Caching Objects from Heterogeneous Information Sources

Athena Vakali and Yannis Manolopoulos
Department of Informatics
Aristotle University
54006 Thessaloniki, Greece
email: {athena,yannis}@delab.csd.auth.gr

Abstract

Information exchange has shown a rapid growth due to Internet expansion and has altered the structure of information sources worldwide. Structured and semistructured data are stored in various heterogeneous information sources and, thus, tools have been developed for facilitating their rapid integration. The goal of this paper is to extend the integration process by the introduction of caching techniques to the client environment, in order to decrease access time to objects distributed over various information sources. In particular, two methods for optimizing distributed objects exchange have been studied and implemented; the first method is based on evolutionary computation, whereas the second one is based on the Least-Recently-Used caching algorithm. In our model, queries ask for documents or objects which are identified by the information source and their placement within that source. Objects are kept in cache area based on their access pattern and the cache is updated by the two proposed algorithms. Both methods are evaluated experimentally, and results show that the evolutionary computing based algorithm is superior than the traditional LRU algorithm.

Index terms: heterogeneous information sources, caching techniques, distributed cache, World-Wide-Web objects, evolutionary computation, genetic algorithms.

1 Introduction

Modern information systems usually involve the integration of distributed heterogeneous information sources. This heterogeneity arises due to the fact that diverse computational and information processing needs have to be satisfied nowadays. Semistructured information is included in most modern information spaces over the World-Wide-Web. In a typical scenario,

queries are posed by a "user" (often identified as the client) to a target source (identified as the server). In such a case, the retrieval of requested objects is a quite difficult task because of the highly heterogeneous object collections. Also, it is notable that semistructured data do not obey a specific schema, and thus query optimization can not be performed as happens in the case of relational databases. In addition, the ongoing Internet expansion increases the access needs to diverse information sources and the integration of such heterogeneous information environments has become a major necessity.

Several prototype systems for facilitating the efficient integration of heterogeneous information sources have been developed. 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. In this respect, well known systems are: Tsimmis [7], Lore [1], Araneus [2], Strudel [8], W3qs [12], WebLog [13], Information Manifold [14] and WebSql [15]. 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 [19]. Architectures with associated algorithms for supporting the query mapping to objects have also been implemented [6, 11].

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 [10]. Each Web object is mastered by a specific site and object's changes and updates are mastered by a specific machine. The Web server is responsible for the preservation of the object's "freshness" while keeping the server load balanced, and facilitating the internet traffic. *Intelligent Caching* has been investigated in [22] where specific software is developed to investigate alternative Web caching techniques. In [10] 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 [20] where an alternative approach suggests the wide distribution of Internet load across multiple servers. The combination of caching and replication is discussed in [3], 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 [9, 16, 17, 21].

Our work addresses the problem of improving the process of accessing diverse distributed objects in order to respond to queries regarding heterogeneous spaces. Our goal is to provide a model for facilitating the querying process between a client and a server. The introduction of caching algorithms which could be adopted by developed heterogeneous spaces integrating tools, 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 drawn from multiple heterogeneous spaces. A model which adapts the evolutionary computation idea to preserve a cache "population" of objects at the client side is suggested compared to a conventional cache scheme based on the Least-Recently-Used (LRU) policy. The model is experimented under various simulated data and the evolutionary computation scheme has been proven to be quite effective.

The remainder of the paper is organized as follows. The next section describes heterogeneous information spaces structure, with emphasis on query processing. Section 3 analyzes the caching schemes that have been developed by most Web servers and introduces the evolutionary computation mechanism by focusing on the genetic algorithm idea and its implementation in scientific problems. Sections 4 and 5 present the simulation model structure and the experimentation results, respectively. Section 6 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) and they are based on specific functions and different components to implement the query processing into a common object model. In the sequel, Tsimmis Project 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

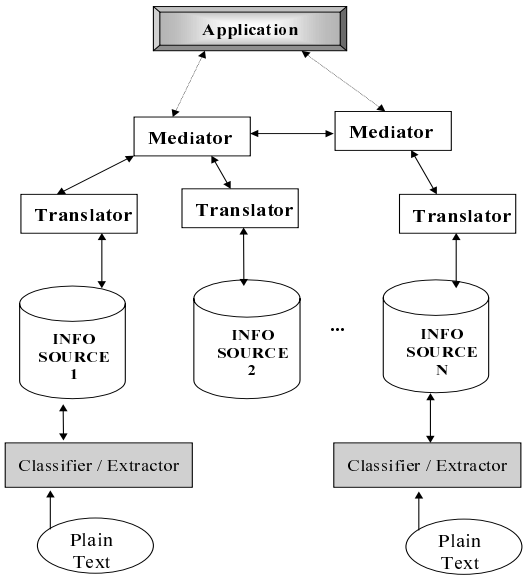

Figure 1: The main parts of the Tsimmis Project Architecture.

data objects to a common information model. The translator, on one one hand, 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 [7]. 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 the processing a specific type of information. End users can access information either by writing applications that 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 constraint and local constraint managers. 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 aspects for the tools regarding integration of heterogeneous information [19]:

- *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*, i.e. 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 and availability of objects to the requesting client. Caching policies are described in the next section.

3 Caching Policies

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 soon. 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 on which data could be stored. In our model the objects are the results to specific queries posed to a variety of information sources. Each object could be placed on 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 could 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 reduce 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).

We have adopted two different cache management policies in order to manage the objects exchange across heterogeneous information sources. The LRU caching is a scheme that has been applied widely in many cache servers. In our case, it is used as a basis for comparisons with the introduced here evolutionary Caching (called *GA* Caching).

3.1 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.

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 in order to serve as a typical basis for comparisons with the proposed Genetic Algorithm cache scheme described in the next subsection.

3.2 GA Caching on the client

Evolutionary programming has been successfully applied to numerous problems from different domains. The basic idea of evolutionary computation is based on the evolution of natural populations which is based on the principles of natural selection and "survival of the fittest". Genetic algorithms (GAs) comprise one of the main evolutionary methods, applied to many computational problems requiring either search through a huge number of possibilities for solutions, or adaptation to a changing environment. The innovation of GAs is that they work with a coding of the parameter set (not the parameters themselves), they search from a population of points and they use probabilistic transition rules.

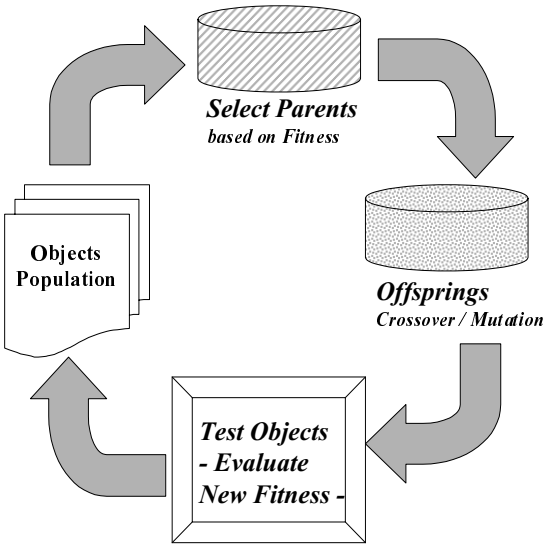

Figure 2: The structure of the typical Genetic Algorithm.

Figure 2 shows the structure and the cycle of a simple Genetic Algorithm. It is obvious that a GA is an iterative procedure which aims in resulting in a refined population based on specified criteria. The GA process involves a constant-size population of individuals each one represented by a finite string of symbols (the "genome"), encoding a possible solution in a given problem space. This space (referred as the search space) comprises of all possible solutions to a given problem. Generally, the GAs are applied to problems demanding optimization out of spaces which are too large to be exhaustively searched. The standard genetic algorithm generates an initial population of individuals, which is updated at each evolutionary step resulting in a new

"generation". The individuals in the current population are decoded and evaluated according to some predefined quality criterion, called *fitness function*. Each individual's fitness evaluation is an important issue, usually given as part of the problem's description. Two genetically-inspired operations, known as *crossover* and *mutation* are applied to selected individuals in order to successively create stronger generations.

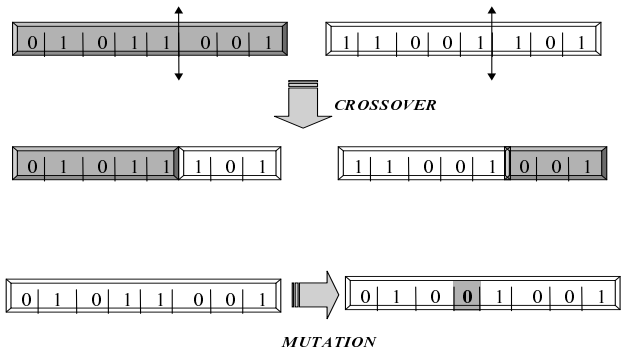

Figure 3: Genetic Algorithm operators: crossover and mutation.

Figure 3 depicts the latter two operations in a 8-bit string individual.

- **Crossover** is performed between two individuals (parents) with some probability, in order to identify two new individuals resulting by exchanging parts of parents' genome. The exchanging of parents parts are performed by cutting each individual at a specific bit position and produce two "head" and two "tail" segments. The tail segments are then swapped over to produce two new full length individual strings.
- **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).

Provided that GA has been correctly implemented, the population will evolve over successive generations such that the fitness of the best and the average individual in each generation is improved towards the global optimum.

4 The Simulation Model

Our model implements the following problem statement:

A client demands objects from various information spaces. Find an improved scheme to support the access to objects from heterogeneous spaces through client caching policies.

Since the client demands objects from various diverse locations, in our model we identify objects as follows:

- **The Information Source:** 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. Therefore, objects $O_{11}, O_{12}, ..., O_{1k}$ are the objects stored in IS_1, objects $O_{21}, O_{22}, ..., O_{2l}$ are the objects stored in IS_2 and so on.
- **The Queries:** each query posed by a client is a pair of the identification of the IS, where the object is located, and the identification number of the object within the specific IS. Therefore, the general query format is the pair: $< IS_i, O_{ij} >$.

Figure 4 depicts the process of the query servicing with the introduction of a cache area between the client and the disperse information sources.

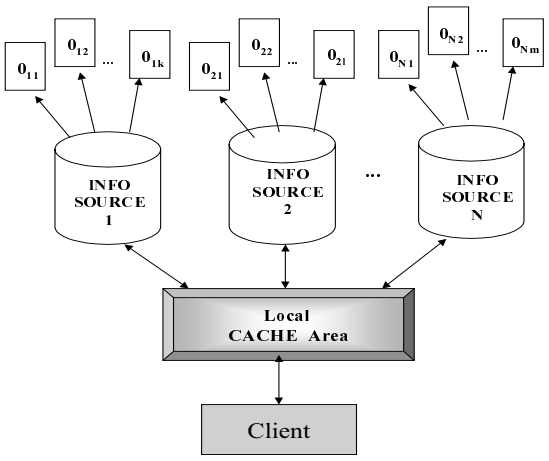

Figure 4: The Object Caching Scheme.

We have implemented the GA evolutionary approach to the cache update and refreshment in order to optimize cacheable objects. In the experimenes section, it will be shown that our method result in an improved cache content. Our GA model follows the Simple GA proposed in [9]. The following heuristics were made in order to adapt the GA approach to the cache update scheme:

- the cache is considered as a population of individuals,
- the individual is the actual cached object, identified by the pair of the IS_i source and the object's number within the latter source. The values of the pair result in a unique number for each object. For implementation purposes, each individual is encoded as a bitstring corresponding to the numbers of the pair values. This string has a length of

$n+m$ bits, where n is the number of bits for the IS_i source, and m is the number of bits for the object identification number.

- each individual is assigned with a fitness value derived by a general fitness function,
- the fitness function is related to a factor declaring the object's access frequency.

The idea is to apply the GA process to the cache population in order to optimize cache content by evolving a successive number of cache generations. Crossover and mutation is applied in each cache generation in order to result in a more fresh cache population. Cache population is characterized by its fitness which is appropriately related to the object's freshness factor. We have implemented a cache update scheme to perform cache reform and re-"generation" at regular time intervals. A pseudo-code version of the GA scheme of our cache-update implementation follows:

```
initialize()     // derive old_population=initial population
generation <- 1
   while (generation <= maxgen)  do
      parent1 <- selection(popsize, fitness, old_population)
      parent2 <- selection(popsize, fitness, old_population)

      crossover(parent1, parent2, old_population, new_population, p_cross)
      mutation(new_population, p_mutate)

      statistical_report(new_population)
      old_population <- new_population
      generation <- generation + 1
```

In the above GA process *maxgen* corresponds to the maximum number of successive generation runs, *popsize* is the cache population size, p_cross and p_mutate are the probabilities for crossover and mutation, respectively, whereas $fitness$ is the cache update factor (fitness evaluation will be described in the next subsection). The implementation of crossover gets old_population and results in new_population either by preserving the parents or by reproducing a new individual based on parents fitness. Mutation is performed on the new_population by affecting few positions in the individuals string towards a better fitness.

5 Experiments - Results

Our simulator modeled a caching system based on the simple GA policy and on the LRU policy. The simulator was tested and validated for a varying number of information spaces and for several queries requesting a random object within some information source. Figures 5 and 6 depict the results for the successful query satisfaction by the cache for 1024 heterogeneous information sources. The cache effectiveness is measured by the percentage of cache "hits" for

a varying number of queries. More specifically, 5 and 6 present the LRU and GA caching hit rates for a cache population reproduced for 1000, 2000, ..., 5000 queries. Crossover probability equals 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 has a stronger impact on systems with many information spaces (Figure 5). It is interesting to note that under a system with 64 information sources, the cache hit rate has a slighting decreasing slope as the number of queries increases, whereas the same figure shows a linear type curve as the number of queries increases under a system with 1024 information sources.

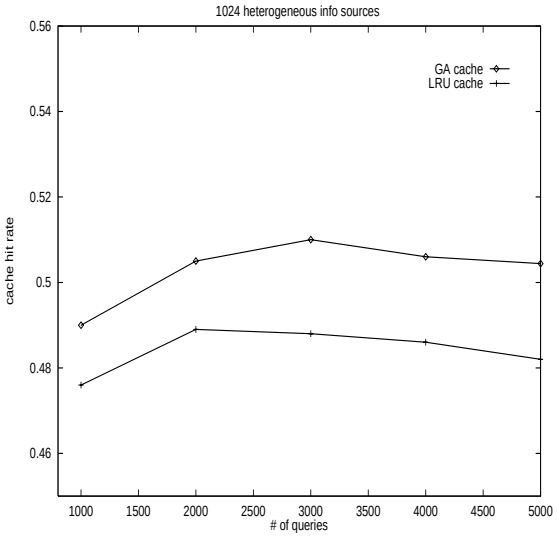

Figure 5: Cache hit rates (1024 IS).

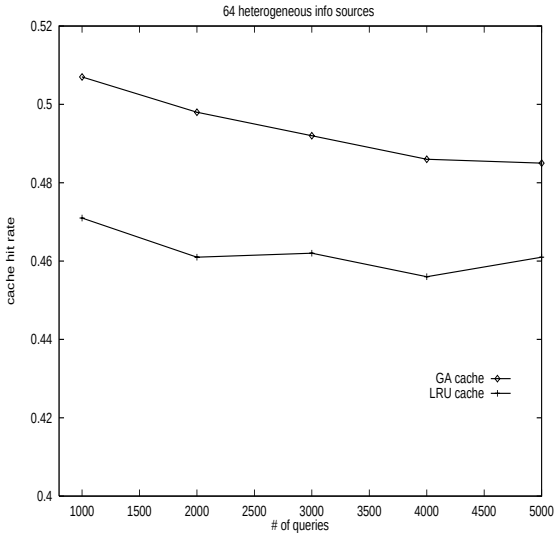

Figure 6: Cache hit rates (64 IS).

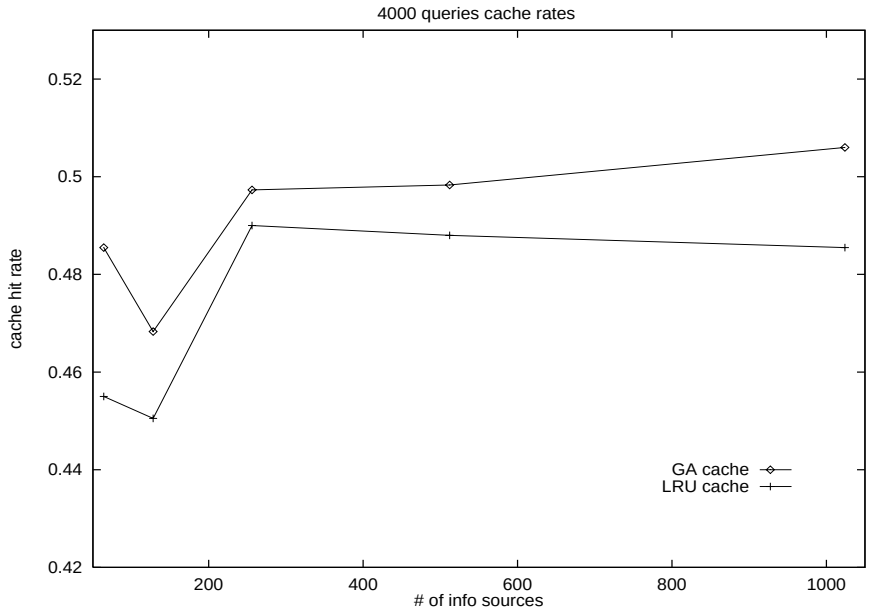

Figure 7: Cache hit rates, 4000 queries.

Similarly, Figure 7 presents the LRU and GA caching hit rates for a cache population reproduced for 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 sources. More specifically both GA and LRU caching seem to be quite unstable for less than 200 information sources. The benefits of GA are pointed out as the number of information sources increases since its cache hit rate shows an increasing cache hit rate whereas LRU has a slight decreasing for the same number of information sources.

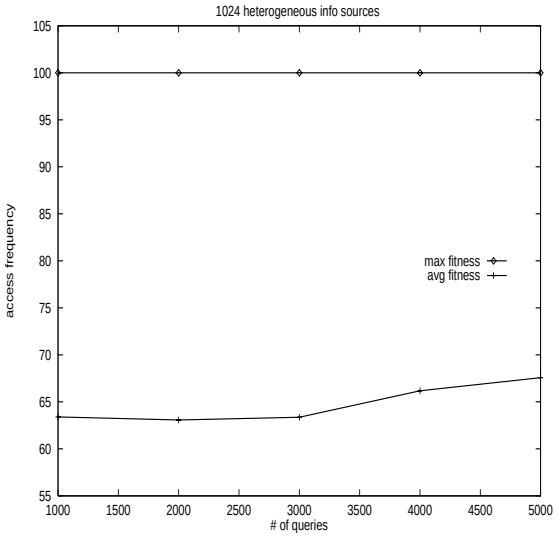

Figure 8: Cache fitness (1024 IS).

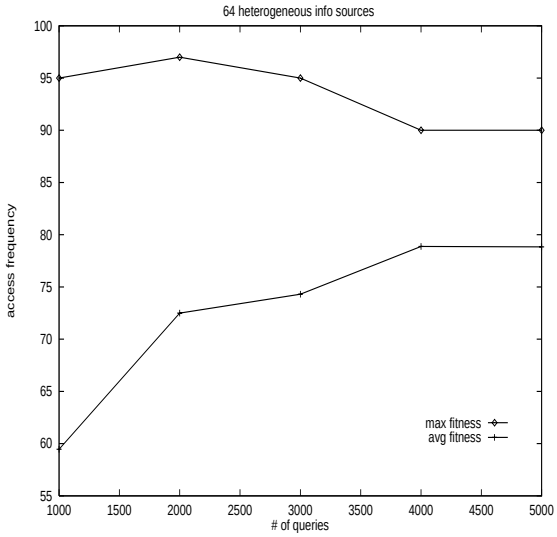

Figure 9: Cache fitness (64 IS).

Figures 8 and 9 depict the results for the average and maximum access frequency of the cached objects under GA caching, for 1024 and 64 heterogeneous information sources, respectively. These figures refer to runs for a cache population reproduced for 1000, 2000, ..., 5000 queries. Note, also, that the cache access frequency is the actual fitness measure needed for the GA process. In case of having 1024 information sources (Figure 8) 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 (Figure 9). 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 these figure, the population fitness stabilizes as the number of information sources is kept high (e.g. 1024 ISs) and the number of queries gets increased.

Figure 10 presents the average and maximum GA caching fitness for a cache population reproduced for 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 5 as well.

The main observation here is that GA caching scheme has shown the best cache hit rate and access frequency. GA scheme has always resulted in better metrics than the corresponding

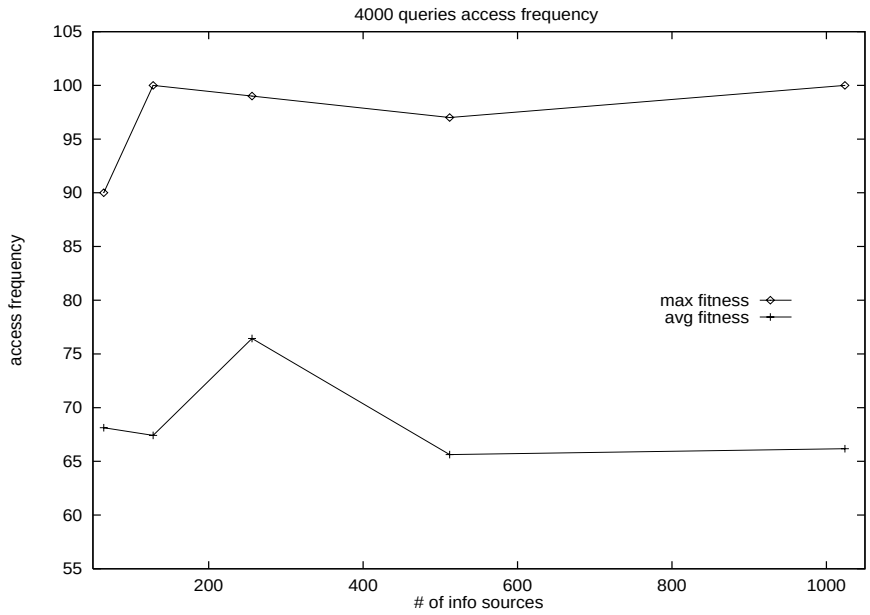

Figure 10: Cache hit rates, 4000 queries.

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.

6 Conclusions - Further Research

The caching technique has been introduced towards an effective object circulation across heterogeneous information sources. The adoption of two different policies in order to cache objects at the client side is proposed in order to enhance the performance of such complex diverse systems. The evolutionary computation type caching, evolved by the use of a typical Genetic Algorithm scheme has been proven to be quite effective. The models were experimented under simulated data sets for a varying number of information sources and certain conclusions were raised about the proposed GA scheme. The GA scheme has been proven superior than the usual LRU caching update since cache population evolved over the simulation time for an increasing number of queries. Crossover and mutation operations play an important role to the cache reform and their probabilities could be kept to the values proposed by the standard GA in order to result to an effective cache content.

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

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

[4] A. Bestavros, R.L. Carter and M. Crovella: Application-level Document Caching in the Internet, *Proceedings 2nd International Workshop in Distributed and Networked Environments*, SDNE 1995.

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

[6] 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, Aug 1996.

[7] 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.

[8] M. Fernandez, D. Florescu, J. Kang, A. Levy and D. Suciu: STRUDEL - a Web Site Management System, *Proceedings ACM SIGMOD'97 Conference*, Vol.26, No.2, pp.549-552, Jun 1997.

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

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

[11] 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, May 1997.

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

[13] 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*, Feb 96.

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

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

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

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

[18] D. Muntz and P. Honeyman: Multi-level Caching in Distributed File Systems, *Proceedings USENIX Winter 1992 Technical Conference*, Jan 1992.

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

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

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

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