Caching across Heterogeneous Information Sources: an Object-based Approach *

ATHENA VAKALI YANNIS MANOLOPOULOS
Department of Informatics
Aristotle University
Thessaloniki 54006
GREECE
{avakali,manolopo}@csd.auth.gr

Abstract: Information exchange has been increased drastically and rapidly due to the major large scale Internet expansion. The structure of information sources worldwide has been altered and both structured and semistructured data are stored in various heterogeneous information spaces. Several tools have been developed for facilitating the rapid integration of the different information structures and accessing needs across heterogeneous sources over the Internet. This paper proposes a caching approach to extend the heterogeneous information sources integration process such that distributed objects are better accessed and retrieved. Our caching approach considers objects that are formed in response to certain queries posed over a specified number of distinct information sources. A methodology based on the notion of genetic algorithms is studied aiming at facilitating distributed objects exchange and accessing. In the proposed model, queries refer to documents or objects identified by their information source and their location within that source. Objects remain in the cache area based on their frequency of accesses, their popularity and their server priority, whereas the cache content is regularly updated. The proposed methods are experimented and results are compared with the corresponding results of the conventional *Least Recently Used (LRU)* cache replacement algorithm. The proposed evolutionary approach based algorithm is proven to be superior than the traditional LRU algorithm with respect to both cache and byte hit ratios.

Key-words: heterogeneous information sources, object-based models, caching policies, distributed object caching, evolutionary and genetic algorithms.

1 Introduction

Current large scale information systems involve the integration of distributed objects residing over a vast number of heterogeneous information sources. This heterogeneity arises from the fact that diverse computational and information processing needs have to be facilitated and guaranteed over networks and the Internet. Semistructured information is the typical form of data available in most modern information sources over the World-Wide-Web. In a typical scenario, queries are posed by a user (identified as the client) to a target source (identified as the server). In such a case, the retrieval of requested information

*A preliminary version of this work has been presented at the 9th International Database Conference (IDC'99). Work funded by the project "Modeling and management of semi-structured data for dynamic WWW applications" in the context of INTAS-RFBR program.

is performed once the related objects are identified by a number of difficult tasks or actions due to the high heterogeneity of object collections. It should be noted that semistructured data do not obey a specific schema, and thus query optimization can not be performed as in relational databases. In addition, the ongoing Internet expansion increases the access needs to diverse information sources and thus the integration of heterogeneous (and often diverse) information environments has become a major research issue.

The facilitating of efficient integration under heterogeneous information sources has been investigated in several developed prototype systems. These systems include components that extract properties from unstructured objects, translate information into a common object model, combine information from several sources, allow browsing of information and manage constraints across heterogeneous sites. Well known systems in this framework are: Tsimmis [8], Lore [1], Araneus [4], Strudel [9], W3qs [14], WebLog [15], Information Manifold [16] and WebSql [18]. Tsimmis and Lore projects are the most important efforts. Object exchange across diverse and dynamic information sources is the main aspect that has been studied and substantial progress has been shown on database integration techniques [21]. Architectures with associated algorithms for supporting the query mapping to objects have also been implemented [7, 12].

In order to improve performance of accessing heterogeneous information sources, *World-Wide Web caching* has been proposed as an effective solution to the problem of insufficient bandwidth caused by the rapid increase of web objects demands. In simple words, Web caching provides mechanisms to faster web access and improves load balancing without demanding more bandwidth. Most information servers are enhanced with appropriate tools which bring web objects closer to end users by adding specific cache consistency mechanisms and cache hierarchies. Since many web caches often fail to maintain a consistent cache, the problem of maintaining an updated cache has gained a lot of attention recently. Web caching differs from a distributed file system mainly in its access patterns since Web is orders of magnitude larger than any distributed file system [6, 11]. Furthermore, Web differs from conventional caching due to objects sizes nonhomogeneity [2]. The most critical research issues in Web caching concern cache replacement strategies as well as cache consistency and validation. Proxy cache server applications are in charge of supporting the server load balanced, as well as of facilitating the internet traffic. At the same time, a proxy cache server is responsible for the preservation of the cached object's "freshness". *Intelligent Caching* has been investigated in [28] where specific software is developed to investigate alternative Web caching techniques. In [11] a survey of contemporary cache consistency mechanisms in Internet is presented and examines recent research in Web cache consistency. The introduction of trace-driven simulation shows that a weak cache consistency protocol reduces network bandwidth and server load more than prior estimates of an objects life cycle or invalidation protocols. The need for replication is discussed in [22], where an alternative approach suggests the wide distribution of Internet load across multiple servers. The combination of caching and replication is discussed in [5], where the performance of a proxy cache server is evaluated and validated. The latter scheme has been proved beneficial with respect to both the circulation of web objects and the Web server's functionality.

Evolutionary strategies have been used to solve many computational problems demanding optimization and adaptation to a changing environment. The idea in all these systems was to evolve a population of candidate solutions to a given problem, using operations inspired by natural genetic variation and natural selection. Usually grouped under the term evolutionary algorithms or evolutionary computation, we find the domains of genetic algorithms, evolution strategies, and genetic programming. More specifically, genetic algorithms have been applied in the areas of scientific modeling and machine learning, but recently there has been a rapidly growing interest in their application in other fields [10, 19, 20, 23]. A web-based evolutionary model has been presented in [25] where cache content is updated by evolving

over a number of successive cache objects populations and it is shown by trace-driven simulation that cache content is improved. A genetic algorithm model is presented in [26] for Web objects replication and caching. Cache replacement is performed by the evolution of Web objects cache population accompanied by replication policies employed to the most recently accessed objects.

The present work addresses the problem of improving the process of accessing diverse distributed objects in order to respond to queries posed across heterogeneous information sources. The presented model extends the model of [24] so that the cache content is optimized due to both the cache being formed by objects with high access frequencies and the cache replacement scheme which is further applied. Our goal is to provide a model for facilitating the response and service of the querying process between clients and heterogeneous servers located across the Internet. The introduction of caching algorithms which could be adopted by developed heterogeneous sources integrating tools, since is shown to be quite effective towards objects availability and locality. More specifically, in this paper we focus on the retrieval of objects identified by their information source and their specific location within their source (server). A model based on the evolutionary idea (such as the genetic algorithm), is simulated in order to develop a cache with a "population" of objects at the client side. The population of objects evolves over the simulated time under two operations and the proposed model is compared to a conventional cache scheme based on the LRU approach. The cached objects population is formed with respect to their popularity, frequency of access and their server priority. The model is experimented under artificial queries workload and the proposed approach has been proven to be quite effective.

The remainder of the paper is organized as follows. The next section describes the structure of heterogeneous information sources, with emphasis on query processing. Section 3 analyzes the considered object-based caching model whereas the proposed caching approach is fully described and explained in Section 4. Sections 5 and 6 present the simulation model structure and the experimentation results, respectively. Finally, Section 7 points some conclusions and discusses potential future work.

2 Heterogeneous Information Spaces

A common problem found on various organizations and systems is that of multiple diverse information sources such as databases, electronic mail systems, digital libraries, knowledge bases and information retrieval systems. Most often the required information resides on various sources and is not always available due to the difficulties of accessing different systems as well as due to the network traffic. Therefore, the development of tools to access multiple heterogeneous sources in an integrated manner became quite crucial. Some data integration projects have been developed (e.g. Lore, Tsimmis), which are based on specific functions and different components to implement the query processing into a common object model. In the sequel, the project Tsimmis is further described as a representative case of modern heterogeneous information sources integration tool.

Figure 1 shows the most basic structural parts of the Tsimmis architecture. This architecture consists of a number of different heterogeneous information sources referred by the queries posed by a client machine. Above each source is the translator that logically converts the underlying data objects to a common information model. On one hand, the translator converts queries over information in the common model into requests that the source can execute, and, on the other hand, it converts the data returned by the source into the common model. The Object Exchange Model (OEM) serves as a simple tagged object model [8]. Above the translators lie the mediators, which refine in some way information from one or more sources. A mediator embeds the knowledge that is necessary for processing a specific type of information. End users can access information either by writing applications that

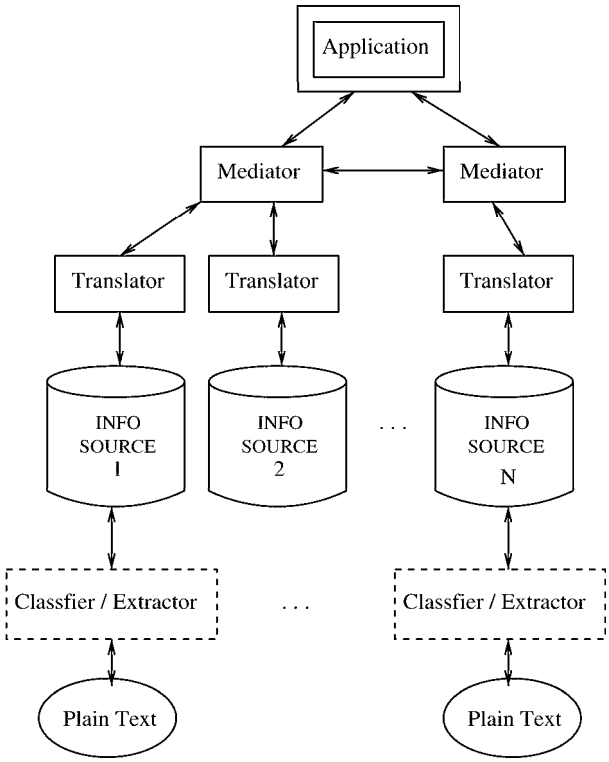

Figure 1: The main parts of the Tsimmis Project Architecture.

request OEM objects or by using generic browsing tools. The latter process involves a user writing a query on an interactive Web page, whereas the answer is received as a hypertext document. Constraint management is performed by additional managers dedicated to this task. Finally, Tsimmis system provides a mechanism for exploring heterogeneous information sources that is easy to interact with and that is based on commonly used Web interfaces.

Generally, there are three main issues with respect to the tools for heterogeneous information integration [21]:

information exchange, since the information systems need to exchange data objects residing on various sources. Therefore, there is a need for specifying the way the objects will be requested, represented and transferred over a network infrastructure.

information discovery and browsing, because users demand the exploration and browsing of objects residing among heterogeneous and probably distant servers.

mediators, which are programs that collect information from multiple sources in order to process, combine and export the resulting information to the user.

In this paper we focus on the information discovery problem by proposing schemes to improve object availability, since this is a major problem in the overall information demand process. We introduce evolutionary caching policies to strengthen the consistency, the discovery and the availability of objects to the requesting client.

As discussed in [8, 21] an object-based information model needs to be identified in order to formulate the information exchange and the query processing. OEM is the typical model proposed in order to define the objects located across several diverse heterogeneous information sources. The most common definition and specification of each object in OEM follows.

Definition 1: Each object is a tuple the following structure:

[Label, Type, Value, Object-ID]

where *Label* is a string describing the object representation, *Type* is the object data type, *Value* is the variable length value of the object and *Object-ID* is the object identifier.

The above object definition guides the query processing since several querying languages have been proposed based on this definition. A client is considered to issue a query in a language similar to SQL-like languages in the form of typical SELECT expressions:

```
SELECT Fetch_expression
FROM Object
WHERE Condition
```

The result to the above query is an object itself as given in Definition 1 with the particular *Label* = *answer*. Here, we consider queries which result in certain objects and we focus on appropriate actions in order to cache the most often requested objects at locations closer to the client(s). Our model focuses on the query responses and considers the objects that result while answering specific queries posed over a number of distinct heterogeneous information sources.

3 The Object-based Caching Model

Caching was initially introduced to provide an intermediate storage space between main memory and processor. Thus, the locality of reference is enhanced since the most recently accessed data have higher probability to be accessed again in the near future. The caching principle was extended to Web servers by considering them as another level in the memory hierarchy.

Cache area is a finite local area in which data could be stored. As mentioned in the previous, in our model the objects are the results to specific queries posed to a variety of information sources. Each object could be placed in the cache area so that it will be further available in a future request for the same object. The cache has a limited space, so that the cache content has to be updated regularly in order to be beneficial. Clients request information that might be located on an object residing at some source across a network (e.g. Internet). If the client poses another request for the same object, cache will use its own copy instead of requesting the original source again for the same object. The two main caching advantages are the reduction in both latency (e.g. requests are satisfied by the cache which is closer to the client) and network traffic (e.g. each object is retrieved from the server once, thus reducing the bandwidth used by a client). A question arising in relation to all caching actions is the object's freshness. Formally, an object is considered *stale* when the original server must be contacted to validate the existence of the cache copy. Object staleness results from the lack of awareness on server cache about the changes of original object. In practice, almost each proxy cache server implementation is reinforced with specific staleness confrontation. The goal of caching across heterogeneous information sources could be stated by the answers to the following questions:

- *when* to cache object(s) from heterogeneous information sources ?
- *which* objects(s) to cache from heterogeneous information sources ?
- *which* objects(s) to be replaced when the cache space becomes full?

The answers to the above questions and the criteria to perform the proposed caching policy are given in the next section.

Our model proposed a caching scheme in order to facilitate the following actions:

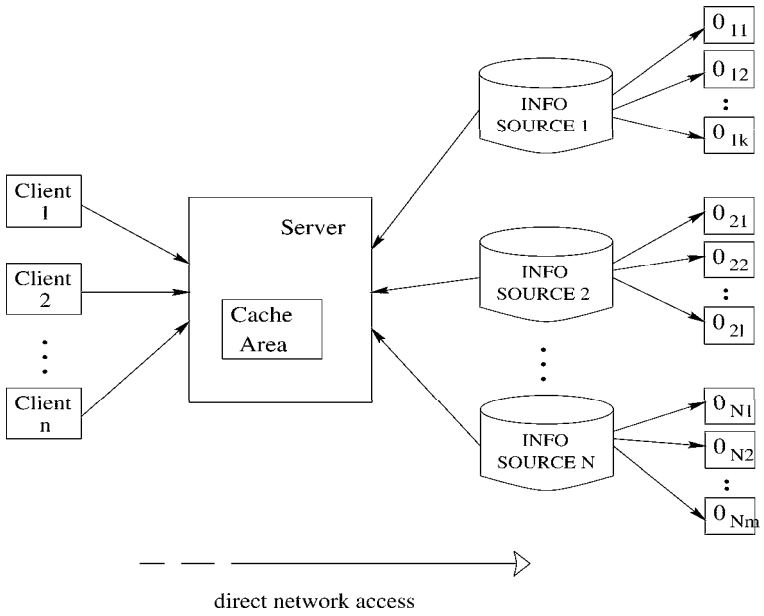

Figure 2: The object-based caching model.

1. A client demands objects from various information sources.
2. Appropriate client caching policies are applied such that the access to objects from heterogeneous spaces is performed at improved rates.
3. An update caching scheme is supported to eliminate cached objects staleness.

Figure 2 depicts the process of the query servicing with the introduction of a cache area between the client and the disperse information sources. Our model proposes an evolutionary approach in order to confront with the following key caching issues:

- The *identification* of *Objects* to be cached in the considered reserved cache area. The objects chosen form a "population" of objects to be copied to the reserved area in a considered local server. The choice of objects will be made under certain criteria such as frequency of access, popularity and retrieval cost.
- The *replacement* and *update* of the cache area will be performed at regular intervals in order to meet the criteria of reliability and accessibility. The time intervals to apply a caching algorithm are identified by certain access frequency information.
- The *information discovery* of objects residing at different information sources with variation in access and retrieval costs.

For each requested object the file system has to determine the location of the requested data blocks in order to perform the request servicing. The present paper supports this process by having the cache being searched first, prior to accessing the original information source. Information discovery is a challenging issue due to the large number of information sources. In the proposed model, a priority scheme is proposed for the considered sources/servers, to support the distinct information sources (dedicated to advances services and applications) which need to be prioritized. To better manage the information objects residing at the most crucial sources, we assign a number in the range $[1 \dots N]$ to each of the considered information sources in decreasing order of priority, i.e. the smallest priority number the larger priority of the considered information sources. The following metric is introduced in order to manage the sources priority scheme.

Definition 2: The information source *priority coefficient*, pc, for an information source s which has been assigned priority number pn_s $(1 \leq pn_s \leq N)$ is defined by:

$$pc_s = \frac{1}{pn_s}$$

Therefore, it is true that the higher the values of pc_s, the most important the considered server s is.

Furthermore, the cached objects should be also been prioritized in order to favor the most popular and frequently accessed objects. Here objects popularity is defined as follows.

Definition 3: The *popularity* of an object i is defined by:

$$pop_i = \frac{hits_i}{hits_{tot}}$$

where $hits_i$ refer to the number of hits for the cached object i out of the total number of hits $hits_{tot}$ on the considered cached objects.

4 The Evolutionary Caching Approach

4.1 The Caching Criteria

The selection of the objects to be cached should be done based on a priority decision scheme which should favor the choice of objects which are accessed most frequently at a desirable retrieval rate. Some criteria are needed in order to better formulate the problem statement. The following definitions are needed for the criteria which will identify the selection of objects to be cached.

Definition 4: The *dynamic frequency* of object i is defined by:

$$df_i = \frac{pop_i}{a_i}$$

where a_i is the metric to identify the number of accesses to other objects since object i was last referenced (see Table 1). Therefore, it is true that the higher the values of df_i, the most popular and recently the object i was accessed.

parameter	description
N	total number of information sources
S	number of objects considered for caching
pn_s	priority number assigned to source s, $1 \leq s \leq N$
pc_s	priority coefficient for source s, $1 \leq s \leq N$
s_i	size (in KBytes) of object i, $1 \leq i \leq S$
pop_i	popularity of object i, $1 \leq i \leq S$
df_i	dynamic frequency of object i, $1 \leq i \leq S$
a_i	number of accesses since the last time object i was accessed

Table 1: The most useful attributes of each cached object i.

At each selection step the algorithm considers whether an object should be cached or not. Therefore, there are two actions related to caching process, either the object will be chosen for caching or not. A function is needed to identify the action that should be taken for each considered object.

Definition 5: The *action function* for object i is defined by:

$$act_i = \begin{cases} 1 & \text{if object } i \text{ is chosen for caching} \\ 0 & \text{otherwise} \end{cases}$$

Problem Statement: Suppose that S is the number of requested objects which could be considered for caching and $AREA$ is the total capacity of the area reserved for caching. The caching area content optimization problem is defined as follows:

$$\text{Maximize} \quad \sum_{i=1}^{S} act_i \times df_i \times pc_s[i] \tag{1}$$

$$\text{subject to} \quad \sum_{i=1}^{S} act_i \times s_i \leq \text{AREA}$$

where s_i is the size of object i and $pc_s(i)$ refers to the priority coefficient of server s where the object i resides (see Table 1). In the optimization formula the quantity $df_i \times pc_s[i]$ is used as a "weight" factor associated with each object, since it is related to the objects popularity, access frequency and its source priority. The basic aim of the proposed caching approach is to support and enforce the caching of the most popular, frequently accessed objects, which are retrieved by the most important information sources.

4.2 Identifying Objects to be Cached

The genetic algorithm (GA) approach is considered for the selection of the objects to be cached due to the following two main reasons:

- First, the basic idea of the genetic algorithms is based on the evolution of populations by the criterion "survival of the fittest" and the objects to be cached should be the "fittest" (i.e. the most popular and frequently accessed at best retrieval rates) objects.

- Second, the GAs are applied to problems demanding optimization out of spaces which are too large to be exhaustively searched and all information sources have a huge amount of objects, impossible to be searched exhaustively in a realistic amount of time.

As depicted in Figure 3 a GA is an iterative procedure on a constant-size population of individual objects encoding a possible solution in a given problem space. In the present paper, the objects considered for caching are modeled as the individuals considered for evolution. Mapping the solution to the genetic space is an important task for the GA. The proposed algorithm starts with a specified number of objects S, which have been requested at the last evolutionary cycle. Each object is uniquely identified by its id, which is kept as a pointer to the relevant object record. Furthermore, an array A is used to keep record of the objects caching status, i.e. $A[i]$ is 1 if the object i is cached and 0 otherwise. Therefore, the optimization process uses an one-dimensional matrix suitable for the bit-string representations in the GA. Each GA cycle results in a bit-based representation string, which identifies the objects to be cached. Thus, each element at the genetic space can be decoded to a feasible element in the solution space.

The individual objects are assessed according to predefined quality criterion, called *objective or fitness function.* Two genetically-inspired operations, known as *crossover* and *mutation* are applied to selected cached objects (considered to be the population individuals) to successively create stronger "generations" of considered objects. The proposed GA model follows the simple GA proposed in [10].

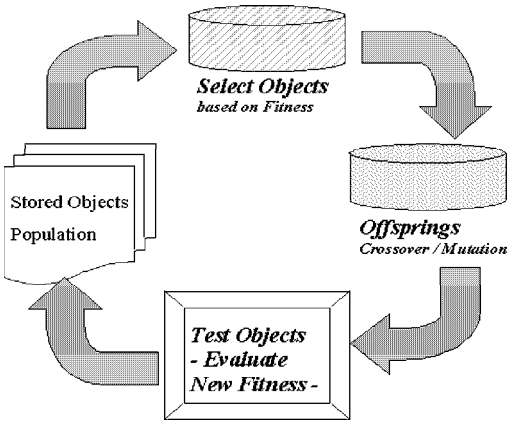

Figure 3: The Genetic Algorithm Cycle.

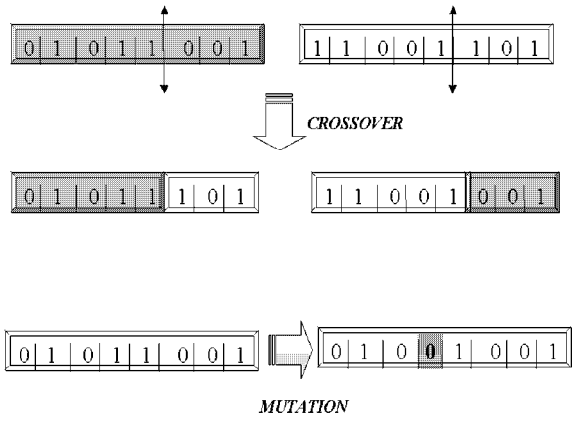

Figure 4: GA operators: crossover and mutation.

Encoding and Operators

Each object individual must be identified according to a predetermined encoded string. The encoding scheme is chosen such that the potential solution to our problem may be represented as a set of parameters. These parameters are joined together to form the encoded string. In order to consider the identification of each object individual, the objects identifications are mapped to the integer values $1, 2, \ldots, S$, where S is the total number of objects considered for caching. According to the problem statement (as defined in Formula 1) the parameters act_i, df_i, pc_s and s_i are the ones to guide the optimization problem, therefore they are included in the proposed encoded string. Each parameter is assigned a value and the presence of that parameter is signaled by the presence of that value in the ordered encoded string. The standard GA manipulations, namely the crossover and mutation, mix and recombine "genes" of an initial objects population to form the new objects to be cached for the next generation.

Crossover is performed between two object individuals ("parents") with some probability, in order to identify two new individuals resulting by exchanging parts of the parent strings. These exchangings are performed by cutting each individual at a specific position and produce two head and two tail segments. The tail segments are then swapped over to produce two new full length individual strings. Figure 4 presents the crossover operation on an example of an 8-bit binary encoded string, partitioned after its 5th bit, in order to result into two new 8-bit individuals.

Mutation is introduced in order to prevent premature convergence to local optima by randomly sampling new points in the search space. Mutation is applied to each "child" individually after crossover. It randomly alters each individual with a (usually) small probability (e.g. 0.001). Figure 4 depicts the mutation operation in an binary 8-bit string where the 4th bit is mutated to result in a new individual.

The objects population will evolve over successive generations such that the fitness of the best and the average object individual in each generation is improved towards the global optimum.

The Objective Function

An objective (or fitness) function must be devised based on the need to have a figure of merit proportional to the utility or ability of the encoded object individual. The following formula defines the fitness function, $F(x)$, that considers both access frequency and retrieval rate for an object population x:

$$F(x) = \sum_{i=1}^{S} act_i \times df_i \times pc_s[i] \tag{2}$$

The above fitness function has been introduced in the present research effort in order to consider the effect of access frequency and retrieval cost in the overall caching process.

The GA Caching Algorithm

The proposed GA-based caching algorithm generates an initial population of objects located at various information sources. To make the search efficient, the initial population consists of objects characterized of being as frequently accessed and faster retrieved as possible. Thus, each population is formed by the most promising and strong objects of all considered information spaces. Then, the standard operators defined above mix and recombine the encoded strings of the initial population to form offspring of the next generation. In this evolution process, the fitter object individuals will create a larger number of offspring, and thus have a higher chance of "surviving" to subsequent generations. The GA-based caching evolves under a termination criterion, which is identified by the number of generations i.e. the number of successive GA cycle runs. A pseudo-code version of the GA-based caching algorithm follows.

```
// Algorithm cache_form(CACHE, population_C)

    old_pop <- population_C // current objects population
    generation <- 1
    while (generation <= maxgen)  do
        par1 <- selection(popsize, fitness, old_pop)
        par2 <- selection(popsize, fitness, old_pop)
        crossover(par1,par2,old_pop,new_pop,p_cross)
        mutation(new_pop, p_mutate)
        evaluate_fitness(new_pop)
        statistical_report(new_pop)
        old_pop <- new_pop
        generation <- generation + 1
```

In the above algorithm the cache content is formed by using the variable *maxgen*, which corresponds to the maximum number of successive generation runs, whereas *popsize* is the objects population size and $fitness$ is the objects update metric as described above by function $F(x)$ in Expression (2). Variables $par1$ and $par2$ define the parents chosen for the reform of each generation, *p_cross*, *p_mutate* are the probabilities for the crossover and mutation operators, respectively. The *old_pop* refers to the initial population in every GA cycle, whereas the *new_pop* is the resulting population of each GA run. The above algorithm is appropriately structured such that it provides a solution to the problem statement as defined by Expression 1.

Furthermore, we do consider the cache replacement process in order to maintain the most often requested non-stale objects. We employ several "runs" in order to question the cache content "staleness" and our proposed approach is summarized by the following main algorithm:

```
evaluate(frequencies, Objects)
t  <- 0
while (t <= TIME)  do
    request_arrival()
    if ( frequencies < RATE ) then
        population_C <- most_frequent_objects
        cache_form(CACHE, population_C)
```

```
request servicing()
evaluate(frequencies, Objects)
t <- t + 1
```

In the above algorithm (considered to run for a time period determined by the variable $TIME$), the population to be cached is determined by the *cache_form* routine. It is important to note that the caching control process is performed at regular time intervals according to the access frequencies evaluated by the *evaluate* routine. The objects remain cached until the next running cycle. A parameter ($RATE$) defines the upper bound for performing the cache replacement, i.e. when frequencies are under the specified $RATE$ the current cached objects do not match with the requests access patterns and thus, reconsidering of the cache content has to be performed. The $CACHE$ and *population_C* are the input parameters for routine *cache_form* in order to appropriately place the chosen objects population on the reserved cache area.

5 The Simulation Approach

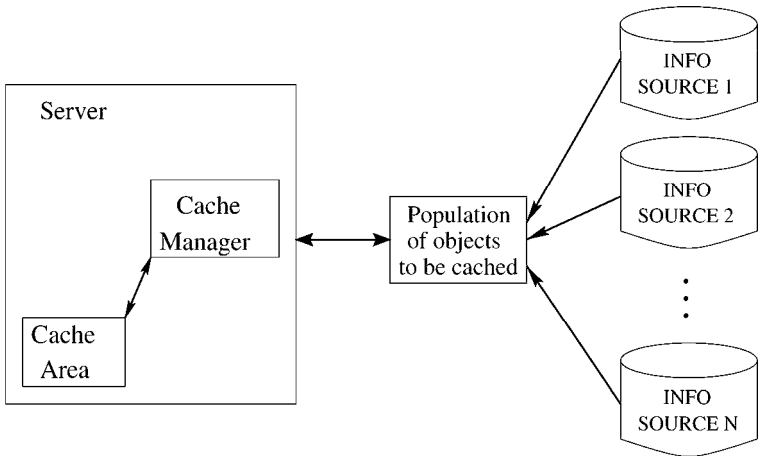

Figure 5: The modules of the Caching simulation model.

The following distinct structural modules are supported in our model in order to better formulate the simulation model (see Figure 5).

The Information Sources. A finite number of information sources (IS), namely IS_1, IS_2, ..., IS_n, is supported. Each IS_i contains a number of distinct objects that might be the response to a query posed by a client.

The Objects. Each object within an IS is identified by a distinct number as given by the Object-ID (see Definition 1). Therefore, objects $O_{11}, O_{12}, ..., O_{1k}$ are the objects located in IS_1, objects $O_{21}, O_{22}, ..., O_{2l}$ are the objects located in IS_2 and so on.

The Queries. Each query posed by a client includes the pair of the identification of the IS, where the object is located, and the identification number of the object within this specific IS. Therefore, the general query format includes the pair: $< IS_i, O_{ij} >$.

The proposed simulation model is composed of distinct components for the workload, the cache and the information sources. Figure 6 represents the different components or modules of the analytic model and their interaction in relation to request servicing. The model supports a workload of requests posed by the clients to a central storage server which guides the request servicing. Aspects of resource

management are simplified for both cache and information sources, in order to support timing and synchronization on request servicing. Each information source has a corresponding manager/controller that guides the requests servicing. The cache area is questioned and request servicing is performed when the cached objects match the requested data. The Cache manager takes a formalized workload as input, transforms this workload to an appropriate format adapted to the objects structure, the request servicing is performed and the performance metrics can be evaluated.

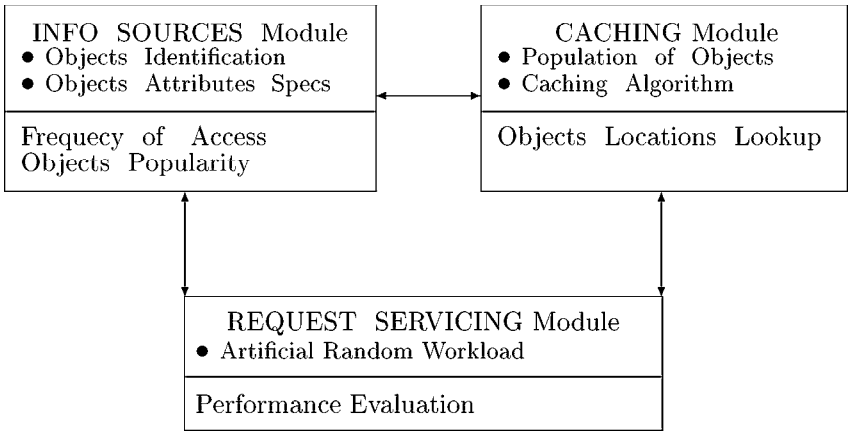

Figure 6: Simulation Model Components and their interaction.

The Workload and the Request Servicing

A typical request consists of the attributes for:

- *Information Source*, i.e. the id of the information source to serve the request,
- *Objects Id* , i.e. the identification number assigned to objects within the information sources,
- *Operation*, in the present model only reads are considered in order to study the request servicing,
- *Size* , i.e. the amount of data blocks to be read, and the
- *Arrival Time* , i.e. the time when the request arrived at the controller.

According to a conventional controller, the above request pattern will be directed to the appropriate information source in order to be served. In the presented approach the request is modified by the *Cache Manager*, which checks the caching area before requesting the objects from its original storage location as identified by its request pattern. It should be noted that the present model concerns reading requests since this is the typical operation involved in information sources objects exchange. Requests arrive to the system randomly by various independent processes. Some requests arrive while others are being serviced. Requests arrival rate could be either constant or independent and exponentially distributed or bursty. Here, both random and correlated requests are considered. The correlated requests arrivals refer to higher priority sources, so the pc_s for an information source s (Definition 2) is involved in the requests arrival pattern.

Workload is characterized by the number of queries that will have to be serviced. Therefore, the simulator needs to be experimented under different query sets. The Query servicing routine guides the queries to the cache area in order to identify whether the requested objects are cached or not. Figure 7 depicts the most important parameters of both query workload and cache area structures. The pair of *IS_id* and *O_id* are the IS and Object identifications respectively (noted as $< IS_i, O_{ij} >$ above). As shown in Figure 7 this pair is the actual index for identifying the objects residing in the cache area. *Size*

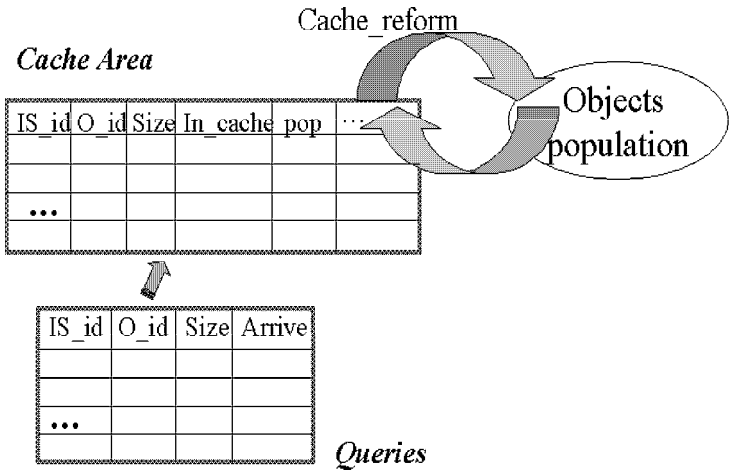

Figure 7: The Workload and Request servicing.

is the requested object's size and the *Arrive* is the time of this specific query arrival. Arrival times are produced randomly by a workload generator routine such that the request patterns follow the priority values as assigned to the information sources.

The Caching Component

The Caching Component consists of a *timing module* to model the timing devoted to caching such that synchronize and coordinate the request servicing. The cache manager serves requests in an FCFS basis and appropriate indexing is used to identify the requested object. The cache size is determined in the model and the cache area can be occupied by a certain number of data objects. The cache manager module is appropriately configured such that the performance metrics can be estimated. The cache area has a specific cache retrieval rate (in MB/sec) which estimates the consuming rate of the requested objects which belkong in the cache.

The Heterogeneous Information Sources

All information sources could be contacted by clients as in the developed prototype systems (see Section 2). Here, we model the distinct information sources by identifying them uniquely and by considering them to have appropriate storage units to host the data objects. Their own storage units are defined to have similar configuration and metrics.

The performance metrics used in the presented simulation model focus on the cached objects cache-hit ratio and byte-hit ratio, defined as follows:

Cache hit ratio represents the percentage of all requests being serviced by a cache copy of the requested object, instead of contacting the original object's information source.

Byte hit ratio represents the percentage of all data transferred from cache, i.e. corresponds to ratio of the size of objects retrieved from the cache server. Byte hit ratio provides an indication of the network bandwidth use.

The above metrics are considered to be the most typical ones which capture and analyze the cache replacement policies (e.g. [2, 3]) and the results for these metrics are presented in the next experimentation section.

Convergence of the Genetic Algorithm
A GA is a randomized parallel search method modeled on natural selection. Since natural selection uses diversity in a population to produce adaptation, the GA converges when most of the population is identical, i.e. when the diversity is minimized. There is no mathematical proof of convergence or any guarantee that the GAs find the global minimum [13]. There have been some efforts in specifying bounds or prediction for times and speed of convergence for GAs. For example, in [17] average hamming distance is introduced as a measure for population diversity and bounds for time to minimal diversity are derived.

As a stochastic search method successful GAs must maintain a balance between the exploration of a wide area of the solution space and there must be an exploitation of beneficial aspects of existing solutions. In the considered maximization problem, the GA continues for a specified simulation time and the *cache_form* algorithm is repeated for a certain number of iterations as identified by a maximum number of GA generations (Section 4.2). As estimated by the *statistical_report* routine, the same cached objects tend to appear for a number of iterations. As pointed in [27] convergence and homogeneity are not typical problems in simulations that last for a number of generations up to 100, whereas these issues become critical in simulations which last hundreds or thousands of generations. Therefore, in the proposed simulation model the *cache_form* routine is repeated for a maximum number of 100 generations as an indicative parameter that will guarantee that the proposed GA converges at a beneficial solution within a specified number of repetitions.

6 Experimentation - Results

We have adopted two different cache management policies to manage the objects exchange across heterogeneous information sources. The LRU caching is a scheme that has been applied widely in many cache servers. Cache is usually implemented as a hash table with a fixed number of locations that could host objects. In our model, the LRU caching was implemented to serve as a typical basis for comparisons with the proposed genetic algorithm cache scheme (called *GA Caching*). We have considered two different types of request workloads: random and bursty. The bursty request arrivals favor the most important information sources since the priority coefficient pc_s for sources s determines the request arrival pattern.

LRU Caching on the client
The LRU algorithm is a typical page replacement algorithm which is used in modern operating systems and database systems. The same method has been followed by many modern cache servers. It has been proven that applying the LRU policy helps in maintaining a consistent cache content, as well as in improving the overall performance of many Web servers.

The idea of the LRU schemes applied to caching is that objects should be removed from the cache at the same rate they are added, since objects have to be removed when the used cache space reaches its upper capacity. Each object in the cache is associated with the time it was last used. This way, when a new object has to be fetched in the cache area, LRU chooses to replace the object that has not been used for the longest time period, i.e. objects with large LRU values are removed before objects with small LRU values.

The Results
The simulator was tested and experimented under artificial query workloads defined by a varying number of information spaces and by several queries requesting distributed objects. Figure 8 depicts the results

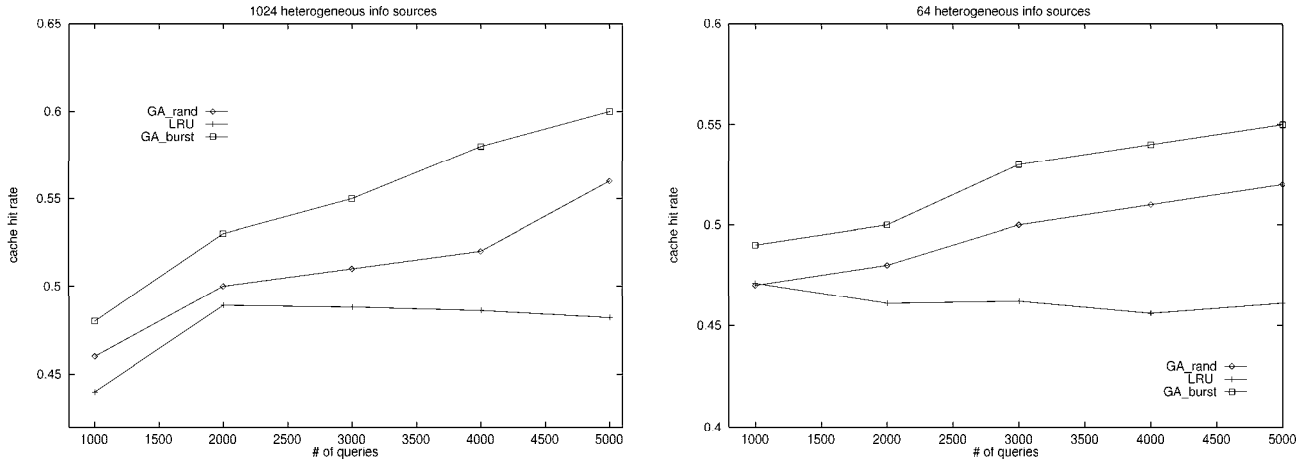

Figure 8: Cache hit rates (1024 IS and 64 IS).

for the query processing of a cache population formed of objects among 1024 and 64 heterogeneous information sources (left and right part respectively). The cache effectiveness is measured by the percentage of cache "hits" for a varying number of queries. More specifically, Figure 8 presents the LRU and GA caching hit rates for a cache population reproduced for 1000, 2000, ..., 5000 queries. Therefore, the simulator has been experimented under various workloads of various query sets. Crossover probability is 0.6, whereas mutation probability is 0.033, since these values have been suggested as a representative trial set for most GA optimizations. The importance of GA and LRU caching is obvious on the considered model with more information spaces (see Figure 8). It is interesting to note that the GA performs better when we have bursty request arrivals, which was expected due to the introduction of the sources priority coefficients in the GA fitness evaluation. It should be also noted that the GA outperforms the conventional LRU caching in all cases the improvement due to GA caching reaches high rates. For example, there is an almost 25% improvement due to the GA under bursty request arrivals, in the case of 1024 heterogeneous sources and 5000 queries.

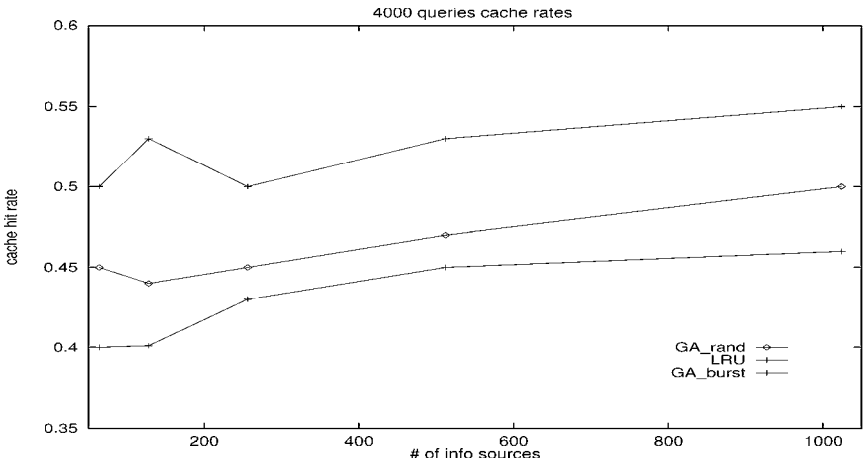

Figure 9: Cache hit rates (4000 queries).

Similarly, Figure 9 presents the LRU and GA caching hit rates for a cache population reproduced over 64, 128, ..., 1024 distinct information sources under a typical data trial set of 4000 queries. GA caching proves to be more effective as the number of heterogeneous information sources increases. This is a positive remark for the adoption of such a scheme in the integration of responding over heterogeneous

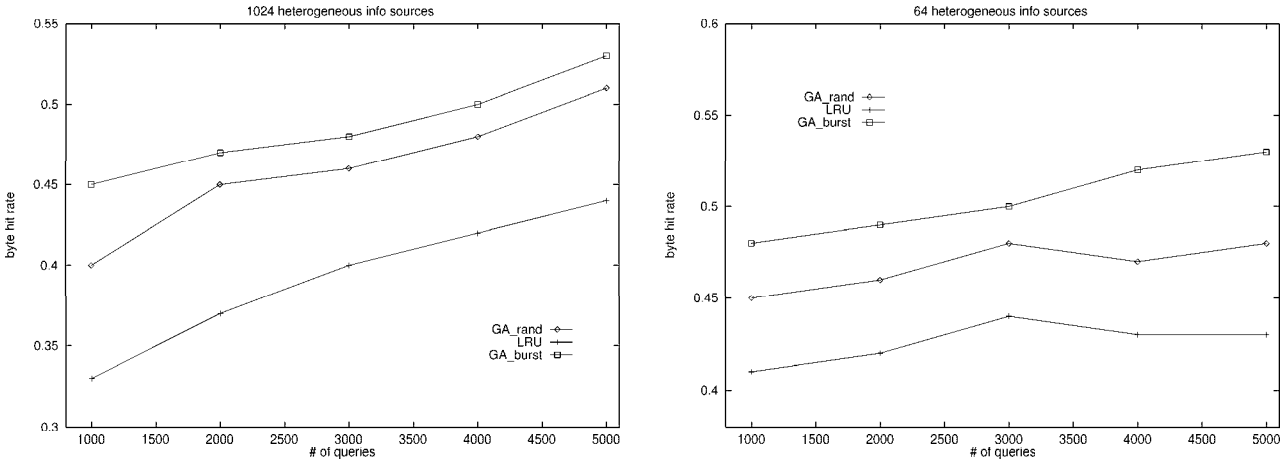

Figure 10: Byte hit rates (1024 IS and 64 IS).

sources. More specifically, both GA and LRU caching seem to be unstable for less than 200 information sources and the GA under bursty arrivals has resulted in better hit rates in all cases. The benefits of GA are emphasized for an increased number of information sources since it shows an improved cache hit rate, as opposed to the slight decreasing of LRU (under the same number of information sources).

Figure 10 depicts the results for the byte hit rates by a cache of objects collected across several heterogeneous information sources. The cache effectiveness is measured by the percentage of byte hits for a varying number of queries. In this figure the LRU and GA caching hit rates are presented for a cache population reproduced for 1000, 2000, ..., 5000 queries, under 1024 and 64 information sources (left and right parts of Figure 10, respectively). It should be noted that the byte hit rates show similar curves as the cache hit rates (see Figure 8). The byte hits are not as high as the cache hits but the behavior of LRU and GA caching algorithms is similar to that when cache hits are considered. Again, in Figure 11 the LRU and GA byte hit rates are presented, for a cache population reproduced over 64, 128, ..., 1024 distinct information sources under a typical data trial set of 4000 queries. GA caching proves to be more effective as the number of heterogeneous information sources increases and still the GA under bursty arrivals is considered to be the best choice as compared to the typical LRU and the GA under random request arrivals.

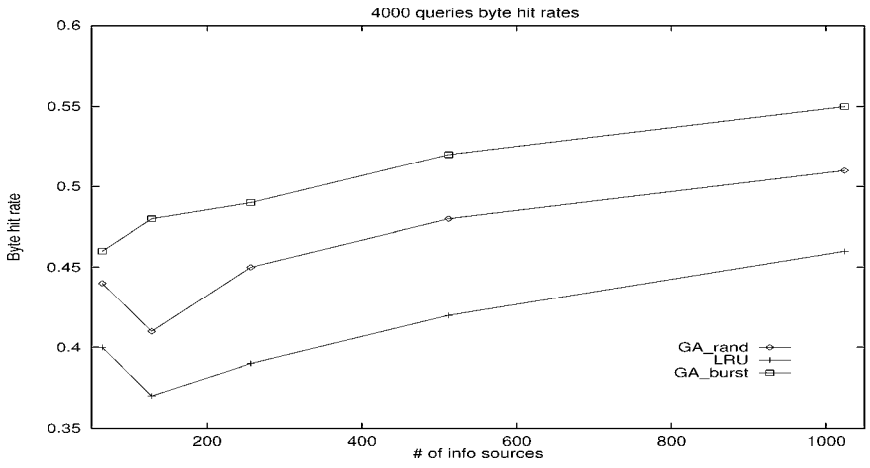

Figure 11: Byte hit rates (4000 queries).

Figure 12 depicts the results for the average and maximum access frequency of the cached objects

under GA caching, over 1024 and 64 heterogeneous information sources (left and right parts of Figure 12, respectively). This figure refers to runs for a cache population reproduced for 1000, 2000, ..., 5000 queries under random request arrivals. Note, also, that the cache access frequency is the actual fitness measure needed for the GA process. In case of 1024 information sources (see left part of Figure 12) both average and maximum fitness have a linear-like slope, with the average access frequency being beneficial as the number of queries increases. The values of maximum and average access frequency converge when the system supports fewer information sources (see right part of Figure 12). This fact was expected since the system with more information sources has a broader collection of objects which might not be accessed as frequently as the objects within a system with fewer information sources. As depicted in this figure, the population fitness stabilizes as the number of information sources is high (e.g. 1024 ISs) and the number of queries increases.

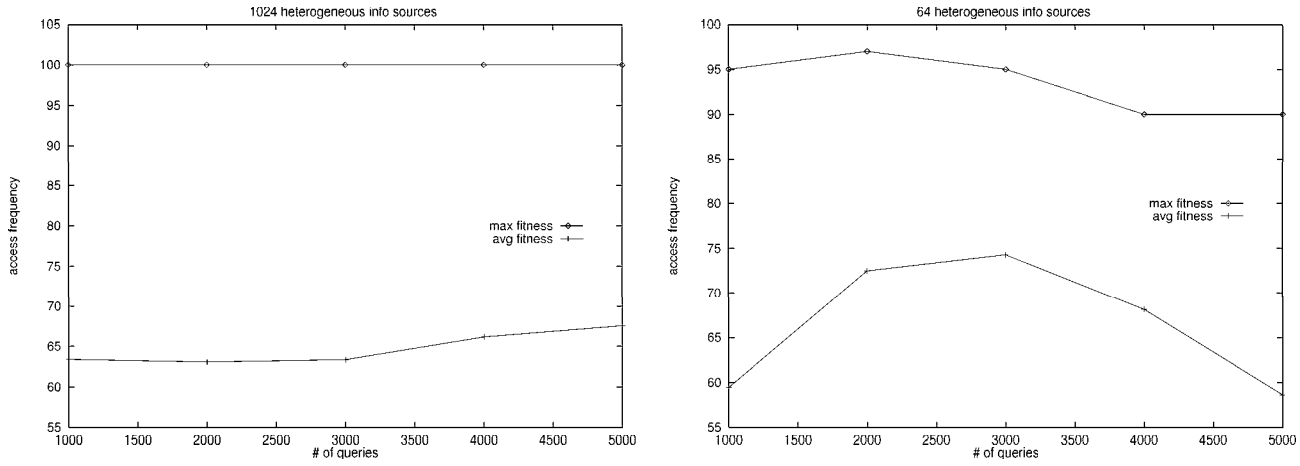

Figure 12: Cache fitness (1024 IS and 64 IS).

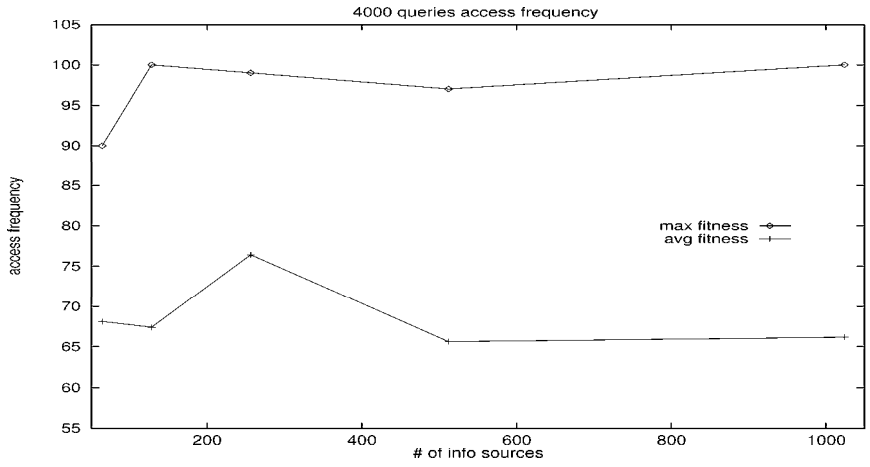

Figure 13: Cache hit rates (4000 queries).

Figure 13 presents the average and maximum GA caching fitness for a cache population reproduced over 64, 128, ..., 1024 distinct information sources under a typical data set of 4000 queries. Both average and maximum fitness values are unstable for fewer information sources. The values of these metrics seem to stabilize for more than 500 information sources which is depicted in Figure 8 as well.

Overall the GA caching scheme has shown the best cache and byte hit rates and access frequency. GA scheme has resulted in better performance metrics than the corresponding LRU caching scheme. As

presented in the above figures applying caching on objects from a large number of different information sources is quite significant since almost half of the objects are found locally with no need to overload the client system with network search and transfer costs.

7 Conclusions - Further Research

Caching has been introduced towards an effective object circulation across heterogeneous information sources. The adoption of an object-based caching according to a genetic algorithm scheme, has been proposed to enhance the performance of such complex diverse systems. The proposed caching scheme, included a cache replacement policy employed at regular intervals in order to optimize the cache content. A simulation model has been experimented under both random and bursty artificial query workloads of a varying number of information sources and certain conclusions were discussed about the proposed GA scheme. The GA scheme has been proven quite beneficial in comparison to the typical LRU caching since cache population evolved over the simulation time for an increasing number of queries.

Further research should examine the above scheme in conjunction with a developed heterogeneous information source tool by mapping query results to the objects identified as in the presented work. The proposed scheme could be experimented under real traces and query types and over different fitness selection policies, which might include rates of object file types or bytes length. Other evolving computation schemes, such as simulated annealing and threshold acceptance, could be adopted in objects caching in order to study their effect on object refreshment policy towards a better cache content. Replication is another subject which can be applied to GA caching since the crossover operation replicates individuals between generations. The basic idea here is to keep replicated individuals over successive generations to various servers in order to test the caching and replication of heterogeneous objects.

References

[1] S. Abiteboul, D. Quass, J. McHugh, J. Widom and J. Wiener: The Lorel Query Language for Semistructured Data, *International Journal on Digital Libraries*, Vol.1, No.1, pp.68-88, 1997.

[2] C. Aggarwal, J. Wolf and P.S. Yu: Caching on the World Wide Web, *IEEE Transactions on Knowledge and Data Engineering*, Vol.11, No.1, pp.94-107, 1999.

[3] M. Arlitt, R. Friedrich and T. Jin: Performance Evaluation of Web Proxy Cache Replacement Policies, Hewlett-Packard Technical Report HPL 98-97, to appear in *Performance Evaluation Journal.*

[4] P. Atzeni, G. Mecca and P. Merialdo: To Weave the Web, *Proceedings 23nd VLDB Conference*, pp.206-215, Athens, Greece, 1997.

[5] M. Baentsch et al.: Enhancing the Web's Infrastructure: from Caching to Replication, *IEEE Internet Computing*, Vol.1, No.2, pp.18-27, 1997.

[6] M.A. Blaze: Caching in Large-Scale Distributed File Systems, Ph.D. dissertation, Princeton University, 1993.

[7] K.C.C. Chang, H. Garcia-Molina and A. Paepcke: Boolean Query Mapping Across Heterogeneous Information Sources, *IEEE Transactions on Knowledge and Data Engineering*, Vol.8, No.4, pp.515-521, 1996.

[8] S. Chawathe, H. Garcia-Molina, J. Hammer, K. Ireland, Y. Papakonstantinou, J. Ullman and J. Widom: The Tsimmis Project - Integration of Heterogeneous Information Sources, *Proceedings 10th Anniversary Meeting of the Information Processing Society of Japan*, Tokyo, Japan, 1994.

[9] M. Fernandez, D. Florescu, J. Kang, A. Levy and D. Suciu: STRUDEL - a Web Site Management System, *Proceedings ACM SIGMOD'97 Conference*, pp.549-552, Tucson, AZ, 1997.

[10] D. Goldberg: *Genetic Algorithms in Search, Optimization, and Machine Learning*, Addison-Wesley, 1989.

[11] J. Gwertzman and M. Seltzer: World Wide Web Cache Consistency, *Proceedings USENIX 1996 Annual Technical Conference*, pp.141-151, San Diego, CA, 1996.

[12] J. Hammer, H. Garcia-Molina, J. Cho, R. Aranha and A. Crespo: Extracting Semistructured Information from the Web, *Proceedings Workshop on Management of Semistructured Data*, Tuscon, AZ, 1997.

[13] R. L. Haupt and S.E. Haupt: *Practical Genetic Algorithms*, John Wiley, 1998.

[14] D. Konopnicki and O. Shmueli: W3QS - a Query System for the World Wide Web, *Proceedings 21st VLDB Conference*, pp.54-65, Zurich, Switzerland, 1995.

[15] L.V.S. Lakshman, F. Sadri and I.N. Subramanian: A Declarative Language for Querying and restructuring the World-Wide-Web, *Proceedings IEEE RIDE Workshop*, pp.12-21, New Orleans, LO, 1996.

[16] A.Y. Levy, A. Rajaraman and J.J. Ordille: Querying Heterogeneous Information Sources using Source Descriptions, *Proceedings 22nd VLDB Conference*, pp.251-262, Bombay, India, 1996.

[17] S.J. Louis: Genetic Algorithms as a Viable Computational Tool for Design, Ph.D. dissertation, Computer Science Department, Indiana University, 1993.

[18] A. Mendelzon, G. Mihaila and T. Milo: Querying the World Wide Web, *Proceedings 4th PDIS Conference*, pp.80-91, Miami, FL, 1996.

[19] Z. Michalewicz: *Genetic Algorithms + Data Structures = Evolutionary Programs*, Springer Verlag, New York, 1992.

[20] M. Mitchell: *An Introduction to Genetic Algorithms*, The MIT Press, Cambridge, London, 1998.

[21] Y. Papakonstantinou, H. Garcia-Molina and J. Widom: Object Exchange Across Heterogeneous Information Sources, *Proceedings 11th ICDE Conference*, pp.251-260, Taipei, Taiwan, 1995.

[22] D. Povey and J. Harrison: A Distributed Internet Object Cache, *Proceedings 20th Australasian Computer Science Conference*, Sydney, Australia, 1997.

[23] T. Starkweather, D. Whitley and K. Mathias: *Optimization Using Distributed Genetic Algorithms - Parallel Problem Solving*, Springer Verlag, 1991.

[24] A. Vakali and Y. Manolopoulos: Caching Objects from Heterogeneous Information Sources, *Proceedings 9th International Database Conference (IDC'99)*, pp.377-389, Hong Kong, China, 1999.

[25] A. Vakali: A Web-based Evolutionary Model for Internet Data Caching, *Proceedings 2nd International Workshop on Network-Based Information Systems (NBIS'99)*, pp.650-654, Florence, Italy, 1999.

[26] A. Vakali: A Genetic Algorithm scheme for Web Replication and Caching, *Proceedings 3rd IMACS/IEEE International Conference on Circuits, Systems, Communications and Computers, (CSCC'99)*, Athens, Greece, 1999.

[27] M.G. Cooper and V. Vemuri: Genetic Algorithms, Chapter in *CRC Handbook on Industrial Electronics*, CRC Press, 1997.

[28] D. Wessels: Intelligent Caching World-Wide Web Objects, *Proceedings INET'95 Conference*, 1995.