24th International Symposium on Automation & Robotics in Construction (ISARC 2007)
Construction Automation Group, I.I.T. Madras

EFFECT OF CROSS OVER OPERATOR IN GENETIC ALGORITHMS ON ANTICIPATORY SCHEDULING

R. Muthu Selvi
Lecturer
Department of Information Technology
Thiagarajar College of Engineering
Madurai, India
rmsit@tce.edu

R. Rajaram
Professor & Head,
Department of Computer Science
Thiagarajar College of Engineering
Madurai, India
hodcse@tce.edu

ABSTRACT

Disk schedulers in current operating systems schedule a request as soon as the previous request has been completed. In many common applications disk read requests are issued synchronously with short period between them In this case, the scheduler switches to a request from another process with the assumption that the last process has no more requests. This condition is called deceptive idleness. It can be overcome by anticipatory disk scheduling framework, which introduces a short, controlled delay period, during which the disk scheduler waits for additional requests to arrive from the process that issued the last serviced request.

Genetic algorithms, powerful and broadly applicable optimization techniques and the most widely known types of evolutionary computation methods can be used for optimizing the anticipatory scheduling. In this paper, we propose to use the cross over operators while tuning the anticipatory scheduling with Genetic algorithms. In general, a cross over operator is regarded as a main genetic operator and the performance of genetic algorithms depends to a great extent on the performance of the cross over operator used. We make the study on Linux Operating Systems kernel 2.6.18. We use various cross over operators and make a comparison on the performance of anticipatory scheduling.

KEYWORDS

Anticipatory Scheduling, Genetic Algorithms, Crossover Operator

1. INTRODUCTION

Disk schedulers select a request for service as soon as (or before) the previous request has completed. Now consider processes issuing disk requests synchronously. After one request has finished, each process issues a new request. So, there is one request at any time. The scheduler assumes that there is no further request from the last process. It selects a request from other process. The scheduler is not able to consecutively service more than one request from the processes. This is the condition called as deceptive idleness. If the data requested is sequential, this problem is common. The scheduler has to reorder the requests correctly and hence there is performance degradation.

The anticipatory disk scheduling framework introduces a short, delay period in such a way that the scheduler waits for additional requests to be issued by the process, which issued the last serviced request. Utilization is improved since the scheduler is able to service more than one request from the same process. The anticipatory scheduling framework is an adaptive heuristics based on a simple cost-benefit analysis [1].

Genetic Algorithms (GA) are stochastic search algorithms, which borrow some concepts from nature. GA maintain a population pool of

candidate solutions called strings or chromosomes. Each chromosome is a collection of building blocks called genes. A fitness value is associated with each chromosome. A user-defined function called fitness function determines the fitness value. The function returns a magnitude that is proportional to the candidate solution‘s suitably and/or optimality. At the start of the algorithm, an initial population is generated randomly or according to some rules. The reproduction operator selects chromosomes from the population to be parents for a new chromosome. The crossover operator oversees the mating process of two chromosomes. It decides what genetic material from each parent is passed on to the child chromosome. The mutation operator takes each chromosome in the offspring pool and randomly changes part of its contents. Thus each new generation of chromosomes are formed by the action of genetic operators on the older population. The chromosomes are compared via their fitness value to derive a new population, where the weaker chromosomes may be eliminated [2].

The work outlined in this paper involves optimizing the read and write requests in anticipatory scheduling using genetic Algorithms and preliminary results are provided. Since the source code is open, we implement the Genetic Algorithms on anticipatory scheduling as external modules in Linux Kernel 2.6.18. We use the results of the study show that significant improvement in performance is possible using Genetic Algorithms for scheduling.

2. OVERVIEW OF ANTICIPATORY SCHEDULING

In disk scheduling, the following three conditions cause deceptive idleness: (1) multiple disk-intensive applications concurrently issuing synchronous disk requests, (2) the intrinsic non-preemptible nature of disk requests, and (3) a work-conserving disk scheduler, which schedules a request immediately upon completion of the previous request. After the request completes, the framework potentially waits briefly for additional requests to arrive, before dispatching a new request to the disk. Applications, which quickly generate another request, can do so before the scheduler takes its decision. Here, deceptive idleness is thus avoided. The fact that the disk remains idle during this short period is not necessarily detrimental to performance. In practice, the framework waits for the shortest period of time over which it expects, in high probability, for the benefits of waiting to outweigh the costs of keeping the disk idle [1].

The anticipatory scheduling framework consists of three components: (1) The original disk scheduler, which implements the scheduling policy and is unaware of anticipatory scheduling; (2) a scheduler-independent *anticipation core*; and, (3) adaptive scheduler-specific *anticipation heuristics* for seek reducing and proportional-share schedulers.

Figure 1 depicts the architecture of the framework. The anticipation core implements the generic logic and timing mechanisms for waiting, and relies on the anticipation heuristic to decide if and how long to wait. This heuristic is implemented separately for each scheduler, and has access to the internal state of the scheduler. To apply anticipatory scheduling to a new scheduling policy, one merely has to implement an appropriate anticipation heuristic.

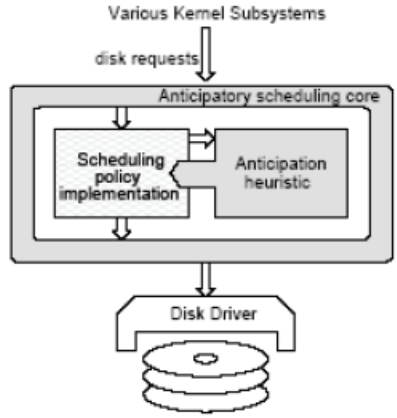

Figure 1 Anticipatory Scheduling Framework

3. GENETIC ALGORITHMS METHODOLOGY

GAs are a population-based model, which uses selection and recombination operators to generate new sample points in the solution space. GAs encode a potential solution to a specific problem on a chromosome-like data structure and applies

recombination operators to these structures in a manner that preserves critical information. Reproduction opportunities are applied in such a way that those chromosomes representing a better solution to the target problem are given more chances to reproduce than chromosomes with poorer solutions. GAs are a promising heuristic approach to locate nearoptimal solutions in large search spaces.

There are three phenotypes that are just fitness measures. The fitness routine looks at the delta of number of I/O operations completed during a child's lifetime. This fitness routine helps balance out the idea of pure throughput. This gives a small fitness bonus to a large number small I/O's. In the throughput phenotype, the fitness simply looks at the number of sectors read or written in during a child's lifetime. This phenotype makes sure data is actually moving, and not just servicing a lot a small requests. The latency phenotype measures the time all requests sit in the queue. This should help combat I/O starvation.

Selection is the process of keeping and eliminating chromosomes in the population based on their relative quality or fitness. In most practices, a roulette wheel approach, rank-based or value-based, is adopted as the selection procedure. In a rank based selection scheme, the population is sorted according to the fitness values. Each chromosome is assigned a sector of the roulette wheel based on its ranked-value and not the actual fitness value. In contrast, a value based selection scheme assigns roulette wheel sectors proportional to the fitness value of the chromosomes. In this paper, natural selection scheme is used. Advantage of natural selection based fitness assignment is it provides uniform scaling across chromosomes in the population and is less sensitive to probability-based selections.

Generally, GAs are composed of two main components that are problem dependent: the encoding problem and the evaluation function. The encoding problem generates an encoding scheme to represent the possible solutions to the optimization problem. In this research, a candidate solution is encoded to represent the sequential, random reads and writes for disk scheduling. The evaluation function measures the quality of a particular solution. Each chromosome is associated with a fitness value, which in this case is the waiting time in the queue. For this research, the smallest value represents the better solution. The fitness of a candidate is calculated here based on its simulated performance.

New chromosomes, called off springs, are formed by (a) merging two chromosomes from the current population together using a crossover operator or by (b) modifying a chromosome using a mutation operator. Crossover, the main genetic operator, generates valid offspring by combining features of two parent chromosomes. Chromosomes are combined together at different crossover rate, which is defined as the ratio of the number of offspring produced in each generation to the population size.

Mutation, a background operator, produces spontaneous random changes in various chromosomes. Mutation serves the critical role of either replacing the chromosomes lost from the population during the selection process or introducing new chromosomes that were not present in the initial population. The mutation rate controls the rate at which new chromosomes are introduced into the population. In this paper, results are based on the implementation of a set of crossover operators, which is mostly used in scheduling problems, and insertion mutation operator that selects a gene randomly and inserts it in a random position.

4. NUMERICAL RESULTS

In the experiments reported in this section, we use Linux kernel 2.6.18.

For each genetic-based scenario, 20 random schedules were generated for the initial population. The poorest 10 schedules were eliminated from the initial population, and the GA population size was kept a constant 10. Natural selection scheme is used that selects the top performers and eliminates the worst performers. The various recombination operators used at a crossover rate 0.9.

4.1. Partially Mapped Crossover

It uses a special repair procedure to resolve possible illegitimacy. Two cut points are selected along the string at random. The sub strings defined

by the two cut points are called mapping sections. Two sub strings are exchanged between parents to produce protochildren. The mapping relationship between two mapping sections is determined. Offspring is legalized using the mapping relationship [3].

4.2. Order Crossover

It can be viewed as a kind of variation of PMX that uses a different repair procedure.

A sub string is selected from one parent at random. A protochild is produced by copying the sub string into positions corresponding to those in the parent. All the symbols from the second parent, which are already in the sub string, are deleted. The resulting sequence contains the symbols the protochild needs. The symbols are placed into unfixed positions in the protochild from left to right according to the order of the sequence to produce an offspring [3].

4.3. Position-based Crossover

It is essentially a uniform crossover for literal permutation encodings incorporated with a repair procedure. It first generates a random mask and then exchanges relative genes between parents according to the mask. It uses a repair procedure to resolve the illegitimacy. It selects a set of positions from one parent at random. A protochild is produced by copying the symbols on these positions into the corresponding positions in the protochild. The symbols already selected from the second parent are deleted. The resulting sequence contains only the symbols the protochild needs. The symbols are placed into unfixed positions in the protochild from left to right according to the order of the sequence to produce an offspring [3].

4.4. Linear Order Crossover

It tends to transmit the relative positions of genes rather than the absolute positions. It selects sub lists from parents randomly. Sublist2 is removed from parent p1, leaving some holes and then slide the holes to the cross section. Similarly, sublist1 is removed from parent1 to form offspring o1 and sublist2 is inserted into the holes of parent p1 to form the offspring O_2 [3].

Insertion mutation is used with probability 0.1. The stopping criteria is that the current population converged (i.e., all the chromosomes have the same fitness value).

4.5. Linux Kernel

Linux kernel 2.6.18 is used for this research.[4] The proc filesystem,which is a pseudofilesystem rooted at /proc contains user-accessible objects that pertain to the runtime state of the kernel and, by extension, the executing processes that run on top of it. The proc filesystem exists only as a reflection of the in-memory kernel data structures it displays. This is used to port the results to domains outside disk scheduling.

Table 1 Fitness Values for Different Crossover Operators

	No of Population			
Crossover	100	200	300	400
Partially mapped Crossover	63.009	71.093	67.987	61.129
Order crossover	62.267	71.093	86.002	61.129
Position-based crossover	61.129	75.489	71.093	62.980
Linear order crossover	63.980	76.987	75.400	64.491

From the table, it is shown that as number of population is increased, Genetic Algorithms performance is improved.

5. CONCLUSION

In conclusion, preliminary data demonstrates that using GAs for Anticipatory scheduling in Linux scheduling can provide improved performance. But there is a strong argument that it adds extra complexity to the kernel. Future work will be conducted to more completely study the effect of changing parameters of the GA, including crossover and mutation rates as well as the methods used for crossover and mutation. Finally, future studies will be conducted to determine the performance improvement by using Lamarckian parameters.

6. REFERENCES

[1] I/O,Sitaram Iyer Peter Druschel, Anticipatory scheduling: A disk scheduling framework to overcome deceptive idleness in synchronous Department of Computer Science, Rice University.

[2] David E.Goldberg, Genetic Algorithms in Search,Optimization and Machine Learning,.

[3] Mitsuio Gen,Ranwei Cheng,. Genetic Algorithms

[4] http://lxr.linux.no/