An Analysis of User Behavior and Demand Swapping Policies in Time-Sharing Systems

YASUFUMI YOSHIZAWA* and TOSHIYUKI KINOSHITA*

One bottleneck encountered in large scale *time-sharing systems* (TSS) is excessive interactive swaps. The storage management of operating systems should minimize the number of physical interactive swaps. If working sets can be left resident until users complete their inputs, the number of swaps can be minimized. *A demand swapping policy* which maintains working sets in the main storage and swaps them only when a shortage of free storage space develops is a useful technique for resolving such bottlenecks.

One important aspect of the *demand swapping policy* is the algorithm to determine which working set to swap out. To develop this *demand swapping algorithm*, the trace data of the behavior of actual TSS users at a terminal are accumulated and user input processes are analyzed. Five *demand swapping algorithms* (**LRU**, **RAND**, **LUFO**, **PRED** and **SLRU**) are proposed from the results. The number of physical interactive swaps that come from each *demand swapping algorithm* is compared using a *trace-driven simulator*. From this analysis, it is found that **LRU** is the best algorithms among the *fixed-space demand swapping algorithms*. However, **SLRU**, which is a kind of variable-space demand swapping algorithm, reduces the number of physical interactive swaps more than **LRU** within a given *critical time* range.

1. Introduction

Virtual storage operating systems with *working-set* (*WS*) strategy [12] transfer pages between the main storage and the auxiliary storage. These page transfers can be divided into two classes. The first is caused by the program behavior. This results in operating systems (OS's) transferring pages in order to maintain a *WS* for each process and to dynamically assign each process to its storage area. The second is caused by operating systems in order to dynamically distribute scarce system resources among processes. The first class is called *paging* and the second class is called *swapping*.

Furthermore, swapping can be classified into *forced swapping* and *interactive swapping*. *Forced swapping* is used to control the distribution of scarce system resources among processes with long-running times, which are observed in the batch processing environment. The objective of this resource distribution (*scheduling*) [20] is to maximize the performance objective specified by the installation manager. That is, the objective of *forced swapping* is decided by the scheduling policy [8] of each operating system.

On the other hand, *interactive swapping* occurs typically in the TSS (Time Sharing System) environment whenever a process enters a *long-wait* situation (for example, this situation occurs when a TSS process waits for input from the corresponding terminal). The performance objective of TSS systems is to assure the system responsiveness specified by the installation manager. Therefore, the TSS operating system has to minimize the swapping delay time caused by *interactive swapping*.

As the TSS scale becomes larger, *interactive swapping* becomes more frequent than *forced swapping*. J. P. Buzen [6] has pointed out that in virtual storage operating systems, the number of transfers caused by interactive swapping often exceeds the number of transfers caused by *paging*. Therefore, the number of page transfers caused by interactive swapping has to be minimized by the TSS operating system. This minimization can be realized by shortening the swapping delay time.

Two policies are considered for *interactive swapping*: these are *immediate swapping policy* and *demand swapping policy*. An operating system with an *immediate swapping policy* immediately swaps out the process' *WS* when the process enters a *long-wait* situation (or at the end of the time slice) [27]. The number of physical *interactive swaps* is almost proportional to the number of interactions. On the other hand, an operating system with a *demand swapping policy* does not automatically swap out the process' *WS even if the process enters a long-wait situation. In the demand swapping policy [31], WS*'s are swapped out only when the amount of free storage space is insufficient for required swap-in. As a result, an operating system with the demand swapping policy can decrease in number of physical page transfers and reduce both CPU and I/O load caused by

*Systems Development Laboratory, Hitachi, Ltd., 1099 Ohzenji, Asao, Kawasaki 215 Japan.

interactive swaps.

Demand swapping algorithms [31] are those algorithms which swap out a process only when its memory is needed. These algorithms correspond to the page replacement algorithm which decrease in number of page transfers caused by *paging* in the demand paging system [1][9][30]. When a user types a TSS command at a terminal, the operating system checks to see if the working set still remains. If so, the command may be accepted with no swapping required. If the working set is not resident, a swap is required. When the required *WS* has been swapped out, the TSS operating system has to immediately free a storage space in order to swap in the required *WS*. This storage space can be freed by swapping out a particular *WS* which is selected as a swap-out candidate by a demand swapping algorithm. In the demand swapping policy, the above swapping operations are performed whenever a user types a TSS command. This paper shows that if the demand swapping algorithm uses the past history of the time interval between successive TSS commands for each individual user, the number of physical page transfers caused by interactive swapping can be reduced.

Demand swapping algorithms using the past history of the time interval between successive TSS commands during each TSS session are considered in this paper. Four demand swapping algorithms (Least Recently Used: LRU, Random: RAND, Last Used First Out: LUFO, and Prediction: PRED) have been proposed and analyzed in [31]. A new demand swapping algorithm, Shrunken Least Recently Used (SLRU), is proposed in this paper based on the analysis of the actual TSS user behavior at a terminal. LRU, RAND and LUFO are obtained by analogy with the page replacement algorithm [1][15][22]. That is, the time interval between successive references to the same *WS* is substituted for the time interval between successive references to the same page in the demand paging system. PRED predicts the next command input time for a TSS user by using his past history. SLRU exploits the correlation between successive interval times of TSS commands.

We introduce the performance criterion defined as the number of times which the required *WS* was found in the main storage divided by the total number of commands issued by a TSS user. Those five demand swapping algorithms are evaluated by the performance criterion.

Trace data of the actual TSS user behavior at a terminal were accumulated at the Computer Centre of the University of Tokyo [16][17] for evaluation of demand swapping algorithms. This Computer Centre has been designated as the site for large scale TSS's in Japan. The Computer Centre had a heterogeneous, tightly-coupled multiprocessor consisting of four CPU's (two HITAC 8800's and two HITAC 8700's) which shared a main storage capacity of eight million bytes [23][24]. This system was controlled by a Hitachi operating system (*OS7* [25]), which supported multiple virtual storage space of 2048 million bytes each as well as dynamic linking and a ring protection mechanism both of which were patterned after MULTICS [11]. The HITAC 8800/8700 was the first system to employ both multiple virtual storage space and cache memories.

2. Analysis of Input Process

2.1 Summary of Trace Data and Terminal Usage Patterns

The data for this analysis were collected by a software monitor which is invoked from the terminal input/output controller routine and TSS command

Table 1 Summary of the trace data.

Item	Case 1	Case 2	Case 3	All Cases
Observation period	477 min.	497 min.	599 min.	1,573 min.
Number of interactions	23,574	23,211	24,834	71,619
Number of output interactions	14,814	14,606	15,988	45,408
Average URT*	19.2 sec	18.4 sec	20.9 sec	19.5 sec
Average SRT**	1.53 sec	2.84 sec	1.89 sec	2.08 sec
Average OUT***	10.6 sec	10.3 sec	9.51 sec	10.2 sec
Average cycle time	31.6 sec	31.5 sec	32.3 sec	31.8 sec
Number of sessions	563	563	577	1,703
Average session time	22.9 min.	22.7 min.	25.0 min.	23.6 min.
Average interactions in one session	43.5	43.2	46.4	44.5
Average number of TSS users	26.8	25.7	24.3	25.5

URT*: User Response Time, SRT**: System Response Time, OUT***: Output time

Table 2 Classified terminal inputs

Classified Inputs	Frequency
Commands for Program Execution (compiler, application, etc.)	6.4(%)
Commands for Job Control (logon, logoff, etc.)	3.8(%)
Commands for Text Editting (insert, delete, replace, etc.)	45.2(%)
Parameter Inputs for Program Execution	38.0(%)
Miscellaneous (file maintenance, attention, etc.)	6.6(%)
Total	100.0(%)

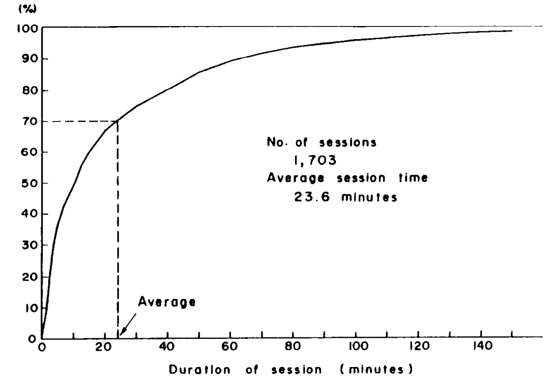

Fig. 1 Distribution of session time.

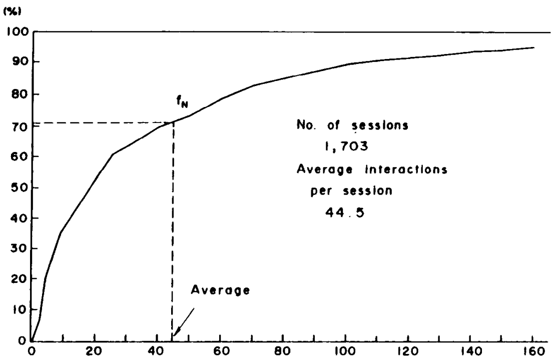

Fig. 2 Distribution of the number of interactions per session.

analyzer routine. This software monitor records the following items:

(1) user identification code;
(2) system and user response times where system response time includes the queueing delays which depend on the system load; and
(3) the leading eight characters of the input.

This information was written out on magnetic tapes during three days in February, 1978, at the Computer Centre of the University of Tokyo. The summary of the measured data is shown in Table 1. The analysis of the

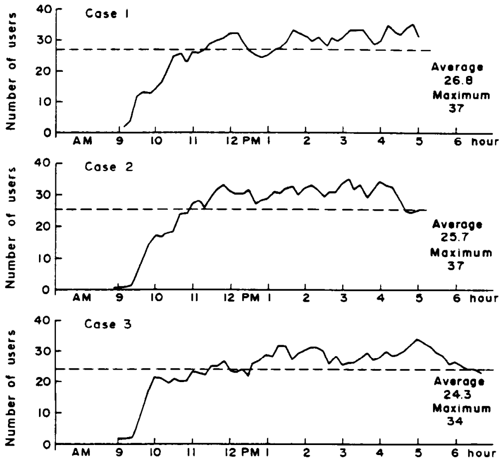

Fig. 3 Number of TSS users.

user input process is based on terminal sessions. The duration of a terminal session is defined as the total time between a user's signing on the system (typing *logon* command) and disconnecting the terminal from the system (typing *logoff* or next *logon* command). Table 2 shows the classified terminal inputs and their frequency.

During the three day period, the system operated for about 26 hours, and there were 1,703 terminal sessions, and 71,619 interactions. Fig. 1 is a plot of terminal session times. The average terminal session time was 23.6 minutes. The plot shows that there were many short terminal sessions; that is, more than 50% were less than 10 minutes long. Fig. 2 shows the distribution of the number of interactions in a terminal session. The average was 44.5. There were 45,408 interactions with printing messages, which represented 63% of all interactions. The mean cycle time, which is the average interval between two successive requests, was 31.8 seconds.

The number of sign-on users was averaged every ten minutes. The results are shown in Fig. 3. The average number of sign-on users for the three days were 26.8, 25.7, and 24.3. In order to prevent deterioration of heavy batch processing, the maximum number of simultaneously signed-on users was set to 39 for this system.

2.2 Components of Cycle Time

The *cycle time* (*CT*) is the interval between two successive requests to the system from a terminal. In general, *CT* can be divided into three portions:

(1) the user response time (*URT*),
(2) the system response time (*SRT*), and
(3) the output time (*OUT*), as shown in Fig. 4.

URT is defined as the time between the system's prompting for the user to enter the next command and the user's typing of the carriage return (sending a *transaction* for processing). *SRT* is defined as the time between

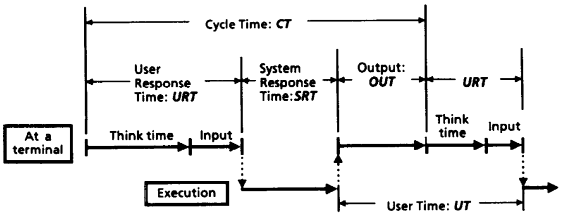

Fig. 4 Typical behavior in TSS interactions.

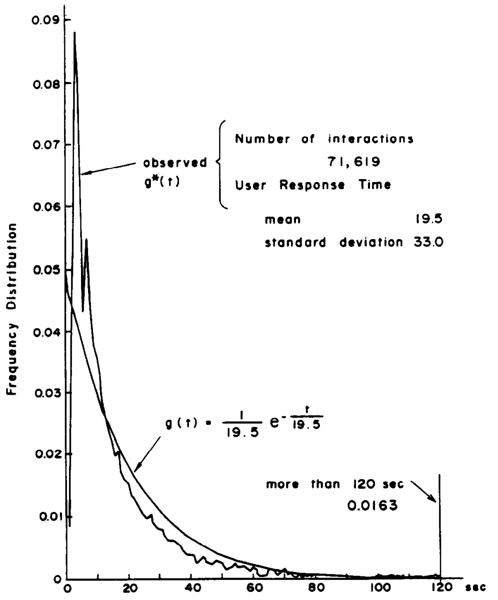

Fig. 5 Comparison of observed URT with that of assumed exponential distribution.

the user's typing the carriage return and the system's next output to the terminal. *OUT* is defined as the time between the system's first output to the terminal and the system's next prompting. If the system's first output is a prompting message, *OUT* is zero. In our observation, the ratio of interactions with non-zero *OUT* is 63%. If we let CT_i denote the *i*-th *CT* in a session, $CT_i = URT_i + SRT_i + \mathrm{OUT}_i$. We define the user time (UT_i) as the time of $OUT_{i-1} + URT_i$, because *WS*'s are not needed in the main storage for this period.

The frequency distribution of *URT*, $g^*(t)$, is shown in Fig. 5. For comparison with the observed *URT* distribution, an exponential distribution with the same mean value, $g(t)$, is also shown in Fig. 5. The mean *URT* is 19.5 seconds. In this TSS, the operating system terminates a terminal session if a *URT* is longer than 600 seconds. The observed distribution of *URT* looks more or less exponential, but the observed data shows more *short URT*'s than does an exponential curve with the same mean value. On the other hand, the exponential curve overestimates the frequency in the 13 to 70 second range. Moreover, the ratio of observed *URT*'s of longer than 120 seconds is 1.63%. This is 7.7 times the cumulative percentage of the *URT*'s, which are longer than 120 seconds in the exponential distribution.

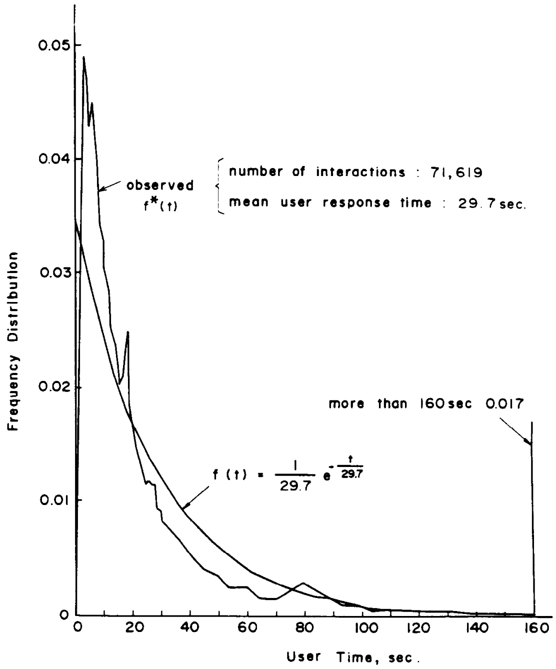

Fig. 6 Comparison of observed UT with that of assumed exponential distribution.

Comparison of the observed *UT* distribution, $f^*(t)$, with an exponential distribution, $f(t)$, is also shown in Fig. 6. The mean *UT* is 29.7 seconds.

The observed *UT* distribution shows a tendency to a biphase or triphase hyperexponential distribution rather than an exponential distribution. This result is like the distribution of interarrival times in *SDC-ARPA* TSS [10]. The ratio of observed *UT*'s which are longer than 160 seconds is 1.70%. In the exponential distribution, the cumulative distribution of *UT*'s of this range is 0.457%. Therefore, the observed ratio of long *UT*'s, it is possible to divide *UT*'s into three ranges:

(1) $0 \leqq UT < 10$ (sec). About 40% of all *UT*'s are concentrated in this range. User responses in this range consist of easy or simple replies.

(2) $10 \leqq UT < 70$ (sec). Half of all *UT*'s are contained in this range. user responses include simple data set editing or file maintenance operations.

(3) 70 (sec) $\leqq UT$. In this range, a long time is probably required for users to consider to observe computation or debugging results.

2.3 Correlation Coefficients

It is desirable to predict the next user time (*UT*) at the end of the system response time (*SRT*) for the storage

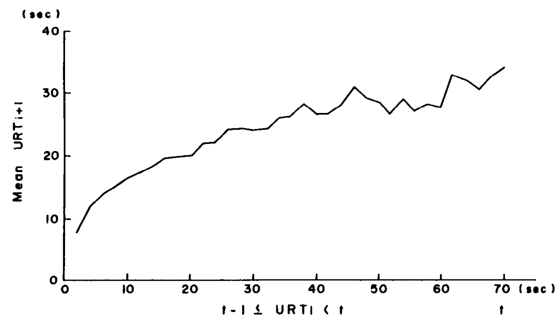

Fig. 7 Mean URT_{i+1} when URT_i falls in the interval $[t-1, t]$.

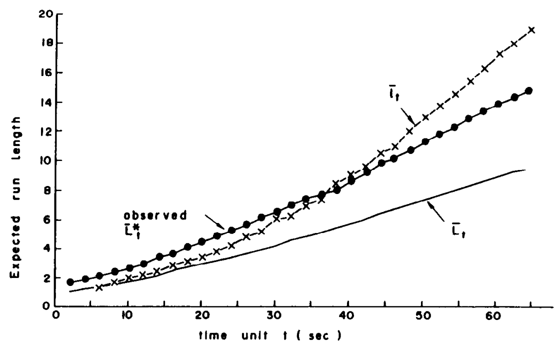

Fig. 8 Expected run length of short URT's.

management in an operating system which employs the demand swapping policy. If an accurate prediction of the next *UT* is possible, the storage management could select the *WS* for swap-out whose next *UT* is farthest in the future when storage space is unavailable for another user. If this is possible, the storage management could maintain *WS*'s whose next input may be completed in the near future. Thus, the storage management could reduce interactive swapping.

In general, the possibility of predicting the next value of variables depends on the correlation coefficient of the two successive variables. Therefore, we calculated the correlation coefficients of the variables between the user response time (*URT*) and elements of cycle time (*CT*).

Correlation coefficients of values between elements of *CT* are shown in Table 3. The values of correlation coefficients except $Cor(URT_i, URT_{i+1})$ are almost zero. Thus, they seem independent of each other. E. G. Coffman Jr. and R. C. Wood showed the serial correlation function ($Cor(UT_i, UT_{i+n})$) computed for $n=0, 1, 2, \ldots, 10$. From these computations in the *SDC-ARPA* TSS, they concluded that "*the length of any given interarrival period is statistically independent of the length of all previous periods*" [10]. Our observations also support their results. However, in our analysis, the serial coefficients of *URT*'s are larger than the others.

The observed serial correlation coefficients for three separate days are 0.189, 0.213, and 0.150. These values are not so large statistically. The observed distribution of *URT* is more skewed than the exponential distribution. Accordingly, the correlation between the two successive *URT*'s should be examined from another angle. The mean values of URT_{i+1}, when the URT_i falls in the interval $(t-1, t)$, are investigated and shown in Fig. 7. This shows that URT_{i+1} tends to be short if the previous URT_i is short.

2.4 Run of the Short-*URT*'s

The mean run length of the short *URT*'s is calculated in this section in order to investigate the dependence between successive *URTs*. Let s denote a *URT* which is less than or equal to t seconds, and let r denote a $URT>t$. Thus, the sequence of *URT*'s in a terminal session can be represented by the sequence $\{s, r\}$. For example, if $t=5$ seconds, the sequence of *URT* $\{15, 3, 5, 2, 8, \ldots\}$ is described as $\{r, s, s, s, r, \ldots\}$. The successive s's are called a *short-URT run*. Let l_t be the length of a *short-URT run*. If *URT*'s are mutually independent, l_t is geometrically distributed; that is, $P\{l_t=k\}=p(t)^{k-1}\times q(t)$, for $k\geq 1$, where $p(t)=P\{URT\leq t\}$, $q(t)=1-p(t)$. The mean $\bar{l}_t$ is: $\bar{l}_t=\sum_{k=1}^{\infty} k\times P\{l_t=k\}=1/q(t)$.

However, the number of interactions in a terminal session is finite and distributed, as in Fig. 2. When the number of interactions in a session is finite number N, the mean run length of short *URT*'s is: $E_N=N/\{1+(N-1)q(t)\}$ for $N\geq 1$. The derivation of E_N is shown in Appendix A. Now let f_N be the distribution of N, as shown in Fig. 2. The mean run length of short *URT*'s is given by: $\bar{L}_t=\sum_{N=1}^{\infty} f_N\times E_N$.

Mean lengths of *short-URT* runs ($\bar{l}_t$ and $\bar{L}_t$) which were previously defined by $p(t)$, E_N and f_N, are calculated with respect to t. Furthermore, let $\bar{L}^*_t$ be the observed value of $\bar{L}_t$. Thus, the three mean lengths of *short-URT* runs, $\bar{l}_t$, $\bar{L}_t$ and $\bar{L}^*_t$ are shown in Fig. 8. The observed mean length $\bar{L}^*_t$ of *short-URT* runs is larger than $\bar{L}_t$. This fact also supports the conclusion that *the successive URT's are not mutually independent*. For example, the mean run length of *short URT*'s which are shorter than 30 seconds, is 6.5 in the observed data, but with the assumption that *URTs* are mutually independent, $\bar{L}_t=4.2$. From this result, we propose a demand swapping algorithm, SLRU, using this characteristics in the next section.

3. Demand Swapping Policies

3.1 Objectives

With the *immediate swapping policy* of TSO [27], and the early version of MVS [28][29], excessive swapping is the system bottleneck usually encountered, especially when the TSS becomes large scale [3][6]. Therefore, two important design considerations are (1) estimation of the main storage capacity required and (2) the swapping rate due to interactive swapping in such a swapping policy.

For the first design feature we estimated the main storage capacity required for TSS in the *immediate swapping policy*. Here, the total demand of *WS*'s, M, accessing to CPU is approximately:

$$M=n\times SRT\times w/CT, \tag{1}$$

where n is the number of users and w is the average *WS*'s size. For instance, it is assumed that CT is 20 seconds, w is 800 KB (10^3 bytes of the main storage), and SRT is 2 seconds as reported by T. Bertvas in [3]. Thus, $M=80\times n$ KB in Eq. (1). With a large scale TSS, such as $n=300$, M can be estimated as 24 MB (10^6 bytes). Such main storage capacity would not be considered large in today's large scale computer systems. Moreover, CT will be longer than 20 seconds, judging from the result reported in [4] and our measurements. Consequently, the total amount of the main storage, which must be provided for the TSS user while his *WS* is swapped in (so that it can be accessible to the CPU), may be smaller than 24 MB in this situation.

For the second design feature, we estimate the swapping rate (*SPR* [3]) which is the number of transferred pages per second between main storage and auxiliary storage due to swapping. We assume that *WS* is copied out to auxiliary storage or brought into the main storage by the swapping, rather than the paging mechanism. Thus, $SPR=f\times w\times n/CT$, where f reflects the fact that both swap-in and swap-out take place. The value of f is almost 2, so, $SPR=24$ MB/sec under the above assumption. Thus, twenty three channels may be dedicated to swapping, where the channel transfer rate is 3.0 MB/sec and channel utilization is limited to 35% practically. In addition, several channels are necessary for file access and paging. Therefore, the design of a large scale TSS is probably limited by the system configuration—especially by the number of channels.

In the *immediate swapping policy*, it is not possible to use large scale storage effectively, even when it is available; many channels are needed for swapping. Today, the main storage of 64 to 128 MB in large scale computer systems is much more than required for the *immediate swapping policy*. Thus, a more favorable balance between the main storage and channel usage can be achieved using the *demand swapping policy*.

In *demand swapping policy*, *WS*'s are not swapped out immediately at the completion of a TSS command processing (called *transaction*), and can be left in the storage until it is required by another user. So, if a *WS* is still resident, the system can start working on the *transaction* without encountering a swapping delay.

Now let us define the *working set hit ratio* (*WSHR*) for the performance measurement of the demand swapping algorithm. *WSHR is defined as the number of times the desired WS was found in the main storage (a hit), divided by the total number of transactions made by TSS users. WSHR* depends on the main storage capacity, the user's input process, and the demand swapping algorithm. The demand swapping algorithm selects a swap-out candidate only when the storage is needed for another user whose *WS* is already swapped out.

3.2 Demand Swapping Algorithms

The object of the demand swapping algorithm is to gain high *WSHR*. The demand swapping algorithm can be regarded as the page replacement algorithm when the time interval between successive references to the same *WS* is substituted for the time interval between successive references to the same page in the demand paging system. Now our attention is concentrated on the demand swapping algorithm in relation to the input process. Therefore, the working set size is assumed constant in this paper.

The optimal demand swapping algorithm, which yields the maximum *WSHR* over the space of all demand swapping algorithms for every input process and every main storage capacity, has the following characteristics.

Whenever *WS* must be swapped out from the main storage, the chosen *WS* is the one whose next input is farthest in the future. This optimal demand swapping algorithm is analogous to MIN [2] or OPT [22] in the page replacement, where the input process of TSS users is substituted for the page reference string. We can regard the input process of TSS users as the reference string of *WS*'s. Unfortunately, the optimal demand swapping algorithm cannot be achieved in an actual operating system because it requires knowledge of future input processes, such as MIN or OPT. Therefore, we proposed five demand swapping algorithms. These five demand swapping algorithms are considered here, since they are of practical importance:

Least Recently Used (LRU) Algorithm

In this algorithm, the swap-out candidate of *WS* is the one with the longest resident time since the last completion of a *transaction*.

Random (RAND) Algorithm

When there are m_0 *WS*'s in the main storage, any given *WS* is chosen for swap-out with a uniform probability of $1/m_0$.

Last Used First out (LUFO) Algorithm

In contrast to LRU, this algorithm chooses a *WS* with the shortest resident time since the last completion of a *transaction*. In this algorithm, users who have been thinking for a long time are expected to complete their inputs sooner.

Prediction (PRED) Algorithm

In this algorithm, the next user time (*UT*) is predicted from past *UT*'s. From this prediction, PRED chooses a *WS* whose predicted completion input time is farthest in the future. In this algorithm, it is assumed that each user has a characteristic input process. That is, each user continuously employs roughly similar *UT*'s during his session. The exponential smoothing method [5] is used to predict the next *UT*. This prediction formula is: $\hat{X}_i = \hat{X}_{i-1} + \alpha \times (x_{i-1} - \hat{X}_{i-1})$, for $i = 1, 2, 3, \ldots$ where $\hat{X}_i$ is the predicted value of the next *UT*, x_{i-1} is a previously observed value, and $\alpha (0 < \alpha < 1)$ is the smoothing constant [31].

Shrunken Least Recently Used (SLRU) Algorithm

From the analysis of the input process, especially the run of the *short-URT*, we know that successive *URT*'s are not independent. This algorithm takes advantage of this fact. That is, at the end of *SRT*, *WS* is swapped out if the previous *URT* is greater than t_0. Otherwise, the *WS* is kept in the main storage. In the virtual storage operating system, adequate amount of free storage is required because of preventing paging and swapping delay [32]. SLRU can achieve the replenishment of free storage and demand the swapping policy at the same time. The selection method of *WS* for swap-out—when the storage becomes insufficient for another user—is the same as for LRU. This t_0 is called *critical time*. If we make t_0 infinite, *WS* is always kept resident at the end of *SRT*. Therefore, SLRU is the same as LRU when t_0 is infinite. On the other hand, SLRU is the *immediate swapping policy* when t_0 is zero. This critical time, t_0, is an important parameter in this SLRU algorithm.

Fixed-space demand swapping algorithms (LRU, RAND, LUFO, and PRED) can be compared in terms of their *WSHR* at equal storage sizes. But SLRU may be *a variable-space demand swapping algorithm*; depending on t_0, some *WS*'s are swapped out at the end of *SRT*. The main storage capacity for the demand swapping policy varies dynamically with time and is less than or equal to the total capacity of the main storage. From this reason, the *WSHR* of SLRU could be analyzed differently from the others.

4. Analysis of Demand Swapping Algorithms

The actual behavior of TSS users at a terminal is obtained as time series data in order to analyze the relationship with the five proposed demand swapping algorithms. For this analysis, *a trace-driven type* [7][19] simulation model has been developed. The input information is the trace data. The real time behavior of the original workload with a varying level of multiprogramming can be reproduced by this simulator.

4.1 Simulation Model

4.1.1 TSS User Model

This simulation model bases on the input process during a terminal session. Four terminal states are considered by this simulator. Three of those four states corresponds to system response time (*SRT*), user response time (*URT*) and output time (*OUT*). The fourth corresponds to a *logged-off* state which enters by a *logoff* command and left by a logon command. A state of *SRT* is entered from a state of *URT* at a command input and left in a state of *URT* or *OUT*. A state of *URT* is entered from a state of *SRT* directly or from a state of *OUT* at the end of the output request. This simulator makes a terminal state transition when an event is found in the trace record which is either (1) a *logon/logoff* command, (2) a system prompting for the user to enter the next command (3) a user depressing the carriage return, or (4) output messages to a terminal.

4.1.2 Swapping model

Fig. 9 shows the structure of the trace-driven simulator for the demand swapping analysis. This simulator maintains m_0^* *WS*'s in the main storage. That is, the main storage capacity, *M*, is equal to $m_0^* \times w$. Where *w* is the average *WS* size. There are three kinds of queues: (1) active queue, (2) survival queue, and (3) swap-out queue.

The queue element represents the process state containing a terminal state and statistical information related to the TSS process. The *active queue* contains elements that represent terminals during SRT. *WS*'s related to the *active queue* element are used for accessing to the CPU. The element of the *survival queue* represents the terminal whose *WS* is kept in the main storage during user time (*URT* or *OUT*). *Swap-out queue* elements represent the terminal whose *WS*'s have been already swapped out. the total number of active and survival queue elements is less than or equal to m_0.

When the simulator gets a record of a user typing the carriage return, the corresponding queue element is searched in the survival queue. If the corresponding queue element is not found in the survival queue, and the total number of elements in the active and survival queues is equal to m_0, a queue element corresponding to a swap-out candidate is removed from the survival queue and placed in the swap-out queue. This swapping decision is made by a *demand swapping algorithm* which is specified by a parameter of this simulator. This simulator neglects the swapping time, and the corresponding element is placed in the active queue.

This queue element is created at *logon* command, transferred between the three queues, and destroyed at *logoff* command. At the end of *SRT*, the corresponding element of the active queue is removed and placed at the end of the survival queue, except for in PRED and SLRU. In PRED, the simulator calculates the next predicted user time (*UT*) using previously observed *UT* values and the smoothing constant. It then merges the queue element into the survival queue based on the predicted *UT* value. In SLRU, if the previous *URT* is greater than the critical time t_0, the queue element is removed from the active queue and placed in the swap-out queue. Otherwise, the queue element is placed in the end of the survival queue. The critical time t_0 is given as a parameter to this simulator.

Accordingly, a swap-out candidate is the first element

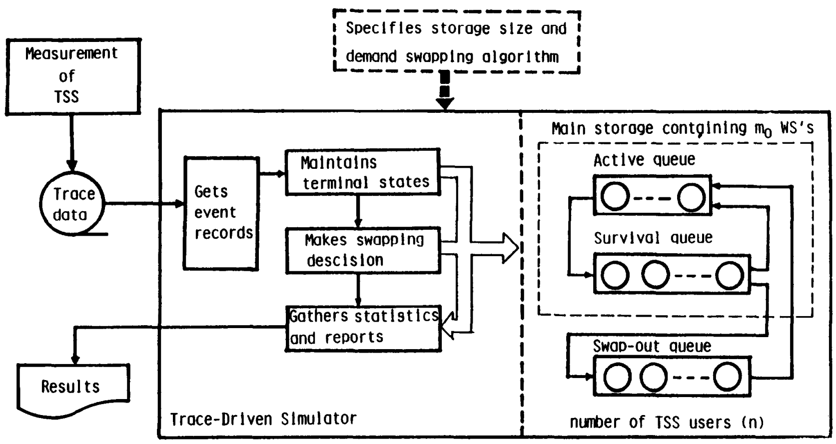

Fig. 9 Demand swapping analysis with trace-driven simulator.

of the survival queue in LRU, PRED, and SLRU. On the other hand, the last element of the survival queue is a swap-out candidate in LUFO. In RAND, the simulator selects the swap-out candidate using the pseudo random number, which is an integer uniformly distributed between one and the number of the survival queue element.

4.2 Fixed-Space Demand Swapping Algorithms

4.2.1 Simulation Results of WSHR

WSHR's of the *fixed-space demand swapping algorithms* are plotted in Fig. 10. The number of *WS*'s in the main storage, m_0^*, is given by a parameter to this simulator. Let a be the average number of *WS*'s whose process are running and in *SRT* state. Thus a equal to $SRT \times n^*/CT$. Where n^* is the number of logged on users. Let n be the average number of *WS*'s during *UT*, so $n=n^*-a$. Thus, the average number of *WS*'s in the main storage waiting for the completion of *UT*, m_0, is given by m_0^*-a. As mentioned in 3.2, we are concerned with the *demand swapping algorithm*. Therefore, we assume the working set size is constant, although *WS* varies in the real world [18][21][26]. The smoothing constant in PRED is assumed to be 0.5.

From simulation results based on the trace data, the *WSHR*'s of four demand swapping algorithms from the highest to the lowest are LRU, RAND, PRED, and LUFO. The differences of *WSHR* between LRU and RAND are slight. However, the differences in WSHR between RAND and PRED, and between PRED and LUFO, are relatively large. For instance, if the mean number of *WS*'s in the main storage is 15, the *WSHR*'s of LRU, RAND, PRED, and LUFO are 56%, 54%, 43% and 33%, respectively. Therefore, there are differences in *WSHR* among *demand swapping algorithms*.

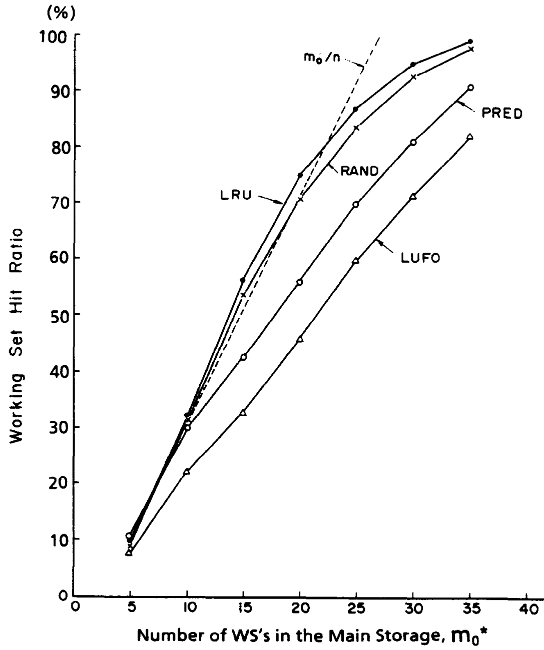

Fig. 10 Simulation results of WSHR.

4.2.2 Main Characteristics of each Demand Swapping Algorithm

The relation between the user input process and demand swapping algorithm is investigated in this section. Two distribution functions, p_i^* and q_i^*, are introduced from the user input process and *WSHR*'s of LRU and RAND are represented by them. The other *WSHR*'s are explained by the results of the analysis of input process in section 2. It is assumed that the main storage capacity, M, contains m_0 *WS*'s; that is, $m_0=M/w$ where w is

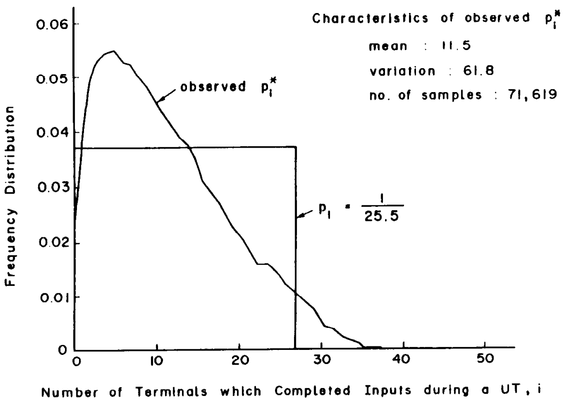

Fig. 11 Distribution of number of terminals which completed inputs during a UT.

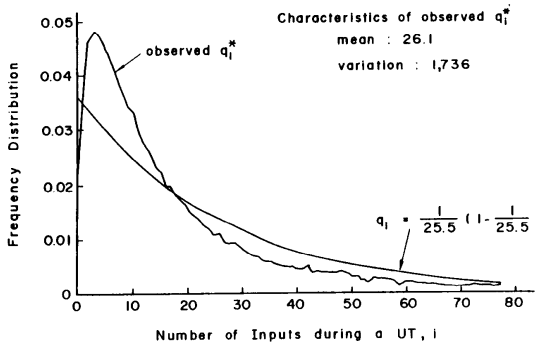

Fig. 12 Distribution of number of inputs during a UT.

the average *WS* size.

LRU

In this algorithm, the condition under which *userA*'s *WS* is still resident at completing his next input is that, at most, m_0-1 distinct users complete their inputs during *A*'s *UT*. The distribution, p_i^*, in which *i* distinct users complete at least one input each within a *UT* is a main factor in LRU. The distribution, p_i^*, is obtained from the trace data and shown in Fig. 11. The mean number of users who complete their input is 11.5. From this distribution, and *WSHR* is LRU is obtained: $WSHR_{LRU}=\sum_{i=0}^{m_0-1} p_i^*$.

RAND

The resident *WS* is selected for swap-out with a uniform probability of $1/m_0$ when the storage is needed for another user whose *WS* has been already swapped out. The number of opportunities of this selction during a *UT* is an important factor in RAND. Thus, the distribution, q_i^*, in which *i* inputs (some possibly from the same user) are made during a *UT*, is investigated and shown in Fig. 12. The average number of inputs is

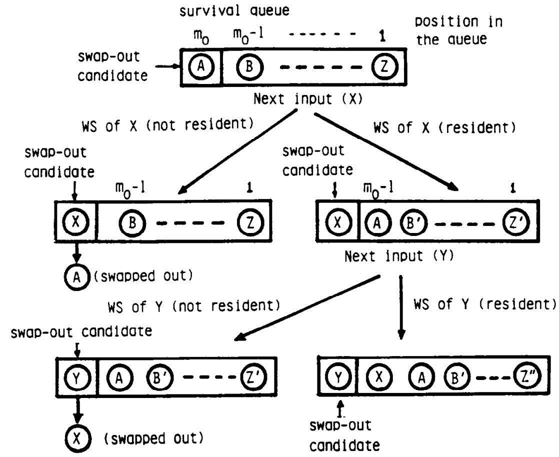

Fig. 13 Schematic model of the transfer of the survival queue in LUFO algorithm.

26.1.

Now consider the situation that a *user X* completes his input during *user A*'s *UT*. Let $P(n, m_0)$ be the probability for which *user A*'s *WS* is kept resident after *X*'s input. Thus, the *WSHR* of RAND is:

$$WSHR_{RAND}=\sum_{i=0}^{\infty} q_i^* \times P(n, m_0)^i. \tag{2}$$

The probability $P(n, m_0)$ is given in *Appendix C.*

LUFO

Let queue elements be merged (in order of priority, longer *UT*'s first) in a survival queue. This schematic model is shown in Fig. 13, where m_0 *WS*'s are still retained in the main storage and users are in *UT*. In Fig. 13, the *user Z* spends the longest *UT* time and the *user A* the shortest. *WS* of *A* is now the swap-out candidate. That is, the swap-out candidate is the last element in the survival queue in LUFO. In this situation, we assume that the *user X* completes his input. If *X*'s queue element is not in the survival queue, *A*'s *WS* is selected for swap-out.

However, if *X*'s queue element is in the survival queue, *A*'s *WS* is kept in the main storage. *X*'s queue element is placed at the end of the survival queue, assuming that the system response time is negligible. *X*'s *WS* becomes a swap-out candidate next, and *A*'s queue element does not go to the end of the survival queue until *A*'s next input comes, even if *A*'s *UT* is long. This means that *A*'s *WS* is never swapped out during his *UT*. To verify this fact, we assume that a *user Y* completes his next *UT*. The same operation is performed in the survival queue at the *X*'s input. As the results, *A*'s element stays in the (m_0-1)th position or is transferred to the (m_0-2)th position of the survival queue. Even if *Y* is *X*, *A*'s element stays in the (m_0-1)th position of the survival queue.

Table 3 Correlation coefficients.

	Case 1	Case 2	Case 3
Cor (SRT_i, URT_{i+1})	0.0129	0.00529	−0.00531
Cor (OUT_i, URT_{i+1})	0.0835	0.00336	0.00432
Cor (SRT_i+OUT_i, URT_{i+1})	0.0840	0.0324	0.0327
Cor (UT_i^*, UT_{i+1})	0.0478	0.0522	0.0849
Cor (SRT_i, UT_{i+1})	0.0123	0.0295	0.0269
Cor (URT_i, URT_{i+1})	0.189	0.213	0.150

*$UT_i=OUT_{i-1}+URT_i$

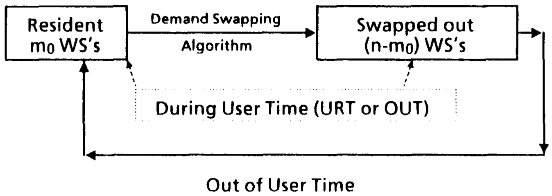

Fig. 14 Transaction flow in the stochastic model.

In the above consideration, *UT* is an important factor, especially for long periods in LUFO when *WS*'s are resident at the completion of input. If the ratio of long *UT*'s is large, *WS*'s waste the main storage. Thus, the long *UT* time frequency of the *UT* distribution is especially important. If the ratio of long *UT* is large, the main storage will be wasted by those *WS*'s. The distribution of *UT*, as shown in Fig. 6, is more skewed than exponential distribution with the same mean value. The long period portion is greater than the exponential one, as previously mentioned. Consequently, there may be more *WS*'s maintained in the main storage in LUFO for longer periods than in the other demand swapping algorithms.

PRED

The major feature of PRED is prediction accuracy. For example, when a *user A* has had several short *UT*'s in succession, PRED will predict that the *A*'s next *UT* will also be short. So, *A*'s *WS* will be kept in the main storage even if *A*'s next *UT* is long. Therefore, the main storage space will be wasted. On the other hand, when a *user A* has had several long *UT*'s PRED will predict that *A*'s next *UT* will also be long. So, *A*'s *WS* will be swapped out when storage space for another *user*'s *WS* becomes necessary to swap in even if *A*'s next *UT* is short. Consequently, PRED is not effective when *UT*'s are variable.

Therefore, the serial correlation coefficient, *Cor* (UT_i, UT_{i+1}), is important in PRED. The observed values, as shown in Table 3, are close to zero. We can see that predicting the next *UT* from the past *UT*'s yields undependable results.

4.2.3 Comparison of Simulation and Stochastic Model Results

Four demand swapping algorithms use the past history of the time interval between successive TSS commands for each individual user. A stochastic model, in which the time interval between successive TSS commands is independent and exponentially distributed, is introduced in order to make clear how *WSHR*'s depend on the input process. As pointed out in the previous section, the main factors of each *WSHR* are presented by distributions, p_i^*, q_i^*, $f^*(t)$, and the serial correlation coefficient, $Cor(UT_i, UT_{i+1})$. Using this model, we will examine these distributions. We will analyze the relationship between *WSHR* and the input process in LRU and RAND which show high *WSHR* in the trace-driven simulator.

The input process of TSS users can be regarded as a stochastic process. Therefore, it is assumed that *UT*'s are independent and exponentially distributed with mean $1/\lambda$. The probability density is: $f(t)=\lambda\, exp(-\lambda t)$, $(t\geqq 0)$. This assumed input process is called a *stochastic model*. In this section, the *WSHR*'s with *trace-driven simulation* are compared with those of the assumed input process.

In this stochastic model, a TSS transaction enters the system via the swapping state of a queueing network model. Let n be the average number of users in *UT* as defined in 4.2.1. The transaction flow can be shown as Fig. 14, where m_0 is the average number of *WS* residing in the main storage. So, $(n-m_0)$ *WS*'s are swapped out. From the assumption that *UT*'s are exponential, in any Δt interval time, the probability that every user stops his *UT* is equal to $\lambda\Delta t$ and that all users are independent. In addition, *WSHR* is not affected by any demand swapping algorithms. Therefore, the probability that *WS* is still resident at the stopping *UT* is proportional to the number of *WS*'s in the main storage. That is, *WSHR* is equal to m_0/n.

The *WSHR* of the stochastic model, or, m_0/n, is also shown in Fig. 10 where n is an average number of TSS users based on the trace-data. The *WSHR*'s of LRU and RAND closely approximate m_0/n. However, the *WSHR*'s of PRED and LUFO differ from m_0/n. Therefore, the differences in *WSHR* between m_0/n and the simulation results are examined more precisely in the following:

LRU

As previously mentioned, the distribution, in which i distinct users complete at least one input each within a *UT*, is an important factor. Let p_i be this probability in the assumed input process. Thus, p_i is: $p_i=1/n$, for $i=0, 1, 2, 3, \ldots, n-1$ (proved in Appendix B), which is a uniform distribution. Comparison of the observed distribution, p_i^*, with a probability p_i of a uniform distribution is shown in Fig. 11. In p_i^*, the number of users varies as shown in Fig. 3, and the maximum number of users which complete input during a *UT* is 37.

The difference in *WSHR* between the simulation and the stochastic model can be formulated using p_i and p_i^*. That is:

$$\Delta WSHR_{\text{LRU}} = \sum_{i=0}^{m_0^-1} (p_i^* - p_i) \tag{3}$$

The observed p_i^* is larger than p_i when $i \leqq 15$. On the other hand, observed p_i^* is smaller than p_i in the $16 \leqq i \leqq 26$ range. From *Eq.* (3) and Fig. 11, $\Delta WSHR_{\text{LRU}}$ is positive in the $0 < m_0 \leqq 22$ range. This explains why the *WSHR*'s in the simulation are greater than m_0/n in the stochastic model. These results are shown in Fig. 10.

RAND

In the stochastic model, n users are always in *UT* if *SRT* is negligible. In any Δt interval, the probability that a user will complete a *UT* is equal to $\lambda\Delta t$ for every user. So, the probability that *user A* completes his *UT* as the next input to the system is $1/n$. The probability that the next input to the system will be made by others is $(1-1/n)$. Accordingly, at every input to the system, *Bernoulli trials* [14] are made with the probability $1/n$. Thus, the probability q_i that i inputs will be made during a *UT* is a geometric distribution. That is: $q_i = 1/n(1-1/n)^i$, for $i = 0, 1, 2, \ldots\ldots,$.

The observed q_i^* is compared with a probability q_i of a geometric distribution in Fig. 12. The difference in the mean number of inputs between q_i^* and q_i during a *UT* is small. The observed q_i^* is larger than q_i when $i \leqq 18$. However q_i^* is smaller than q_i in the $i \geqq 19$ range.

From *Eq.* (2), the difference in *WSHR* between the simulation and stochastic model is: $\Delta WSHR_{\text{RAND}} = \sum_{i=0}^{\infty} (q_i^* - q_i) \times P(n, m_0)^i$. The difference, $\Delta WSHR_{\text{RAND}}(j) = \sum_{i=0}^{j} (q_i^* - q_i) \times P(n, m_0)^i$, is obviously positive when $j \leqq 18$, because $q_i^* \geqq q_i$ in the $i \leqq j \leqq 18$ range and $P(n, m_0) \leqq 1$. The probability $P(n, m_0)$ is provided in Appendix C. Accordingly, the *WSHR* of RAND, which is evaluated in the simulation, is higher than in the stochastic model. This reasoning explains the behavior shown in Fig. 10.

4.3 Variable-Space Demand Swapping Algorithm

4.3.1 WSHR vs. Critical Time

From the analysis of the observed user input process, successive *URT*'s are not mutually independent. In SLRU, *WS*'s are swapped out at the end of *SRT* when the previous *URT* is larger than t_0. This simulation model assumes that *WS* is invalid immediately upon being swapped out even if it can be reclaimed. That is, freed spaces by swapped out are wasted and unused for demand swapping policy. From this assumption, we can compare the difference in *WSHR* between SLRU and LRU and know the average storage capacity by using SLRU. This t_0 is an important parameter in SLRU. The sensitivity of t_0 in SLRU is investigated. The relationship between *WSHR* and t_0 is shown in Fig. 15. The *WSHR* of LRU are also plotted in Fig. 15, because LRU is a case of SLRU when t_0 is infinite.

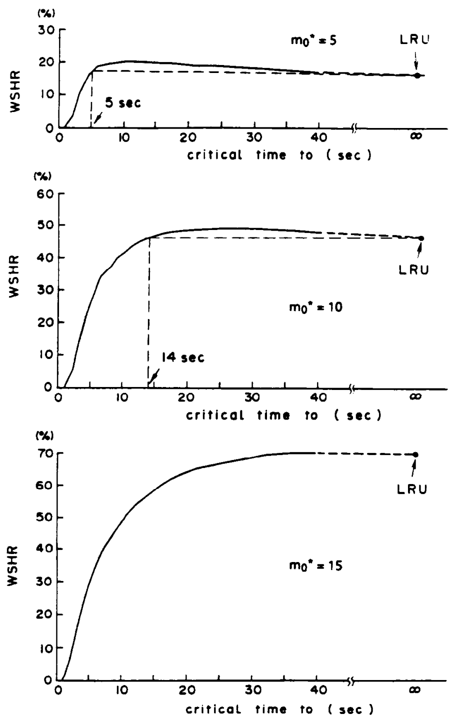

Fig. 15 *WSHR* vs. critical time in SLRU.

The *WSHR* of SLRU, where $m_0^* = 5$, $t_0 \geqq 5$ and where $m_0^* = 10$, $t_0 \geqq 14$, is greater than that of LRU. However, the *WSHR* of SLRU increases with respect to t_0 when $m_0^* = 15$. Thus, a higher *WSHR* can be achieved in SLRU than in LRU if a suitable *critical time* t_0 is chosen. These results show that SLRU takes advantage of the correlation between the two successive *URT*'s.

4.3.2 Comparison of SLRU with LRU

From the simulation study, LRU is the best algorithm among the *fixed-space demand swapping algorithms*. However, SLRU shows higher *WSHR* than LRU in a certain t_0 range. The characteristics of SLRU are examined more precisely in this section.

SLRU is a kind of *variable space demand swapping algorithm*. This means the storage area using demand swapping varies. To compare SLRU with LRU, the average storage size allocated for demand swapping policy in SLRU is shown in Fig. 16. Fig. 16 shows the amount of main storage used for demand swapping policy as a function of critical time t_0. "Used for accessing CPU" in Fig. 16 means that the storage area is used by active processes. This area size is almost $a \times w$ where a is the average number of *WS*'s during *SRT* and w is the average working set size. In SLRU, the storage area is freed when *WS*'s are swapped out. Thus "Unused" in Fig. 16 means freed area. "Using for DS" means that this storage area is used for demand swapping policy

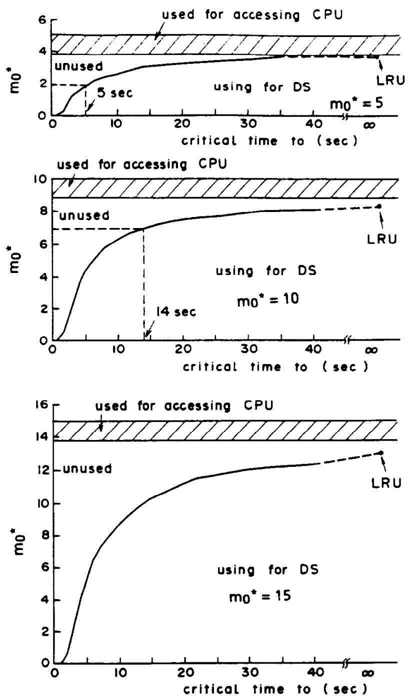

Fig. 16 The allocation of memory use vs. critical time in SLRU.

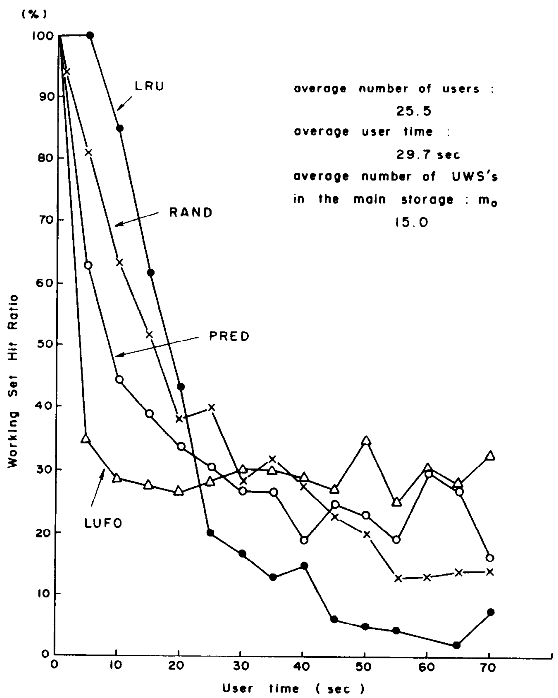

Fig. 17 User time vs. working set hit ratio.

and contains *WS*'s during *UT*.

Obviously, the average size increases with respect to the critical time t_0. When $m_0^*=5$ and $t_0 \geqq 5$, the *WSHR* of SLRU is greater than that of LRU, as shown in Fig. 15. The average storage size, m_0, for demand swapping policy is 2.0 in SLRU where $t_0=5$ seconds, but in LRU, an m_0 of 3.8 is required, as shown in Fig. 16. If $m_0^*=10$, $WSHR_{\mathrm{SLRU}} \geqq WSHR_{\mathrm{LRU}}$ where $t_0 \geqq 14$, as shown in Fig. 15. In order to achieve the same *WSHR*, an m_0 of 7.1 is required in SLRU while an m_0 of 8.4 is required in LRU. Consequently, SLRU can achieve higher *WSHR* with smaller storage capacity than LRU.

4.4 Consideration of Demand Swapping Algorithms

WSHR of each demand swapping algorithm has been analyzed. This *WSHR* is the most important performance measurement in the demand swapping policy, because interactive swaps can be minimized. However, from the user's point of view, another criterion must be examined for the evaluation of demand swapping algorithms.

Our observations show that trivial TSS transactions, such as simple data set editing operations, occupy more than 80% of the total interactions. The consumed system resources for swapping are greater than those of a trivial transaction. From the analysis of the cycle time, the *UT*'s for those trivial transactions are relatively short. That is, the system should be substantially more responsive to trivial transactions than to transactions which take several minutes of CPU time [13]. From this point of view, it is advisable that the *WSHR* for a short *UT* be higher than for a relatively long *UT*. The relationship between *UT* and *WSHR* is the second important criterion in the demand swapping policy, as shown in Fig. 17.

WSHR of LRU is highest where *UT* is less than 21 seconds (Fig. 17). However, *WSHR* of LRU is lowest where *UT* is greater than 24 seconds. In contrast to LRU, the *WSHR* of LUFO is almost constant at 30% and is not related to *UT*. In RAND and PRED, *WSHR* slowly decreases with respect to *UT*. From this we see that LRU is the most effective in terms of this important factor. That is, when a user completes his input quickly, he can expect to shorten the swapping delay with LRU.

LRU and SLRU adopt the same approach of selecting *WS* for swap-out when the main storage becomes insufficient. As we have shown, they are both useful demand swapping algorithms.

5. Conclusions

Five demand swapping algorithms were evaluated using the trace-driven simulator. The main factors of each *WSHR* in the demand swapping algorithm depend on the past history of the time interval between successive TSS commands, (*UT*).

We conclude that LRU, or SLRU, is the best demand swapping algorithm. These two algorithms have the ability to automatically swap out working sets in the main storage for long *UT*. This ability is most important in effectively utilizing the main storage for the demand swapping policy. Therefore, LUFO showed the worst performance in *WSHR*, because it lacks this ability. PRED also will lose this ability unless the next *UT* is accurately predicted. However, PRED does not lose this ability completely even when the prediction is inaccurate.

LRU, or SLRU, takes advantage of the correlation between the two successive *URT*'s. This ability is the second important function. The *URT*'s which were analyzed showed that successive *URT*'s are mutually dependent. Therefore, LRU showed higher *WSHR* than RAND.

SLRU is an improved version of LRU. The critical time t_0 in LRU is the parameter in setting higher *WSHR*. From the simulation study, SLRU showed higher *WSHR* than LRU in a given critical time range.

Acknowledgments

This research has been conducted as part of a project concerning a large scale time-sharing system at the University of Tokyo. Many persons from the staff of the Kanagawa Works, Software Works, and Systems Development Laboratory (SDL) of Hitachi, Ltd., took part. Authors would like to thank Dr. Takeo Miura, (former General Manager of SDL), and Mr. Y. Hattori (of the Software Works) for their encouragement in this research. We are deeply grateful to Prof. H. Ishida of the University of Tokyo who gave a willing consent to accumulate trace data of TSS user process and reviewed this paper. The authors also gratefully acknowledge the help of Dr. K. Noguchi of the Software Works for suggesting this area of research and for his constructive comments, and the help of Mr. A. Harada and his colleagues of the Software Works for their work in developing the demand swapping policy in VOS3. Thanks are also due to Prof. H. Morimura of the Tokyo Institute of Technology for his theoretical discussions, Prof. T. Masuda of University of Tokyo for his constant encouragement, and Dr. S. Mitsumori and Mr. T. Kubo of SDL for their penetrating comments as well as for reviewing this paper.

References

1. Belady, L. A. A study of replacement algorithms for a virtual-storage computer, *IBM Syst. J.*, **5**, 2 (1966), 78-101.
2. Belady, L. A. and Palermo, F. P. On-line measurement of paging behavior by the multivalued MIN algorithm, *IBM J. Res. Dev.*, **18** (January 1974), 2-19.
3. Beretvas, T. Performance tuning in OS/VS2 MVS, *IBM Syst. J.*, **17**, 3 (1978), 290-313.
4. Boies, S. J. User behavior on an interactive computer system, *IBM Syst. J.*, **13**, 1 (1974), 2-18.
5. Brown, R. G. Smoothing, Forcasting and Prediction of Discrete Time Series, *Prentice-Hall, Inc.*, 97-105.
6. Buzen, J. P. A queueing network model of MVS, *ACM Computing Survey*, **10**, 3 (September 1978), 320-331.
7. Cheng, P. S. Trace-driven system modeling, *IBM Syst. J.*, **8**, 4 (1969), 280-289.
8. Chow, We-Min and Chu W. W. An analysis of swapping policies in virtual storage systems, *IEEE Trans. Softw. Eng.*, SE-3, 2 (March 1977), 150-156.
9. Chu, W. W. and Opderbeck, H. The page fault frequency replacement algorithm, *1972 FJCC, AFIPS Conf. Proc.*, **41**, *AFIPS Press, Montvale, N.J.* (1972), 597-609.
10. Coffman, E. G. Jr. and Wood, R. C. Interarrival statistics for time sharing systems, *Comm. ACM*, **9**, 7 (July 1966), 500-503.
11. Corbato's, F. J. and Vyssotsky, V. A. Introduction and overview of the multics system, *1965 FJCC Conf., AFIPS Press Conf. Proc., 27, Spartan Book, Washington* (1965), 185-196.
12. Denning, P. J. The working set model for program behavior, *Comm. ACM*, **11**, 5 (May 1968), 323-333.
13. Doherty, W. J. Scheduling TSS/360 for responsiveness, *1970 FJCC, AFIPS Conf. Proc.*, **37**, *AFIPS Press, Montvale, N.J.* (1970), 97-111.
14. Feller, W. An Introduction of Probability Theory and its Applications. **1**, *John Willy & Sons, Inc., New York* (1957) Ch. 6.
15. Gelenbe, E. A unified approach to the evaluation of replacement algorithm, *IEEE Trans. Comput.*, C-**22**, 6 (June 1973).
16. Ishida, H. A 4-CPU multiprocessor system of the University of Tokyo, *Journal of the Information Processing of Japan*, **15**, 7 (July 1974), 534-541.
17. Ishida, H., Nomoto, S. and Ozawa, H. Graphic monitoring of the performance of a large 4-CPU multiprocessor system, *Proc. of the second USA-Japan Computer Conf.* (1975), 271-275.
18. Jeffrey, R. S. and Denning, P. J. Experiments with program locality, *1972 FJCC, AFIPS Conf. Proc.*, **41**, *AFIPS Press, Montvale, N.J.* (1972), 611-621.
19. Kobayashi, H. Modeling and Analysis: An Introduction to System Performance Evaluation Methodology, *Adison-Wesley Publishing Company Inc.* (1978), Ch. 4.
20. Lynch, H. W. and Page, J. B. The OS/VS2 release 2 system resources manager, *IBM Syst. J.*, **13**, 4 (1974), 274-291.
21. Madison, A. W. and Batson, A. P. Characteristics of program localities, *Comm. ACM*, **19**, 5 (May 1976), 285-294.
22. Mattson, R. L., Gecsei, J., Slutz, D. R. and Traiger, I. L. Evaluation techniques for storage hierarchies, *IBM Syst. J.*, **9**, 2 (1970), 78-117.
23. Nakazawa, K., Murata, K., Ishihara, K., Iwakami, H., Horikoshi, H., Nishino, H. and Noda, K. The development of the high speed national project computer system, *Proc. of the first USA-JAPAN Computer Conf.* (October 1972), 173-181.
24. Noguchi, K., Ohnishi, I. and Morita, H. Design considerations for a heterogeneous tightly coupled multiprocessor system, *1975 NCC, AFIPS Conf. Proc.*, **44**, *AFIPS Press, Montvale, N.J.* (1975) 561-565.
25. Ohnishi, I., Totune, S. and Ishida, H. Command language in OS7, *Proc. of IFIP Working Conf. of Command Languages* (July 1974).
26. Rodriguez-Rosell, J. Empirical working set behavior, *Comm. ACM*, **16**, 9 (September 1973), 556-560.
27. Scherr, A. L. and Larkin, D. C. Time-sharing for OS, *1970 FJCC, AFIPS Conf. Proc.*, **37**, *AFIPS Press, Montvale, N.J.* (1970), 113-117.
28. Scherr, A. L. Functional structure of IBM virtual storage operating systems part 2: OS/VS2-2 concept and philosophies, *IBM Syst. J.*, **12**, 4 (1973), 382-400.
29. Scherr, A. L. The design of IBM OS/VS2 release 2, *NCC, AFIPS Conf. Proc.*, **42**, *AFIPS Press, Montvale, N.J.* (1973), 387-394.
30. Smith, A. J. A modified working set paging algorithm, *IEEE Trans. Comput.* C-**25**, 9 (September 1976), 907-914.
31. Yoshizawa, Y., Kinoshita, T. and Arai, T., Analysis of Demand Swapping Policies in Large Scale Time-Sharing Systems, *Journal of the Information Processing of Japan*, **21**, 4 (July 1980), 314-324.
32. Yoshizawa, Y., Arai, T., Kubo, T. and Shinozaki, T. Adaptive Control for Page Frame Supply in Large Scale Computer Systems, *Proc. of the 1988 ACM SIGMETRICS*, **16**, 1 (May 1988), 235-243.

(Received February 2, 1988; revised August 17, 1988)

Appendixes

A

It is assumed that the number of interactions in a session is N (a finite number) and each *URT* is stochastically independent. Under these assumptions, the sequence of *URT*'s described in $\{s, r\}$ form, for example $B_N=\{r, s, s, r, \ldots, r, s\}$ is regarded as a sequence of *N Bernoulli trials.*

Let $B(N, k)$ denote a sequence of *N Bernoulli trials* which includes k r's and the following symbols are introduced.

L_t : a random variable of the run length of *short URT*'s
$S(i)$: a just *i times* successive run of *short URT*'s
$F(N, k, i)$: the number of $S(i)$ in a $B(N, k)$

Then it is clear that the probability of $B(N, k)$ appearing is $p(t)^{N-k}\times q(t)^k$, where $p(t)=P\{URT\leqq t\}$ and $q(t)=1-p(t)$. So, $P\{L_t=i\}$ is proportional to $\Sigma_{k=0}^{N-i} F(N, k, i)\times p(t)^{N-k}\times q(t)^k$.

The following boundary conditions are obvious.

(1) $F(N, k, i)=0 \quad (i<0, N-k<i)$

(2) $F(N, 0, i)=\begin{cases}0 & (i<N)\\ 1 & (i=N)\end{cases}$

(3) $F(N, N, i)=\begin{cases}N+1 & (i=0)\\ 0 & (0<i)\end{cases}$

Proposition 1. For $i+k\leqq N$ and $k=1, 2, 3, \ldots, N$,

$$F(N, k, i)=C(n-i-1, k-1)+\sum_{j=i+k}^{N} F(j-1, k-1, i)$$

where $C(n, r)$ represents the number of r-conbinations of n distinct objects.

(proof)

When the last r in $B(N, k)$ is in the j-th position, the number of $S(i)$ before the last r is $F(j-1, k-1, i)$ $(j=i+k, \ldots, N)$. On the other hand, $S(i)$ after the last r appears only when the last r is in the $(N-i)$-th position in $B(N, k)$. The number of these cases is $C(N-i-1, k-1)$. $F(N, k, i)$ is the summation of these terms.

Proposition 2. For $i=0, 1, 2, \ldots, N-1$. and $k=1, 2, \ldots, N-i$,

$F(N, k, i)=(k+1)\times C(N-1, k-1)$

(proof)

According to *Proposition 1*, the above formula can be proved by mathematical induction with N or k.

$P\{e_t=i\}$ is proportional to $G(N, i)=\Sigma_{k=0}^{N-i} F(N, k, i)\times p(t)^{N-k}\times q(t)^k$ $(i=1, 2, \ldots, N)$ and normalized, i.e. $\Sigma_{i=1}^{N} P\{L_t=i\}=1$, so, $P\{L_t=i\}=G(N, i)/\Sigma_{i=1}^{N} G(N, i)$. From the result of *Proposition 2*, $G(N, i)$ and $\Sigma_{i=1}^{N} G(N, i)$ can be calculated as follows.

Proposition 3.

$$G(N, i)=\begin{cases}(N-i-1)p(t)^i q(t)^2+2\times p(t)^i q(t)\\ \quad (i=1, 2, \ldots, N-1)\\ p(t)^N \quad (i=N)\end{cases}$$

$$\sum_{i=1}^{N} G(N, i)=p(t)+(N-1)\times p(t)\times q(t)$$

Therefore,

$$P\{L_t=i\}=\begin{cases}\dfrac{(N-i-1)p(t)^{i-1}q(t)^2+2p(t)^{i-1}q(t)}{1+(N-1)q(t)}\\ \quad (i=1, 2, \ldots, N-1)\\ \dfrac{p(t)^{N-1}}{1+(N-1)q(t)}\\ \quad (i=N)\end{cases}$$

Finally E_N which is the expectation of L_t can be represented as:

$$E_N=\sum_{i=1}^{N} i\times P\{L_t=i\}=N/\{1+(N-1)q(t)\}.$$

B

The probability, p_i, that i distinct users complete at least one input each within a *UT* is evaluated. Let t be the *UT* of a user A. The probability that user X (not A) stops his *UT* during t is: $p(t)=1-\exp(1-\lambda t)$. Thus, the probability, $p_i(t)$, that i out of $n-1$ users stop their *UT* within t, is given by: $p_i(t)=C(n-1, i)\times P(t)^i q(t)^{n-i-1}$, where $q(t)=1-p(t)$, which is a binomial distribution. From the assumption of *UT*, probability p_i is: $p_i=\int_0^\infty P_i(t)\times f(t)dt=1/n$, which is a uniform distribution.

C

Let us show the probability, $P(n, m_0)$, for which user A's *WS* is kept resident after X's input. There are two situations in which user A's *WS* remains in the main storage after X's input. One is when user X's *WS* is still resident. This probability is $(m_0-1)/(n-1)$. In the other situation, user X's *WS* are not resident with probability $(n-m_0)/(n-1)$. In this case, the probability that user A's *WS* is still resident is $(n-m_0)(1-1/m_0)/(n-1)$. Consequently, $P(n, m_0)=(m_0-1)/(n-1)+(n-m_0)(1-1/m_0)/(n-1)=n(1-1/m_0)/(n-1)$.