 Gb (with 10 Mb allocated to the object cache in a dual cache).



(a) Mean response time (s)



(b) Ratio of queries resolved by the server (%)



© Volume of data transferred (Gb)

Figure 5. Impact of semantic locality (0.5 Gb cache)

Such figures confirm that using a cache in a distributed querying system enables to reduce bandwidth consumption and load on servers. However, it has to be noted that results in a system without cache are better than the ones in a system with an elementary cache (except for a 90 % semantic locality). In fact, with a cache data flows are read both by the cache and the client. Since gain of an elementary cache is quite low in such a context, it does not compensate for this double processing. On the contrary, semantic caching largely compensates for reading processes. In fact, semantic caches enable reducing load on servers, since many queries generate exact or extended hits. As a consequence, the amount of data transferred is greatly reduced. That is why, mean response times with semantic caches are generally better than mean response times without cache or with an elementary cache. However, it is important to note, that without any semantic locality (R_0), only dual cache is more relevant than a system without cache. Such a result shows the efficiency of a dual cache, as well as it confirms that a semantic cache is reserved to contexts presenting semantic locality.

Results confirm that semantic cache efficiency increases with semantic locality. In fact, the more query are subsumed, the higher is the cache reuse ratio. That is why for the different semantic caches, load on servers decrease. Note that data transfers also decrease since queries are more precise when semantic locality increase and, as a consequence, answers are smaller, as it can be observed with a system without cache. That is why an elementary cache is also impacted by semantic locality, as it is illustrated by the evolution of mean response time.

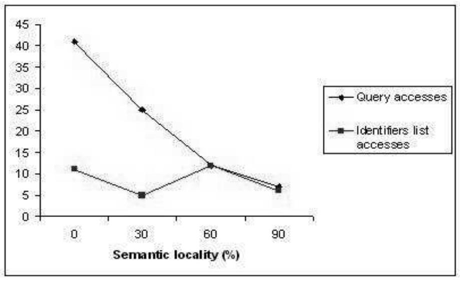
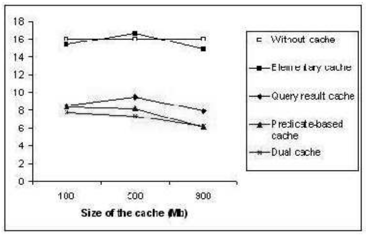


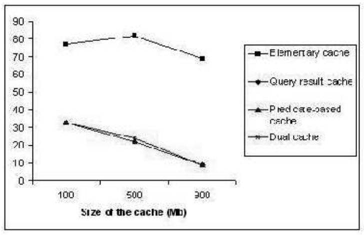
Figure 6. Impact of semantic locality on dual cache (0.5 Gb)

It is important to note that for a dual cache, load on servers corresponds to accesses both by queries or identifiers lists. Figure 6 shows that load in terms of queries is not impacted by semantic locality. In fact, even if the query cache is small (10 Mb), it can manage many query answers. On the contrary, since the object cache is more limited according to entries manipulated, it is impacted by semantic locality. That is why load in terms of identifiers lists decreases when semantic locality increases.

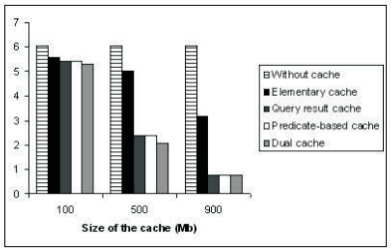
3.2.2 Impact of the size of the cache



(a) Mean response time (s)



(b) Ratio of queries resolved by the server (%)



(c) Volume of data transferred (Gb)

Figure 7. Impact of size of the cache (60 % semantic locality)

Figures 7 present the impact of the size of the cache on the response time (7(a)), the ratio of queries resolved by caches (7(b)) and the amount of data transferred between the client/cache and the server (7(c)) for a system without cache, with an elementary cache, and with the different semantic caches. The semantic locality is fixed to 60 %, simulating a context with a high semantic locality, an usual situation in bioinformatics data querying. It has to be noted that for dual cache, only the size of the object cache varies, whereas the size of the query cache is fixed to 10 Mb.

A first analysis confirms that using a semantic cache is relevant with a high semantic locality. In fact, even with a big size cache, load on servers, as well as bandwidth consumption, are quite more important with an elementary cache than with semantic caches.

About semantic caching, results confirm that efficiency of a cache is proportional to the size. In fact, the bigger is a cache, the higher are exact and extended hit rates. It is important to note that when the size of a cache enables to store the entire data source, all caches present similar results in terms of load on servers and bandwidth consumption. However, it has to be noted that query result cache is less efficient than predicate-based cache and dual cache, due to subsumed query management. In fact, when a query is included in a cache entry, the answer is processed locally, but not kept in the cache. As a consequence, successive refinements will lead the cache to use the same entry without taking advantage of previous computations.

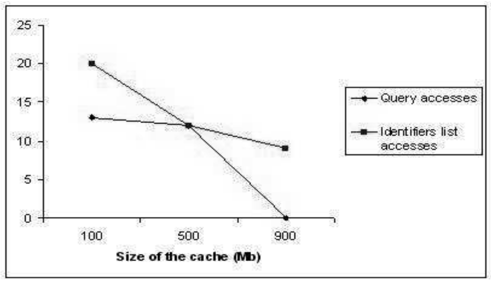


Figure 8. Impact of the size of dual cache

Recall that dual cache accesses servers to evaluate queries and to retrieve objects given a list of identifiers. Figure 8 shows that the size of the cache does not impact query accesses. In fact, even for a 100 Mb dual cache, the query cache is big enough to store many queries. However, results confirm the intuition that the size of the cache directly impacts identifiers list accesses.

3.3 Semantic caching in a grid context

Table 1. Technical behaviors of nodes of Grid5000 used for experimentations

Site	Machine	Processor	Memory	Disk
Lille	IBM eServer 326	AMD Opteron 248 2.2Ghz	4GB	SATA
Nancy	HP ProLiant DL145G2	2x AMD Opteron 246 2.0GHz	2GB	SATA
Rennes	Sun Fire V20z	2x AMD Opteron 248 2.2GHz	2GB	SCSI
Sophia-Antipolis	Sun Fire X4100	2x dual core AMD Opteron 275 2.2GHz	4GB	SAS
Toulouse	Sun Fire V20z	AMD Opteron 248 2.2Ghz	2GB	SCSI

This experiment aims to analyze the impact of different semantic caches in a grid context. Experiments have been performed on Grid5000 [7], the very large French platform for grid experiments. Technical behaviors of nodes of Grid5000 used for this experiment are presented in table 1. Query evaluation involves different servers distributed across the grid. Data has been distributed in three equivalent size files, managed by three different clusters. Clusters used in this experiment are located at Lille, Rennes and Sophia-Antipolis. When a query is submitted, it is forwarded to the three clusters for a parallel evaluation. As usual in grids, clients are distributed. Fifty clients uniformly distributed on four clusters (twelve at Lille, thirteen at Rennes, thirteen at Sophia-Antipolis and twelve at Toulouse) generate queries according to the R_{60} workload and each of them uses its own 0.5 Gb cache. The total number of submitted queries is 5000. For a single experiment all caches are of the same type, either predicate-based cache, query result cache or dual cache.

Table 2. Specific performance metrics for semantic caches in a grid context

Semantic cache	Response Time	Exact hit	Extended hit	Load on servers	Queries evaluated on servers	Data transfered
Query result cache	73.52 s	19.16 %	56.38 %	24,46 %	24,46 %	187.526 Gb
Predicate-based cache	71.01 s	26.46 %	49.70 %	23.84 %	23.84 %	185.464 Gb
Dual cache	47.26 s	52.94 %	39.02 %	23,34 %	8.04 %	132.197 Gb

Table 2 describes the impact of semantic caching in a middleware for data management in grids. It can be noted that caches lead to a dramatic reduction in the number of communications with the servers since many queries result in exact or extended hits. As a consequence, the amount of data transferred from servers to clients is highly reduced. We can then conclude that a system with caches is more robust, as it saves server and network resources.

Experiment shows that dual cache is the best approach for such a context, since with a dual cache the mean response time is the lowest (about a 35 % reduction in comparison to other semantic caches). Such an improvement can be explained by the reduction of the amount of data transferred between servers and clients, as well as the greatest exact and extended hit rates. In fact, dual cache enables to keep a large number of queries, regardless the storage of the corresponding objects. That is why even if load on servers is the same for all semantic caches (about 24 %), it has to be noted that for dual cache only a third (about 8 %) consists of query evaluations, whereas two thirds consist of identifier lists. As a consequence, when data accesses are more efficient by identifiers than by the corresponding queries, using dual cache is recommended.

3.4 Cooperative dual cache

A last experiment enables to show the impact of cache cooperation in dual cache. Query evaluation involves different servers located in three different sites (Nancy, Rennes and Sophia-Antipolis, which technical behaviors are presented in table 1). Fifty clients uniformly distributed on four clusters (twelve at Lille, thirteen at Rennes, thirteen at Sophia-Antipolis and twelve at Toulouse, which technical behaviors are presented in table 1) generate queries according to a specific R_{40} workload and each of them uses its own 325 Mb cache (corresponding to half of Swiss-Prot), with 10 Mb allocated to the object cache.

In addition to the semantic locality, this specific R_{40} workload introduces the notion of community. Community is used to group users having the same interests. The requests from the members of a community tend to focus on a particular subset of records. In the particular case of Swiss-Prot, we have created groups of interest according to the tree of life. Each record belongs to one of four different groups: `Eukaryota`, `Archaea`, `Viruses` and `Bacteria`. Thus, for each of these groups, we defined a community of users supposed to be specifically interested in this group. In our experiments, 60 % of the queries issued by any users concerns the records shared by its community. The last 40 % requests are uniformly distributed among the other records. The total number of submitted queries is 2500.

Results present the different metrics (response time, load on servers and amount of data transferred between clients and servers) for a basic dual cache, i.e. a dual cache resolving both query and object misses directly via servers, and a cooperative dual cache, resolving query misses according to a semantic proximity (in this experiments query caches belonging to a same community cooperate) and resolving object misses according to a physic proximity (in this experiments object caches in a same cluster cooperate). In this context, a flooding approach has been used for both query caches and object caches cooperations.

Table 3. Specific performance metrics for dual caches in a grid context

Dual cache	Response time	Evaluation on servers	Transfered data	Transfered data between clients and servers
Basic	103,3 s	34 %	30,4 Gb	30,4 Gb
Cooperative	24,4 s	9 %	25,1 Gb	11,5 Gb

Table 3 presents results for the different dual caches. Results show that using cooperation enables to reduce mean response time (about a 70 % reduction). On the one hand, semantic proximity between query caches enables to reduce the number of evaluations on servers, as well as the amount of data transferred, since data sources are used via identifier accesses avoiding retrieving already stored objects. On the other hand, physic proximity between object caches enables load balancing, reducing bandwidth consumption between clients and servers. In fact, some objects are retrieved via object caches belonging in the same cluster, providing high speed data access.

4 Related work

This paper tackles different domains related to caching. In this section we present some of the main works related to each domain: grid, semantic and cooperative caching.

Our experiments compared dual cache to predicate-based caching [19] and query result caching [12] which are representative of the two main approaches used in semantic caching, those separating query and objects [19, 21] and those handling them together [8, 12]. However, there are other proposals. Finkelstein [17] proposes to cache parts of queries that can be reused for further evaluations. This solution considers semantic aspects but does not manage extended hits. Roussopoulos [26] proposes a cache of views in a centralized system. Caching views is quite interesting, but as object caching is not considered such a solution may be of limited use in a distributed environment. On the same principles, Lee et al. [22] proposes caching views if they cannot be obtained using already materialized ones.

Resolution protocols have been intensively studied in web caching. In this context many protocols have been proposed. They can be decomposed in two main categories [5]: flooding and catalogue based. In this paper, a flooding resolution has been used. This choice is orthogonal to proximity-based resolution and can be changed according to the context.

None of the mentioned works have been deployed in a grid. Uddin Ahmed et al. [2] and Cardenas et al. [10] do by adopting a mediation like approach [35]. Intelligent Cache Management [2] focuses on the problem of network latency. It proposes to store data in distributed DB replicated across the grid. User SQL queries are submitted to the cache which decomposes them into sub-queries for local and remote domains, and builds afterwards the final results. Cardenas et al. [10] propose a distributed cache service for grid computing. Such a service offers semantic cache functionalities by using hierarchical cache architecture. A kind of global cache federates grid node caches by using a global catalogue. A metadata catalogue helps to localize data in data sources. Finally some proposals focus on load balancing and fault tolerance in large scale environments, like dCache [16], a cache for grids, used for data management in particles physics. Our proposal is orthogonal to these works.

Dual cache is not the first system combining two caches. For example, Trystram et al. [30] proposes a cache solution for parallel multiple sequence alignments. Such solution is composed of a cache for pair wise alignments and a second one to store multiple alignments. Pair wise entries are used to compute multiple ones. Contrary to dual cache, the proposal of Trystram et al. [30] is very context specific.

5 Conclusion

This paper presents a cache solution to improve querying in a grid context. The dual cache is based on the cooperation of a query cache and an object cache. This proposition has been implemented, tested and compared to other semantic cache propositions. Experiments have been done in a grid context using a data management middleware. Results prove the efficiency of our solution in this context. Our proposal saves computation time since it maximizes query caching and saves memory as the object cache avoids "in cache object replication". Cooperation between the two caches never introduces overhead related to consistency issues. Moreover this approach allows tuned configuration for each cache. This is particularly useful in a grid environment where caches may be deployed on heterogeneous sites.

Furthermore, thanks to a clear separation of calculus caching (query cache) and access caching (object cache) dual cache leads to new opportunities. First different kind of querying capabilities (filtering, grouping, ordering, etc.) may be used in the cache solution. Then, it is possible to resolve a cache miss in a different way for the query cache and the object cache, according to specific cooperation policies related to semantic or physic proximity.

Our solution seems well suited for contexts where shared data has low update rate. However, even in this context, consistency issues have to be developed. In addition, future work involves the study of various application contexts including warehouse oriented systems. We also plan more in-deep work on cooperation policies and replacement strategies, specifically cooperative ones. For example, to avoid entries to be evicted, they can be placed in other caches. Finally, we are interested in context-aware caching strategies for developing self-adaptive and autonomous caches in dynamic environments.

Acknowledgement

Thanks to S. Albrand for remarks on this paper, O.Valentin and Y. Denneulin for their contribution to this work, the *Gedeon* and *Hadas* teams for fruitful discussions, *Ministère délégué à l'Enseignement supérieur et à la Recherche* and *Institut National Polytechnique de Grenoble* for financial support.

References

[1] Jussara Almeida and Pei Cao. Measuring proxy performance with the Wisconsin Proxy Benchmark. Computer Networks and ISDN Systems, 30(22-23):2179–2192, 1998.

[2] Mobin Uddin Ahmed, Raja Asad Zaheer, and M. Abdul Qadir. Intelligent cache management for data grid. In The Australian workshop on Grid computing and e-research, pages 5–12, 2005.

[3] Brigitte Boeckmann, Amos Bairoch, Rolf Apweiler, Marie-Claude Blatter, Anne Estreicher, Elisabteh Gasteiger, Maria J. Martin, Karine Michoud, Claire O'Donovan, Isabelle Phan, Sandrine Pilbout, and Michel Schneider. The swissprot protein knowledgebase and its supplement trembl in 2003. Nucleic Acids Res, 31(1):365–370, 2003.

[4] Eric Bruneton, Thierry Coupaye, Matthieu Leclerq, Vivien Quéma, and Jean-Bernard Stefani. An Open Component model and its support in Java. In The International symposium in Component based Software Engineering, pages 7–22, 2004.

[5] Greg Barish and Katia Obraczka. World Wide Web caching: Trends and techniques. Communications Magazine, IEEE, 38(5):178–184, 2000.

[6] Lionel Brunie, Jean-Marc Pierson, and David Coquil. Semantic collaborative web caching. In Proceedings of the International Conference on Web Information Systems Engineering, pages 30–42, 2002.

[7] Franck Cappello. Grid'5000: A large scale, reconfigurable, controlable and monitorable grid platform. In The IEEE/ACM International Workshop on Grid Computing, 2005.

[8] Boris Chidlovskii and Uwe M. Borghoff. Semantic caching of web queries. The Very Large Data Bases Journal, 9(1):2–17, 2000.

[9] Anawat Chankhunthod, Peter B. Danzig, Chuck Neerdaels, Michael F. Schwartz, and Kurt J. Worrell. A hierarchical internet object cache. In USENIX Annual Technical Conference, pages 153–164, 1996.

[10] Yonny Cardenas, Jean-Marc Pierson, and Lionel Brunie. Uniform distributed cache service for grid computing. In The International Workshop on Database and Expert Systems Applications, pages 351–355, 2005.

[11] David J. DeWitt. The wisconsin benchmark: Past, present, and future. In The Benchmark Handbook. Morgan Kaufmann, 1993.

[12] Shaul Dar, Michael J. Franklin, Björn T. Jonsson, Divesh Srivastava, and Michael Tan. Semantic data caching and replacement. In Proceedings of the international conference on Very Large Data Bases, pages 330–341, 1996.

[13] Laurent d'Orazio, Fabrice Jouanot, Yves Denneulin, Cyril Labbé, Claudia Roncancio, and Olivier Valentin. Distributed semantic caching in grid middleware. In Proceedings of the international conference on Database and Expert Systems Applications, 2007.

[14] Laurent d'Orazio, Fabrice Jouanot, Cyril Labbé, and Claudia Roncancio. Building adaptable cache services. In Proceedings of the international workshop on middleware for grid computing, pages 1–6, 2005

[15] Li Fan, Pei Cao, Jussara Almeida, and Andrei Z. Broder. Summary cache: a scalable wide-area web cache sharing protocol. IEEE/ACM Transactions on Networking, 8(3):281–293, 2000.

[16] Patrick Fuhrmann and Volker Gulzow. dcache, storage system for the future. In The European Conference on Parallel and Distributed Computing, pages 1106–1113, 2006.

[17] Sheldon Finkelstein. Common expression analysis in database applications. In Proceedings of the ACM SIGMOD international conference on Management of data, pages 235–245, 1982.

[18] Syam Gadde, Jeff Chase, and Michael Rabinovich. A taste of crispy Squid. In Proceedings of the Workshop on Internet Server Performance, 1998.

[19] Arthur M. Keller and Julie Basu. A predicate-based caching scheme for clientserver database architectures. The VLDB Journal, 5(1):35–47, 1996.

[20] Ari Luotonen and Kevin Altis. World-wide web proxies. In The International World Wide Web Conference, pages 147–154, 1994.

[21] Dongwon Lee and Wesley W. Chu. Semantic caching via query matching for web sources. In The international conference on Information and knowledge management, pages 77–85, 1999.

[22] Ken. C. K. Lee, H. V. Leong, and Antonio Si. Semantic query caching in a mobile environment. ACM SIGMOBILE Mobile Computing and Communications Review, 3(2):28–36, 1999.

[23] Qiong Luo, Jeffrey F. Naughton, Rajasekar Krishnamurthy, Pei Cao, and Yunrui Li. Active query caching for database web servers. In The International Workshop on The World Wide Web and Databases, pages 92–104, 2001.

[24] Polygraph. http://polygraph.ircache.net/.

[25] Michael Rabinovich, Jeff Chase, and Syam Gadde. Not all hits are created equal: cooperative proxy caching over a wide-area network. Computer Networks and ISDN Systems, 30(22-23):2253–2259, 1998.

[26] Nicholas Roussopoulos. An incremental access method for viewcache: concept, algorithms, and cost analysis. ACM Transactions on Database Systems, 16(3):535–563, 1991.

[27] Alex Rousskov and Duane Wessels. Cache digests. Computer Networks and ISDN Systems, 30(22-23):2155–2168, 1998.

[28] T. T. Tay, Y. Feng, and M. N. Wijeysundera. A distributed internet caching system. In The IEEE Conference on Local Computer Networks, pages 624–633, 2000.

[29] TPC. http://www.tpc.org/.

[30] Denis Trystram and Jaroslaw Zola. Parallel multiple sequence alignment with decentralized cache support. In The European Conference on Parallel and Distributed Computing, pages 1217–1226, 2005.

[31] Tuyet-Trinh Vu and Christine Collet. Adaptable query evaluation using qbf. In Proceedings of the International Database Engineering and Applications Symposium, pages 265–270, 2004.

[32] Olivier Valentin, Fabrice Jouanot, Laurent d'Orazio, Yves Denneulin, Claudia Roncancio, Cyril Labbé, Christophe Blanchet, Pierre Sens, and Claude Bonnard. Gedeon, un intergiciel pour grille de données. In Conférence Française en Système d'Exploitation, 2006.

[33] Paul Vixie and DuaneWessels. Rfc 2756: Hyper text caching protocol (htcp/0.0), 2000.

[34] Duane Wessels and K Claffy. ICP and the Squid Web cache. IEEE Journal on Selected Areas in Communication, 16(3):345–357, 1998.

[35] Gio Wiederhold. Mediators in the architecture of future information systems. Computer, 25(3):38–49, 1992.