An evolutionary scheme for Web Replication and Caching.

Athena Vakali
Department of Informatics
Aristotle University
54006 Thessaloniki, Greece
email: avakali@csd.auth.gr

Abstract

Design and implementation of effective caching schemes has been a critical issue with respect to World Wide Web objects circulation and availability. Caching and replication have been combined and applied in prototype systems in order to reduce the overall bandwidth and increase system's fault tolerance. This paper presents a model for optimizing access performance when requesting Web objects across distributed systems. The replication and caching scheme is designed by the use of an evolutionary computation algorithm. Cached data are considered as a population evolving over simulated time, replicating the most prominent data to dedicated replication servers. The simulation model is experimented and tested under cache traces provided by the Squid proxy cache server at the Aristotle University of Thessaloniki. Cache hit rates and bytes hit length are reported showing that the proposed evolutionary mechanisms improve cache consistency and reliability.

Index terms: World-Wide Web caching, Web replication and caching, cache consistency, evolutionary computation, genetic algorithms.

1 Introduction - Previous Work

The continously rapid growth and worldwide expansion of the Internet has introduced new issues such as World-Wide Web (WWW) traffic, bandwidth insufficiency and distributed objects exchange. Web caching has presented an effective solution, since it provides mechanisms to faster web access, to improved load balancing and to reduced server load. Cache efficiency depends on its content update frequency as well as on the algorithmic approach used to retain the cache content reliable and consistent. Most web servers are reinforced with proxy cache servers which result in web objects coming closer to end users by adding specific cache consistency mechanisms and cache hierarchies. Several approaches have been suggested for more effective cache management and the problem of maintaining an updated cache has gained a lot of attention recently, due to the fact that many web caches often fail to maintain a consistent cache. Several techniques and frameworks have been proposed towards a more reliable and consistent cache infrastructure [6, 3, 18].

Cache consistency mechanisms have been included in almost every proxy cache server (e.g. [9, 21]) and their improvement became a major research issue. In [12] a survey of contemporary cache consistency mechanisms in Internet is presented and 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 potential of document caching at the application level is discussed in [2] to address the need for better resource management towards documents latency reduction. The design and efficiency of a *Cache hierarchy* is a major issue in most proxy caches and in various research efforts [4, 9, 21]. The improvement in performance of Internet information systems supporting hierarchical proxy caches is argued since cache hierarchy is introduced in order to result in better systems scale. In [4] it is shown that hierarchical caching of ftp files could eliminate half of all file transfers, whereas the efficiency of proxy cache operations is argued in [14] under a distributed WWW cache by using election algorithms and an hierarchy similar to the xFS, while in [8] a low-level simulation of a proxy cache considers further details as connection aborts in order to extend the high-level metrics being used so far.

Caching and replication is discussed in [1] where the performance of a proxy cache server is evaluated and validated. Caching and replication have proved to be beneficial in both the circulation of web objects and the Web server's functionality. The need for replication is discussed in [22] where an alternative approach suggests the wide distribution of Internet load across multiple servers. Furthermore, prefetching and caching are techniques proposed to reduce latency in the Web. Several bounds on the performance improvement seen from these techniques have been derived under specific workloads [13]. The replication and caching methodology has raised a lot of research and implementation interest. Some working groups and research teams have been established for a co-ordinated replication and caching framework within the Internet community [16, 10].

Evolutionary computation policies have been used to solve scientific problems demanding optimization and adaptation to a changing environment. The idea in these approaches is to evolve a population of candidate solutions to a given problem, using operations inspired by natural genetic variation and natural selection (expressed as "survival of the fittest"). Usually grouped under the term evolutionary algorithms or evolutionary computation, we find the domains of genetic algorithms, evolution strategies, and genetic programming. 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. More specifically, GAs have been applied in the areas of scientific modeling and machine learning, but recently there has been a growing interest in their application in other fields [11, 23, 5, 17, 15, 7].

This paper presents a model based on an evolutionary computation approach in order to design and simulate an effective Web replication and caching scheme. The model is implemented by an algorithmic approach adapted to the Genetic algorithm process. The implementation is based on the Squid proxy-cache server specifications for representing the Web objects as individuals to be cached and replicated. The simulated model is experimented under real Squid cache traces and cache log files. The contributions of the paper are twofold. First, a caching scheme is maintained by the use of evolution over a number of successive "populations" of cached objects. Second, replication is introduced to extend the caching scheme and the objects chosen for replication are identified by their preservation on the successive steps of the evolutionary scheme.

The remainder of the paper is organized as follows. The next section describes Web proxy cache environments and various cache infrastructures, with emphasis on the Squid proxy cache. Section 3 presents the design and structure of the replication and caching model which is based on evolutionary computation. Section 4 discusses the model's implementation details and operational functions whereas results from trace driven experimentation are presented in Section 5. Results refer to cache hit rates, byte hit lengths and file types hit rate. Section 6 points some conclusions and discusses potential future work.

2 Web Proxy Caches

Caching was initially introduced to provide an intermediate storage space between the main memory and the processor, relying on locality of reference by assuming that the most recently accessed data has the highest potential of being accessed again soon. Caching was extended to Web servers in order to improve client latency, network traffic and server load.

A Web cache is an application residing between Web servers and clients. Cache server watches requests for Web objects (html pages, images and files). If there is another request for the same object, cache will use the copy it has, instead of asking the original server for it again [20]. The main Web caches advantages are the reduce in both latency since request is satisfied by the cache being closer to the client and traffic since each object is gotten from the server once, thus reducing the bandwidth used by a client. Proxy caches are reinforced with set of rules to determine whether an object will be served by the cache or not. Some of these rules are set in the http protocols and some are set by the cache administrator.

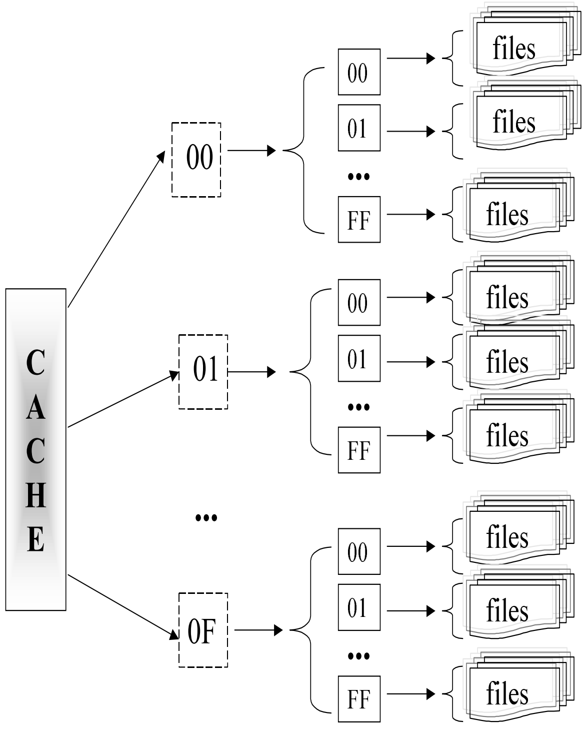

Figure 1: Structure of the Squid proxy cache area.

Nowadays a variety of cache servers are available for the World-Wide Web caching, most of them freely-distributed on the Internet. Most of the recent Web servers application include caching modules (e.g. Apache, Spinner, Jigsaw, Purveyor). A brief description of the three most wide-spreaded proxy cache servers follows :

- *CERN proxy server* has been widely adopted since there was a large infrastructure of the CERN web servers already installed. A heuristic known as *time-to-live* (TTL), was used to manage object's staleness. TTL is implemented by using the last date modification header included in every reply from a Web server. A TTL timing frame based on that date, accompanies each document places in the cache [24, 12].

- *Netscape Proxy Server* has been available commercially since 1995 and checks object's staleness by supporting TTL frame based on object's age when it is cached. This server also supports pre-emptively fetch groups of linked web pages according to a schedule and has a variety of filtering options for use as a firewall proxy.

- *Harvest cache* software was developed with the aim of making effective use of the information available on the Internet, by sharing the load of information gathering and publishing between many servers. Harvest produced the ICP protocol for co-operation between individual caches. Newest Harvest developments are available commercially whereas a team from the National Laboratory for Advanced Networking Research (NLANR) have continued to provide a free version under the name Squid [26]. Squid has evolved by additional features for objects refreshment and purging, memory usage and hierarchical caching. Harvest and Squid have been adopted widely by many institutions and research organizations as a new proposal for efficient caching.

The Squid Proxy Cache is further discussed since the present paper develops a simulation environment based on the Squid cache model and experiments are made by the use of Squid trace log files. Squid caching software has gained a lot of attention lately, since it is used on an experimental network of seven major co-operating servers across U.S.A., under a project framework by NLANR [19]. These servers support links to collaborating cache projects in other countries. Aristotle University has installed Squid proxy cache for main and sibling caches and supports a Squid mirror site. The present paper uses data from this cache installation for experimentation.

Figure 1 represents the organization of Squid cache hierarchy storage-wise, consisting of a two-level directory structure. Assuming approximately 256 objects per directory there is a potential of a total of 1,048,576 (=16 ×256×256) cached objects. Squid uses a lot of memory for performance reasons since an amount of metadata for each cached object is kept in memory. Squid switched from the TTL base expiration model to a Refresh-Rate model. Objects are no longer purged from the cache when they expire. Instead of assigning TTLs when the object enters the cache, now a check of freshness requirements is performed when objects are requested. The refresh parameters are identified as *min_age, Percent* and *max_age*. *Age* is how much the object has aged since it was retrieved whereas *lm_factor* is the ratio of age over the how old was the object when it was retrieved. *expires* is an optional field used to mark an object's expiration date. *Client_max_age* is the (optional) maximum age the client will accept as taken from the http cache-control request header. The following algorithm is used by Squid to determine whether an object is stale or fresh.

```
if Age > Client_max_age then
                Return "STALE"
else if Age <= min_age then
                Return "FRESH"
else if (expires) then                  // expires field exists
         if (expires <= NOW) then Return "STALE"
         else  Return "FRESH"
else if Age > max_age then
                Return "STALE"
else if lm_factor < Percent  then
                Return "FRESH"
else Return "STALE"
```

Squid keeps size of the disk cache relatively smooth since objects are removed at the same rate they are added and object purging is performed by the implementation of a Least-Recently-Used

(LRU) replacement algorithm. Objects with large LRU age values are forced to be removed prior objects with small LRU ages. Squid cache storage is implemented as a hash table with some number of hash "buckets" and store buckets are randomized so that same buckets are not scanned at the same time of the day [26, 25].

3 The Replication and Caching Model

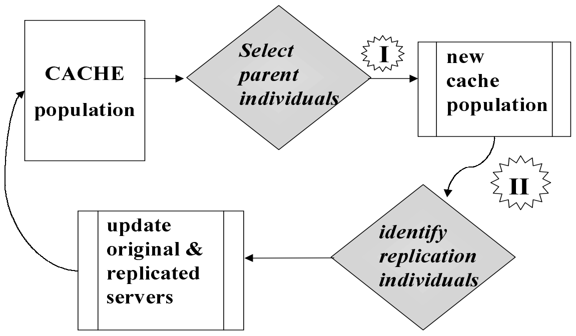

Figure 2: Structure of the algorithm for Caching and Replication.

As reported already caching alone cannot solve problems related to document retrieval latency times, objects availability, reduction in data transfered and redistribution of network accesses [1]. Replication has been suggested in order to increase availability of data while it imposed the need for Web object changes propagation between the original and replicated sites. Therefore, in the replication and caching scheme there is a need to maintain a mechanism in order to result in consistency and reliability between the original and the replicated servers. The basic idea of the model presented here is to support caching and replication under a scheme which is evolved over simulated time by an iterative approach resembling the GA process.

1. *Problem Statement* :
   A cache is maintained on a primary server with its entries being the file objects stored in the cache area. The problem is to improve the cache content on a primary server by reinforcing it with accompanying caches formed by replication of selected file objects on nearby servers.

2. *Encoding* :
   A string representation was used to identify each cached file object. The actual Web object that can be cached and replicated, is identified by the filename where it's stored. Squid objects are the last level in the cache storage hierarchy (Figure 1), stored in files with filenames coded as hexadecimal numbers strings (e.g. 001af200, 000be301 are the actual object's filenames). Therefore, each individual cached object is encoded as a binary bit string corresponding to the hexadecimal string of the object's filename.

3. *Objective Function* :
   each cached object is assigned with a "fitness" value derived by a function used to characterize its "freshness". Since fitness function drives the evolution of the population, it is important to reward the stonger(improved) cache content individuals. Therefore, a metric characterizing cache object's freshness will be the best choice for the evolution of the replication and caching scheme. As described in Section 2 all proxy caches relate their object's refresh policy with timing object's last modification period. Therefore, in our implementation for the replication and caching model, each individual object's fitness will be evaluated by a factor corresponding to the ratio of object's "ages" since its retrieval and its last modification.

   Therefore, the fitness function is given by,

$$Fitness_{object} = \frac{T^{Object-retrieval}}{T^{Object-age}}$$

   where the nominator corresponds to the time that passed since the object's retrieval and the denominator is the age of the object at the time of its retrieval. The fitness function for each cached object considers its "cost" while it remains in cache. It is important to allow infeasible solutions into the population because good solutions are often the result of breeding between a feasible and infeasible solution.

4. *The Algorithm* :
   The algorithm commences with an initial population of individual cached Web objects, which is updated at each evolutionary step resulting in a new "generation". A Web object requested by the client, could be in cache area or not. If its not in cache the caches of the replicated servers are checked. The service of each request is performed according to the following algorithm :

```
if (Request in Primary_Cache) then
                 Return cache_hit
else if (Request in Replica_cache) then
                 Return cache_hit
else    file_in_cache
        Return cache_miss
...
if (Refresh_time) then
         Cache_Update
         Replica_Update
...
```

   The Refesh_time is modeled as a flag in the algorithm to identify whether to perform the cache and replica refreshment according to the GA process. As depicted in Figure 2 the cache refreshment is based on the evolution of a cache population by updating the replicated sites at each evolution cycle. The Cache_Update and the Replica_Update (marked as I, II in Figure 2) are performed over simulated time by preserving successive generations of objects to be cached according to the following criteria :

- *Update I* : the current population is refreshed by selection of individual objects which could remain or be purged from the cache area based on their fitness value and the operations of crossover and mutation. (The operations of crossover and mutation are further discussed in the following paragraph).
- *Update II* : individuals to be replicated will be identified by their strength at remaining on the cache area. More specifically, objects that are present to the previous and the resulted new generation are chosen for replication at the appropriate replication server. The replication process directs the chosen Web objects to the dedicated cache area at the appropriate replication server.

5. *Operators* :
   The two genetically-inspired operations, known as *crossover* and *mutation* are applied to

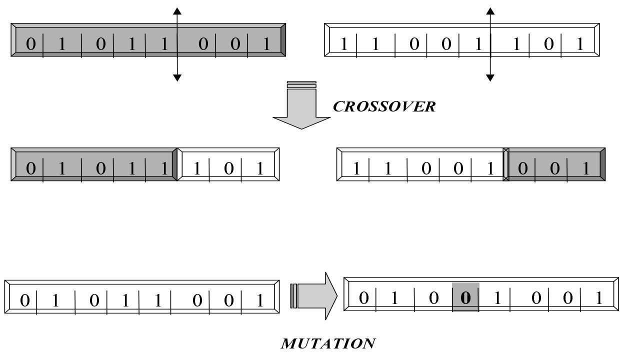

Figure 3: Genetic Algorithm operators: crossover and mutation.

selected individuals in order to successively create stronger generations. Figure 3 depicts an example of applying these two operations in a 8-bit string individual. *Crossover* is applied between two individuals (parents) with some probability. The crossover probability determines whether the two parents will survive in the next generation or whether they will be exchanged in order to result in two new offsprings. 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).

4 Implementation

The model described in the previous section has been implemented in order to optimize the cache consistency and the Web objects access process by the replication of selected cacheable

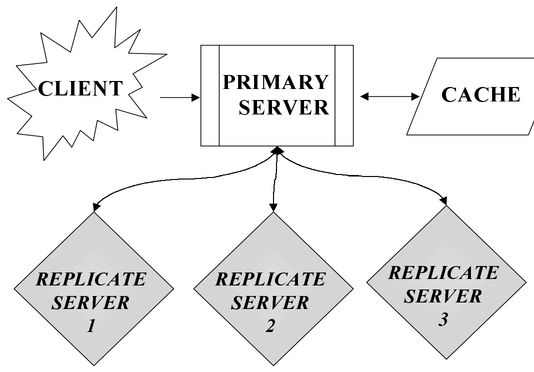

Figure 4: Overview of the Caching and Replication scheme.

objects among the primary and replicated servers. Our GA model follows the Simple GA proposed in [11]. The GA can be adapted to the cache management process since cache consists of a large space of objects (stored files). Figure 4 presents the basic framework followed at the simulation process in order to implement the GA model for replication and caching. In this figure there are three supported replication servers each one in close collaboration with the primary server.

Caches in primary as well as on replicated servers are modeled as hash tables and replication servers are implemented such that each one stores a specific Web object type (e.g. .html pages, .gif, .jpeg files). The population evolves over successive generations progressing within a loop limited by a number of maximum generations specified at each cache and replica update at the simulation run. The evolution process is implemented such that the fitness of the best and the average individual in each generation is improved towards the global optimum. The evolutionary cache environment is simulated such that individual objects are encoded as the cached files under Squid proxy cache. The Squid proxy cache server is installed at the Aristotle University of Thessaloniki (AUTh) and is the top proxy server of Greek academic institutes. Traces from AUTh logfiles have been used to test our cache update model. Squid (in its default configuration) produces four logfiles:

- *logs/access.log*: requests issued to the proxy server with information regarding how many people use the cache, how much each one requested etc.
- *logs/cache.log*: information Squid wants to know such as errors, startup messages etc.
- *logs/store.log*: information of what's happening with our cache diskwise; it shows whenever an object is added or removed from disk.
- *cache/log*: contains the mapping of objects to their location on the disk.

In order to identify the object's fitness function the necessary fields from Squid's log files are used. As pointed out in the previous section each object's fitness function is related to the

Store log fields	
time	the time this entry was logged.
action	either RELEASE, SWAPIN, or SWAPOUT. RELEASE: the object has been removed from cache. SWAPOUT: the object has been saved to disk. SWAPIN: the object existed on disk and has been swapped into memory.
status	the HTTP reply code.
datehdr	the value of the HTTP Date: reply header.
lastmod	the value of the HTTP Last-Modified: reply header.
expires	the value of the HTTP Expires: reply header.
type	the HTTP Content-Type reply header.
expect-len	the value of the HTTP Content-Length reply header.
real-len	the number of bytes of content actually read.
method	HTTP request method.
key	the cache key ; often it's simply the URL.

Table 1: The fields for each individual object in *store.log*

objects freshness / staleness factor) which will be implemented as :

$$fitness_{object} = \frac{\mathrm{n}ow - datehdr}{\mathrm{d}atehdr - lastmod}$$

such that the nominator of the fraction corresponds to the time that passed since the object's retrieval and the denominator presents the age of the object at the time of its retrieval. Fields of *store.log* (described in Table 1) are used to evaluate each of these parameters.

5 Experiments - Results

The present simulator modeled a replication and caching system based on the idea of the GA policy such that the cache reform and the replication process evolves over a number of generations. The simulator was tested under Squid cache traces and by the use of their -corresponding log files. Traces refer to the period from November to December 1998. The proposed replication and caching scheme was applied to cache population at simulated time of reduced request stream. The figures of the present paper, refer to a typical 5-day run. Simulation runs where tested with crossover probability = 0.6 and mutation probability = 0.0333. These probability values have been suggested as a representative trial set for most GA optimizations [11].

Figures 5, 6 and 7 depict the effect of the number of generations to the cache metrics. More specifically, Figure 5 presents the cache hit rate (percentage) for a cache population

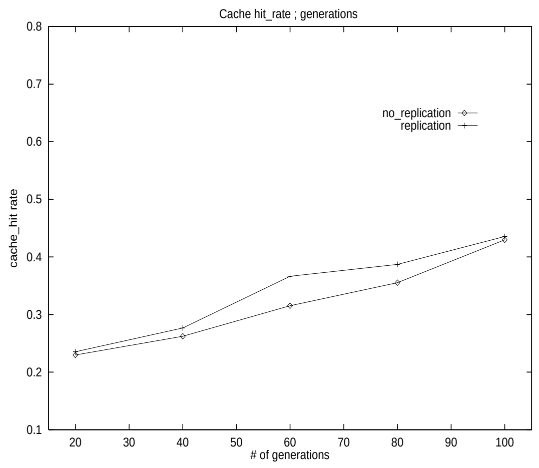

Figure 5: Cache hit rate over generations

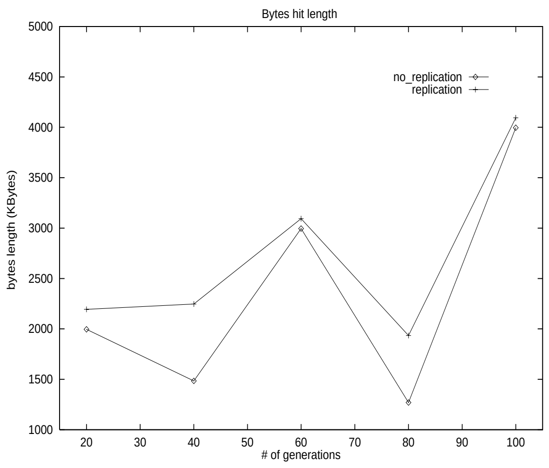

Figure 6: Bytes hit length over generations

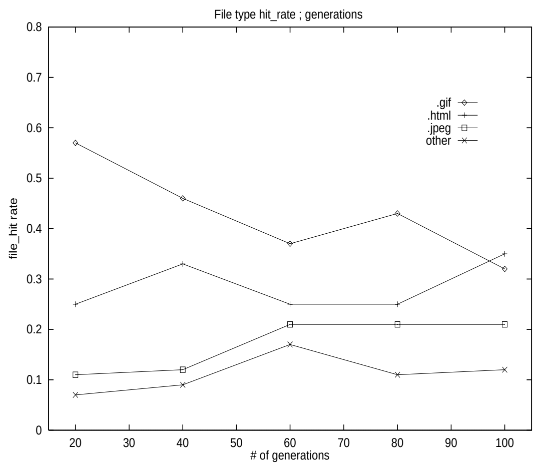

Figure 7: File type hit rates ; generations

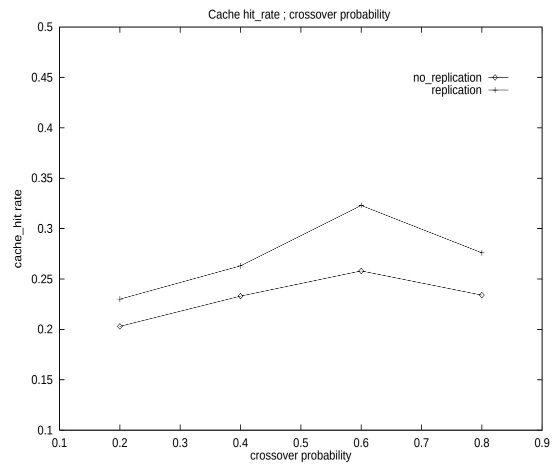

Figure 8: Cache hit rate over crossover

being reproduced for $10, 20, \cdots, 100$ generations. The cache hit rate curves refer to caching with replication and caching without replication. As shown in Figure 5 the replication and caching scheme is beneficial to the overal Web object access since it results in increased cache hit rates. The support of replication together with caching has its best improvement (reaching 14 %) compared to the simple caching when cache update evolves for 60 successive generations. The two schemes seem to converge for a quite small as well as for a quite large number of generations.

Figure 6 presents the Byte hit length (in KBytes) for simulated runs of $10, 20, \cdots, 100$ generations. As shown in this figure, the two schemes result in quite similar curve slopes with the replication and caching overpassing the simple caching scheme at almost 34% for various maximum number of generations (e.g. 40, 80 generations). These results emphasize the importance of adoption of the replication to the caching environment. Figure 7 presents the file type hit curves for the same numbers of $10, 20, \cdots, 100$ generation runs. Files are categorized to *html, gif, jpeg* types which are the most common in cache populations and all *other* types

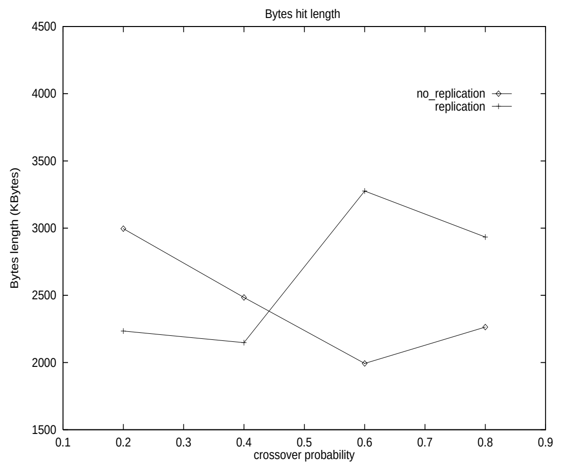

Figure 9: Bytes hit length over crossover

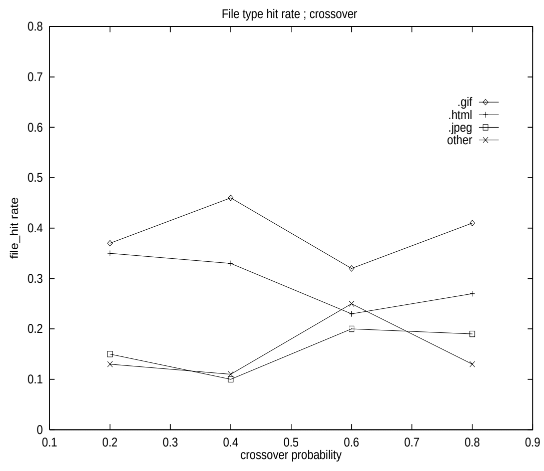

Figure 10: File type hit rates ; crossover

include mostly plain/text files as well as application files. These curves show that the log files include mostly requests for gif and html files, whereas jpg and other files are kept at similar lower rates.

Figures 8, 9 and 10 depict the effect of crossover probability to the cache metrics. Figure 8 presents the cache hit rate (percentage) for a cache population being reproduced under crossover probability $0.2, 0.4, 0.6, 0.8$, for 50 successive generations and with mutation probability $= 0.0333$. The crossover probability is a metric characterizing the overall evolutionary computation run and it is quite interesting to note that the most beneficial result for the replication and caching occurs under the 0.6 crossover probability as suggested by the simple GA trial set. Both replication and no-replication schemes seem to have a similar curve slope with their bigger deviation under the value of 0.6 crossover probability. Similarly, Figures 9 and 10 present the Byte hit length (in KBytes) and the file type hit rate respectively, for simulated runs of $0.2, 0.4, 0.6, 0.8$ crossover probability. The byts hit length is independent of the crossover probability values whereas the .gif files seem to be the most often cached objects for varying crossover probabilities.

6 Conclusions - Further Research

The Web replication and caching problem is studied under an evolutionary computational scheme based on the genetic algorithm idea. A replication and caching model is implemented in a simulation environment. The simulation process included almost all of the necessary parameters to study the model under real traces, such that the most indicative cache metrics (last modification factor, cache length, actions and file types) are represented. The model was tested with the use of real traces provided by a Squid proxy cache server and certain conclusions were raised about the proposed scheme. The replication and caching scheme has been proven quite effective for cache populations evolved over the simulation time under increasing numbers of generations. The replication combined with caching has resulted in beneficial cache hit rates with respect to maximum generation number and crossover probability.

Further research should adapt and examine the above scheme to a real environment and collect results and statistics from a longer period of usage and experimentation. The benefits of supporting Web replication and caching by evolutionary mechanisms should be documented and formalized. Furthermore, the present scheme could be experimented under different fitness selection policies which might consider the number of times an object was accessed while in cache or percentages of object file types. Other evolving computation schemes such as simulated annealing and threshold acceptance, could be adopted in repliation and caching, in order to study their effect on maintaining a reliable cache while increasing data availability and system's fault tolerance.

Acknowledgments

The author thanks Panayotis Jounakis (System administrator) and Savvas Anastasiades (technical staff) of the Network Operation Center at the Aristotle University, for providing access to the Squid cache traces and trace log files.

References

[1] 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.

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

[3] P. Cao, J. Zhang and K. Beach: Active Cache : Caching Dynamic Contents on the Web, *Proceedings of the IFIP International Conference on Distributed Platforms and Open Distributed Processing* , pp. 373-388, Middleware 1998.

[4] A. Chankhunthod, P. Danzig and C. Neerdaels: A Hierarchical Internet Object Cache, *Proceedings of the USENIX 1996 Annual Technical Conference*, pp.153-163, San Diego, California, Jan 1996.

[5] R. Collins and D. Jefferson: Selection in Massively Parallel Genetic Algorithms, *Proceedings of the 4th International Conference on Genetic Algorithms*, Morgan-Kaufmann, pp 249-256, 1991.

[6] P. Danzig: NetCache Architecture and Deployment, *Proceedings of the 3rd International WWW Caching Workshop*, Manchester, England, Jun 1998.

[7] B. Dengiz, F. Atiparmak, A. E. Smith : Local Search Genetic Algorithm for Optimization of Highly Reliable Communications Networks, *IEEE Transactions on Evolutionary Computation*, Vol.1, No. 3, pp. 179-188, Aug 1997.

[8] R. Caceres, F. Douglis, A. Feldmann, C. Glass, M. Rabinovich : Web Proxy Caching : The Devil is in the Details, *Proceedings of the SIGMETRICS Workshop on Internet Server Performance*, Jun 1998.

[9] R. Fieldings et al.: Hypertext Transfer Protocol HTTP/1.1, HTTP Working Group Internet Draft, August 1998.

[10] J. Gettys, T. Bl and H. F. Nielsen : Replication and Caching Position Statement, *W3C position statement*, http://www.w3.org/Propagation/, 1997.

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

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

[13] T. Kroeger, D.D.E. Long and J. Mogul : Exploring the Bounds of Web Latency Reduction from Caching and Prefetching, *Proceedings of the USENIX Symposium on Internet Technologies and Systems*, pp.13-22, Monterey, California, Dec 1997.

[14] M. Kurcewicz, W. Sylwestrzak, A. Wierzbicki: A Distributed WWW Cache, *Proceedings of the 3rd International WWW Caching Workshop*, Manchester, England, Jun 1998.

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

[16] J. Martin : Web Replication and Caching (WREC) , *Internet Engineering Task Force Working Group on Web Replication and Caching*, http://www.terena.nl/ tech/ wrec/, Dec 1998.

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

[18] S. Michel, K. Nguyen, A. Rosenstein and L. Zhang: Adaptive Web Caching : Towards a New Global Caching Architecture, *Proceedings of the 3rd International WWW Caching Workshop*, Manchester, England, Jun 1998.

[19] A Distributed Testbed for National Information Provisioning, http://ircache.nlanr.net/, 1998.

[20] M. Nottingham: Web Caching Documentation, http://mnot.cbd.net.au/cache_docs/, Nov 1998.

[21] O. Pearson: The Squid Cache software, Squid Users Guide, http://www.auth.gr/ SquidUsers/, 1998.

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

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

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

[25] D. Wessels : Configuring Squid Caches. Squid Hierarchy Tutorial, http://www.auth.gr/ Squid/, Aug 1997.

[26] D. Wessels: Squid: Squid Internet Object Cache, http://www.auth.gr/Squid/, 1998.