An Address Pattern Generator for a On-Demand Paging System Simulation

Yoichi MURAOKA*

Abstract

An address pattern generator based on a two stack model is proposed in this paper. Paging behavior of an address pattern generated by this generator is compared with that of an example of an actual pattern, and its usability is indicated.

To show the effectiveness of the generator, paging behavior simulation of a multiprogrammed on-demand paging system is performed using the address pattern generator. The main emphasis is put on the comparison of various scheduling algorithms. It is shown that to give equal CPU time to each program in a multiprogramming mix, memory allocation rather than time slicing plays a key role. Associated overhead to accomplish this goal is also described.

1. Introduction

There are two approaches to analyze the performance of an on-demand paging system. The first approach, an analytic model approach, is suited to model steady state behavior. In this case, it is common to assume that all jobs in a multiprogramming mix are of the same characteristic to make the analysis simple. Furthermore, it is difficult, in general, to analyze dynamic behavior of the system. Another approach is a simulation using a program address pattern (a series of memory addresses generated by a program executuion)[1]. In this case, the result of the simulation is guaranteed if real address patterns are used. It is, however, inconvenient in developing e.g. a scheduling algorithm where the use of various address patterns of predetermined characteristics are desireble.

To meet the above need, we developed an address pattern generator (APG). In this paper, the algorithm of APG and examples of its use are reported.

APG generates an artificial address pattern using a random number generator to drive a discrete event simulator. Although it is impossible for APG to generate any meaningful address pattern, it is enough if its paging characteristic mimics that of a real one. We believe that our address pattern generator is more realistic than those reported elsewhere because of the reason explained in Section 2. By changing APG parameters, address patterns of various paging characteristics can be easily generated.

2. Address Pattern Generator

Address Pattern Generator (APG) generates a memory reference address for

---

This paper first appeared in Japanese in Joho－Shori (Journal of the Information Processing Society of Japan), Vol. 16, No. 8 (1975), pp. 671～677.

* Yokosuka Electrical Communication Laboratory, Nippon Telegraph and Telephone Public Corp.

each instruction. Since our main concern is in paging behavior, it is sufficient to generate a page address rather than a full word/byte address. In [2], address pattern generators based on the LRU stack model and the VSL model were reported.

The LRU stack model is based on a stack in which program page numbers are kept in a First-In-First-Out manner. For each stack position s_i is associated probability p_i. The size of the stack is equal to the number of program pages N. Probability p_i is assigned to each stack position in such a manner that $p_1 \geq p_2 \geq \ldots \geq p_N$ holds, where

$$\sum_{i=1}^{N} p_i = 1.$$

At each step of the address pattern generation, the random number r is generated, where $0 \leq r < 1$, and the stack position s_K, for which

$$\sum_{i=1}^{K-1} p_i < r$$

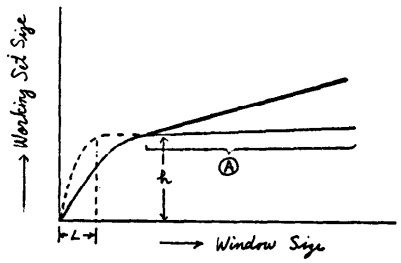

Fig. 1 Working Set Size Characteristic

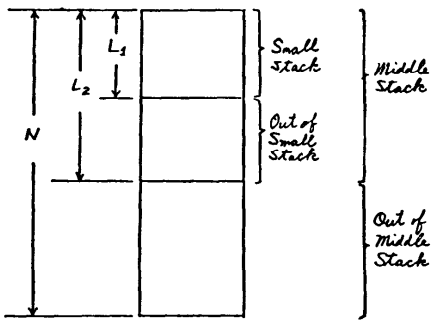

Fig. 2 Two Stack Model Stack Configuration

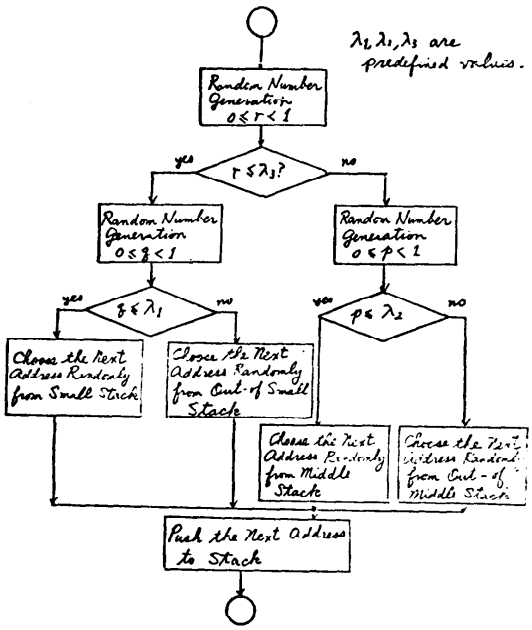

Fig. 3 Flow of Address Pattern Generation

and

$$\sum_{i=1}^{K} p_i \geq r$$

holds, is determined (for the sake of convenience, it is assumed that $\sum_{i=1}^{0} p_i = 0$).

The page number X, which is at the Kth stack position, is taken out of the stack, generated as the page number for this step and is pushed at the top of the stack.

The VSL model can be thought to be a degeneration of the LRU stack model. The specific stack position s_L and probability λ are defined. For the stack position s_i, probability $p_i = \lambda/L$ is associated if $i \leq L$, and $p_i = (1-\lambda)/(N-L)$ otherwise. The process of address generation is similar to the LRU model.

While, in the LRU model, probability must be specified for every stack position; in the VSL model, it is sufficient to define only two parameters, λ and L. Hence the VSL model is more practical than the LRU model. The VSL model, however, has the following shortcoming. The working set size has been accepted as a good measure to characterize the program behavior [3]. Fig. 1 gives an example of the relation between the working set size and the window size. It is rather difficult for the VSL model to generate an address pattern that has a gentle rise as shown in this figure. To simulate part A characteristic, the value of L of the VSL model must be set nearly equal to h, and the value of λ nearly equal to unity. This implies, however, that pages in stack positions s_1 to s_L are used with all equal probability, i.e. 1/L; and it is most likely that the working set size gets to L at window size L as shown in Fig. 1 by the dotted line. If we make the value of λ very small, the working set size will rise monotonously as shown by the heavy line. To circumvent these shortcomings, we have developed a two stack model which is an extension of the VSL model.

In the two stack model, the VSL stack is divided into three parts as

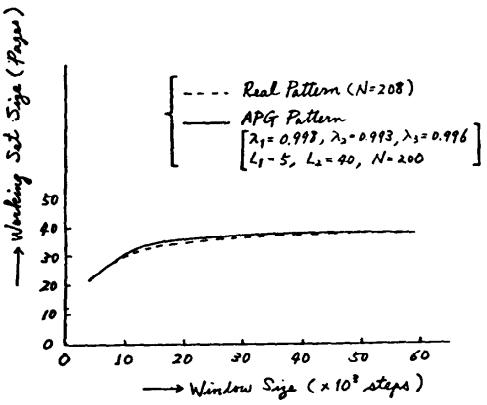

Fig. 4 Comparison of a Real Pattern and an APG Generated Pattern — Working Set Size Characteristic

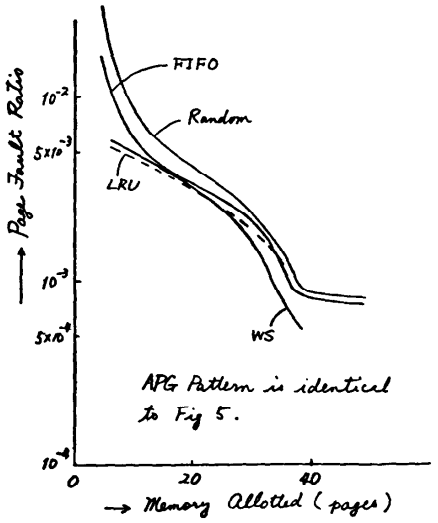

Fig. 5 Comparison of Paging Algorithms Using An APG Generated Pattern

shown in Fig. 2. The address pattern generation algorithm is given in Fig. 3. By setting L_2=N and λ_3=1, the two stack model degenerates to the VSL model. In the model, pages are likely to be selected from the small stack for a short period. In the long run, pages eventually will be selected from a middle stack. Hence, an address pattern of a gentle rise working set size characteristic as shown in Fig. 1 can be generated with ease.

The behavior of the two stack model can be interpreted as follows: A program usually executes using a small number (L_1) of pages for a while. Then it jumps to the other portion of the address space. The place where it jumps usually has some dependence (L_2) to pages that it has been using. In other words, the small stack of the two stack model rules the rise of the working set size characteristic curve, and the middle stack rules A part (Fig. 1). To the contrary, a program may jump to anywhere in the address space in the VSL model. The LRU model depends on the locality principle where the most recently used page has the highest probability to be re-accessed.

Fig. 4 is an example to show how an address pattern generated by APG can match a real one in its paging behavior. Four paging algorithms are compared for the above APG generated address pattern in Fig. 5. Of course, we can not draw a conclusion on the paging algorithm from this. Fig. 5, however, shows a generally accepted tendency that the working set algorithm outperforms the other three; then follows the LRU algorithm, and the random algorithm is the poorest. Judging from this, we may conclude that the address pattern generated by the APG simulates the real one well in terms of the paging behavior.

In general it is difficult to show the correctness and the generality of any simulation. In the case of APG, we assert that the correctness is assured at least qualitatively judging from Figs. 4 and 5.

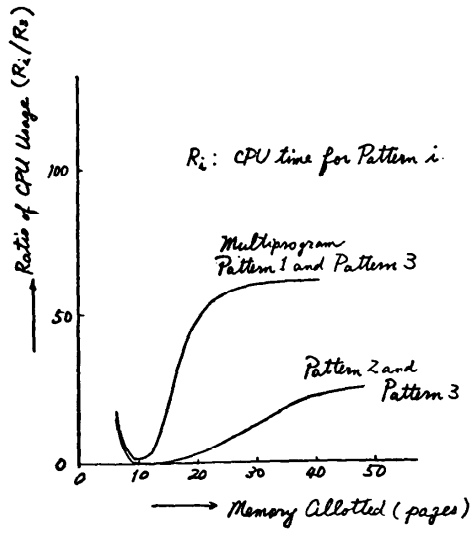

Fig. 6 Run Time Ratio of Programs in Multiprogramming Mix

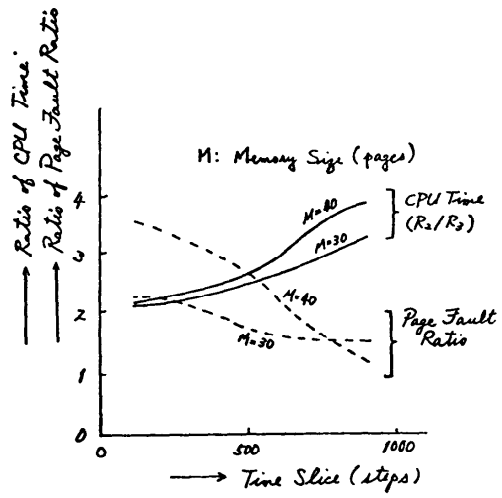

Fig. 7 Effect of Time Slicing

3. The Use of APG

The advantages of simulation using APG are that:

(1) it is possible to discover the phenomena which are otherwise difficult to find, and

(2) it is possible to evaluate the dynamic effect of scheduling algorithms.

In this section we give, as an example of APG usage, a result of the multiprogrammed on-demand paging simulation. Address patterns used for the simulation are generated by the following parameters;

(1) Pattern 1

λ_1=0.998, λ_2=0.9993, λ_3=0.995, L_1=1, L_2=10, N=200.

(2) Pattern 2

λ_1=0.998, λ_2=0.9993, λ_3=0.995, L_1=2, L_2=20, N=200.

(3) Pattern 3

λ_1=0.998, λ_2=0.9993, λ_3=0.995, L_1=5, L_2=40, N=200.

Although these values are adopted in reference to real address patterns, their generality can not be asserted. Hence, what follows is an example of the simulation to show the effectiveness of APG.

We examine the relation between the CPU time usage and the paging algorithm by simulation. The main objective is to see the effect of time slicing in a multiprogrammed on-demand paging system.

If programs in a multiprogrammed mix are dispatched in the FIFO order without time slicing, a program with good characteristic (i.e. a program with the smaller working set size) is expected to run longer and enjoy more CPU time. Fig. 6 verifies this by simulation. In this case, the grobal LRU algorithm is used to multiprogram Patterns 1 and 3 or Patterns 2 and 3. It is clear that programs with smaller working set size (i.e. Patterns 1 or 2) have used up more CPU time than Pattern 3.

To equal CPU time usage among programs in a multiprogrammed mix, time slicing is a commonly accepted scheme. Fig. 7 is a result of simulation of multiprogramming Patterns 2 and 3 incorporating time slicing.

From Fig. 7, we can draw the following observation:

(1) Time slicing is effective to equalize CPU time usage.

(2) The number of page fault increases as a price.

The reason for (2) is thought to be that the dispatching opportunity for a program with the larger working set size increases.

4. Conclusion

In this paper, we presented a new algorithm for address pattern generation and gave some examples of its use. As described in Section 3, by simulation using APG, many new phenomena which are otherwise hard to find can be brought into the light. As mentioned earlier, the results given in this paper are examples. However, by choosing more appropriate address patterns, it becomes possible to analyze the behavior of a multiprogrammed on-demand paging system where programs of various characteristics are in a mix.

The speed of APG simulation is approximately one-hundredth of real time, and the time to stabilize is short. Since various address patterns can be generated by a

small number of parameters, it can be of great help to analyze many phenomena which are otherwise difficult to find.

References

[1] J.W.Boyse: Execution Characteristics of Programs in a Page-on-Demand Systems, CACM, Vol.17, No.4, pp.192-196(1974).

[2] J.R.Spirn & P.J.Denning: Experiments with Program Locality, Proc.FJCC, pp.611-621(1972).

[3] P.J.Denning: The Working Set Model for Program Behavior, CACM, Vol.11, No.5, pp.323-333(1968).