Roll in/out and Usage of Large Capacity Core Memory in a Time-sharing System

By

Hiroshi HAGIWARA* and Hajime KITAGAWA**

(Received March 31, 1970)

In this paper the analysis of roll in/out operation and the usage of large capacity core memory (LCM) in a time-sharing system (TSS) are described. A TSS model based on a few assumptions is proposed, and the relation between roll in/out and system performance is analyzed. The system is classified into two types, i.e. swapping limited and CPU limited, according to whether the maximum system performance is restricted by capability of swapping devices or central processing units, and using LCM in TSS is very effective for the swapping limited case because of decrease of system overhead due to roll in/out operation and increase of system performance. The usage of LCM is divided into two main classes, i.e. as roll-out area and roll-in area, and a system model with LCM is analyzed and discussed in each class. One of the typical cases of using LCM as roll-in area is the implementation of conversational language in interpretive mode, then, the condition on which the interpreter program using LCM is operated in swapping limited state in spite of decrease of swapping overhead and increase of CPU time is obtained, and a numerical example is showed. Finally, an optimum algorithm for the use of LCM, which betters system performance in man-computer communication, is suggested.

1. Introduction

In recent years considerable attempts have been made to implement a general-purpose time-sharing system by large-scale, high-speed computer. In such a time-sharing system (TSS), the system performance which includes not only efficiency of central processing unit (CPU) but also response interval for request from a remote terminal and maximum number of active users is often restricted by roll in/out operation. If N is the number of active terminals ('active' means 'in use', i.e. in the state of input, output, waiting or thinking), it can be said that the system is operated in multiprogramming of N degree, but because all of these N programs can not reside in core memory on account of its capacity limitation, it becomes necessary to roll in/out (or swap) user's programs between core memory and file device (drum, disk etc.). Especially, in recent computers the processing

* Department of Applied Mathematics and Physics
** Data Processing Center, Kyoto University

rate of CPU is rather higher than the transfer rate of file device via input/output channel unit, therefore, system overhead due to roll in/out operation has increased and becomes one of the important factors affecting system performance.

With this point in view, it is very effective to use large capacity core memory (LCM) besides conventional high-speed core memory (HCM) as one of the methods decreasing system overhaed caused by roll in/out operation. This paper describes the analysis of the relation between roll in/out and system performance and some considerations about the usage of LCM in a TSS model based on a few assumptions.

It is supposed that whether there is a paging facility in the system or not has a considerable effect on the treatment and result of system analysis[4,7]. In the system which is analyzed and discussed here, there is no paging facility, the whole of each user's program is rolled in/out, reallocation of core memory is achieved by use of base-registers, and there is no supervisor's overhead time for scheduling and switching of tasks.

2. Roll in/out and System Performance

In a TSS, the users at remote terminals can communicate with the computer simultaneously, independently and immediately. As each user generates a request for computer processing, he in effect enters into the queue whose members are ready for processing by CPU. Thus each program in turn is serviced according to the specific scheduling algorithm, that is, transferred into core memory, operated upon and transferred out. Such a swapping of programs is called a roll in/out operation. After completion of the processing for request the computer responds to each user as output to terminal. Short response interval makes TSS valuable

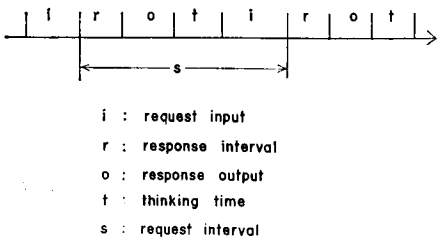

Fig. 1. Transition of terminal status.

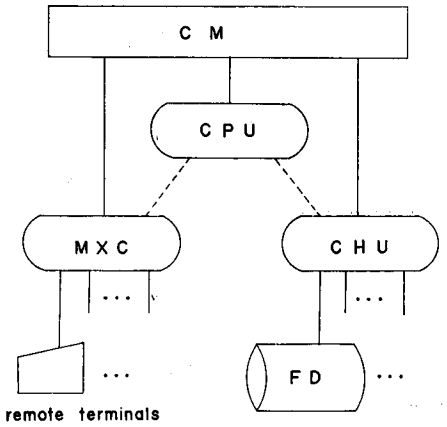

Fig. 2. System model.

and convenient for such jobs as online program debugging, trial and error procedure and so on. In order to respond to all requests as fast as possible it is effective to allow each request for CPU service no more than a specified amount of time-slice at a stretch and such quantum service is the most common scheduling of CPU service in TSS. Fig. 1 shows the cyclic transition of terminal status.

Consider the system model indicated in Fig. 2. In this model there are one CPU, one input/output channel unit (CHU) and some file devices (FD) for roll in/out and sufficient number of terminals via multiplexer channel unit (MXC). The following symbols are used:

- t_r : average response interval
- t_s : average request interval
- N : number of active terminals
- n : number of ready terminals, 'ready' means the status of waiting response
- X_r : CPU time of request, X_r is assumed to be a random variable with the distribution function given by $\Pr(X_r \leq t) = 1-\exp(-t/\lambda)$[2,3]
- λ : average CPU time of request
- q : time-slice of quantum service
- θ : average CPU time of quantum service
- L : average program size
- h : average number of programs residing in core memory (CM) user area, h is assumed to be more than the total number of CPU and CHU (now $h \geq 2$)
- d : average number of programs residing in FD
- τ : average CHU occupancy time of roll in/out operation for one quantum service
- a : access time of FD
- b : transfer rate of FD.

Let X_q be CPU time of each quantum service, then because of the basic property of the exponential distribution X_q becomes a random variable with the distribution function given by

$$\Pr(X_q \leq t) = \begin{cases} 1-\exp(-t/\lambda) & t < q, \\ 1 & t \geq q. \end{cases}$$

This means that it is no matter how many times the request being taken up for the present quantum service has been operated before.

Therefore θ, average of X_q, is given by the following equation:

$$\theta = \lambda(1-\exp(-q/\lambda))^2+q\exp(-2q/\lambda)\,. \tag{1}$$

τ, average time of roll in/out for one quantum service, depends upon N and let

$$\tau(N) = \begin{cases} 0 & N \leq h\,, \\ \dfrac{N-h}{N}\tau_1 & h < N \leq h+d\,, \end{cases} \tag{2}$$

where

$$\tau_1 = 2\left(a+\frac{L}{b}\right). \tag{3}$$

Let

$$W(N) = \max(\tau(N), \theta)\,, \tag{4}$$

then it is obvious that $W(N)$ is a critical parameter function of system operation because it strongly influences system performance such as maximum number of active users, $N_{\max}$, and average response interval. In the following analysis and discussion, the system is classified into two types, i.e. swapping limited and CPU limited, according to whether $W(N_{\max})$ is $\tau(N_{\max})$ or θ.

Next, consider the request which is $(i-1)q<X_r\leq iq$, then its processing is completed after i quantum services and the probability of generation of such a request is

$$\Pr((i-1)q<X_r\leq iq) = \exp(-(i-1)q/\lambda)-\exp(-iq/\lambda) \qquad i=1, 2, 3, \cdots\,.$$

Let M_q and M_r denote the number of quantum services and requests for a unit of time respectively, the following relation is obtained.

$$\begin{aligned} M_q &= M_r\sum_i i(\exp(-(i-1)q/\lambda)-\exp(-iq/\lambda)) \\ &= M_r\sum_i \exp(-(i-1)q/\lambda) = \frac{M_r}{1-\exp(-q/\lambda)}\,. \end{aligned} \tag{5}$$

Now, the number of quantum services which can be processed in the duration of t_s and t_r are given respectively by $t_s/W(N)$ and $t_r/W(N)$, therefore, $N_{\max}$ and $n_{\max}$, maximum of n processed in t_r, are obtained by using (5) on the assumption that there is no generation of next request during response interval at each terminal:

$$N_{\max} = \frac{(1-\exp(-q/\lambda))t_s}{W(N_{\max})}\,, \tag{6}$$

$$n_{\max} = \frac{(1-\exp(-q/\lambda))t_r}{W(N)}\,, \tag{7}$$

where d is assumed to be so large that there is no restriction to $N_{\max}$, i.e.

$d > N_{max} - h$.

In the case of swapping limited system, using (2), (6) and $W(N_{max}) = \tau(N_{max})$, the maximum number of active users is obtained as follows:

$$N_{max} = h + \frac{(1 - \exp(-q/\lambda))t_s}{\tau_1}. \qquad (8)$$

Function $W(N)$ is indicated in Fig. 3, where N_c is given by $h\tau_1/(\tau_1 - \theta)$. If $N > N_{max}$, each user at terminal can not communicate with the computer by request interval given by t_s.

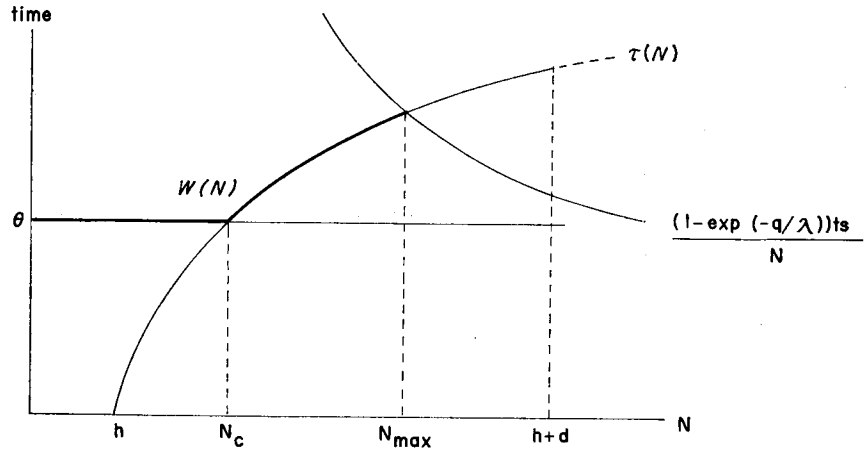

Fig. 3. Function $W(N)$ and maximum number of active terminals.

Besides N_{max}, one of the important system performance measures is n_{max}/t_r. The maximum number of ready terminals, n_{max}, which means the congestion of the system is directly proportional to the average response interval, t_r, and $n_{max}\theta/t_r$ also gives the busy ratio of CPU used for request, so it can be said that the larger n_{max}/t_r becomes, the better the system performance is for users and system. This is given by

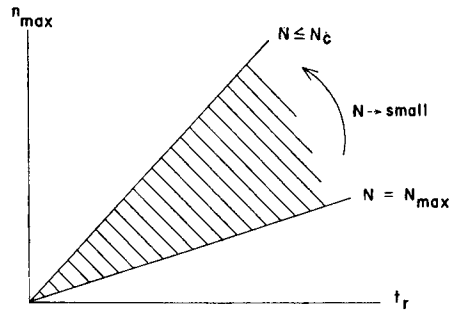

Fig. 4. Relation between response interval and maximum number of ready terminals.

$$\frac{n_{\max}}{t_r} = \frac{1-\exp(-q/\lambda)}{W(N)}, \tag{9}$$

and indicated in Fig. 4.

In the other case, the system is CPU limited, using (6), (7) and $W(N_{\max})=\theta$, $N_{\max}$ and $n_{\max}/t_r$ are obtained as follows:

$$N_{\max} = \frac{(1-\exp(-q/\lambda))t_s}{\theta},$$

$$\frac{n_{\max}}{t_r} = \frac{(1-\exp(-q/\lambda))}{\theta} = \frac{N_{\max}}{t_s}. \tag{10}$$

3. Use of Large Capacity Core Memory

Since large capacity core memory (LCM) is, in comparison with conventional high-speed core memory (HCM), not so fast but of low cost, to install LCM is effective for the system whose memony capacity is intended to be as large as possible. Table 1 shows the comparison of speed and cost between HCM and LCM in the case of two popular systems, FACOM 230-60 and IBM 360[7,8] (in IBM 360, respectively called HSS-high speed store and LCS-large capacity store).

Table 1. Comparison between HCM and LCM in two popular systems.

		cycle time	cost/month
FACOM 230-60	HCM	0.92 μsec	61 yen/word
	LCM	6 μsec	15 yen/word
IBM 360	HSS	0.75 μsec	3.7 ¢/byte
	LCS	8 μsec	0.6 ¢/byte

LCM is also directly-addressible core memory, and the only difference between HCM and LCM is in cycle time and access time. Thus the drum, disk and other peripherals can read from or write to either core memory via input/output channel, and CPU can directly access as execution of instructions or load/store of data on LCM as well as HCM. It is desirable to be able to transfer program or data between HCM and LCM faster than between core memory and file devices. Usually such high-speed channel unit for the faster transmission between HCM and LCM is installed, and called high-speed transfer unit (HTU) here. Using LCM in a time-sharing system, it is possible to design an efficient system by decreasing $\tau(N)$ and increasing $N_{\max}$ or $n_{\max}/t_r$.

Now, consider the system model with LCM indicated in Fig. 5. In this model there are one CPU, one HTU, one CHU and some file devices for roll in/out,

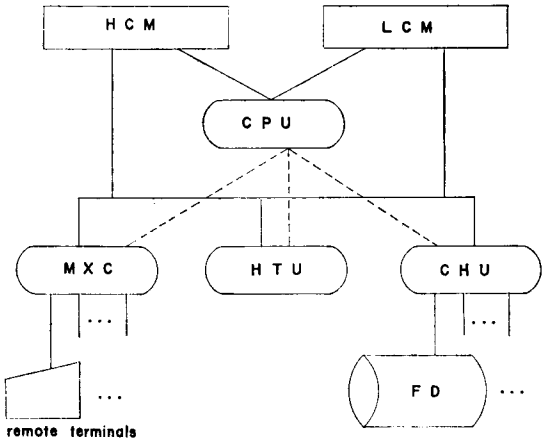

Fig. 5. System model with LCM.

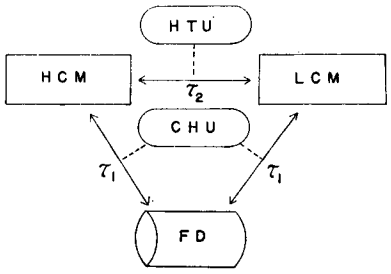

Fig. 6. Three ways of roll in/out.

and sufficient number of terminals via MXC. h, k and d denote the average number of programs which can reside in HCM, LCM and FD respectively, and it is assumed that $h \geq 2$ and d is so large that there is no restriction to $N_{\max}$. Other symbols are used in the same meaning as the previous section.

In such a system three ways of roll in/out operation, that is, between HCM and FD, LCM and FD, or HCM and LCM, are possible as indicated in Fig. 6. τ_1 and τ_2 are

τ_1 : roll in/out time of program (size L) between FD and HCM or LCM by CHU, given by (3),

τ_2 : roll in/out time of program (size L) between HCM and LCM by HTU, given by

$$\tau_2 = \frac{2L}{e},$$

where e is transfer rate of HTU.

Next, the usage of LCM in the system is roughly divided into two main classes:

Use of LCM as roll-out area

LCM is the roll-out area as well as FD. Access to the user's program by CPU for request is always performed on HCM after transferred from LCM or FD. Swapping of programs between HCM and LCM is done by HTU.

Use of LCM as roll-in area

Access to the user's program by CPU for request is always directly performed on LCM after transferred from FD as the roll-out area. HTU is not used.

As in both cases the use of LCM intends to make the system efficient by decreasing frequency and time of roll in/out operation, the analysis and discussion is useful in the case of swapping limited system, then it is assumed to be swapping limited, that is, $W(N_{\max})=\tau(N_{\max})$ in the following.

(i) Use of LCM as roll-out area

Let the average roll in/out time, $\tau(N)$, be approximately given by

$$\tau(N)=\begin{cases}0 & N\leq h\,,\\ \dfrac{N-h}{N}\tau_2 & h<N\leq h+k\,,\\ \dfrac{k}{N}\tau_2+\dfrac{N-h-k}{N}\tau_1 & h+k<N\leq h+k+d\,,\end{cases}\tag{11}$$

then, using (6) and $W(N_{\max})=\tau(N_{\max})$, the maximum number of active users is obtained as follows:

$$N_{\max}=h+k+\frac{(1-\exp(-q/\lambda))t_s-k\tau_2}{\tau_1}\,.\tag{12}$$

Function $W(N)$ is indicated in Fig. 7, where N_c is given by

$$N_c=\begin{cases}\dfrac{h\tau_2}{\tau_2-\theta} & \theta\leq k\tau_2/(h+k)\,,\\ \dfrac{(h+k)\tau_1-k\tau_2}{\tau_1-\theta} & \theta>k\tau_2/(h+k)\,,\end{cases}\tag{13}$$

and $n_{\max}/t_r$ is given by (9).

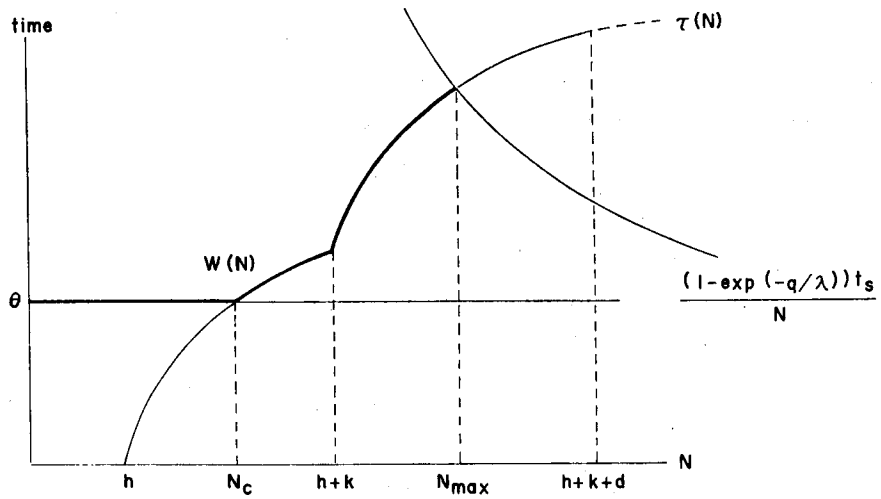

Fig. 7. Function $W(N)$ for use of LCM as roll-out area.

(ii) Use of LCM as roll-in area

Direct access to LCM from CPU increases CPU time for request because of its larger cycle time. Then, assuming that the average value of CPU time for request is $\lambda\eta$ where $\eta>1$, the average value of CPU time for quantum service, ω, is obtained by using $\lambda\eta$ instead of λ in (1):

$$\omega = \lambda\eta(1-\exp(-q/(\lambda\eta)))^2+q\exp(-2q/(\lambda\eta)) . \tag{14}$$

Now, consider the two typical cases of direct access to the user's program on LCM. First, each user's program is a self-executable object program in machine language and access by CPU during processing for request is always performed on LCM. In this case η is given by

$$\eta = \frac{g_2}{g_1}, \tag{15}$$

where g_1 and g_2 are the average duration for execution of one instruction on HCM and LCM respectively. Second, each user's program is a symbolic program which is interpreted and executed by the interpreter on HCM. Usually the processing of conversational language which is indispensable for TSS is implemented in interpretive mode because of its special facilities such as incremental compilation, controlled execution, online debugging and so on. If the interpreter program is re-entrantable and almost resident in HCM, direct access to LCM is the only load/store of data by the interpreter. In this case η is given by

$$\eta = \frac{g_1 j+f}{g_1 j}, \tag{16}$$

where f is cycle time of LCM and it is assumed that access to user's program area by the interpreter is every j steps on the average.

Let the average roll in/out time, $\tau(N)$, be given by

$$\tau(N) = \begin{cases} 0 & N\leq k, \\ \dfrac{N-k}{N}\tau_1 & k<N\leq k+d, \end{cases} \tag{17}$$

then, using $\lambda\eta$ instead of λ in (6) and $W(N_{\max})=\tau(N_{\max})$, the maximum number of active users is obtained as follows:

$$N_{\max} = k+\frac{(1-\exp(-q/(\lambda\eta)))t_s}{\tau_1} . \tag{18}$$

Function $W(N)$ is indicated in Fig. 8, where N_c is given by $k\tau_1/(\tau_1-\omega)$, and $n_{\max}/t_r$ is given by

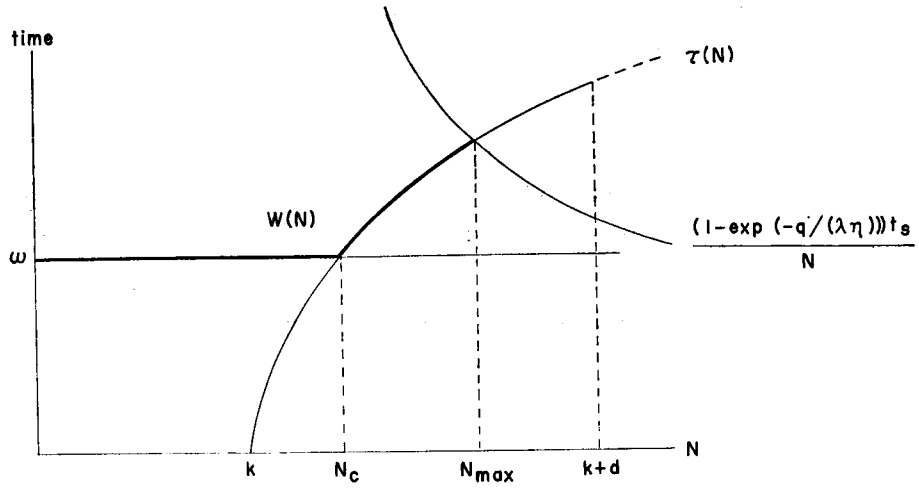

Fig. 8. Function $W(N)$ for use of LCM as roll-in area.

$$\frac{n_{\max}}{t_r} = \frac{1-\exp(-q/(\lambda\eta))}{W(N)} . \tag{19}$$

Next, it can be said that the average value of CPU time for quantum service always increases because the larger η becomes, the more frequent quantum-over CPU service is. Then the use of LCM as roll-in area may make system CPU limited because of increase of ω or much more decrease of $\tau(N)$ than ω. Then, the condition of the swapping limited system is obtained as follows by using $\omega \leq \tau(N_{\max})$:

$$\left(\frac{k}{t_s}+\frac{z}{\tau_1}\right)\left(\lambda\eta z+\frac{q(1-z)^2}{z}\right) \leq 1 , \tag{20}$$

where $z = 1-\exp(-q/(\lambda\eta))$,

thus it is necessary for η to be less than η_c, where η_c is the value of η derived from the relation of equality in (20). And the condition on which the interpreter program using LCM is operated in swapping limited state is obtained by using (16) and $\eta < \eta_c$ as follows:

$$j > j_c = \frac{f}{(\eta_c-1)g_1} . \tag{21}$$

Some numerical examples on the condition of the interpreter program using LCM are showed in Table 2 and Table 3, where the following values are used:

$a=17$ msec, $b=39$, 78 kw/sec, $g_1=1.6$ μsec, $g_2=6.1$ μsec,

f=6.0 μsec, λ=0.1, 0.2, 0.5 sec, q=0.5, 1.0 sec,
t_s=20, 30, 40 sec, L=5, 10, 20 kw,
$k=\dfrac{760}{L}$, $1<\eta\leq\dfrac{g_2}{g_1}$,

and values of η_c and j_c are calculated by using (3), (20) and (21). In these tables 'S' and 'C' mean respectively swapping limited and CPU limited system at $1<\eta\leq g_2/g_1$.

Table 2. Numerical examples of η_c (upper) and j_c (lower) at b=39 kw/sec.

L	t_s \ λ (q=0.5)	0.1	0.2	0.5	1.0	0.2	0.5
5	20	C	C	C	C	C	C
	30	1.31 12.1	C	C	1.19 19.8	C	C
	40	1.72 5.2	C	C	1.40 9.4	C	C
10	20	3.78 1.4	1.89 4.3	C	1.84 4.5	C	C
	30	S	S	S	2.53 2.5	1.27 13.9	C
	40	S	S	S	3.25 1.7	1.63 6.0	C
20	20	S	S	S	S	3.35 1.6	1.34 11.1
	30	S	S	S	S	S	S
	40	S	S	S	S	S	S

Table 3. Numerical examples of η_c (upper) and j_c (lower) at b=78 kw/sec.

L	t_s \ λ (q=0.5)	0.1	0.2	0.5	1.0	0.2	0.5
5	20	C	C	C	C	C	C
	30	C	C	C	C	C	C
	40	1.06 62.5	C	C	1.01 375.0	C	C
10	20	1.72 5.2	C	C	1.40 9.4	C	C
	30	S	S	S	1.72 5.2	C	C
	40	S	S	S	1.95 4.0	C	C
20	20	S	S	S	3.21 1.7	1.61 6.2	C
	30	S	S	S	S	2.81 2.1	S
	40	S	S	S	S	S	S

(iii) Optimum use of LCM

In the above discussion the use of LCM has been considered to divide into two classes, roll-out area and roll-in area. But usually these two kinds of usage are mixed in the practical system where strategy of using LCM depends upon each program. Now, if each program is designed to be executed or interpreted in both HCM and LCM, the following algorithm will be optimum for the use of LCM.

In the case of $\omega>\theta$, the use of LCM as roll-out area for small N and roll-in area for large N makes system efficient because of decrease of $W(N)$ and consequently increase of n_{max}/t_r as clearly indicated in Fig. 7 and Fig. 8. If N_p denotes the value of N by which $\tau(N)$ for the use as roll-out area given by (11) is equal to ω, then the usage as roll-out area when $N\leq N_p$ and as roll-in area when $N_p< N\leq N_{max}$ is an optimum algorithm which makes $W(N)$ as small as possible. Fig. 9 indicates function $W(N)$ in this case, and N_p is given by

$$N_p=\begin{cases}\dfrac{h\tau_2}{\tau_2-\omega} & \omega\leq k\tau_2/(h+k)\ ,\\ \dfrac{(h+k)\tau_1-k\tau_2}{\tau_1-\omega} & \omega>k\tau_2/(h+k)\ .\end{cases} \tag{22}$$

However, in such a system where there are any differences in the memory allocation and protection scheme between HCM and LCM, this algorithm is not so effective because of increase of the system overhead.

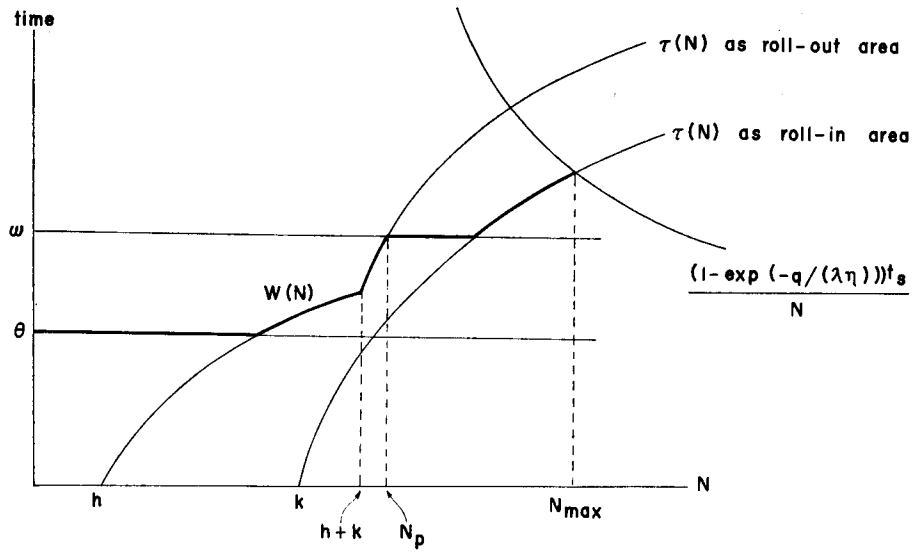

Fig. 9. Function $W(N)$ for optimum use of LCM at $\omega>\theta$.

4. Conclusion

The relation between roll in/out operation, which is inevitable and an important factor in a recent large-scale, general-purpose TSS, and system performance in man-computer communication was analyzed in the system model based on a few assumptions, then, the usage and the effect of LCM in TSS was discussed.

The condition on which the interpreter program using LCM as data area is operated in swapping limited state was obtained and it will be useful when a conversational language processor in TSS with LCM is designed and implemented.

The time-slice of quantum service is also an important and easy adjustable factor for system performance, and it is necessary to decide its value with the policy of which performance measure is placed great emphasis on.

However, this entire discussion has been based on the analysis by average value as for several random variables, so, it is desirable to analyze the system performance dynamically in real-world application. For this purpose, the digital simulation and the real-data collection are considered to be effective and will be tried.

References

1) J.I. Schwartz, E.G. Coffman and C. Weissman: A general-purpose time-sharing system, Proceedings of the 1964 SJCC, pp. 397–411
2) B. Krishnamoorthi and R.C. Wood: Time-shared computer operations with both interarrival and service times exponential, Journal of the ACM, Vol. 13, No. 3, pp. 317–338 (1966)
3) E.G. Coffman and R.C. Wood: Interarrival statistics for time-sharing system, Communications of the ACM, Vol. 9, No. 7, pp. 500–503 (1966)
4) J.E. Shemer and G.A. Shippey: Statistical analysis of paged and segmented computer system, IEEE Transactions, Vol. EC–15, No. 6, pp. 855–863 (1966)
5) A.L. Scherr: An analysis of time-shared computer systems, The M.I.T. Press (1967)
6) M. Tsujigado: Multiprogramming, swapping and program residense priority in the FACOM 230–60, Proceedings of the 1968 SJCC, pp. 223–228
7) R.E. Fikes, H.C. Lauer and A.L. Vareha: Steps toward a general-purpose time-sharing system using large capacity core storage and TSS/360, Proceedings of the 1968 ACM national conference, pp. 7–18
8) D.N. Freeman and R.R. Pearson: Efficiency vs responsiveness in a multiple-services computer facility, Proceedings of the 1968 ACM national conference, pp. 25–34
9) T.B. Pinkerton: Performance monitoring in a time-sharing system, Communications of the ACM, Vol. 12, No. 11, pp. 608–610 (1969)
10) Fujitsu Limited: FACOM 230–60 / Hardware, System Manual EX–011–1–4