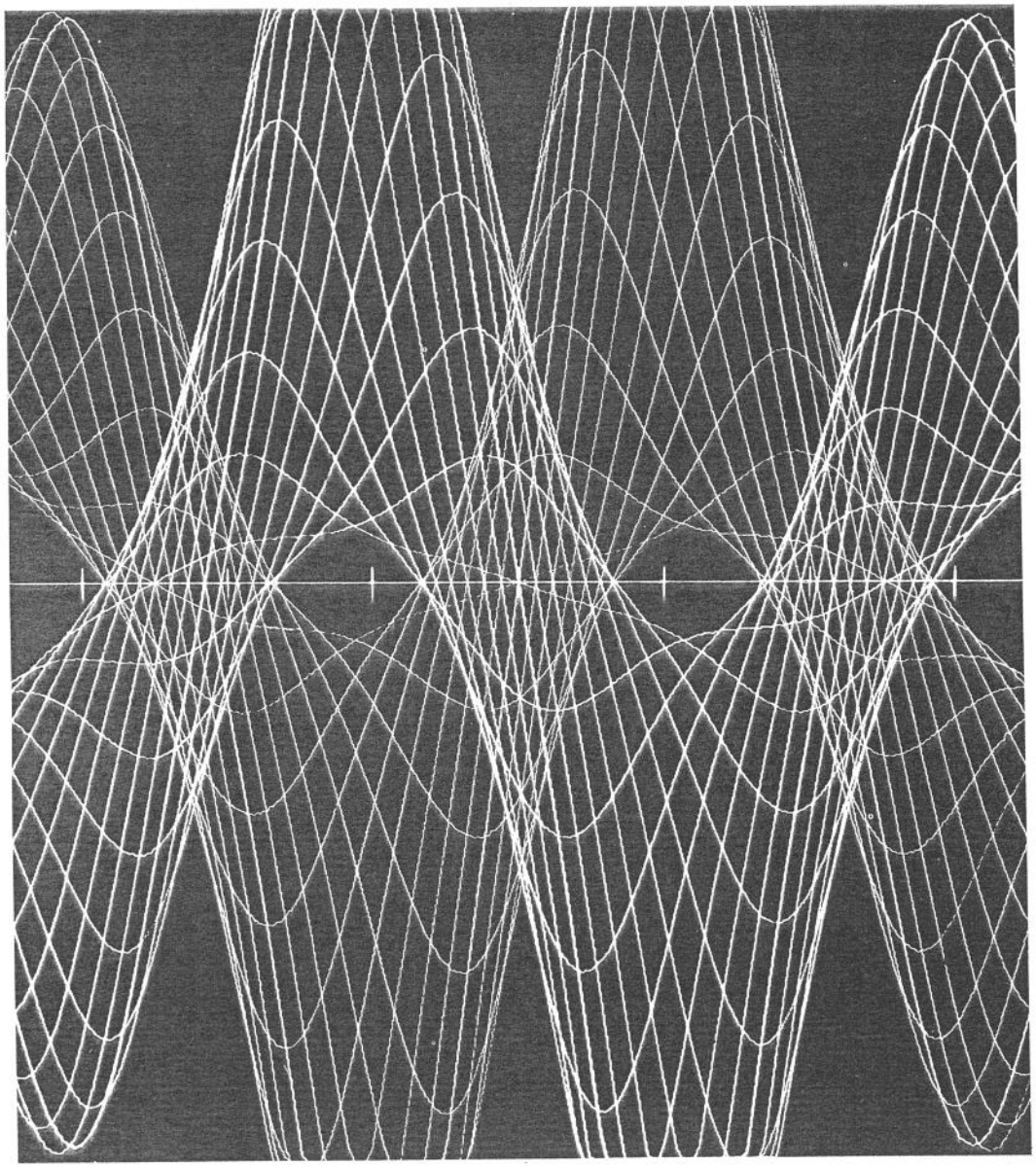

RC 2233

THE LOAD LEVELER

A. J. Shils

October 7, 1968

IBM RESEARCH

THE LOAD LEVELER

A. J. Shils
Research Division
Yorktown Heights, New York

ABSTRACT: Conversationally interactive computer systems display throughput which varies in accordance with fluctuations in load. Experiments have been made on a means of dynamically controlling the load for this type of system to automatically keep throughput at a high level.

A program called the "Load Leveler" has been built into the M44/44X operating system to set the instantaneous level of multiprogramming in order to improve system throughput. The factors involved are described and results of the Load Leveling operation are presented. Results are encouraging since system performance is improved with use of the Load Leveler. The improved performance is believed due to a relexation of contention for page space during squeezed core conditions. Comparison of system performance with and without the load leveler is made.

RC 2233 (# 11082)
October 7, 1968
Systems organization

1. Introduction

The M44/44X time shared system provided the framework for the Load Leveler investigation. This system allowed up to 16 simultaneous users, each one appearing to have private access to 2 million words of memory, a CPU, and an I/O complement. [1,2,3] During heavy load the system exhibited erratic throughput. It was believed that changes in throughput were linked to changes in CPU and paging activity. In an effort to provide an acceptable throughput, especially during heavy loads on the system, the development and implementation of a "Load Leveler" was begun. The load leveler is a mechanism for setting and maintaining a selected level of activity in the system to increase overall throughput. The design and performance of the Load Leveler are described in this paper.

2. System Load

System load was defined as all the work the system must perform to yield desirable results and a means had to be found to characterize this load.

The M44/44X Load Leveler uses the following simple, yet effective, characterization of the load:

(i) The total amount of work load being processed by the system is (a) the activity (% of time busy) of the CPU during Δt*, and (b) the number of pages replaced during Δt.

(ii) The total amount of work the system is required to complete is described by the number of jobs requesting processing.

The two factors, CPU activity and page rate were used to characterize system performance as in Figure 1. In Figure 1 the vertical axis represents CPU activity during the i^{th}, Δt interval. The horizontal axis represents the rate of page replacement in the same interval, where the end point, s, is equal to the maximum number of pages which can be replaced during any interval. The maximum, s, cannot be precisely defined in practice because on the M44/44X system page replacement involves 0, 1, or 2 page I/O transmissions depending on whether a blank page is requested and whether replacement requires preserving an updated copy in backup storage. The upper limit, s, is reached when no channel use is required during Δt

* Δt is the period the operating system permits to elapse between executions of the code comprising the load leveler.

and pages are constantly being replaced by setting all words of each page requested to zero utilizing the CPU at its CPU/Memory rate. But this is theoretical since a job which only requested page replacement would not produce practical results, unless perhaps it is a diagnostic. In practice, the observed maximum s represents mainly channel use and as such is limited to approximately 800 normalized pages* per minute for each of the two IBM 1301 II disk files used for back-up.

CPU activity versus page rate yield a graph which can conveniently be divided into three areas as in Figure 1. Each area is characteristic of a mode of the system's behavior as follows:

Area A: When the system is operating in Area A greater than "y" percent of CPU activity is observed regardless of the page activity. For high values of "y" this is desirable performance.

Area B: In Area B, both the CPU use and the page rate are low and it can be assumed that there is a lack of demand for these activities. Thus, system resources are being underutilized and it is assumed that the load the system was presented with is less than the amount of work the system can efficiently process.

Area C: System operation in Area C is characterized by relatively low CPU utilization and a relatively high rate of page activity.

It can be assumed that CPU use is low because the page rate is high. The CPU resource is consumed awaiting pages for tasks so as to have some work to process. This situation can be self-sustaining and is often termed "paging-to-death". The Load Leveler mechanism has a beneficial effect in this unfortunate situation by reducing the contention for page space.

The Load Leveler acts to maintain the highest CPU activity it can which means it attempts to maximize the amount of time the system spends in Area A of Figure 1. Load Leveler operates as follows:

(i) If the load leveler detects that the system is operating in Area C it sets aside one (or more) jobs in an attempt to decrease page competition and raise CPU activity.

(ii) If load leveler finds the system to be operating in Areas A or B, and there exists a job (or jobs) previously set aside by the Load

*Since various page sizes can be prescribed at system initialization time it is convenient to describe the rate in terms of the equivalent number of 256 word pages, e.g. "normalized pages".

Leveler, a job is returned to the regular job queue to compete for CPU and memory resources. Figure 2 shows the Load Leveler algorithm.

If Δt is chosen small enough (e.g. close to the page transfer time) the LL will run at a frequency which makes any short term page or idle condition appear to be significant and cause Load Leveler action. For example, suppose Δt is approximately the time needed to transmit a page and page activity occurs during a Δt. The page density relative to the short Δt is high, however, perhaps for n successive (small Δt) periods the overall page activity may be very low. When dealing with such time dependent densities one must be careful not to make a mountain out of a molehill.

If Δt is chosen large enough to include starting through completion of a job (or set of jobs) then Load Leveler is in effect, not active.

x: x is the page replacement rate used by LL to designate the amount of paging above which it is undesirable for the system to perform provided y is low enough.

y: y is the percent of CPU activity above which the CPU is considered overly idle. For example, if y=30% then less than 70% CPU activity during Δt is undesirable.

To check on the validity of the parameter, a test was made to determine the effect on job throughput time by varying the acceptable CPU activity parameter. The results of such a test are shown in Figure 3. Δt equaled 10 seconds and the page rate (x) equaled 500 (normalized) pages. The CPU parameter (y) was permitted to vary over eleven runs. Each run consisted of processing four Fortran compilation tapes, each tape containing a batch of source jobs. For the runs with y=90% and 99% idle LL took no action at all which is equivalent to operation without LL initially activated. When y equaled 90% and 99% and LL was not active, the execution times differed by about 5%. This is quite acceptable and testifies to interaction of the jobs at a level which precludes exact duplication of experiments. Below 90% CPU use the improvement was over 30% in run time to complete all jobs. Some factors responsible for the improvement are:

(i) When Load Leveler dynamically removed a job from contending for page space by removing it from the normal job queue, the pages which that job would have demanded were available for other

jobs' demands

(ii) Whenever more pages were available to other jobs, regardless of the reason, the contention among jobs for page space was lowered and the CPU ran relatively longer as the page group per job increased.

(iii) Due to lower contention for space because space increased by setting one or more jobs aside the amount of page I/O was reduced which further reduced operating system overload.

When x and y boundaries are specified for the region defined in Figure 1 it is possible to calculate the probability of successive points crossing such boundaries. The calculation took the form of a "from/to" matrix whose elements were the probability of boundary crossings and non-crossings. This matrix can be tabulated whether Load Leveler was activated or not because it is a calculation based only on passive recorded system activity. Boundaries can be arbitrarily applied to the recorded activity in a Figure 1 format to get any desired view of the system. Matrices are shown in Figures 4A and 4B for operation exhibited by the system with and without the LL, respectively. The data is the same as that used for Figure 3, i.e., multiple batches of Fortran compilations.

Both matrices were made based on parameters of Δt=10 seconds, y=70% and x=500 (normalized) pages. These correspond exactly to the LL parameters for Figure 4A. In Figure 4B, the matrix represents transitions for the system, when the Load Leveler was not being used.

Figure 5 is a representation in terms of Figure 1 of the data in Figures 4A and 4B. Observations of Figure 5 can be made:

(a) Without the Load Leveler the system made its transition from overload to overload condition with much greater frequency than with the LL. Thus, the LL provided an improvement in that it reduced the tendency of the system to stay bogged-down in an overload condition and reduced the tendency of the CPU to be idle.

(b) The more than 10% greater frequency of staying in the underload condition without the LL indicated that better use was being made of the system resources with the LL active.

(c) Without the LL there was a 10% greater frequency of going from an OK condition to an overload condition as compared to running with LL. Therefore, LL tended to smooth system operation as follows:

(i) kept it in OK rather than overload

(ii) when in overload, got it out sooner

(iii) when in the underload condition LL tried to force-feed some load to the system

(d) With the LL there was a 2% greater frequency of staying OK as compared to operation without the LL, however, the 2% can be disregarded. We have seen that at least approximately 5% differences can be attributed to non-repeatability of experiments.

Another representation of the data for Figures 4 and 5 is a plot of the pairs of y and x values recorded. These plots show the time ordered pattern of transitions via a Figure 1 format. Such a plot indicates that there are concentrations of activity. Figures 6A and 6B represent the data used in matrix calculations of Figure 4A and 4B respectively.

For Figures 6A and 6B only the overload area has been outlined. The large dot is the center of mass of all the points. In Figure 6A, with LL active, the central dot lies outside of the overload area but below the OK area. There is significant activity (Figure 6B) around coordinates 20%, 800 with very little activity for Figure 6A in the region.

On days when the LL was purposely disabled for comparison purposes users voiced complaints about system performance such as, ".....it took me (n) times longer to do a certain job of mine today than it usually takes". These complaints were justified because evidence about system response time from the recorded data confirmed them. The program which recorded the data used in Figures 4 and 6 had a delay instruction of 10 seconds which it executed at the end of each brief data collection activity. The amount of delay greater than the requested amount has been quantized on 1/10 second intervals, and the amount of excess time was a measure of the system's failure to respond immediately to demand. The number of occurrences of delay for each quanta is graphed in Figure 7. Ideally, all the responses would fall between 10.0 and 10.1 seconds (0.0&0.1 of the horizontal axis), however, there was a spread of response time.

Over 800 quick responses were obtained with LL active (Figure 7A) as compared to about 500 without LL (Figure 7B), and there were numerous delays beyond 35 quanta (3.5 seconds) when LL was not active. There were 1298 points recorded in Figure 7A, and 1062 in Figure 7B. Since 800/1289 is greater than 500/1062, there was improved response time with LL active. While only one set of data is quoted here, this phenomenon was consistent over several months of system operation when users were treated to unannounced

deactivation of the LL for comparison purposes.

3. General Notes on Load Leveler/System Integration

(A) the main work queue for the system distinguished between background (e.g. card or tape, non-terminal) jobs and time shared terminal (foreground) initiated jobs. Load Leveler set aside non-terminal jobs before selecting from the terminal user set. Conversely, when work was to be returned to the main job queue during underload, terminal jobs were selected first.

(B) When it was necessary to set aside terminal jobs, those terminal jobs having the least frequent human interaction were chosen for set aside first, e.g. LL tried to favor the person who more frequently used his terminal to interact with the system. (Jobs which required little if any conversational mode had the closed shop background facility available during prime time.)

(C) After LL acted to set a job aside, LL set its delay parameter to $t/4$ to have itself reactivated sooner than if it did not take any action. This delay variable was reset to t from $t/4$ if no action was taken. LL provided this "Quick-check" on its own performance to see if:

(i) The situation did not improve (e.g. still in overload) and another job should be set aside or

(ii) Things had improved and work could be returned.

(D) The interface between the Load Leveler and the replacement algorithm is deserving of further study. For example, an overall strategy governing the Load Leveler's choice of which job to set aside and the page replacement algorithm's [4,5,6] choice of which job to favor has obvious potentials. Favoring a job means that pages belonging to the favored job will not be usurped for other jobs or the favored job. There is no point to favoring a job which is set aside because that job can use no pages, nor should a job be set aside if it is the job which is favored by the replacement algorithm. This relation should exist as long as there is non-trivial replacement algorithm.

4. Conclusion

The Load Leveler has been successful in increasing the throughput of the M44/44X system. Any overhead incurred through use of the Load Leveler has been unnoticeable and seemed to be completely

negligible. One might do two things to further the worthwhile experience which has been gained.

(a) Even though the M44/44X system no longer exists one can speculate on the effect of minor changes that could have been made. For example, could an average page rate per job have been used to advantage to select which job was set aside in contrast to rules previously described and used? Also when no jobs are set aside and it is necessary to set the first one aside might it not be better to set two jobs aside initially? The quick-check feature described was an attempt at this mechanism.

(b) It should be interesting to see how the problem to which the load leveler addressed itself could be generalized. Then perhaps one could generalize the LL solution and consider a broader application to a future system. For example, there may be important load leveling interactions with schedulers, file manipulation, preventive maintenance, and human users, supervisors, and operators. The problems appear to be large and one should first get a more exact feel for their size.

References

1. Experience Using a Time-Shared Multi-Programming System with Dynamic Address Relocation Hardware, R. W. O'Neill, SJCC Proceedings, Vo. 30, 1967, pp. 611-621.

2. Adding Computers - Virtually, Computing Report, March 1967, Vol. III, No. 2.

3. Experience with the M44/44X Virtual Computing System, R. W. O'Neill, Computer Technology Conference, IEE, U.K. July 1967.

4. "Biased" Replacement Algorithms for Multi-Programming, L. A. Belady, IBM Systems Journal Vol. 5, No. 2, pp. 78-101.

5. A Study of Replacement Algorithms for a Virtual-Storage Computer, L. A. Belady, IBM Systems Journal, Vol. 5, No. 2, pp. 78-101.

6. Mapping Devices and the M44 Data Processing System, R. A. Nelson, Report RC 1303, IBM T. J. Watson Center, Yorktown Heights, New York, October 1964

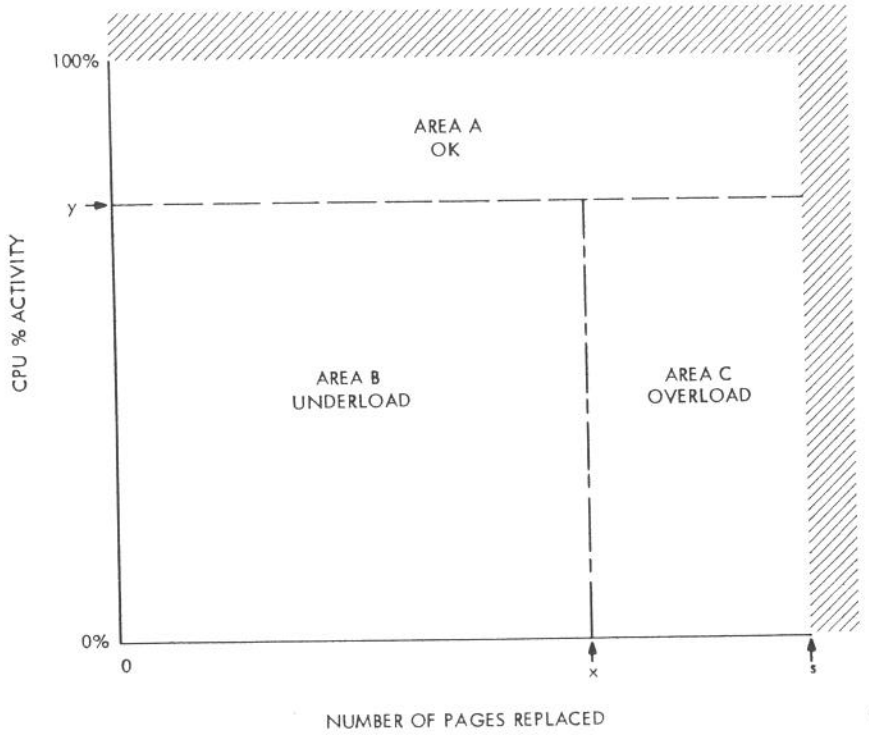

Figure 1. Areas of the System's Operation.

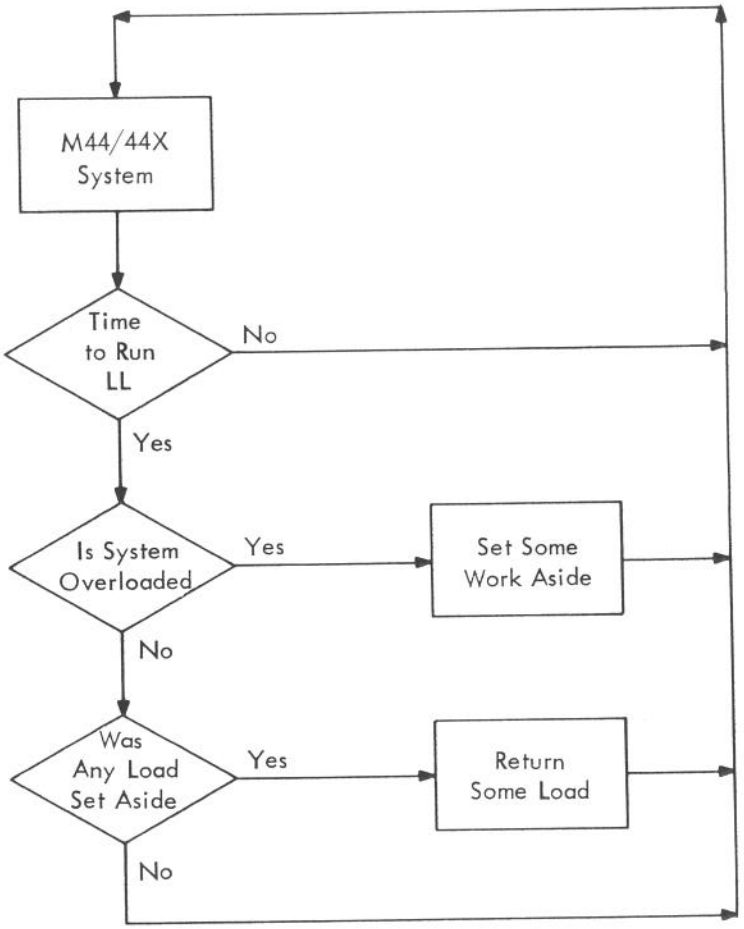

Figure 2. The Load Leveler Algorithm.

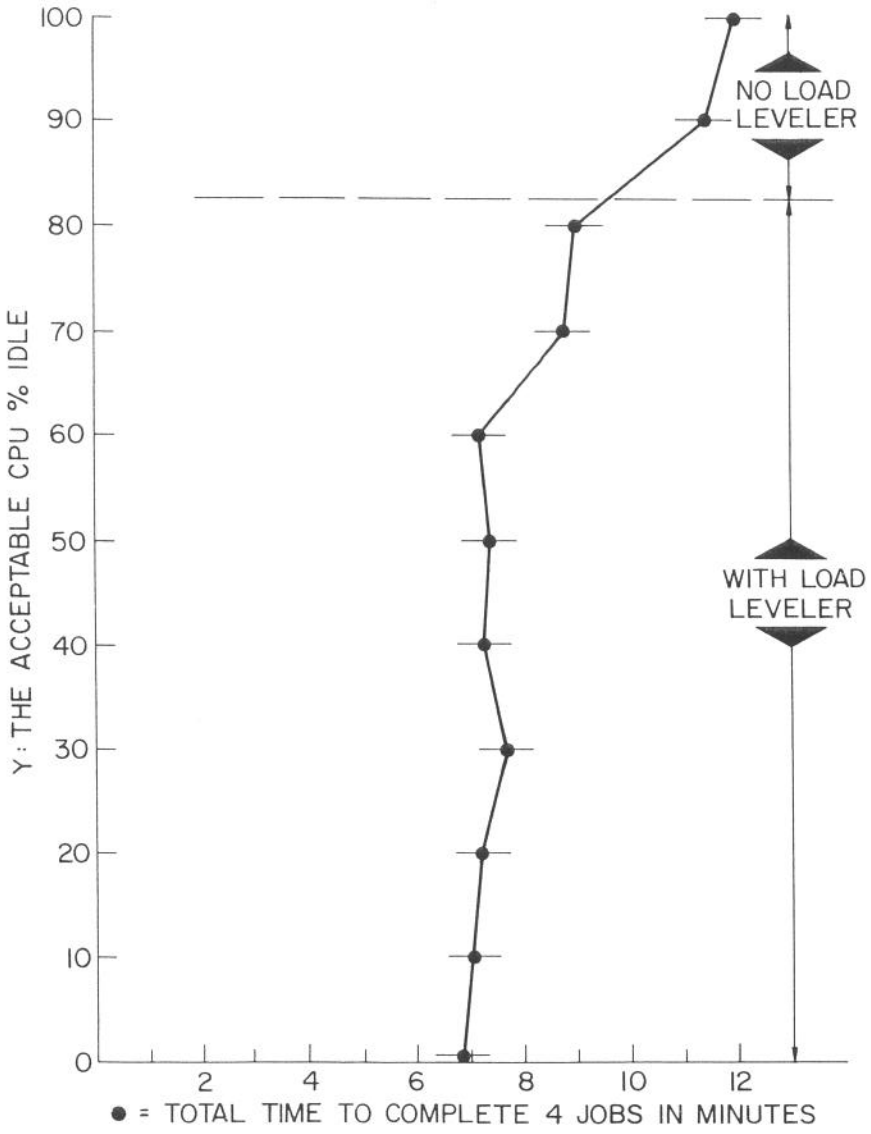

Figure 3. Job Turn Around Time As a Function of CPU Activity.

		To Area		
		UNDER	OK	OVER
From Area	UNDER	0.73	0.11	0.16
	OK	0.19	0.69	0.13
	OVER	0.33	0.23	0.44

Figure 4a. Probability of System Operation Transition WITH the Load Leveler.

		To Area		
		UNDER	OK	OVER
From Area	UNDER	0.86	0.09	0.05
	OK	0.10	0.67	0.23
	OVER	0.04	0.19	0.77

Figure 4b. Probability of System Operation Transition WITHOUT the Load Leveler

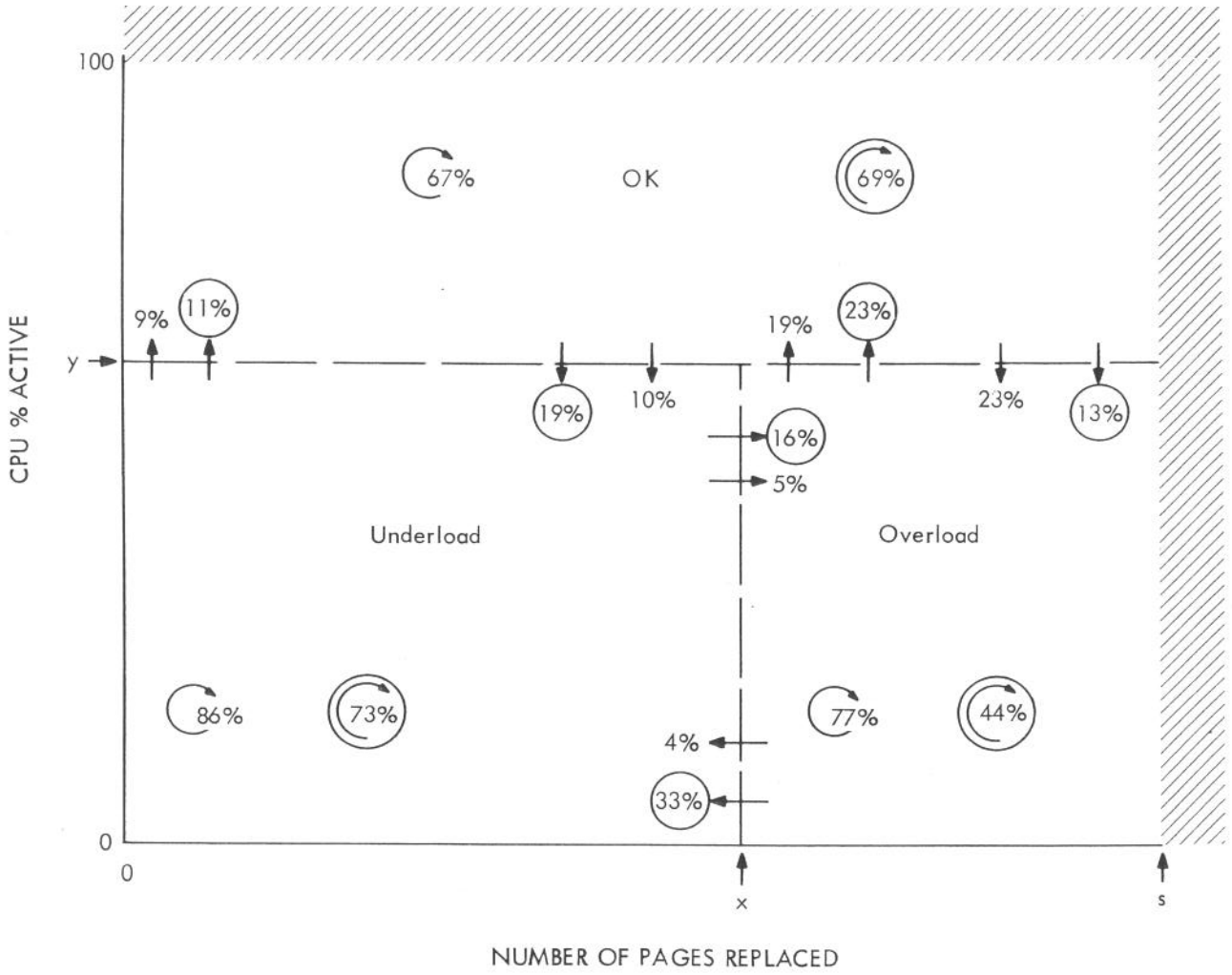

Figure 5. Comparison of Data in Figures 4a and 4b on the Field defined in Figure 1.

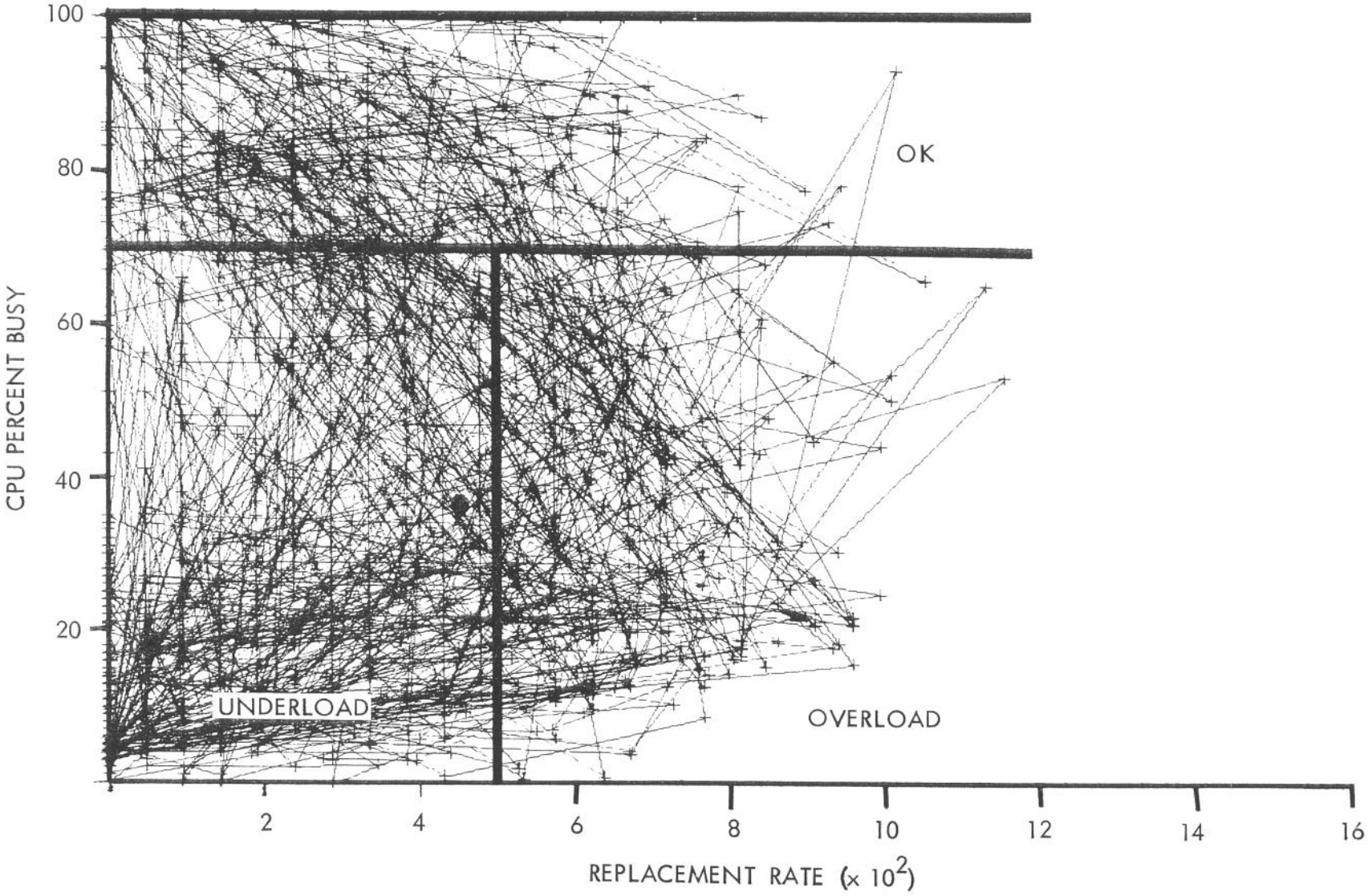

Figure 6a. System Activity Chart WITH the Load Leveler Active.

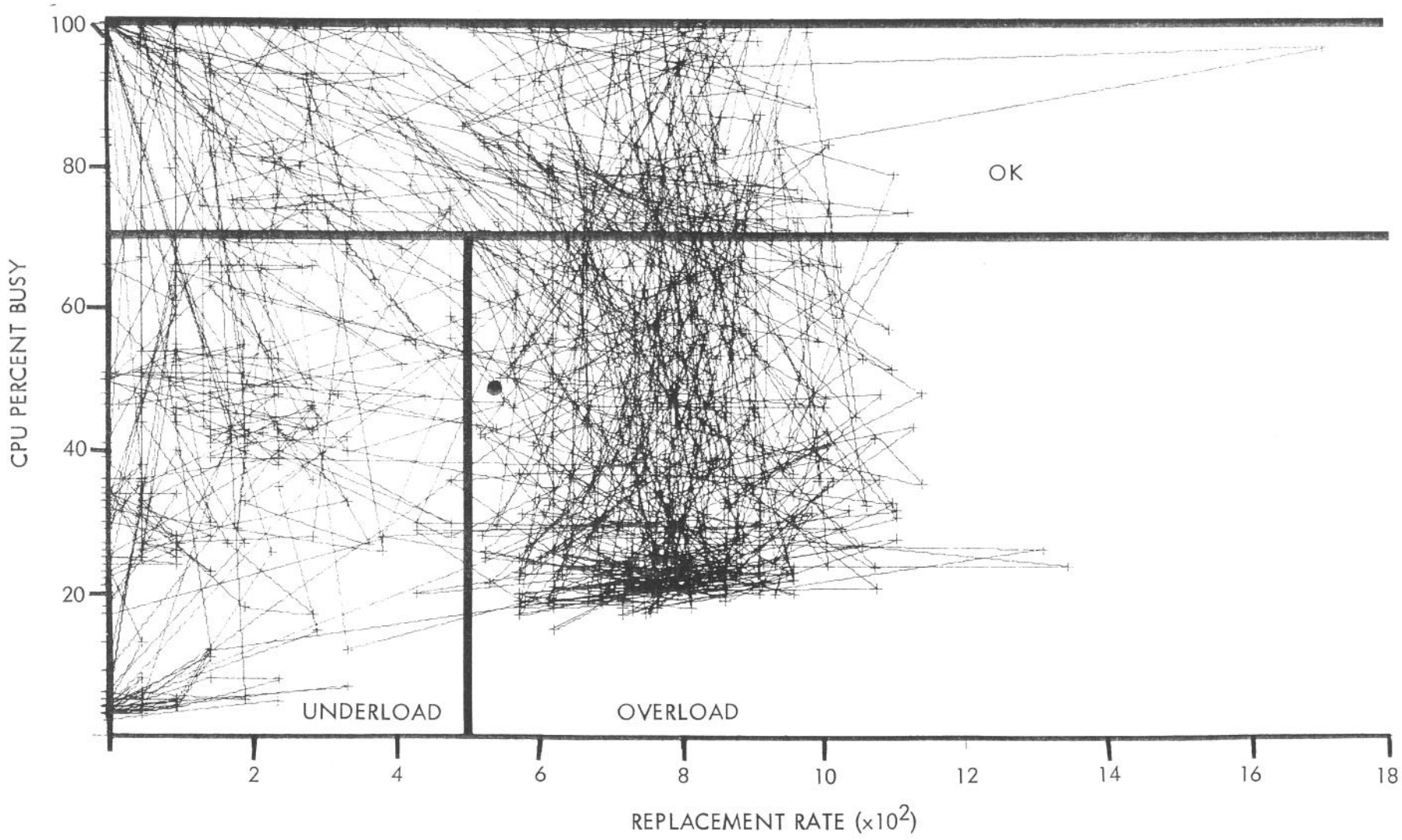

Figure 6b. System Activity Chart WITHOUT the Load Leveler Active.

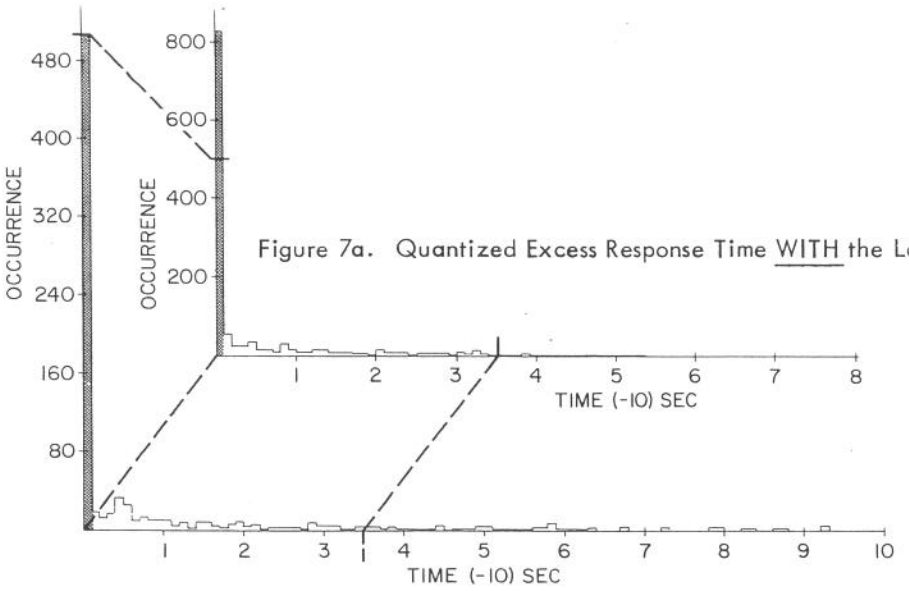

Figure 7a. Quantized Excess Response Time WITH the Load Leveler.

Figure 7b. Quantized Excess Response Time WITHOUT the Load Leveler.