Caching Techniques for Parallel I/O Servicing

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

Abstract *Parallel and distributed systems architectures support parallel I/O components.Caching has been applied to distributed I/O subsystems as a standard solution to the problems of fastening data accessibility and increasing data reliability. Cache consistency mechanisms have been implemented in order to influence the cache usefulness in a positive way. This paper presents a new caching technique based on the genetic algorithm idea and examines the effect of this technique on the parallel I/O cache consistency and updating process. Cached data blocks on parallel disks are considered as a population evolving over simulated time and are updated at regular intervals towards an improved cache content. The proposed cache update scheme is compared with the LRU caching scheme which has been widely adopted. The proposed technique shows improved performance compared to conventional caching under simulation runs for various workloads.*

Keywords: parallel and distributed I/O subsystems, distributed caching, I/O bottleneck, genetic algorithm applications.

1 Introduction

Performance in parallel and distributed systems has been related to the I/O processing since the drastic increase in processors speeds has been followed by a much slower increase rate in I/O servicing. As it is expected, I/O requirements will keep increasing in parallel and distributed applications and operating systems should be designed to deal directly with the I/O problems [8, 6]. *I/O bottleneck* is a major research subject for parallel and distributed systems since it demands attention and management at all levels of system design. The most widely adopted solution to the I/O bottleneck problem is the use of storage subsystems with parallel functionality and capabilities. Multiple disks are used in parallel in order to icrease data availiability and access parallelism. Data redundant storage has been implemented by several multiple disk schemes attached to one or more controllers and supported by identical or various disk drive configurations.

Caching is a powerful and transparent technique used in order to optimize performance in parallel and distributed systems. It has been applied in a client - server model in order to facilitate and fasten the necessary system interactions. Caching is implemented in the client environment, by retaining copy of server's data for a later use, avoiding the need to re-contact the server's environment. The usefuleness of caching depends upon the idea of accesing again soon the data having been accessed most recently [1, 2, 9]. Cache efficiency depends on its content update frequency as well as on the algorithmic approach used to retain the cache content reliable and consistent. 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 distributed system caches of-

ten fail to maintain a consistent cache. Several techniques and frameworks have been proposed towards a more reliable and consistent cache infrastructure [3].

Genetic Algorithms(GA) belong to the evolutionary methods, used to solve many computational problems demanding optimization and adaptation to changing environments. GA are search algorithms based on the mechanics of natural selection and natural genetics. 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. The main idea in the GA approache 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"). GAs have been applied in various research areas such as scientific modeling, machine learning as well as network infrastructure [7, 10, 4].

This paper deals with the problem of maintaining consistent and updated caches in a parallel I/O subsystem. The caches are updated by adapting the evolutionary computation idea to preserve a consistent cache 'population' of data blocks. Each cache is modeled as a population of server's data blocks and cache content is updated at regular intervals by a GA approach. The model is experimented under simulation runs and the results are compared with the corresponding cache hit rates under conventional LRU caching mechanism.

The remainder of the paper is organized as follows. The next section describes caching in parallel I/O subsystems and presents the GA technique introduced for these caching schemes. Sections 3 presents the experimentation results produced by the simulator developed for the GA caching. Section 4 points some conclusions and discusses potential future work.

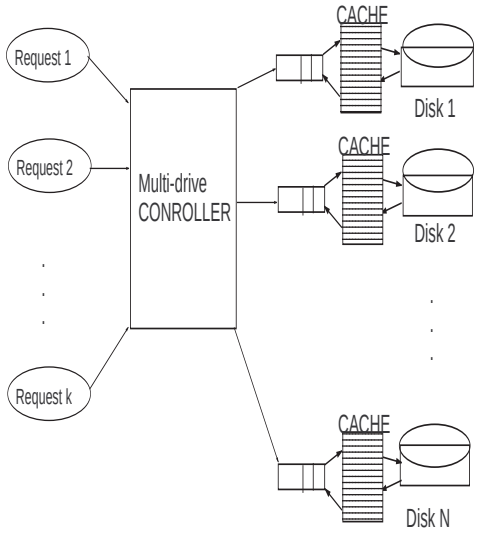

Figure 1: The parallel I/O subsystem.

2 Caching for Parallel I/O

Several parallel I/O subsystems have been suggested in modern parallel and distributed systems. Most of these assumes the hierarchical memory model proposed in [11] where an abstract machine consists of a set of processors interconnected via a high-speed network and each processor access an appropriate I/O controller. Furthermore, modern I/O subsystems are reinforced with quite efficient mechanisms implemented as policies that perform scheduling, reordering of I/O requests or read-ahead. The current complicated storage systems infrastructure hardens the development of analytic as well as simulation models. Disk controller has been considered as the most suitable component for hosting storage systems policies and current technology provides efficient controllers with respect to the disk drive's functionality. Most disk controllers are reinforced with self-managing techniques through standard interfaces used on standard systems without software modifications [5].

Each of these controllers manages a set of disk drives. The controller is responsible for managing and directing read/write requests to the queues of the disk drives. Each disk has an associated cache with server's data blocks be-

ing updated frequently by cache management algorithms (Figure 1).

2.1 Conventional Caching

Caching is a powerful and transparent technique used in order to optimize performance in parallel and distributed systems. Caching was initially introduced to provide an intermediate storage space between the main memory and the processor by relying on *locality of reference* i.e., assume that the most recently accessed data has the highest potential of being accessed again soon. Caching has been applied in a client-server model in order to facilitate and fasten the necessary system interactions.

The caching principle was extended to parallel and distributed systems When servicing an I/O request from the storage device queue the cache is searched first. A request is said to be a *cache hit* when all of the requested data blocks are found in cache. The hit rate is the percentage of all I/O requests being served by the cache. The Least Recently Used (LRU) algorithm is the most popular for the cache updating process and has been used in many caching applications. The cache update is perfomed by replacement of the data blocks used least in the recent past. LRU is quite easily implemented since it takes into account only the time since last access of the cache blocks [9]. Variations of LRU have also been used and implemented in many parallel and distributed caching schemes.

2.2 A Genetic algorithm for Caching

A GA is an iterative procedure that consists of a constant-size population of individuals each one represented by a finite string of symbols, encoding a possible solution in a given problem space. The standard GA generates an initial population of individuals, which is updated at each evolutionary step resulting in a new "generation". The basic idea of the model presented here is to support caching under a scheme which is evolved over simulated time

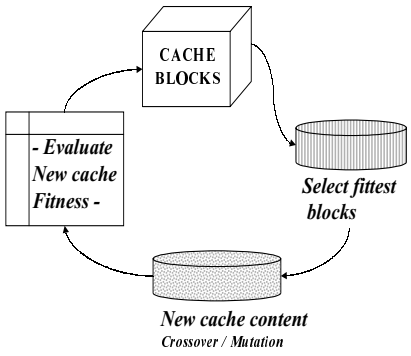

Figure 2: The Genetic Algorithm process.

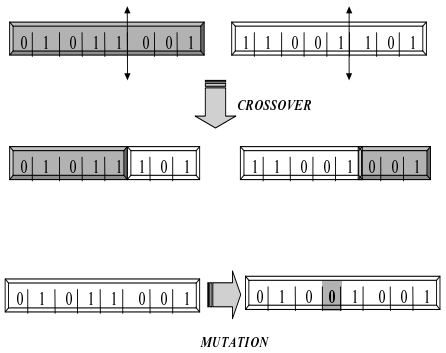

Figure 3: operators: crossover and mutation.

by an iterative approach resembling the GA process.

The individuals in the current population are decoded and evaluated according to some predefined quality criterion, called *fitness function*. Each individual's fitness is an important parameter, 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 depicts these 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' strings. The exchanging of parents parts are performed by cutting each individual at a specific bit po-

sition 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 randomly alters each individual under 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.

In the presented model a cache is maintained on each disk device of the parallel I/O subsystem and cache entries are server's data blocks being requested previously. The proposed model attempts to improve the cache content on each device by applying a GA-based update scheme in order to result in an improved and "stronger" cache content (Figure 2). A string representation was used to identify each cached data block. Each cached object is assigned with a "fitness" value derived by a function used to characterize its "strength" in order to drive the evolution of the cache population efficiently. Therefore, the access frequency of each data block is used as its fitness value. Access frequency has been chosen as the fitness value since it is the most indicative criterion for the decision of data blocks remaining or leaving the cache. Therefore, in the presented model the cache update will result by the cache blocks "re-generation" performed in each disk drive's cache.

3 Experiments - Results

The proposed GA model follows the Simple GA proposed in [7] with a modest crossover probability (0.6) and a low mutation probability (0.0333). The simulator used artificial workload of parallel I/O requests randomly distributed among the disk drives. The GA cache update scheme is implemented in order to perform cache reform and re-"generation"

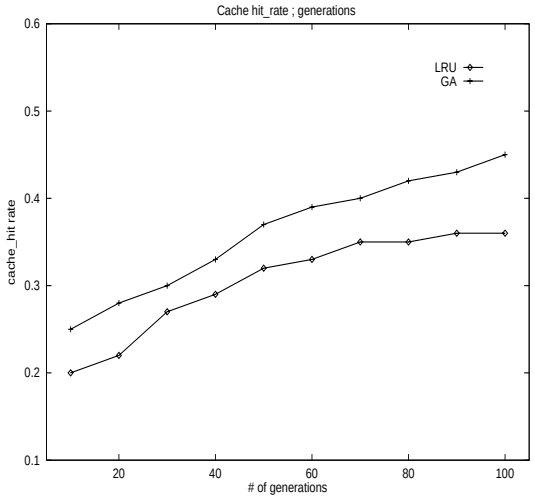

Figure 4: avg/max fitness over generations

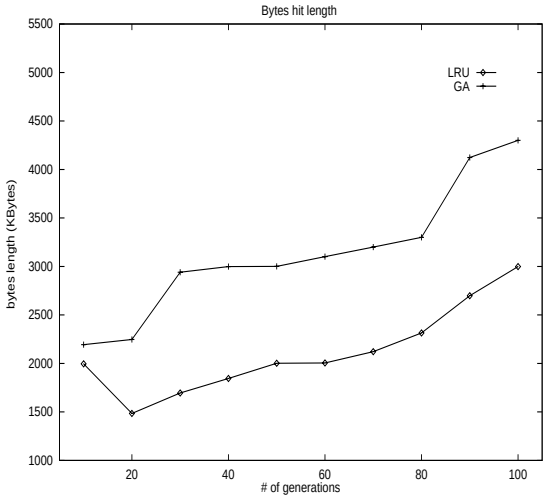

Figure 5: bytes hit; over generations

at regular time intervals. At these regural intervals both LRU and GA schemes have been implemented and tested. Under LRU the cache update is performed by the removal of cache blocks least recently used, whereas under GA the cache is updated by favoring the fittest cache blocks i.e., the data blocks that have been accessed most frequently. The GA scheme has been tested for different number of generations and the corresponding LRU is also evaluated.

The effectiveness of caching is measured by the hit ratio, i.e. the ratio of cache hits to all requests and by the byte hit ratio, i.e. the

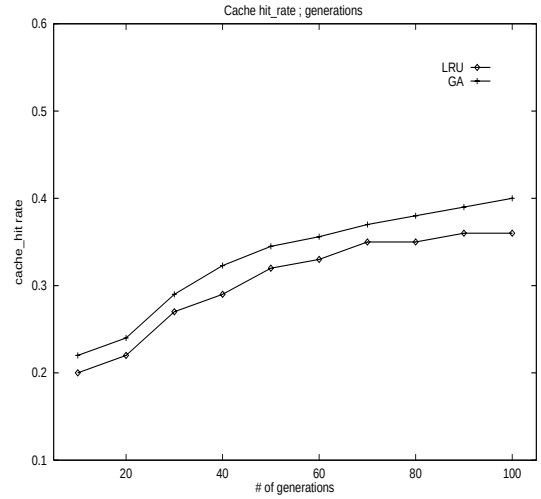

Figure 6: avg/max fitness over generations

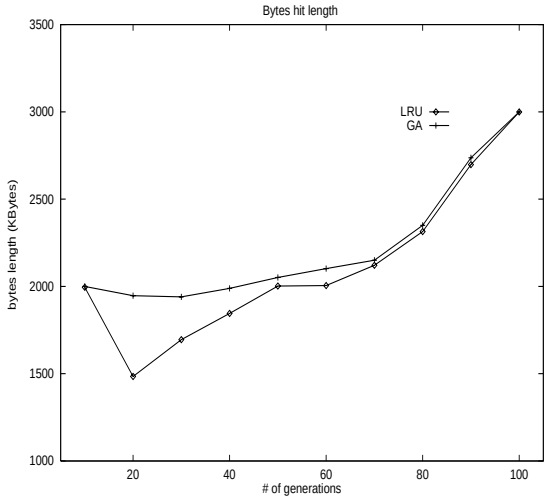

Figure 7: bytes hit; over generations

size of the I/O requests served by the cache. The byte hit ratio is considered as a better metric since it representes the actual volume of the cache served requests. Figures 4, 5, depict the effect of the number of generations to the cache and bytes hit rates respectively, for a cache population being reproduced over $10, 20, \cdots, 100$ generations. As shown in these figures GA caching is shown to be quite beneficial when compared with a conventional caching strategy as LRU algorithm. The improvement is greater as the generation number increases. More specifically, GA is almost 20% better than the corresponding LRU concerning cache hit rates (e.g. cache hit rates for 100 generations run). The GA is also improved considerably (rates of almost 50%) in the bytes hit rate compared to the corresponding LRU scheme.

The crossover operation is crucial with respect to the effectiveness of the GA scheme. Therefore the model is experimented for various crossover probability values, in order to result in a more reliable GA model. Figures 6 and 7, depict the effect of the number of generations to the cache and bytes hit rates respectively, under crossover probability equal to 0.8 and for a cache population being reproduced over $10, 20, \cdots, 100$ generations. The mutation probability remains as in the previous simulation run (Figures 4 and 5). The high crossover probability in the GA will result in a major cache reform and the cache content will be updated more drastically. As shown in these figures, GA caching is shown to be not as beneficial as the similar GA scheme with the lower crossover probability of 0.6. The GA still shows a better performance compared with a conventional caching strategy as LRU algorithm. Again, the improvement is greater as the generation number increases. Therefore, it is quite important for the GA model to run under a "realistic" crossover probability, as suggested by the standard Genetic algorithm model.

4 Conclusions - Future Work

Caching in a I/O subsystem is studied under a Genetic algorithm in order to improve system's responsiveness for parallel I/O request servicing. The simulation process included almost all of the necessary parameters to implement the cache content and the LRU caching policy has been used as a comparative technique. The proposed scheme has been proven quite effective since cache population evolved over the simulation time for an increasing number of parallel I/O requests.

Further research should expand the present scheme under different fitness selection poli-

cies. Other evolving computation schemes (e.g. simulated annealing, threshold acceptance), could be adopted in caching on parallel I/O systems, in order to study their effect on cache consistency and hit rates.

References

[1] M. A. Blaze: "Caching in Large-Scale Distributed File Systems", *Princeton University*, PhD thesis, Jan 1993.

[2] G. Coulouris, J. Dollimore, T. Kindberg: *Distributed Systems, Concept and Design*, 2nd ed., Addison-Wesley, 1994.

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

[4] 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.

[5] R. English and A. Stepanov: "Loge : A Self-Organizing Disk Controller", *HPL-91-179*, HP Labs, Technical Report, Dec. 1991.

[6] 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.

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

[8] R. Jain, J. Werth, J. C. Browne (editors): *Input/Output in Parallel and Distributed Computer Systems*, Kluwer Academic Publishers, 1996.

[9] R. Karedla, J. S. Love and B. G. Wherry: "Caching Strategies to Improve Disk System Performance", *IEEE Computer*, Vol.27, No. 3, pp. 38-46, Mar 1994.

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

[11] J. Vitter and E. Shriver: "Algorithms for Parallel Memory I,II", *Department of Computer Science*, Brown University, Technical Report CS-90-21, Sep. 1990.