Evolutionary Prefetching and Caching in an Independent Storage Units Model

Athena Vakali

Department of Informatics
Aristotle University of Thessaloniki, Greece
E-mail: avakali@csd.auth.gr

Abstract. Modern applications demand support for a large number of clients and require large scale storage subsystems. This paper presents a theoretical model of prefetching and caching of storage objects under a parallel storage units architecture. The storage objects are defined as variable sized data blocks and a specific cache area is reserved for data prefetching and caching. An evolutionary algorithm is proposed for identifying the storage objects to be prefetched and cached. The storage object prefetching approach is experimented under certain artificial workloads of requests for a set of storage units and has shown significant performance improvement with respect to request service times, as well as cache and byte hit ratios.

Index terms: *data prefetching and caching, parallel storage units, object-based storage models.*

1 Introduction

According to [3], the amount of storage sold has been almost doubling each year, and storage demands are rapidly increasing due to the complexity and diversity of many current applications. Research has focused on minimizing the so called " I/O bottleneck". Here, we consider an object-based storage model and we propose a prefetching and caching approach in order to reduce data access times and improve data availability over a specified storage subsystem of a number of independent storage units.

Modern disk drives attributes and characteristics have been identified in [11] and their most significant performance factors have been indicated in [10,12]. Network Attached Storage systems [9,1] and *NASD* [4] have been introduced as new scalable bandwidth storage architectures with an object-based storage interface model.

File prefetching has been proven a quite effective technique for improving file access performance. In [7] an analytical-based prefetching mechanism is proposed and the prefetching approach has been proven quite effective while cache miss ratios have been reduced significantly. Recommendations on how to improve and benefit of file prefetching are pointed in [14]. Cooperative prefetching and

T. Yakhno (Ed.): ADVIS 2000, LNCS 1909, pp. 265–274, 2000.

caching is also discussed in [16], where the use of network-wide global resources support prefetching and caching in the presence of hints of future demands. Traditional caching in a distributed file system is discussed in [2].Finally, a web-based evolutionary model has been presented in [15] 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.

This paper presents a theoretical analytical model which introduces the idea of prefetching and caching of data objects collected by a number of independent storage units. The storage units are considered to store information in the form of the so-called *storage objects* which are variable sized data blocks. The prefetching is not based on continuous data allocation but on request frequency of the data storage objects. Storage objects are considered to formulate the individual members of a large "population" residing among a predefined set of parallel independent storage units. An initial process constructs a population of data storage objects to be prefetched on a local cache server from the various storage units. These data blocks populations are evolved such that their members are as frequently requested and efficiently retrieved as possible. The identification of the data objects to be prefetched is done by the introduction of an evolutionary-type algorithm based on the Genetic Algorithm idea. The model performs the request servicing by searching for requested data at the cache area first, then at the other storage units. The prefetching process is applied at regular intervals such that the populations are updated and confront with the requests access patterns. The remainder of the paper is organized as follows. The next section has the definition of the considered object-based storage model and of the storage units characteristics and parameters. Section 3 presents the prefetching and caching approach whereas the requests workload, the model's experimentation and results are presented in Section 4. Section 5 summarizes the conclusions and discusses potential future work.

2 The Object-Based Storage Model

Figure 1 shows the architecture and topology of the proposed storage model. A number of n clients requests data stored among k parallel independent storage units. A local server is considered to host the cache area which is contacted by the clients and a cache controller is assumed to handle and manage the caching and prefetching.

- **The Cache :** A cache acts as a buffer area for storage of the most frequently requested data. The caching policy is based on the idea that when a user (client) requests a piece of data, the cache should be checked first. The cache area is modeled as an information table which contains information about the cached data block(s). Each row in the cache table has an index number which uniquely characterizes objects stored and is also accompanied by a number of attributes such as size, time of its being cached, storage unit it resides. The cache area has a limited predefined size and there is a specific retrieval

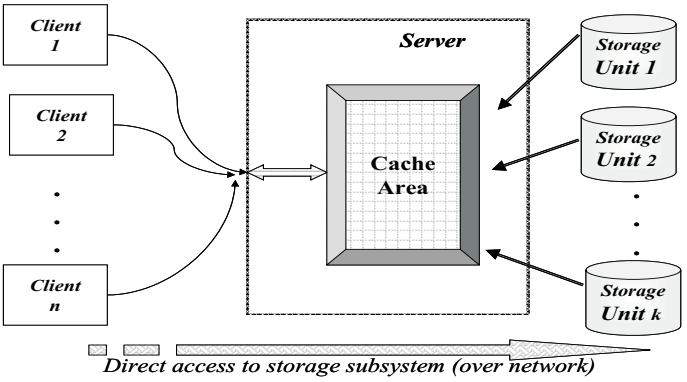

Fig. 1. The Object-based Storage Model

Table 1. The storage model parameters

parameter	description
id_i	object i identification (an index number).
s_i	object's i size, in KBytes.
t_i	time the object i was prefetched.
a_i	number of accesses since the last time object i was accessed.
cr	the cache area retrieval rate, in MBytes per second.
CS	the total cache size, in MBytes.
sr	the storage unit retrieval rate, in MBytes per second.
D	number of independent disks of the storage subsystem .
C	number of cylinders per disk in each storage unit.

rate for request servicing. The parameters associated with the considered object-based storage model are given in Table 1.

The most common performance metrics used for cache characterization are the cache-hit ratio and byte-hit ratio :

- *Cache hit ratio :* represents the percentage of all requests being serviced by a cache copy of the requested data, instead of searching the other original storage unit. Similarly, a *cache miss* is related to requested data not found in cache.
- *Byte hit ratio :* represents the percentage of all data transfered from cache, i.e. corresponds to ratio of the size of data retrieved from the cache area.

– **The Storage Unit :** We mainly concentrate on a multiple disks subsystem where storage units are considered as similar technology disk drives, with similar configuration requirements that can serve requests in parallel under a considered storage subsystem.(Disks parameters are given in Table 1). *Request servicing* is performed by accessing the storage unit which has the

requested data and reading them from this drive. Disks serve requests in parallel in order to exploit the system's responsiveness.
The service time of a request in the disk mechanism is a function of the seek time (ST), the rotational latency (RL) and the data transfer time (TT). [12, 13]. The most widely used formula for evaluating the expected service time involves these time metrics and it is expressed by :

$$\mathrm{E}[\mathrm{D}isk\_Service\_Time] \;=\; \mathrm{E}[ST] \;+\; \mathrm{E}[RL] \;+\; \mathrm{E}[TT] \qquad (1)$$

where E[ST] refers to the expected seek time, E[RL] refers to the expected rotational delay and E[TT] refers to the expected transferring time.
The following function has been used widely for the approximate evaluation of the seek time, which is a major performance factor :

$$Seek\_Time(dist) \;=\; \begin{cases} 0 & \text{if } \; dist = 0 \\ a + b\,\sqrt{dist} & \text{if } \; 0 < dist < cutoff \\ c + d\;dist & \text{if } \; dist \geq cutoff \end{cases} \qquad (2)$$

where a, b, c, d and $cutoff$ are device-specific parameters and $dist$ is the number of cylinders to be traveled. The expected rotational delay is evaluated by $\mathrm{E}[RL] \approx \frac{Revolution\_Time}{2}$ for randomly distributed requests. The transfer time depends on the amount of data to be transferred and is evaluated by $\mathrm{E}[TT] = \frac{Request\_Size}{sr}$ under a constant sr disk drive retrieval rate.

3 The Prefetching and Caching Algorithm

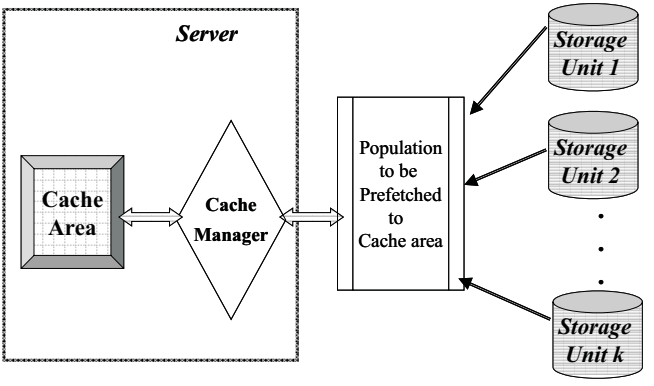

Fig. 2. The Prefetching and Caching Algorithm

Users/clients make requests which refer to data objects stored among the k storage units spread over parallel storage units. Each request refers to an arbitrary amount of data of a certain file. The file system divides the request

into several block-sized segments, each served separately by the file system [14]. Several blocks could be grouped in a cluster or segment in order to define the "storage object" which is either cached or stored at a storage unit. Figure 2 depicts the proposed prefetching and caching process.

Definition 1 : The *storage object* is a group of logically sequential data blocks that are stored consequently on a disk. A number of KBytes corresponding to x data blocks defines the size of each stored object.

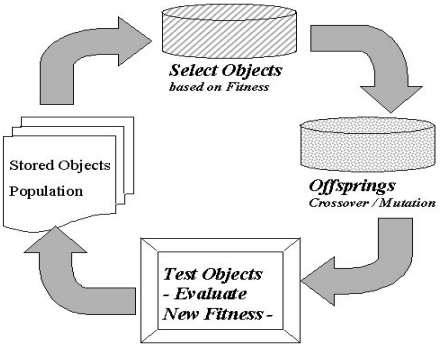

Fig. 3. The Genetic Algorithm Cycle

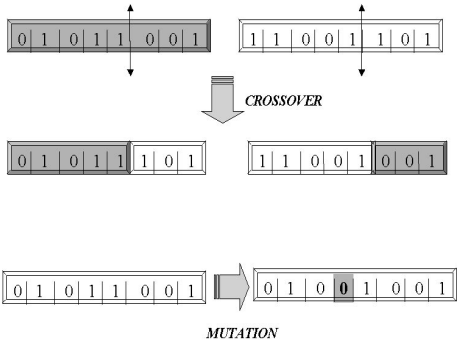

Fig. 4. GA operators: crossover and mutation

The Genetic Algorithm (GA) idea is applied in a considered population of the storage objects as defined above. The GA is used because of two main reasons : First, the basic idea of the GAs is based on the evolution of populations by the criterion "survival of the fittest" and the objects to be prefetched should be the fittest (i.e. the most frequently accessed in best retrieval rates) of the stored objects. Second, the GAs are applied to problems demanding optimization out of spaces which are too large to be exhaustively searched and all storage units have a huge amount of storage objects, impossible to be searched exhaustively in a realistic amount of time. Figure 3 depicts the cycle of a GA applied in a space of individual stored objects. In the present paper, the stored objects are modeled as the individuals considered for evolution. The individuals 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 storage objects. The propose GA model follows the simple GA proposed in [5].

– **Encoding and Operators :** Each stored object individual must be identified according to a predetermined encoded string. The encode 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

encode string. In order to consider the identification of each stored object individual, the stored objects filenames are mapped to the integer values $1, 2, \ldots, O$ where O is the total number of objects to be prefetched in the are reserved for prefetching and caching. Parameters act_i, df_i, cr and s_i are the ones to guide the optimization problem, therefore they are included in the proposed encoding string. Each parameter is assigned a value and the presence of that parameter is signaled by the presence of that value in the ordered encode string. *Crossover* is performed between two stored object individuals ("parents") with some probability, in order to identify two new individuals resulting by exchanging parts of parents' 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. 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 stored objects population will evolve over successive generations such that the fitness of the best and the average stored object individual in each generation is improved towards the global optimum. An objective (or fitness) function is devised based on the need to have a figure of merit proportional to the utility or ability of the encoded stored object individual. Our fitness function is the considered access frequency of each storage object.

– **The GA Prefetching and Caching Algorithm** Each population is formed by the most promising and strong storage objects of all considered storage units. Then, the standard operators defined above mix and recombine the encoding strings of the initial population to form offspring of the next generation. In this process of evolution, the fitter stored object individuals will create a larger number of offspring, and thus have a higher chance of "surviving" to subsequent generations. A pseudo-code version of the GA prefetching and caching algorithm follows :

```
initialize()
old_storage_pop <- initial objects population
evaluate_fitness(old_storage_pop)
generation <- 1
while (generation <= maxgen)  do
    par1 <- selection(popsize, fitness, old_storage_pop)
    par2 <- selection(popsize, fitness, old_storage_pop)
    crossover(par1,par2,old_storage_pop,new_storage_pop,p_cross)
    mutation(new_storage_pop, p_mutate)
    evaluate_fitness(new_storage_pop)
    statistical_report(new_storage_pop)
    old_storage_pop <- new_storage_pop
    generation <- generation + 1
```

In the above GA *maxgen* corresponds to the maximum number of successive generation runs, *popsize* is the stored objects population size, $fitness$ is the

stored objects fitness metric. 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_storage_pop* refers to the initial population in every GA cycle whereas the *new_storage_pop* is the resulting population of each GA run.

4 Experimentation — Results

Table 2. The model's storage units parameters and their values

Secondary Storage Parameters	
Number of Storage Units Drives (k)	10
Seek time parameters (equation 2)	a=3.24ms b=0.4ms. c=8.0ms d=0.008ms. $cutoff$=383.
Number of Cylinders (C)	1936.
Rotational Speed	4002 rpm.
Data Transfer Rate (sr)	10 MB/sec.

Regarding the storage units, the model configuration considered, follows the disk drive configuration for the *HP 97560* disk drive which has been used in previous research [11,6]. The values for the parameters characterizing this particular disk drive are given in Table 2.

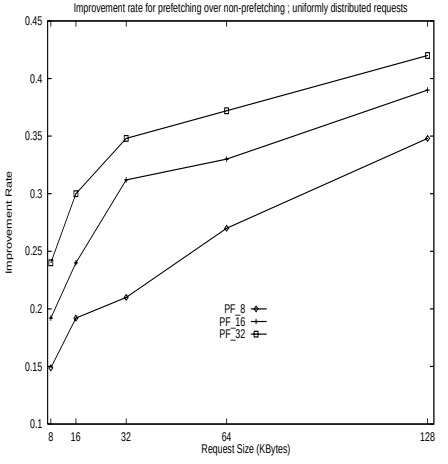

Fig. 5. prefetch/non-prefetch; req size

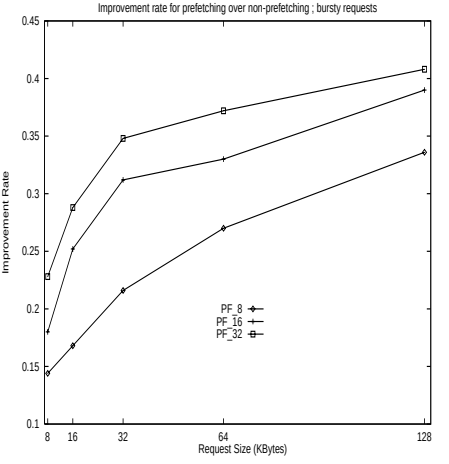

Fig. 6. prefetch/non-prefetch; req size

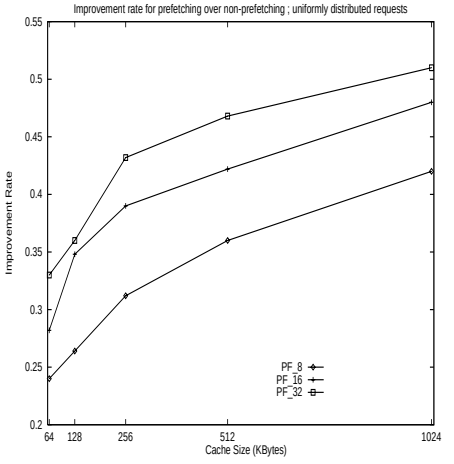

Fig. 7. prefetch/non-prefetch;cache size

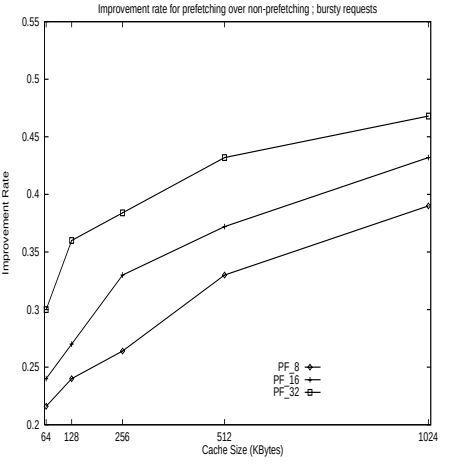

Fig. 8. prefetch/non-prefetch;cache size

The proposed analytic model for prefetching and caching has been experimented under various request workloads. The workload is characterized by its arrival process, its request rate and its burstiness of arrivals. There were more than 100,000 requests generated for every execution cycle and the requests were randomly distributed among the considered storage units. The notations *PF_8, PF_16* and *PF_32* refer to the prefetching and caching algorithm applied on variable sized data blocks grouped into objects of sizes 8, 16 and 32 KBytes, respectively. The prefetching and caching algorithm follows the GA idea and the crossover and mutation probabilities values are $p_{crossover} = 0.6$ and $p_{mutation} = 0.001$ since these values are in the range of suggested representative trial sets for many GA optimizations [5,8]. The initial population for the GA scheme as applied in the considered storage units is generated by a randomly produced population. Furthermore, the typical non-prefetching approach of a conventional storage system has been experimented in order to serve as a basis for comparisons and discussion. The performance metrics evaluated are the metrics defined in Section 2.

The *PF_8*, *PF_16* and *PF_32* approaches were used for prefetching and caching under several requests workloads of various experimentation data sets. The average service time of the storage system has been evaluated by using the results for the service time of each request. The service times have been evaluated for the conventional Non-Prefetching as well as for the three proposed prefetching and caching algorithms, under varying request sizes and under varying cache sizes. The three proposed prefetching and caching algorithms, have been proven to be quite effective compared to the conventional Non-Prefetching strategy.

Figures 5 and 6 represent the percentage rate for the improvement in service times compared to the non-prefetching scheme, under request sizes of $8, 16, \cdots,$ 128 KBytes and under a cache size of 128 KBytes. Figure 5 refers to results

from a workload of 112,000 uniformly ditributed requests and Figure 6 results from a similar 111,800 workload of more bursty requests. All of the prefetching and caching approaches have resulted in better performance metrics compared to the typical non-prefeching scheme. The benefits in service times are improved as the request sizes increase. This is explained since in a non-prefetching scheme, the larger the request size, the more search among storage hierarchies has to be done, resulting in worse service times. The *PF_32* approach is the most beneficial for the storage system since the service times improvement rates get to 42% as the request size increases. This shows that prefetching and caching favors the larger request sizes since the requested data can be accessed faster. Servicing requests by the cache area instead of contacting the original storage unit results in increasing of service times due to the slower characteristics of the disk storage units (ref. parameters Table 2).

Figures 7 and 8 represent the percentage rate for the improvement in service times compared to the non-prefetching scheme, under cache sizes of $64, 128, \cdots,$ 1024 KBytes and under a request size of 32 KBytes. Figure 7 refers to results from a workload of 112,000 uniformly ditributed requests and Figure 8 from a similar 111,800 workload of more bursty requests. All of the prefetching and caching approaches have resulted in better service times compared to the typical non-prefeching scheme. The benefits in service times are improved as the cache sizes increase. This was expected since the data availability in cache is improved when having more space reserved for prefetching and caching. Therefore, the difference in service times between the proposed schemes and the conventional non-prefetching scheme will become more intense as the prefetching and caching area gets increased. Again, the *PF_32* approach is the most beneficial for the storage system since the service times improvement rates can get to about 48% as the request size increases. For example there is a 51% for *PF_32* under cache size of 1,024 KBytes of the first workload and there is a 46% for *PF_32* under cache size of 1,024 KBytes under the second workload. In conclusion, it has been shown by the experimentation that the GA-based prefetching and caching can be a quite effective approach in order to improve the request servicing process in a hierarchical storage model. The performance gain is significant, since service times can be improved at rates of 24%–48% value as the cache sizes and the request sizes increase.

5 Conclusions — Future Work

This paper has presented a study of applying the prefetching and caching idea to a theoretical model of parallel storage units. The proposed approaches for prefetching and caching were based on algorithms of the Genetic Algorithm idea, guided by an objective function in relation to objects frequency of access. The results have shown that the proposed prefetching and caching schemes could be very beneficial for the request servicing process since the improvement in this figure can get as high as 49% for certain request and cache sizes.

Further research should experiment the present scheme under a simulation model which could be based on a certain storage subsystem topology. This is a challenging issue since different storage systems have complex characteristics and requirements that need to be synchronized and parameterized. Furthermore, the use of traced workloads will help in identifying the demerit figures for the proposed model in order to use it in storage system implementations.

References

1. D. Anderson :"Network Attached Storage Research", *http://www.nsic.org/nasd/meetings.html*, Mar., Jun. 1998.
2. M. A. Blaze: Caching in Large-Scale Distributed File Systems, Princeton University, PhD thesis, Jan 1993.
3. G.A. Gibson, J.S. Vitter, J. Wilkes et al.: "Strategic directions in Storage I/O Issues in Large-Scale Computing", *ACM Computing Surveys*, Vol.28, No.4, pp.779–763, 1996.
4. G.A. Gibson, D. F. Nagle, W. Courtright II, N. Lanza, P. Mazaitis, M. Unagst and J. Zelenka : "NASD Scalable Storage Systems", *USENIX 1999 Extreme Linux Workshop* Monterey, California, Jun 1999.
5. D. Goldberg: *Genetic Algorithms in Search, Optimization, and Machine Learning*, Addison-Wesley, 1989.
6. D. Kotz, S.B. Toh and S. Radhakrishnan: "A Detailed Simulation Model of the HP 97560 Disk Drive", Department of Computer Science, Dartmouth College, Technical Report *TR94-220*, Jul 1994.
7. H. Lei and D. Duchamp : "An analytical Approach to file prefetching", *Proceedings of the 1997 USENIX Annual Technical Conference*, Anaheim, California, Jan 1997.
8. Z. Michalewicz: *Genetic Algorithms + Data Structures=Evolution Program*, 3rd edition, Springer-Verlag, 1996.
9. NSIC : National Storage Industry Consortium, *http://www.nsic.org/nasd*, 1999.
10. S.W. Ng: "Advances in Disk Technology — Performance Issues", *IEEE Computer*, Vol.31, No.5, pp.75–81, 1998.
11. C. Ruemmler and J. Wilkes: "An Introduction to Disk Drive Modeling", *IEEE Computer*, Vol.27, No.3, pp.17–28, 1994.
12. E. Shriver: "Performance modeling for realistic storage devices", *Ph.D. Thesis*, Department of Computer Science, New York University, May 1997.
13. E. Shriver, A. Merchant and J. Wilkes: "An Analytic model for disk drives with readahead caches and request reordering", *ACM SIGMETRICS'98*, Conference Proceedings, pp.182–191, Jun 1998.
14. E. Shriver, C. Small and K.A. Smith : "Why does file system prefetching work ?", *Proceedings of the 1999 USENIX Annual Technical Conference*, pp.71–84, Monterey, California, Jun 1999.
15. A. Vakali: A Web-based evolutionary model for Internet Data Caching, *Proceedings of the 2nd International Workshop on Network-Based Information Systems, NBIS'99*,IEEE Computer Society Press, Florence,Italy, Aug 1999.
16. G. M. Voelker et. al : "Implementing Cooperative Prefetching and Caching in a Globally-Managed Memory System", *ACM SIGMETRICS'98*, Conference Proceedings, pp.33–43, Jun 1998.