To: MTB Distribution

From: R. E. Mullen and G. A. Texada

Date: January 11, 1981

Subject: Results of Emacs performance measurements.

PURPOSE

Persuant to MR8 PFS item 3.5.1.4 this MTB describes the measurement process and its results.

METHODOLOGY

CUESTA (Communication User Emulation System for Traffic Analysis) was chosen to "drive" emacs. CUESTA combines a reasonably simple script generation capability with a simple method for controling the speed at which "transactions" are transmitted to the test system. Two CUESTA scripts were created; a "simple" script of only text insertion to validate that CUESTA and emacs could work together, and a "complicated" script to more closely reflect a real editing session. The results contained in this MTB were gathered using only the complicated script.

The complicated script was a endless loop of 14 different editing functions including; text insertion, character deletion while inserting text, backing up lines, deleting words, inserting words, paragraph adjustment and paragraph/buffer wiping (See Appendix A).

To make extrapolation possible, the Multics configuration selected was:

1 CPU
1 MW Memory
1 MSP451 Dual channel
8 MSU451
1 DN6678

The measurement sessions were conducted as follows:

With the test system and the driver system up and running, the script was started. The users log in and invoke emacs and wait

(CUESTA sleep). Once all users were waiting, all lines were restarted with a 10 second delay between each to prevent syncronization. Once all lines had been running approximately 15 minutes, the meters on the test system were reset. At 25 minute intervals, meters were taken.

TUNING

The only Multics tuning parameter changed from the "standard" was tefirst. It was changed to 2 seconds. The key-in rate was 2 characters per second as performed by CUESTA.

METERS

For comparision purposes, the following set of meters is for a 1 user complicated script.

TOTAL TIME METERS:

Total metering time 0:25:49

	%	AVE	
Page Faults	0.01	1671.027	
Loop Locks	0.00	0.000	
RWS Overhead	0.00	0.000	
Getwork	0.10	771.631	
Loop Locks	0.00	0.000	
Post Purging	0.00	153.000	
Seg Faults	0.00	5390.111	
Bound Faults	0.00	11876.000	
Interrupts	0.40	3115.004	
MP Idle	0.00		
Loading idle	0.00		
NMP Idle	0.24		
Zero idle	98.13		
Other	1.10	68.01 % of non-idle	

DEVICE METERS:

Total metering time 0:25:49

	dska
Prior Page I/O	378
ATB	4098.305
Other Page I/O	4
ATB	387289.882
ATB Page I/O	4055.391
Prior VTOCE I/O	30

ATB	51638.650
ATB I/O	3760.095
% Busy	0
Avg. Page Wait	173.811
Avg. Page ^Wait	44.453
Avg. VTOCE Wait	47.198
Avg. Page I/O T	29.083
Avg. VTOCE I/O T	20.491
EDAC Corr. Errs	0
Errors	0
Fatal Errors	0

TRAFFIC CONTROL METERS:

Total metering time 0:25:49

Ave queue length	0.08
Ave eligible	0.08
Response time	0.006 sec

The following meters are from the final 35 user run:

TOTAL TIME METERS:

Total metering time 0:36:14

	%	AVE	
Page Faults	36.61	5503.090	
Loop Locks	0.00	0.000	
RWS Overhead	0.00	0.000	
Getwork	5.23	547.643	
Loop Locks	0.00	0.000	
Post Purging	0.78	4868.765	
Seg Faults	0.03	9253.852	
Bound Faults	0.09	35319.750	
Interrupts	20.96	2159.180	
MP Idle	0.07		
Loading idle	0.05		
NMP Idle	10.93		
Zero idle	0.46		
Other	25.57		28.89 % of non-idle

DEVICE METERS:

Total metering time 0:36:13

	dska
Prior Page I/O	146740
ATB	14.813

Other Page I/O	87479
ATB	24.849
ATB Page I/O	9.280
Prior VTOCE I/O	202
ATB	10761.220
ATB I/O	9.272
% Busy	527
Avg. Page Wait	102.006
Avg. Page ^Wait	1093.319
Avg. VTOCE Wait	185.788
Avg. Page I/O T	48.877
Avg. VTOCE I/O T	44.514
EDAC Corr. Errs	16
Errors	1
Fatal Errors	0

TRAFFIC CONTROL METERS:

Total metering time	0:36:18

Ave queue length	7.46
Ave eligible	7.07
Response time	1.381 sec

CONCLUSIONS

It appears that one L68 cpu, one MW of memory, one dual channel disk controller with 8 logical channels and 8 drives will pretty well support 35 users executing the given script at the given input rate. In order for the reader to judge the applicability of this result to actual or proposed use of emacs certain issues must be considered which relate to the nature of the script and the nature of the load placed on the hardware.

The script is neither entirely trivial nor entirely difficult; it could be characterized as consisting primarily as the inputting of new text with about 50% of the lines requiring immediate correction of a character and additional occasional backing up further in order to make corrections. At the end of each script cycle the paragraph is adjusted and finally deleted. The best thing about the script is that it consumes cpu time at about the same rate (per connect hour) as live users on System M, when run on an empty system. The best measure of the work being accomplished is the number of characters input into the system per second. On the single user run 1.6 cps were input, on the 35 user run .9 cps per user were input. Less progress was made by the simulated users on the loaded system because they paused, waiting for the mainframe to catch up, after each line of input and it took longer for the mainframe to catch up

under load. Because the average mainframe interaction incurred about 50 page waits and because the average page wait time was about 100 milliseconds we can assume the average editor function requiring a mainframe interaction took at least 5 seconds (realtime) of eligibility and 1.3 seconds to become eligible for a total of at least 6.3 seconds.

Looking at the metering output for the 35 user run it can be seen that the system is quite IO bound in spite of having a typical and adequate ratio of memory to processor, and a slightly higher than typical ratio of IO capacity (logical channels and drives) to processor. The seemingly large amount of page fault time is largely (more than half) due to having to loop waiting for an IO to complete before page control could even queue the IO of current interest. Another sure indicator of the IO bound situation is the large average page wait time, which should be more like 70ms rather than 100ms. If the loop waiting time in the disk dim were properly accounted for, it would be apparent that the processor was nearly 30% idle, whereas the IO system was saturated.

The import of the above is this: many Multics sites have found that a configuration similar to the one benchmarked here will support about 40 users without any element of the system being a bottleneck. The problem is that emacs users place a somewhat greater load on the IO system and a somewhat lighter load on the processor. This fact is presumably relatively independent of the particular script and suggests that any future effort to improve emacs performance concentrate on reducing paging. A further advantage of reducing paging is that, as we have seen, a major component of response time delays experienced by emacs users is page wait time. Finally, if the paging can be reduced, a certain amount of processor time will be saved as the number of disk interrupts, page faults, and calls to pxss$wait will be reduced.

APPENDIX A.

This script will have 14 different things going on,
referenced by capital letter below.
There should be as many logical channels as disks in
1 cases.
A full script cycle will take about (ballpark) 10 minutes, so
metering snapshots should be of 10 to 15 minutes duration, more
would be ok.

A. ten char input line
abcd abcde[CR]a^g
B. 20 char input line
abcd abcd abcd abcd [CR]a^g
C. 40 char input
D. 60 char input
E. 70 char input
F. 10 chars with deletion of one in middle
abcde# abcd[CR]a^g
G. 20 char input line with delete
abcd abcde# abcd abcd[CR]a^g
H. 40 char with delete
I. 60 char with delete
J. 70 char with delete
K. backup 3 lines, kill a word, go to end
(note: wait for prompt three times)
^p^p^p^g
ESC-d^g
ESC->^g (thats escape of greater_than char)
/* please put think time of 15 sec before one of those sends */

L. backup 3 words, forward one word, insert a word, go to end
ESC-b ESC-b ESC-b ^g (actually no blanks there ...)
ESC-f ^g
abcd^g
ESC-> ^g
/* please put think time of 10 seconds before one of those sends */

M. Adjust paragraph
/* think 20 seconds */
ESC-q ^g (excape Q)

N. Delete and start over
/* think 20 seconds */
^xh^w^g

---- so one whole script cycle would be as follows
AJBICHDGEFAKBLCJDKELAIBKCLDHEKALBGCKDLEFMN

it is made up of the pieces with letter-names described above.