Transaction Logging and Its Uses

Dennis Heidner
Boeing Aerospace Company
Seattle Washington

For some time database users have been concerned about the integrity of their databases and methods to prevent them from being corrupted. Another concern is performance measurement. When H-P introduced MIT-1918, they also introduced "Transaction Logging." Transaction logging is intended to provide a means of repairing databases which are either damaged or are suspected of being so. There are however many additional benefits to be derived from transaction logging including automatic audit trails, historical records of the database users, and information on the database performance.

The purpose of is paper to discuss the basic concepts of transaction logging, its benefits, and drawbacks. Various logging schemes, such as long logical blocks, and multiple IMAGE databases are discussed. Several different database logging cycles and HP recommended recovery procedures are discussed, and a method of recovering and synchronizing multiple databases is proposed.

Finally this paper covers a user written program which has been used to monitor the database performance, to validate and debug new user-written application software, and provide a complete audit trail for future reference.

INTRODUCTION

Many computers are justified only because they can keep track of large quantities of information in "real time" databases. In such cases it becomes extremely important that the integrity of this information remains consistent.

The database can be destroyed or corrupted in a number of ways. These include program errors, personnel errors, and computer hardware problems. A considerable amount of time and resources can be expended to eliminate most of the program errors, but it is almost impossible to guarantee a perfect program. The second source of inconsistencies is people. While it is possible to protect the information from human error by increasing the complexity of the program or by eliminating the human contact with the machine and its peripherals, both are often undesirable. Finally the third cause is system failures. System failures can be caused by numerous events including such things as fires, earthquakes, vandalism, hardware problems, power failures, and of course, MPE flaws.

We can take steps, however, to protect our investment in the database. There exist several very good programs,[1] such as DBCHECK and DBTEST, which will look for and can correct minor structural problems caused by crashes. But what about the user who must update a critical path in IMAGE? To do so requires a DBDELETE followed by a DBPUT. If the system crashes between the two, there will be no structural damage to be found. If you don't mind losing a $50,000 item or a $100,000 check, you have no worries. . . An effective database protection method is transaction logging. Logging takes many forms, the simplest of which only requires that we file away the paperwork used to generate the modifications to the database. Although this is convenient, it is a poor approach when it comes to recovering the database from a crash or system failure. For instance let's assume that we have a failure after two or three thousand transactions have been entered from terminals at several locations. Who wants to re-enter all the old data, while all the normal work is stacking up?

A better method is to have the computer keep duplicate copies of the information used to make the changes. Then it would only be necessary to instruct the computer to use the duplicate to reconstruct the database following a crash.

There are several ways that computers can be used to generate these duplicate copies. The most efficient method is to write the programs with an intrinsic transaction-logging system. This logging system can either be supplied by HP or could be a custom logging scheme. The problem with custom schemes is that they generally require as much or more design time as many of the applications programs that will use them. Since this work is not readily visible to either the end user or management, there is a temptation to do a quick job. The resultant lack of planning causes poor database and system performance. Additionally, in-house logging schemes only work with the in-house programs. If we use externally-written software (such as QUERY), we may find it difficult or impossible to get these routines to use our logging schemes.

TRANSACTION LOGGING (USERLOGGING)

HP recognized this need for database protection, and developed a version of transaction logging which runs on the HP3000.[2-3] HP's transaction logging is actually a process which runs under the control of the MPE operating system. If the database is enabled for logging, a logging process then attaches itself to the database when it is opened up for any update access. If the database is opened up in a read-only mode, the logging process is not attached. When the logging process is running it intercepts transactions after the IMAGE check has been made, yet before the actual transaction has been made in the database. This captured data (old and new values) are then blocked up in a buffer in memory. When the memory buffer fills up, the transactions are moved out to a logging file on the disc. If we are logging to the disc only, then this becomes our duplicate copy of the transactions. If we are logging to the tape drive, then the disc buffer is periodically moved out to the tape drive (see figure 1).[4]

If we have a system failure (or any other event which could cause a database inconsistency) then we use a database recovery procedure which uses a good copy of the database and the duplicate copy of the transactions to restore the information in the database to its condition only moments before the crash.

The recovery program which HP supplies is called DBRECOV. The program literally re-works all the transactions in the same sequence as originally made; this repetition assures that the database structure is correct and undamaged.

Once the database has been corrected and brought back into a consistent state, a backup copy is made and a new logging media is used. The act of making a backup copy and using a new logging media is known as beginning the logging cycle.

In order to implement transaction logging, HP introduced several new user-callable DBMS procedures: DBBEGIN, DBEND, DBMEMO, WRITELOG, BEGINLOG, ENDLOG, OPENLOG, and CLOSELOG. These new procedures are extremely useful because they let us define how transactions are logically grouped.[5]

To illustrate the importance of logical grouping of transactions, assume we have two mutually-dependent pieces of information. It is important that if any change is made to one item, the change that is made to the second item must also be made. If either item is not changed, then neither should be modified. We can do this by using DBBEGIN to mark the beginning of the dependent changes, and DBEND to mark the end (see figure 2). The intrinsic routines ensure that if there is a system crash or failure between the DBBEGIN and the DBEND, neither transaction is made. While transaction logging does not guarantee that we will not have crashes, it does provide some relief in recovering from their effects.

Now let's talk about the drawbacks. Anytime we ask the CPU to perform additional work, there is an increase in the overhead cost for our process. The object is to balance the additional workload on the computer with the benefits that we hope to gain.

Every time the memory buffer is moved out to disc, or the disc buffer is moved to magnetic tape, these transfers tie up the disc controller. Although this may be for very short periods of time, one of the biggest problems plaguing many HP3000 sites is slow response time due to a large number of disc accesses.

If we install logging then, our response time may become worse. Your alternative of course is to use absolutely no logging at all! Thus transaction loggings may be one of the necessary evils in life.

LOGGING STRATEGIES

The placement of the calls to DBBEGIN and DBEND can play a crucial role in the success or failure of logging. Since each call to DBBEGIN or DBEND causes a logging record to be written, and thus additional overhead, it is tempting not to use these at all. The people in the logging laboratory at HP wrote DBRECOV to handle both blocked and unblocked transactions (QUERY does not block its transactions). However while this is ideal for existing programs, we may be losing some very valuable information about our databases.

By properly placing the DBBEGIN and DBEND it is possible to measure the performance of our database. This information can later be used to tune-up our applications programs. Additionally proper placement of the calls enhances our crash recovery procedures.

The worst possible thing that we can do is to take the easy way out, calling DBBEGIN when we open up the database and calling DBEND as we close the database. This results in large recovery blocks. As long as we never have a crash everything works fine. However the first time we must recover after a crash, we might find that DBRECOV is unable to help us out. This is because the recovery process tries to resolve all transactions made between periods when the database is inactive. With the long blocking scheme the database is almost always active. DBRECOV will attempt to build a monstrous file to look for dependent transactions, and inevitably fail!

HP recommends that we make all the necessary locks on the database, call DBBEGIN, make the transaction, the call DBEND before unlocking (see Figure 3A). This will ensure that we have a minimum chance of large concurrent blocks.[6]

Another strategy that appears to work well is to call DBBEGIN, then lock the database or sets, and make our updates. Conversely we would unlock, and then call DBEND (see Figure 3B). This method allows us to measure the time between the begin and the end, which reflects the performance of our database. This procedure works quite well, as long as the following conditions are met:

- Always use ASSIGN LOCK OPTION OFF in QUERY
- Our transactions are made by terminals, and designed so that they collect the data from the screen, perform edits, then go through the DBBEGIN, DBLOCK, updates, DBUNLOCK, DBEND.

If you cannot operate under these conditions, then stay with HP's recommendations.

MULTIPLE DATABASES

When HP first introduced transaction logging, they did not make any provisions for synchronizing transactions which span multiple databases. The DBBEGIN and DBEND intrinsics work only for a single database at a time.[7] However with MIT 2028, HP introduced the BEGINLOG and ENDLOG intrinsics. These new intrinsics now make it possible to develop a method for synchronizing multiple database transactions. This is done by calling BEGINLOG before any multiple database transaction, and ENDLOG at the completion of the transaction (see figure 4). A user-written program could then scan the transaction log for complete BEGINLOG-ENDLOG blocks and indentify the record number of the last complete transaction.

To recover the database you then run DBRECOV and specify @@CONTROL EOF=recordnum." It may be necessary to run DBRECOV for each database that was involved.

LOGGING CYCLES

The method and length of our logging cycles depends heavily on the application and previous experience with the computer system's reliability. There were several possible methods proposed by HP during the MPE 1918 update course. These include:

- DBSTORE, then start a new logfile
- DBSTORE, start a log tape, when it fills start a new one, when it fills start another . . ..
- SYSDUMP, start a logfile

The first logging cycle method is the perferred method. It is straightforward, the recovery procedure is easy to follow, and in the event of a system failure, downtime is limited to the time needed to recover one logfile.

The second type of logging cycle should only be used on databases which require backing up, but have very little activity. This is because each logfile complicates the recovery procedure, and adds a considerable amount of time to recover each logfile.

The third logging cycle option omits the DBSTORE. We have found that a DBSTORE takes about 2 minutes for 3 megabytes of database (1600 bpi tape, series 33 computer). At first glance it would appear that the use of DBSTORE wastes time. However DBSTORE sets some internal flags and time stamps which SYSDUMP does not. These internal stamps and flags are used by DBRECOV to provide added protection against using logfiles from the wrong time period.

If you use a SYSDUMP tape, you must remember to request SYSDUMP store all the files. If partial backups are done, the database must be restored from the latest full backup, then restored from each succeeding partial, before DBRECOV is used. Because the time stamp and flags were not set by SYSDUMP, we must then specify that DBRECOV is to ignore all time stamps and flags. This is often difficult or dangerous to do, especially if your system operators are inexperienced.

SYSDUMP should only be used as a backup for the previous two logging methods. If you do not want to have your database stored on your backup tapes, then you should look into Alfredo Rego's STORENOT program. STORENOT allows the creator of a database to "tie it up" so that it is not stored by full or partial backups.

The logfile can reside on either the disc or magnetic tape. It is faster to log to the disc; however, if the reason for the system failure is a disc hardware or free space problem, you could lose both your database and the backup copy of the transactions. The other choice is for the logfile to reside on tape. This has two drawbacks: first, it ties up the tape drive, and second, it periodically requires the CPU to move the logging buffer from the disc to tape. If the system is already heavily loaded this can only worsen the problem.

If you decide to log to a disc file, you should be careful to build the logfile large enough to hold all of your expected transactions plus a reserve. You can obtain a rough estimate of the log size by:

```
# of sectors = 4*number of database opens
               + (number of updates * up-
                  date rec len)
               + (number of puts * put re-
                  cord length)
               + (number of deletes * de-
                  lete rec len)
               + 1 for DBEND
               + 1 for DBBEGIN
        update rec len (in sectors)
            = (# of items in list
          + update buffer size)/256
        delete rec len (in sectors)
            = (# of items in list
          + delete buffer size)/256
       put record length (in sectors)
            = (# of items in list
           + put buffer size)/256
```

If the buffer sizes are not known — use the media record size . . . you can get that from a DBSCHEMA compilation.

You can count the # of items in the item list or if "@;" was used then just use the item count in that particular set.

If you are not sure you calculated the size correctly then use the :SHOWLOGSTATUS command to monitor the number of records in the log. If you run out of space in a disc file while logging, you can put the database in a state similar to a crash; this may require that you go through a complete database recovery procedure!

CRASH RECOVERY

HP implies that a recovery procedure must be followed every time there is a database crash.[8] This can be disastrous. On one occasion we followed the recommended crash recovery procedure, purged the database, restored the database, and started DBRECOV. It bombed, and upon investigation we discovered that approximately 500 transactions had been lost because the logtape was blank due to a tape drive malfunction. Moral of the story: You should first determine the cause of the crash, then verify that the logfile is good via LOGLIST or DBAUDIT.

We also found that it is important to write your applications programs so that they abort to prevent further transactions if they detect a logging problem. It is possible for the program to pass the IMAGE checks for DBDELETES, delete an item, then find out there is a logging problem! The end result is one less item in the database. This becomes especially critical if you are one of the many IMAGE users who have to update critical items by deleting and re-adding.

If the crash is because the logfile was too small and filled up, then the end result of trying to recover is that your data-entry personnel spend hours reconstructing previous transactions. It is better to run a program such as LOGLIST, and find out what data have been effected. Then run DBSTORE, build a new, larger logfile, and start a new logging cycle. One note of caution: we found that parity errors on the tape drive cause a crash whose symptoms are almost identical to those of one caused by running out of space on a disc logfile.

If the crash is because of a system failure, the correct procedure is:

- Perform a memory dump for HP
- WARMSTART (if possible); this causes MPE to try to recover the transactions in the internal disc buffer. (THIS IS VERY IMPORTANT!)
- SHUTDOWN
- COOL or COLDSTART
- Run LOGLIST or DBAUDIT to determine who, what, when and how bad the crash is.
- If the database was not open in an update or modify mode then simply start a new logging cycle and get your users back on.
- If the database was open in an update or modify mode, then purge the database using DBUTIL, restore the database using DBRESTOR and recover using DBRECOV. BE SURE TO START A NEW LOGGING CYCLE!

AUDIT TRAILS

Good data processing applications have some form of built-in controls which allow for the verification of the accuracy of the database. This is especially true if the application is in the banking, inventory control, or government fields. In many applications some form of an electronic "paper trail" is mandatory.

The information which is logged by IMAGE exceeds most audit requirements and can provide the required electronic trail. Transaction logging records information about who, when, where, and how an item or entry was modified. This information can be extracted in several ways. Bob Greene has a package called DBAUDIT which can analyze the log.[9] I have contributed a similar program called LOGLIST (via IUG 1982 swaptape) which can expand the transaction log per directions. It is described in a appendix to this paper.

The audit trail recorded by transaction logging can be enhanced by carefully planned use of the 'text' area on DBBEGIN, DBEND and DBMEMO. We record the information which leads to a transaction when we call DBBEGIN. The results of the update or special error conditions are logged on the DBEND. If needed, DBMEMO is used to record special remarks and initials of the person making the change.

If you foresee a requirement for frequent analysis of the transaction log, it is also important to include a time-stamp as an item in individual data entries. This forces IMAGE to log both the present time-stamp and its previous value. The value of this information is apparent when tracing the history of a specfic data entry. With a time-stamp on your data entries, it is possible to pull and analyze only those logfiles which contain the time interval about the time-stamp of interest. Since analysis of a transaction log takes about 10-15 minutes for 40,000 records, the time saved in this manner can be considerable.

Perhaps more importantly from a programmer's point of view, we can use the audit trail as a method of providing continuous software monitoring. The concensus among data-processing people is that it is virtually impossible to guarantee that a complex program will correctly handle all cases regardless of what data is fed to it. When an error does occur at our site, experience indicates that it is generally several months before we notice that something is wrong. By maintaining transaction logfiles for a sufficient length of time (6 months), it is possible to locate the source of most errors. This makes it much easier to correct latent program errors. In addition we have found that if the problem was caused by human error, the hard-copy printout that can be generated from the log tape goes a long way toward refreshing the memory of the person who made the mistake.

For users at sites whose software must be accepted by Quality Assurance, audit trails have an additional advantage. As part of the acceptance testing on new releases of our applications programs, we DBSTORE the database, then run the test programs and fully analyze the log. This enables us to provide a visual check on fields and items in a manner easier than using QUERY.

After using the transaction log as an audit trail and debugging aid during the last two years I would estimate that we have saved probably a hundred man-hours which would otherwise have been spent looking for the cause of "freak errors."

As with all good things in life there is a "Catch-22." IMAGE3000 is structured as a closely-knit group of files tied together with the root file. When modifications are made to the database , only the set number, item number and item buffer are logged. If the root file is altered (by using ADAGER, DBGROOM, etc.), then the link between the database and the transaction log is broken. The most obvious problem occurs when the order of data sets is changed with ADAGER's DETSLIDE. Suddenly your Employee-Detail becomes your Part-Master and the log analysis program either bombs or gives ridiculous answers. You have two choices: either don't use ADAGER (not a very realistic choice), or use ADAGER's SCHEMA to generate a dummy version of the database structure as it appeared before changes were made. Then use the editor to shrink the capacity of all the sets down to 3 or 5. Assign this schema some version number and identify on all logfiles under which version of the schema the logfile was made. I have set up a separate group in our account for these "old, shrunk databases." Then when I need to look at an old logfile, I set up a file equation referencing the old "database" and run LOGLIST under that condition.

TRANSACTION-LOGGING PERFORMANCE

There is a great emphasis on designing systems with better response time. For this reason any type of overhead (regardless of how beneficial) is generally shunned. To make matters worse, when HP introduced transaction logging with MIT 1918, they had indicated that there would be a "through-put reduction of 30% for large modication-intensive online applications running 10 or more concurrent processes."[10] Unfortunately the test environment used for that statement was not completely explained. During the past two years we have been using transaction logging on a Series 33 with 768 kbytes and typically 11 active processes. Our experience has shown that there was probably less than 10% reduction in throughput. So, what is the overhead cost of transaction logging?

In order to find out, I wrote a program (DBPERF) which allows me to benchmark IMAGE transactons with and without logging. The benchmarks are deliberately run with as light a load as possible in order to isolate the overhead caused by logging from the effects of other users' activities (see APPENDIX: DBPERF). The results of the tests are shown in Figures 5-7. In Figure 5 we see the comparison of the time to DBPUT verses pathcount, on series 33 and 44 CPU's. As seen in Figure 5 the added overhead caused by transaction logging, is marginal. The anomalies on series 44 data was caused by a user logging on and using FCOPY during the benchmark test. Figure 6 shows the comparison of the time to DBDELETE verses pathcount, on the series 33 and 44 CPU's. The overhead caused by transaction is marginal, again the anomalies on the series 44 data was caused by a user logging on and using FCOPY during the benchmark test. Figure 7 shows comparisons of the time to DBOPEN, DBUPDATE and block transactions with DBBEGIN and DBEND. Earlier I mentioned that logging blocks up the IMAGE transactions (approximately 32 transactions), then moves this buffer out to disc. The overhead caused by this movement is comparable to the roll-in and -out of an inactive user process by the memory manager (MAM).

In most on-line applications the overhead added to the transaction is considerably less than the threshold point at which the system becomes overloaded. However batch jobs are generally another story, if you have batch jobs which require a considerable amount of system resources, run them without logging. Store your database before the job begins, stream the job, and when it completes, then store the database and start a new logging cycle. If you have a crash during the unprotected batch jobs it will only require that you DBRESTOR and rerun the jobs.

PREDICTION OF RESPONSE TIMES

At this point it will be worthwhile to discuss a little queueing theory and how it is used to estimate response times so that we can illustrate the effects of transaction logging on the system. A queue is just a waiting line.[11] When we analyze queueing systems, we talk about such things as number of servers, arrival rate, transaction rate and number of users. The classical example of queues in operation is the waiting lines at banks. With only one cashier (number of servers), if the customers arrive at a rate of one per hour (arrival rate) and the cashier takes only 15 minutes to complete an average transaction (transaction rate), then there will be no waiting line and the cashier can perform some overhead functions such as washing windows while waiting for the next customer. If, on the other hand, customers arrive every 15 minutes, then we can expect to find a person at the cashier constantly. The windows start to collect dirt and grime since the cashier no longer has time to wash them. When the arrival rate of the customers increases to one every 10 minutes, we soon find that a line is forming. If sufficient time is allowed to pass, customers start to switch banks, the cashier demands a raise and the windows now appear to have several layers of dirt and grime and strange creatures crawling on them.

Transaction processing on an HP3000 performs in a similar manner. As long as the arrival rate is sufficiently slower than the transaction rate, MPE is able to perform its necessary overhead functions and the response time is good. Unfortunately the HP3000 cannot ignore its overhead as the cashier did, so as the arrival rate approaches the transaction rate, response time begins to suffer.

It is possible to estimate the response time of the computer if you are able to estimate the number of users, the average time each user "thinks" about what needs to be done, and the time required to complete the transaction. The average "think time" is equal to:

```
                  arrival rate
Think time =    ---------------
                number of users
```

For example:

The XYZ Company has an HP3000 Series 33 computer on which they wish to implement an application which will support 10 users. The "think time" of these users is about 30 seconds each per transaction. The transactions consist of a DBDELETE and a DBPUT on a detail set with four paths. What will their transaction response time be?

The transaction response time is equal to:

```
        Transaction response time= Queue length * transaction rate

        Queue length = the greater of
                                      1
                                     or
                   number of users * transaction rate
                   -----------------------------------
                              think time

Using the IMAGE benchmark results, we then determine:

        Transaction response time = Queue length * 1.3 sec

                       10*1.3    13
        Queue length = ------  = --  ; as noted above, use 1
                         30      30

    then Transaction response time = 1.3 sec

If XYZ adds logging, it will be:

                       10*1.4    14
        Queue length = ------  = --  ; as noted above, use 1
                         30      30

    then Transaction response time = 1.4 sec
```

Our model works well as long as the computer has time to perform its overhead functions, i.e. code-segment swapping, MAM function, and garbage collection. The time available for the computer was approximately:

```
                             User think time
     Computer idle time = ---------------   - transaction rate
                             number of users
```

In the case of the XYZ company this averaged 1.6 seconds per user transaction (with logging).

The overhead that was added due to transaction logging is:

```
                          transaction    transaction
                          time with    - time without
                          logging        logging
        Added overhead = ----------------------------   * 100
                           transaction time without
                                  logging

    or, for XYZ,

                          (1.4-1.3)
        Added overhead = --------- * 100   =  7.6%
                             1.3
```

If 7.6% overhead is enough to cause XYZ's machine to have problems, can you imagine what an additional user using QUERY, the editor, or any of the compilers would do?

An additional benefit from transaction logging is that we are able to collect the arrival rates, transaction rates, and number of users during our actual production enviroment. With this knowledge we can make more accurate design decisions when developing new and additional applications.[12]

CRASH-PROOF?

How crash-proof is your database? Damage to databases can be caused in several ways. The typical cause of damage is a crash occurring while adding or deleting an item to or from a detail set. If the DBPUT or DBDELETE was manipulating the internal pointers in the database, then you can probably count on having at least one broken chain. Other types of database crashes occur when MPE or some "neat" privelege-mode program adds its own kind words to a random data set! When discovered, this error has the same symptoms as a broken chain; however, you may also be missing a considerable amount of data.

Perhaps the worst kind of database crash is the one you can't find. That is, DBTEST, DBCHECK, ADAGER and even DBUNLOAD-DBLOAD say everything is ok. These errors occur when the data set has a critical path which must be updated. Since IMAGE will not let us update critical paths, we have to delete and re-add. If a crash occurs after the DBDELETE is complete and before the DBPUT re-adds the item, then we have lost an entry in the database though the database structure remains intact (see Figure 8). DBTEST, DBCHECK and the other routines have no way of testing for or detecting this error. If your HP3000 is an accounting system, this is intolerable. This type of error could be prevented by using transaction logging and placing the DBBEGIN at the start of the transaction and DBEND at its finish.

It is possible to estimate your chances of having some form of damage to your database in the event of a crash. This Crash Figure of Merit (CFOM) is given by:

```
              (transaction rate * number of users)
       CFOM = ------------------------------------ * 100
                          think time
```

If your CFOM is high, say 20 or 30 percent, then it is probably worth the effort to run DBTEST and DBCHECK on every database that was open when a crash occurred. It may also be very much worthwhile to try transaction logging. If the CFOM is very low (one to two percent), then it is probably easier to manually correct errors and run DBCHECK at some convenient time.

SUMMARY

This paper discusses the merits and drawbacks of transaction logging, and provides some basic guidelines to aid in the successful implementation of transaction logging. Since most applications are designed to "earn" money, it is only fair to treat transaction logging in the same manner. As summarized in figure 9, the decision to log or not to log should be made only after a careful review of the associated system costs, its performance cost, alternatives, and by establishing values for the intangibles such as improved data security, benefits from audit trails, etc.

ACKNOWLEDGEMENT

I wish to thank the HP sales office in Bellevue, Washington, for allowing me to run DBPERF on their Series 44.

APPENDIX LOGLIST

LOGLIST is a logfile analysis program written by the author; it has the following capabilities:

A. Show who, what, when, and how a database which was running with transaction logging was accessed.

B. Trace the changes made to the database and expand the values in a format similar to QUERY so that the dump is easily readable.

C. Selectively track user-requested database items which fall within user-specifiable limits.

D. Show when the log was opened, closed, or restarted and identify all users that were accessing the database during a crash!

E. Provide statistics showing the database activity, transaction elapsed time, detail sets accessed, the ratio of BEGIN-ENDS to database transactions, average transaction times, and worst-case transaction response time.

F. F. Identify (if any) the processes which had "broken" transactions.

Running LOGLIST

LOGLIST should be run in the same account and group in which the database resides. If the log to be examined is on disc, then that file must also be accessible. LOGLIST cannot analyze a logtape that is currently active. Finally, the log analysis consumes considerable CPU time (even though the elapsed time of the analysis may be very short). It is advisable the log analysis be either streamed in a low JOBPRIORITY (DS or ES) or run during periods of low computer usage.

LOGLIST Commands

LOGLIST commands are listed below, each followed by a short summary of its function.

HELP — print additional instructions

DATABASE=[dbname[.group[.acct]]]
(if not specified the values are set to @.@.@ and no expansion of the log records may be done. Only the Log User Summary and histograms will be generated.)

PROCESS=[program[.group[.acct]]]
(if not specified the values are set to @.@.@)

LOGON=[user[.group[.acct]]]
(if not specified the values are set to @.@.@)

LIST[=range]
expand the transactions made to the database (in the QUERY report format) showing:
- the user that made the modification
- if an UPDATE, what was changed
- if a DELETE, what was deleted
- if a PUT, what was added

The transactions are outlined in asterisks (*) to indicate indicate "logical transactions." When the beginning or end of a transaction cannot be determined, the program leaves the outlined block open (see Figure 10). On such blocks, the LOGID of the process is printed and it is possible to rerun the analysis — specifying that those items be expanded separately.

RANGE — The range field is optional, and is in the following form:
LIST=startingrecord:endingrecord

If the ending record is not supplied then LISTLOG will continue to expand until the end of the log file.

NOLIST disable expansion of the transactions made to the database

DATE=m1/d1/y1 [TO m2/d2/y2]
look only for transactions made between and including the specified dates. The default for m2/d2/y2 is 99/99/99.

TIME=H1:M1 [TO H2:M2]
look only for transactions made during the specified time interval. The default for H2:M2 is 24:00.

FIND dset.itemname (EQ,LT,GT[,IB]) 'value1'[,'value2']
look only for transactions made to dset.itemname and falling within the bracketed area as specified by the relational operators.

FIND dset record#
look only for transactions made to record# of dset.

```
        { TAPE;LABEL=label           }
LOGFILE={                            }
        {filename[.group[.acct]]     }
```

if a filename is specified, you must have exclusive read-access to the file. If tape is specified, you must be able to use this non-sharable device.

RUN — begin processing the transaction log.

EXIT — exit the program and return to MPE.

SHOW — display current parameters.

INIT — initialize the files, plots and data back to the way they were when LOGLIST first started. Any data accumulated so far will be sent to the LP.

LIMIT — limit and identify the "worst" transactions. This causes all transaction response-time data which exceeds 20 times the current running average to be thrown out. The time of day, user and process are printed on $STDLIST. This command has no effect until ten logical transactions have been completed. It is useful in locating deadlocks.

<CONTROL Y> — ("CNTL" and "Y" keys pressed simultaneously) interrupt the program (sessions only). The program will give the the time and date of the transaction which it is currently processing and ask if you wish to continue. A "Y" or "N" is expected.

Interpretation of the Log User Summary (see Figure 11)

USER — Logon user name

GROUP — Logon user's group

ACCT — Logon user's account

DBASE — Database that was accessed

PROCESS — Process run by user

GROUP — Group in which the process resides

ACCT — Account to which the Process belongs

LOGON TIME — Time the process began

LOGOFF TIME — Time the process closed the database

LG# — LOGID # for the process (assigned by MPE)

DEV — Logical device from which the process was run

O — Database open mode

CAPABILITY — User's capability (see WHO intrinsic of MPE)

UP — Number of DBUPDATES

PUT — Number of DBPUTS

DEL — Number of DBDELETES

#BLKS — Number of complete logical transaction blocks

Inferences from the LOGLIST Statistics

Several histograms and charts are derived from the data; these are provided by LOGLIST to aid in the interpretation of the data.

DATABASE ACTIVITY (see Figure 12)

The DATABASE ACTIVITY histogram plots the number of transactions on the y-axis and the time of day (in 15 minute intervals) on the x-axis. This histogram can be useful in determining when the peak database loads occur.

DATABASE RESPONSE TIME (LOG10) (see Figure 13)
The LOG10 plot is a useful tool in determining if a process or processes are suffering from very bad response time or may be causing database deadlocks. The LOG10 plot covers the range from .1 sec to 10,000 seconds.

DATABASE RESPONSE TIME (LINEAR) (see Figure 14)
The LINEAR plot is a useful in determining if a process or processes are suffering from poor database response times. The y-axis represents the number of transactions made. The x-axis represents the time, from 0 to 30 seconds.

LOGICAL BLOCK SIZE (see Figure 15)
The LOGICAL BLOCK SIZE histogram is useful in evaluating the effectiveness of the transaction blocking of a process. This chart may also be used to determine if a program is calling the DBBEGIN-DBEND pair only at the beginning and end of processes or after making single database modifications.

DATABASE RESPONSE TIME (AVERAGE) (see Figure 16)
The AVERAGE histogram can be useful in evaluating modifications made to existing programs by aiding in the determination of whether or not the system (as seen by the database users) is getting slower or faster.

DATABASE RESPONSE TIME (WORST CASE) (see Figure 17)
The WORST CASE histogram is useful in locating processes that may have caused database deadlocks. The histogram is also useful in determining if there are certain times during the day in which stream jobs may be run with little or no impact on the response time for on-line users.

TRANSACTION FREQUENCY (see Figure 18)
The TRANSACTION FREQUENCY histogram is a measure of the time between logical blocks, often called the user's "think time." This plot, in conjuction with the database response time charts, can be helpful in determining if and/or how improvements can be made to the application programs and the system.

ADD-DELETE-UPDATE TO BEGIN-END RATIO (see Figure 19)
The ratio of DBPUTS, DBDELETES, and DBUPDATES to DBBEGINS and DBENDS is a good indication of how the transactions are blocked by the user's application programs. The desirable range is $0 \leq$ [PUTS + DELETES + UPDATES] / [BEGINS + ENDS] < 100.

If the ratio is less than one, this usually indicates that there is a process or processes which are making only one database transaction per BEGIN-END set. Although this is not harmful, it does not fully utilize the benefits of transaction logging, resulting in more overhead during the logging process and during recovery.

AVERAGE + STANDARD DEVIATION
LOGLIST provides the averages for the response time and block lengths. With the averages and the standard deviations which are also supplied, it is possible to determine your chances of attaining desired response times or block lengths. For instance, the interval covered by the sum of the average plus one standard deviation includes approximately 85% of all data base transactions logged.

DETAIL SET (DATA BASE) SUMMARY
The DETAIL SET summary provides totals based on the actual activity in the sets. As shown in Figure 20, this information includes the number of DBDELETES, DBPUTS, and DBUPDATES. The capacity and number of entries are also printed.

How LOGLIST Works

When processes are using the "USER LOGGING" facility of MPE, the process opens up a path to the transaction log for each process and each database enabled for logging. As part of this "opening" procedure the user's name, acct, process name, capability, LDEV, and database (if one) are logged in a special record. LOGLIST looks for these records and builds its internal working tables from them.

As processes make transactions to their databases, the logging process intercepts a copy of the changes, adds a time and date stamp then routes them to the logging file. LOGLIST uses the time stamp from the DBBEGIN and DBEND records to determine the total elapsed transaction time. (If you don't use DBBEGIN or DBENDS then you can never measure your response times with LOGLIST!)

Broken transactions can be located by looking for a special "ABNORMAL END" record, and by checking to make sure that all process issued a DBEND before closing the log and terminating.

If the process did not (or was unable) to close the log before terminating, and LOGLIST detects an EOF on the log then it is assumed that there has been a system crash. System crashes can also be determined by looking for the crash marker which was written out at the time of a WARMSTART recovery.

Transactions are expanded by using the information gathered when the process first opened up the log, and the actual data- base "change" records. (These records are marked with "DE," "PU" or "UP.") LOGLIST uses the item-list recorded as part of the transaction and calls DBINFO to determine the types and lengths of the individual items logged.

APPENDIX DBPERF

This program was written to benchmark the time required to perform a wide range of DBPUTS, DBDELETES and DBUPDATES. The primary area of interest was the overhead added to IMAGE/3000 when the user is using transaction logging.

The benchmark process follows the procedure listed below:

A. Disable the database XYZ for logging

B. Perform 50 DBOPEN's and DBCLOSE's to measure time to initially startup the logging process. (NOTE: this will really clobber the response time for everybody else.)

C. Perform 50 DBPUTS to a detail set which contains a single path and various data types. The data used for these operations is generated using the RAND function from the compiler library.

D. Perform 50 DBDELETES to the detail set.

E. Setup a loop so that we can perform 50 DBPUTS and DBDELETES on detail sets which contain from 0 to 15 paths.

F. Generate the plots and data summaries.

G. G) Enable database XYZ for logging, then repeat steps B) thru F)

The database modifications are performed without signaling the start of the transactions with DBBEGIN or the end with DBEND. This was done so that the comparison could be made, without the overhead added by the BEGIN-END blocking. This type of test is fair since the DBBEGIN and DBEND calls are made only to signify that that are changes which are dependent.

The time required to perform the BEGIN-END block is measured and plotted on a separate chart. It should be noted that since DBBEGIN and DBEND do not require immediate access to the disc drives, the time required to perform these intrinsics is very low. The can however add a significant number of records to the memory buffer, which of course means that there is an additionaly load on the I/O channel which controls the disc drives.

REFERENCES

[1]F. Alfredo Rego, "DATABASE THERAPY: A practitioner's experiences," in HPGSUG 1981 Orlando Florida Proceedings, Vol 1, pp. B12-01 to B12-13

[2]P. Sinclair, "MPE 1918: A BONANZA OF ENHANCEMENTS," in COMMUNICATOR issue 23, pp. 4-17

[3]HP, "MPE III 1918 USER UPDATE COURSE"

[4]HP, "MPE III Intrinsics Reference Manual," pp. 3-92 to 3-96

[5]HP, "IMAGE Data Base Management System reference manual," pp. 4-22 to 4-23

[6]HP, "IMAGE Data Base Management System reference manual," pp. 4-23

[7]P. Sinclair, "MPE 1918: A BONANZA OF ENHANCEMENTS," in COMMUNICATOR issue 23, pp. 14

[8]HP, "MPE III 1918 USER UPDATE COURSE," pp. 60

[9]Robert M. Green, Robelle Consulting Ltd., 5421 10th Avenue, Suite 130, Delta, British Columbia V4M 3T,. Canada.

[10]HP, "MPE III 1918 USER UPDATE COURSE," pp. 71

[11]A. O. Allen, "Queueing Models of Computer Systems," in COMPUTER, pp. 13-24, Apr. 1980 (an IEEE publication)

[12]C. Storla, "MEASURING TRANSACTION RESPONSE TIMES," in 1981 IUG Orlando Florida Proceedings, Vol. 1, pp. C7-01 to C7-08

```
========
= user =
=program
=      ==
======== =
          =
           =    =====================  =======  =====
========    =   =             = T   =  =     =  =   =
= user =     =  =             = r   =  = 8   =  =   =
=program      = =    IMAGE    = a L =  = K B =  =   =
=      ==========             = n o =  =   u =  = D =     ====
========      = =   data base = s g =  = M f =  = i =   =     ===
             =  =             = a g ===> e f ===> s ==>= Tape   =
            =   =  management = c i =  = m e =  = s =  = ( if   =
========   =    =             = t n =  = o r =  =   =   =used)=
= user =  =     =     system  = i g =  = r   =  =   =     ====
=program =      =             = o   =  = y   =  =   =
=      ==       =             = n   =  =     =  =   =
========        =====================  =======  =====
                        =
                        =
                       \/
                    ==========
                    =  disc  =
                    =  drive =
                    ==========
```

Figure 1. IMAGE transaction logging flow

```
CALL DBBEGIN(BASE,...   )
     CALL DBLOCK(BASE,...  )

     CALL DBDELETE(BASE,... )

               o                       At this point,
               o                 if there is a crash we lose
               o                 this data entry!
               o

      change made to search item

               o
               o
               o

     CALL DBPUT(BASE,... )

               o              this item has now been re-added

     CALL DBUNLOCK(BASE,... )
CALL DBEND(BASE,...   )
```

Figure 2. Dependent Changes

```
CALL DBLOCK(... )
     CALL DBFIND(... )
     CALL DBGET(... )

     ** MAKE CHANGES TO ITEM VALUES HERE **

     CALL DBBEGIN(... )

               DBPUT
          CALL {DBUPDATE } (... )
               DBDELETE

     CALL DBEND(... )
CALL DBUNLOCK(... )
```

Figure 3A

```
CALL DBFIND(... )
CALL DBGET(... )

** MAKE CHANGES TO ITEM VALUES HERE **

     CALL DBBEGIN(... )
     CALL DBLOCK(... )

              DBPUT
          CALL{DBUPDATE}(... )
              DBDELETE

     CALL DBUNLOCK(... )
     CALL DBEND(... )
```

Figure 3B

```
CALL BEGINLOG(... )


    CALL DBFIND(BASE1,... )
    CALL DBGET(BASE1,... )

    ** MAKE CHANGES TO ITEM VALUES HERE **

         CALL DBBEGIN(BASE1,... )
         CALL DBLOCK(BASE1,... )

                   DBPUT
              CALL{DBUPDATE}(BASE1,... )
                   DBDELETE

         CALL DBUNLOCK(BASE1,... )
         CALL DBEND(BASE1,... )

    CALL DBFIND(BASE2,... )
    CALL DBGET(BASE2,... )

    ** MAKE CHANGES TO ITEM VALUES HERE **

         CALL DBBEGIN(BASE2,... )
         CALL DBLOCK(BASE2,... )

                   DBPUT
              CALL{DBUPDATE}(BASE2,... )
                   DBDELETE

         CALL DBUNLOCK(BASE2,... )
         CALL DBEND(BASE2,... )


CALL ENDLOG(... )
```

Figure 4

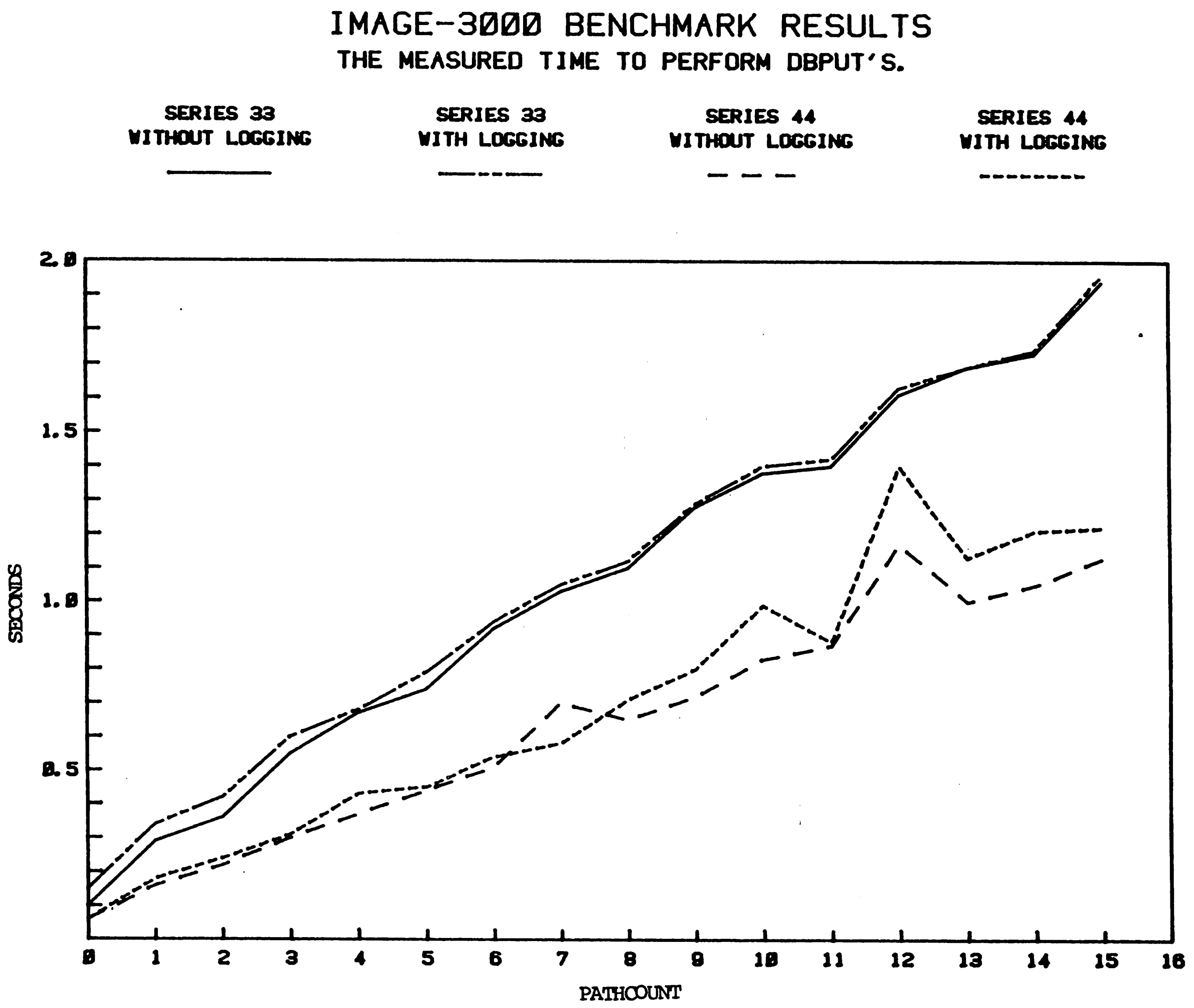


IMAGE-3000 BENCHMARK RESULTS
THE MEASURED TIME TO PERFORM DBPUT'S.
SERIES 33 WITHOUT LOGGING
SERIES 33 WITH LOGGING
SERIES 44 WITHOUT LOGGING
SERIES 44 WITH LOGGING
2.0
1.5
1.0
0.5
SECONDS
0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
PATHCOUNT

Figure 5

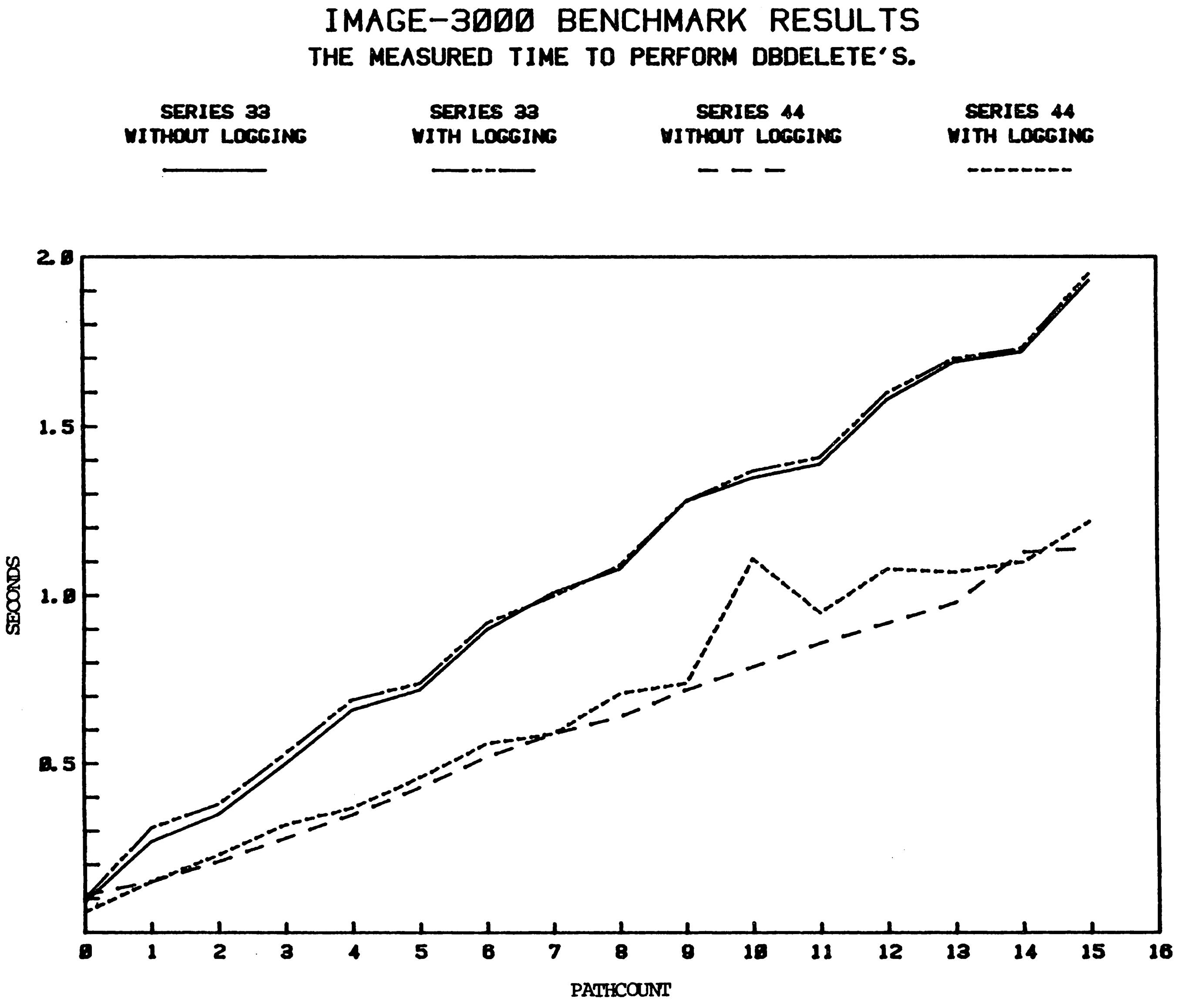


IMAGE-3000 BENCHMARK RESULTS
THE MEASURED TIME TO PERFORM DBDELETE'S.
SERIES 33 WITHOUT LOGGING
SERIES 33 WITH LOGGING
SERIES 44 WITHOUT LOGGING
SERIES 44 WITH LOGGING
SECONDS
2.0
1.5
1.0
0.5
PATHCOUNT
0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16

Figure 6

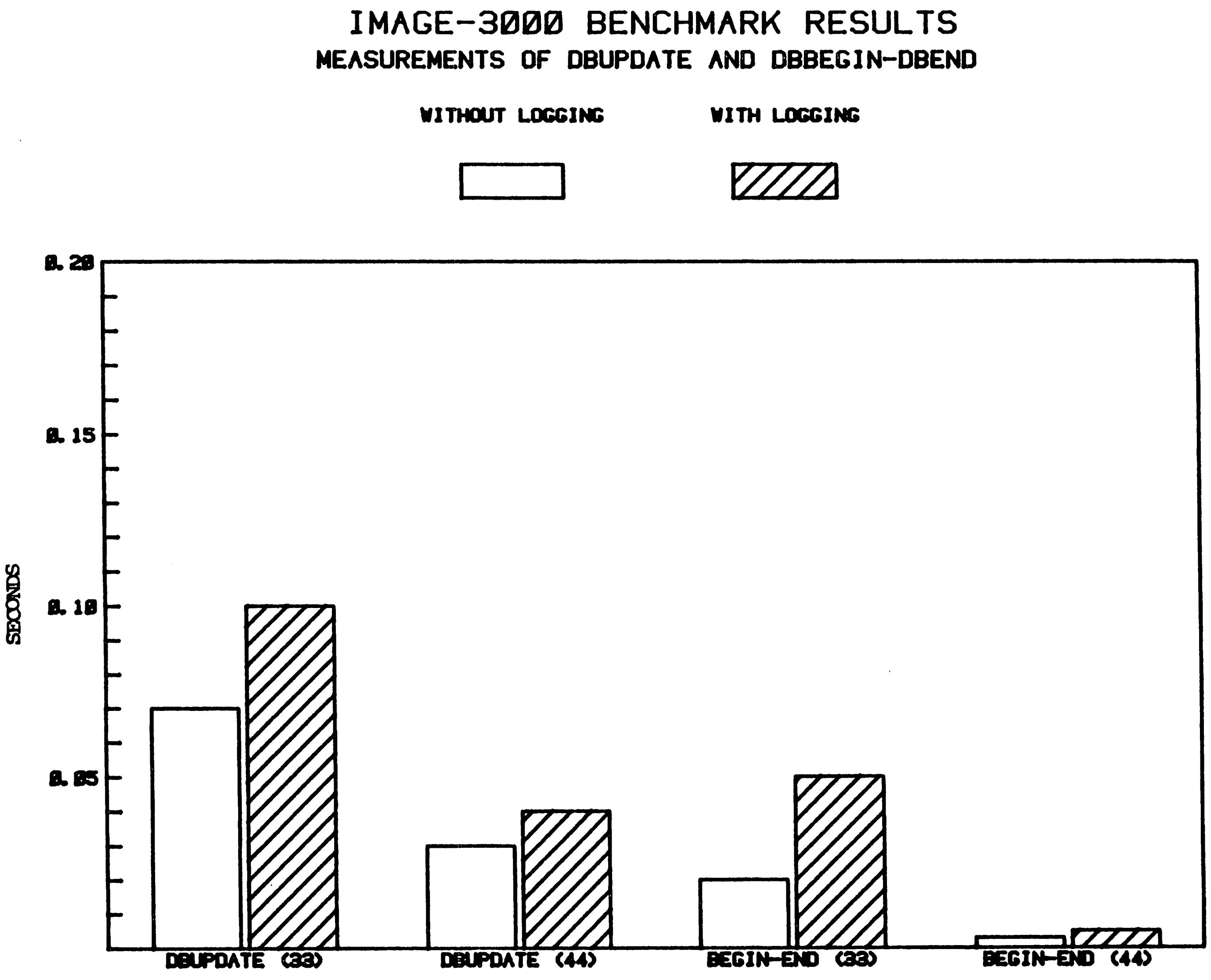


IMAGE-3000 BENCHMARK RESULTS
MEASUREMENTS OF DBUPDATE AND DBBEGIN-DBEND
WITHOUT LOGGING
WITH LOGGING
0.20
0.15
0.10
0.05
SECONDS
DBUPDATE (33)
DBUPDATE (44)
BEGIN-END (33)
BEGIN-END (44)

Figure 7

```
CALL DBBEGIN(BASE,...    )
     CALL DBLOCK(BASE,...   )

     CALL DBDELETE(BASE,... ) <structural damage,if crash occurrs>
                              < for a detail set with 5 paths >
                              < the 'critical' time could be  >
                              < a half second or more!        >

               o                    At this point,
               o              if there is a crash we lose
               o              this data entry!
               o

      change made to search item

               o
               o
               o

     CALL DBPUT(BASE,... ) <structural damage, if crash occurrs>
                              < for a detail set with 5 paths >
                              < the 'critical' time could be  >
                              < a half second or more!        >

               o              The item has now be re-added

     CALL DBUNLOCK(BASE,... )
CALL DBEND(BASE,...    )
```

Figure 8. Crash Modes

```
           Benefits of Logging   VS   Cost of Logging


AUDIT TRAIL                          REDUCED THROUGHPUT?
 Who, What, When and How              Dependent on your
 Ability to list Sets and             application, and system
 fields which are modified.           load.

RECOVERABLE DATA                     COST OF ADDITIONAL MEMORY?
 Ability to recover                   May need more memory
 most if not all                      to maintain current
 transactions, upto the                 system throughput.
 time of the crash.

PERFORMANCE INFO                     TAPE DRIVE OR DISC
 Information available               DEDICATED TO LOGGING?
 which can lead to better             Valuable disc space
 application designs                  or tape drive can be
 in the future.                       tied up with logging.

LOGGING OF ALL CHANGES               STARTUP AFTER CRASH
MADE TO DATA BASE                    MORE COMPLICATED
REGARDLESS OF PROGRAM
 You can use any vendor               Training and "test
 software and still                   recoverys" may be
 maintain an "audit trail".           required to familiarize
                                      the programmers and
                                      operators with the
                                      new system restart
HP  SUPPORT OF LOGGING                procedures.
==============================================================
                            ^
                           / \
                          /   \
                         /     \
                        =========
```

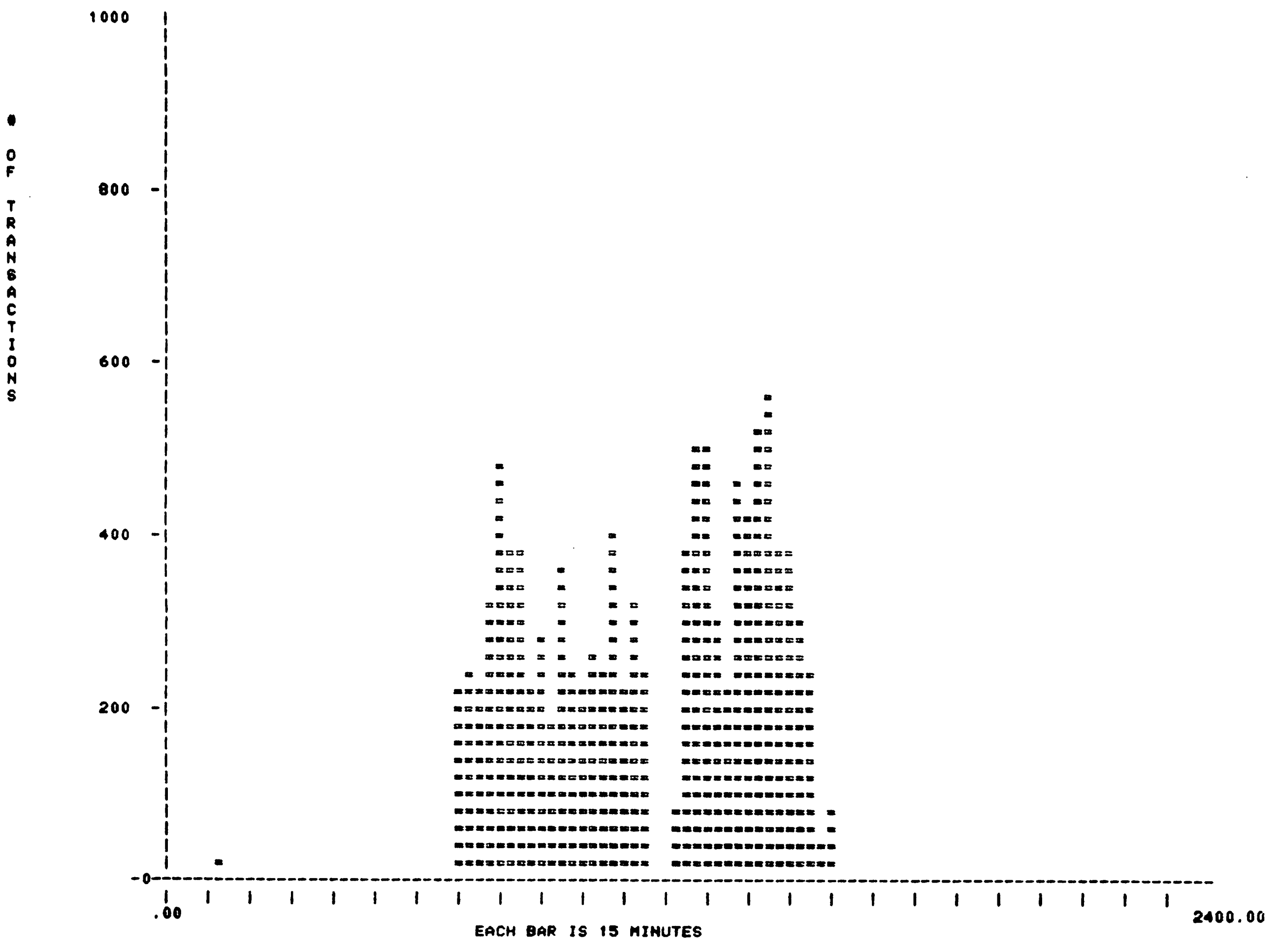
Figure 9

```
*******************************************************************************************************
*    OFFICE  BAC     TEIMS          ACCTPROGBAC     TEIMS          TUE, DEC 29, 1981,  8:58 PM         *
* 1                  ZaE@CE                                                                    *
*                                                                                                     *
*        LOGID#:   1  TRANSACTION# -     4   DELETEING ITEM IN     EQUIP-DETL        DBFILE RECORD#: 23277
*    PROP#           [U12  ] = 1                                                                      *
*    MODELCOD        [U14  ] = 0003FIDDLE                                                             *
*    NOMNCODE        [I1   ] = 3                                                                      *
*    EQUIPLOC        [I1   ] = 10                                                                     *
*    PROGTAG         [I1   ] = 0                                                                      *
*    CURRUSER        [U10  ] = 0000000100                                                             *
*    NEXTUSE1        [U10  ] = NONE                                                                   *
*    PO-SERIES       [U10  ] = NONE                                                                   *
*    HOLDFOR         [U10  ] = NONE                                                                   *
*    BORROWER        [U10  ] = NONE                                                                   *
*    FROMUSER        [U10  ] = NONE                                                                   *
*    MFG             [I1   ] = 0                                                                      *
*    INVTORYTAG      [I1   ] = 0                                                                      *
*    CALDATE         [I2   ] = 811225                                                                 *
*    SERIAL#         [U10  ] = SERIAL                                                                 *
*    ACQUCOST        [I2   ] = 11                                                                     *
*    CAPEXPEN        [U2   ] = E                                                                      *
*    NEWUSED         [U2   ] = N                                                                      *
*    ISSUEDAT        [I2   ] = 811220                                                                 *
*    AOCODE          [U2   ] = AC                                                                     *
*    ACQUDATE        [I2   ] = 811115                                                                 *
*    NEXTSTD1        [I2   ] = 0                                                                      *
*    NEXTSTD2        [I2   ] = 0                                                                      *
*    HOLDATE         [I2   ] = 0                                                                      *
*    NEXTEND1        [I2   ] = 0                                                                      *
*    NEXTEND2        [I2   ] = 0                                                                      *
*    STARTCYCLE      [I2   ] = 0                                                                      *
*    RETNDATE        [I2   ] = 0                                                                      *
*    BORROWDT        [I2   ] = 0                                                                      *
*    MESSTAG1        [I1   ] = 0                                                                      *
*    MESSTAG2        [I1   ] = 0                                                                      *
*    MESSTAG3        [I1   ] = 0                                                                      *
*    ACCESTAG        [I1   ] = 0                                                                      *
*    OPTINTAG        [X2   ] = AA                                                                     *
*    SPECCODE        [I1   ] = 0                                                                      *
*    SAMPUTIL        [U2   ] = YE                                                                     *
*    LASTUSER        [U10  ] = 5176822150                                                             *
*        LOGID#:   1  TRANSACTION# -     4   ADDING ITEM TO        EQUIP-DETL        DBFILE RECORD#: 23277
*    PROP#           [U12  ] = 1                                                                      *
*    MODELCOD        [U14  ] = 0003FIDDLE                                                             *
*    NOMNCODE        [I1   ] = 3                                                                      *
*    EQUIPLOC        [I1   ] = 10                                                                     *
*    PROGTAG         [I1   ] = 0                                                                      *
*    CURRUSER        [U10  ] = 5172870970                                                             *
*    NEXTUSE1        [U10  ] = NONE                                                                   *
*    PO-SERIES       [U10  ] = NONE                                                                   *
*    HOLDFOR         [U10  ] = NONE                                                                   *
*    BORROWER        [U10  ] = NONE                                                                   *
*    FROMUSER        [U10  ] = NONE                                                                   *
*    MFG             [I1   ] = 0                                                                      *
*    INVTORYTAG      [I1   ] = 30003                                                                  *
*    CALDATE         [I2   ] = 811225                                                                 *
*    SERIAL#         [U10  ] = SERIAL                                                                 *
*    ACQUCOST        [I2   ] = 11                                                                     *
*    CAPEXPEN        [U2   ] = E                                                                      *
*-   NEWUSED         [U2   ] = N                                                                      *
*    ISSUEDAT        [I2   ] = 811220                                                                 *
*    AOCODE          [U2   ] = AC                                                                     *
*    ACQUDATE        [I2   ] = 811115                                                                 *
*    NEXTSTD1        [I2   ] = 0                                                                      *
*    NEXTSTD2        [I2   ] = 0                                                                      *
*    HOLDATE         [I2   ] = 0                                                                      *
*    NEXTEND1        [I2   ] = 0                                                                      *
*    NEXTEND2        [I2   ] = 0                                                                      *
*    STARTCYCLE      [I2   ] = 0                                                                      *
*    RETNDATE        [I2   ] = 0                                                                      *
*    BORROWDT        [I2   ] = 0                                                                      *
*    MESSTAG1        [I1   ] = 0                                                                      *
*    MESSTAG2        [I1   ] = 0                                                                      *
*    MESSTAG3        [I1   ] = 0                                                                      *
*    ACCESTAG        [I1   ] = 0                                                                      *
*    OPTINTAG        [X2   ] = AA                                                                     *
*    SPECCODE        [I1   ] = 0                                                                      *
*    SAMPUTIL        [U2   ] = YE                                                                     *
*    LASTUSER        [U10  ] = 5176822150                                                             *
*                    8:58 PM                                                                   *
*******************************************************************************************************

*******************************************************************************************************
*    OFFICE  BAC     TEIMS          ACCTPROGBAC     TEIMS          TUE, DEC 29, 1981,  8:59 PM         *
*                                                                                                     *
*                                                                                                     *
*        LOGID#:   1  TRANSACTION# -     5   UPDATING ITEM IN      EQUIP-DETL        DBFILE RECORD#: 23277
*    NEW VALUES:                                                                                      *
*    CALDATE         [I2   ] = 0                                                                      *
*    AOCODE          [U2   ] = NA                                                                     *
*                                                                                                     *
*    OLD VALUES:                                                                                      *
*    CALDATE         [I2   ] = 811225                                                                 *
*    AOCODE          [U2   ] = AC                                                                     *
*                    8:59 PM                                                                   *
*******************************************************************************************************

         LOGID#:   1  TRANSACTION# -     1   UPDATING ITEM IN      EQUIP-DETL        DBFILE RECORD#: 23277
     NEW VALUES:
     MFG             [I1   ] = 3

     OLD VALUES:
     MFG             [I1   ] = 0
```

Figure 10

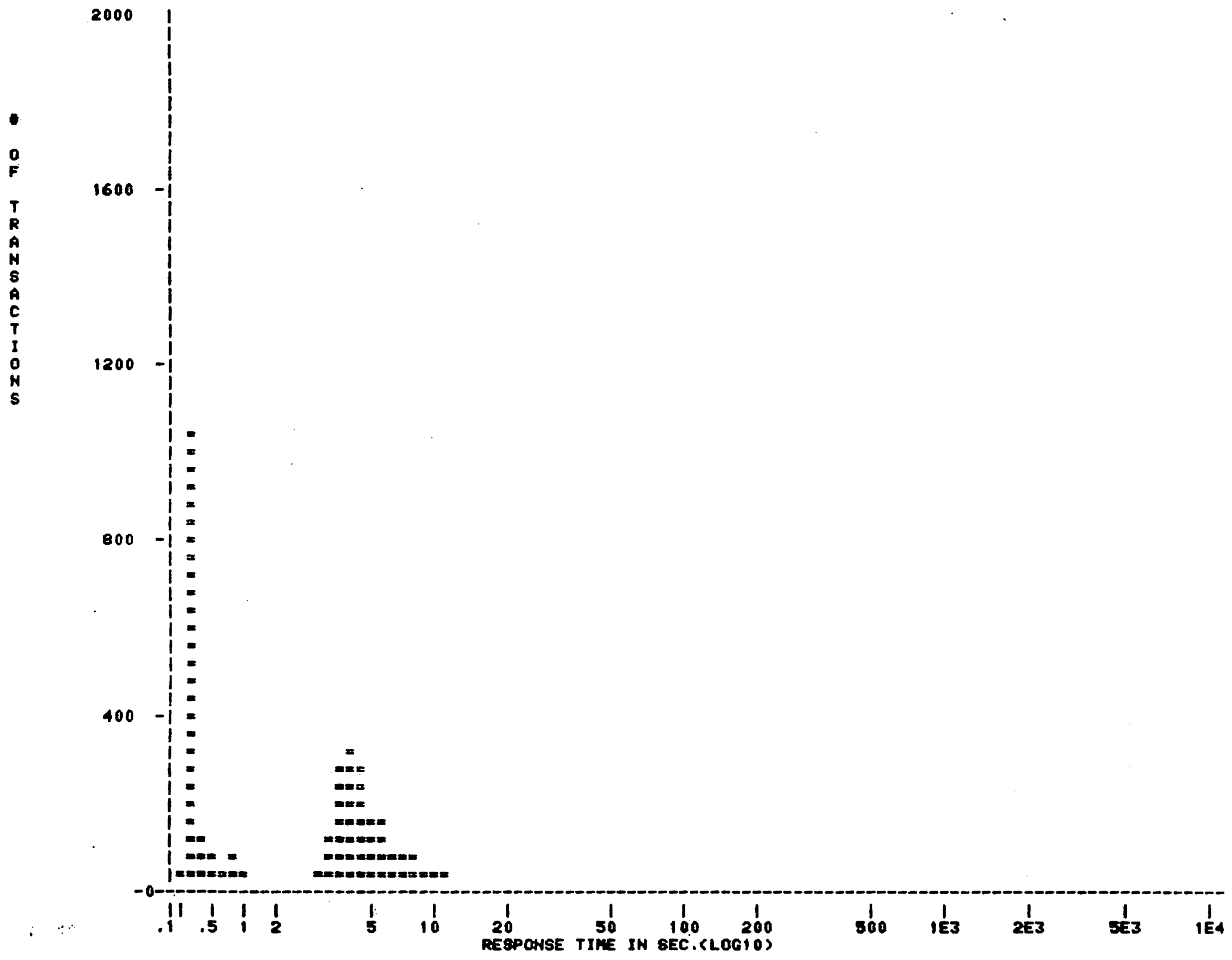
```
  USER    GROUP   ACCT  DBASE PROCESS  GROUP   ACCT        LOGON TIME        LOGOFF TIME  LG# DEV O CAPABILITY   UP  PUT  DEL #BLKS
--------------------------------------------------------------------------------------------------------------------------------------

OFFICE  BAC     TEIMS   TEIM1 QUERY   PUB     SYS     WED,NOV 25,1981, 1:55P NOV 25, 2:02P 163 25 1 4020300611    1    0    0     0
BML     BAC     TEIMS   TEIM1 HAPROG  BAC     TEIMS   WED,NOV 25,1981, 1:50P NOV 25, 2:04P 162 37 1 0060300601    9    3    3     3

        WED, NOV 25, 1981,  2:13 PM  *** PROCESS ABORTED ***  PLTII   BAC     TEIMS
PLTII   BAC     TEIMS   TEIM1 HAPROG  BAC     TEIMS   WED,NOV 25,1981, 2:05P NOV 25, 2:13P 164 36 1 0020300611    3    1    1    1?
OFFICE  BAC     TEIMS   TEIM1 QUERY   PUB     SYS     WED,NOV 25,1981, 2:07P NOV 25, 2:14P 165 25 1 4020300611    1    0    0     0
DC      BAC     TEIMS   TEIM1 HAPROG  BAC     TEIMS   WED,NOV 25,1981, 8:01A NOV 25, 2:17P 132 34 1 0020300611   98   50   38    38
BAC     BAC     TEIMS   TEIM1 ACCTST  BAC     TEIMS   WED,NOV 25,1981, 2:19P NOV 25, 2:20P 169 26 1 5360300613    0    0    0     0
DC      BAC     TEIMS   TEIM1 HAPROG  BAC     TEIMS   WED,NOV 25,1981, 6:55A NOV 25, 2:36P 127 39 1 0020300611  114   54   42    42
PRIMARY BAC     TEIMS   TEIM1 QAPROG  BAC     TEIMS   WED,NOV 25,1981, 2:09P NOV 25, 2:38P 166 37 1 0020300611   18    6    6     6
PRIMARY BAC     TEIMS   TOOLS QAPROG  BAC     TEIMS   WED,NOV 25,1981, 2:09P NOV 25, 2:38P 167 37 1 0020300611    0    0    0    0*
QA      BAC     TEIMS   TEIM1 QAPROG  BAC     TEIMS   WED,NOV 25,1981, 6:32A NOV 25, 2:52P 122 32 1 0020300611  251   99   99    99
QA      BAC     TEIMS   TOOLS QAPROG  BAC     TEIMS   WED,NOV 25,1981, 6:32A NOV 25, 2:52P 123 32 1 0020300611    0    0    0    0*
QA      BAC     TEIMS   TEIM1 QAPROG  BAC     TEIMS   WED,NOV 25,1981, 2:53P NOV 25, 2:56P 172 32 1 0020300611    0    0    0     0
QA      BAC     TEIMS   TOOLS QAPROG  BAC     TEIMS   WED,NOV 25,1981, 2:53P NOV 25, 2:56P 173 32 1 0020300611    0    0    0    0*
QA      BAC     TEIMS   TEIM1 QAPROG  BAC     TEIMS   WED,NOV 25,1981, 2:58P NOV 25, 2:59P 174 32 1 0020300611    0    0    0     0
QA      BAC     TEIMS   TOOLS QAPROG  BAC     TEIMS   WED,NOV 25,1981, 2:58P NOV 25, 2:59P 175 32 1 0020300611    0    0    0    0*
BAC     BAC     TEIMS   TEIM1 QUERY   PUB     SYS     WED,NOV 25,1981, 2:43P NOV 25, 3:04P 171 26 1 5360300613    0    2    1     0
PLTII   BAC     TEIMS   TEIM1 HAPROG  BAC     TEIMS   WED,NOV 25,1981, 2:16P NOV 25, 3:05P 168 36 1 0020300611   49   23   18    18
KENT    BAC     TEIMS   TEIM1 HAPROG  BAC     TEIMS   WED,NOV 25,1981, 6:39A NOV 25, 3:08P 124 33 1 0020300611  220   97   78    83
DC      BAC     TEIMS   TEIM1 HAPROG  BAC     TEIMS   WED,NOV 25,1981, 2:37P NOV 25, 3:20P 170 34 1 0020300611   16    8    6     6
KENT    BAC     TEIMS   TEIM1 HAPROG  BAC     TEIMS   WED,NOV 25,1981, 6:26A NOV 25, 3:24P 120 31 1 0020300611  185   99   71    71
KENT    BAC     TEIMS   TEIM1 HAPROG  BAC     TEIMS   WED,NOV 25,1981, 3:09P NOV 25, 3:26P 177 32 1 0020300611   12    8    5     5
BAC     BAC     TEIMS   TEIM1 ACCTPROGBAC     TEIMS   WED,NOV 25,1981, 3:04P NOV 25, 3:39P 176 26 1 5360300613    4    1    1     4

        WED, NOV 25, 1981,  3.44 PM  *** PROCESS ABORTED ***  TEKSTAFFBAC     TEIMS
TEKSTAFFBAC     TEIMS   TEIM1 TECHPROGBAC     TEIMS   WED,NOV 25,1981,12:58P NOV 25, 3:44P 159 21 1 0020300611  109   20    1   98?
OFFICE  BAC     TEIMS   TEIM1 ACCTPROGBAC     TEIMS   WED,NOV 25,1981,12.13P NOV 25, 3:47P 152 25 1 4020300611  136   76   73   212
BAC     BAC     TEIMS   TEIM1 ACCTST  BAC     TEIMS   WED,NOV 25,1981, 3:55P NOV 25, 4:00P 178 26 1 5360300613    0    0    0     0
BAC     BAC     TEIMS   TEIM1 HAPROG  BAC     TEIMS   WED,NOV 25,1981, 4:27P NOV 25, 4:45P 179 31 1 5360300613    0    0    0     0
                     * - INDICATES PROCESS DID NOT QUALIFY IN SELECTIVE SEARCH    ? - BROKEN TRANSACTION
```

Figure 11



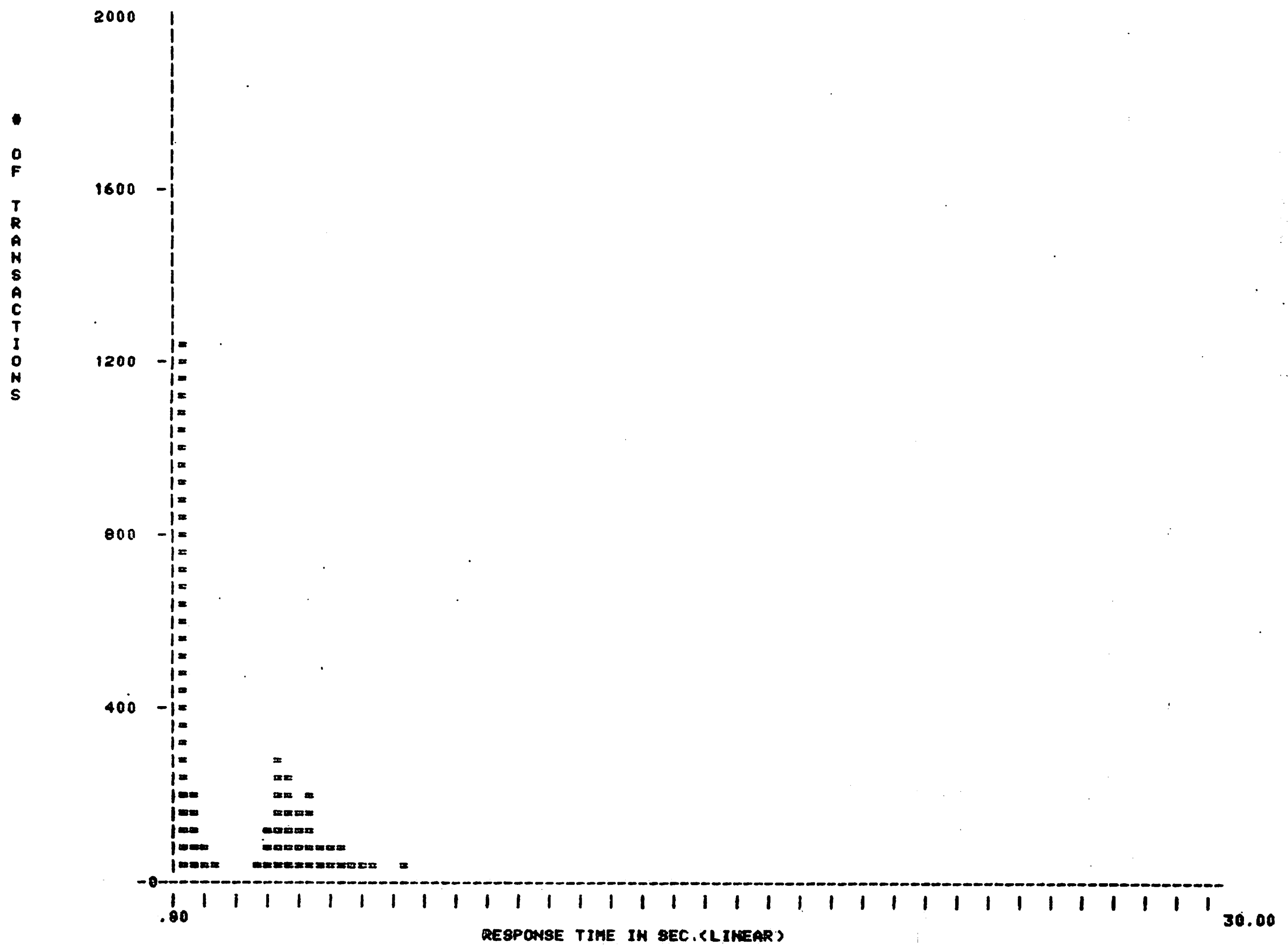
```
MAXIMUM VALUE:   560.0 MIN IS :    .0 SCALE FACTOR:   20.0 AVG :   111.86 Y AXIS MAX: 1000 TOTAL # IN ALL CELLS: 11186
```

Figure 12



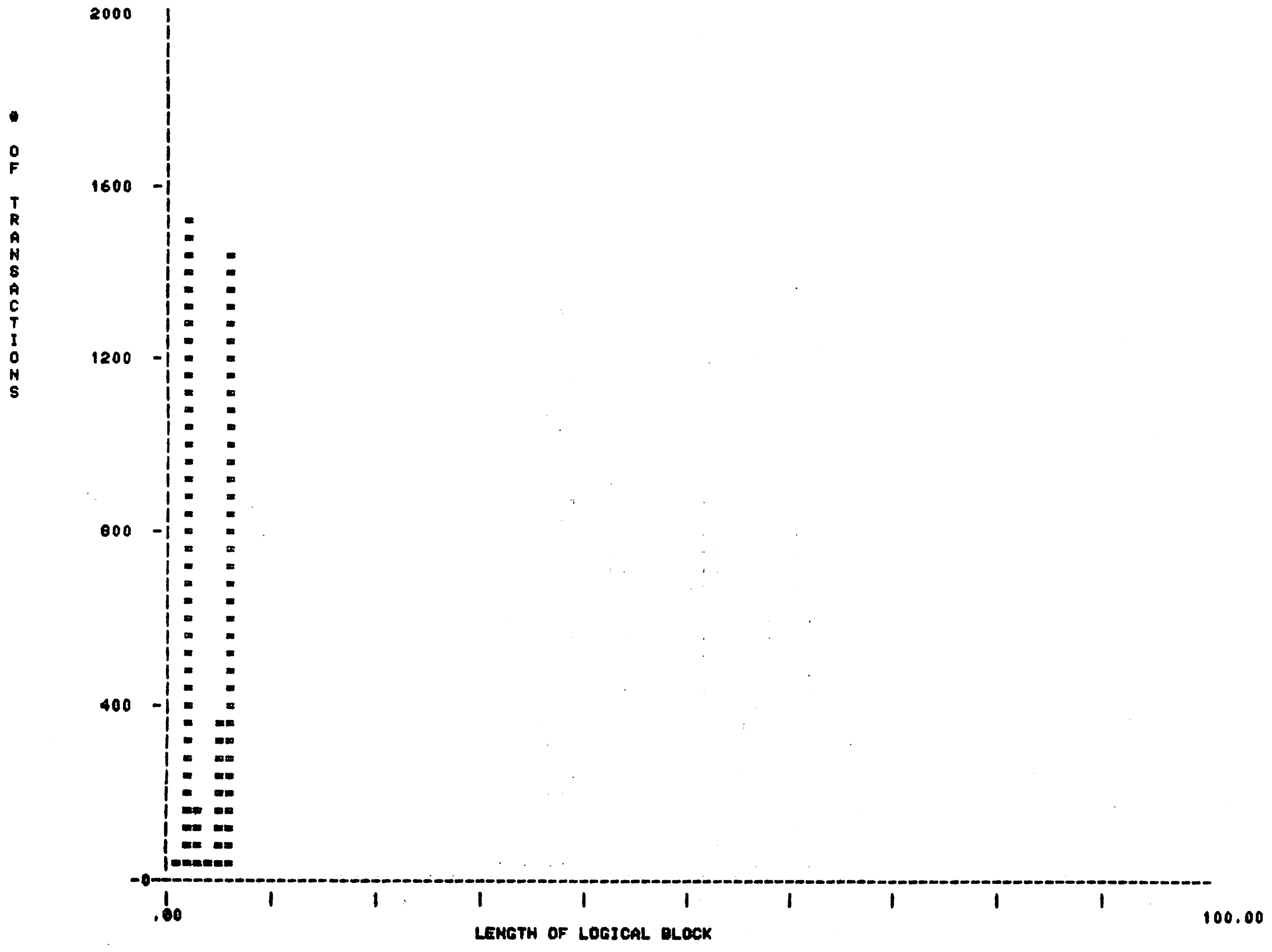
```
MAXIMUM VALUE:  1066.0 MIN IS :      .0 SCALE FACTOR:   40.0 AVG :    37.06 Y AXIS MAX: 2000 TOTAL # IN ALL CELLS:  3706
```

Figure 13



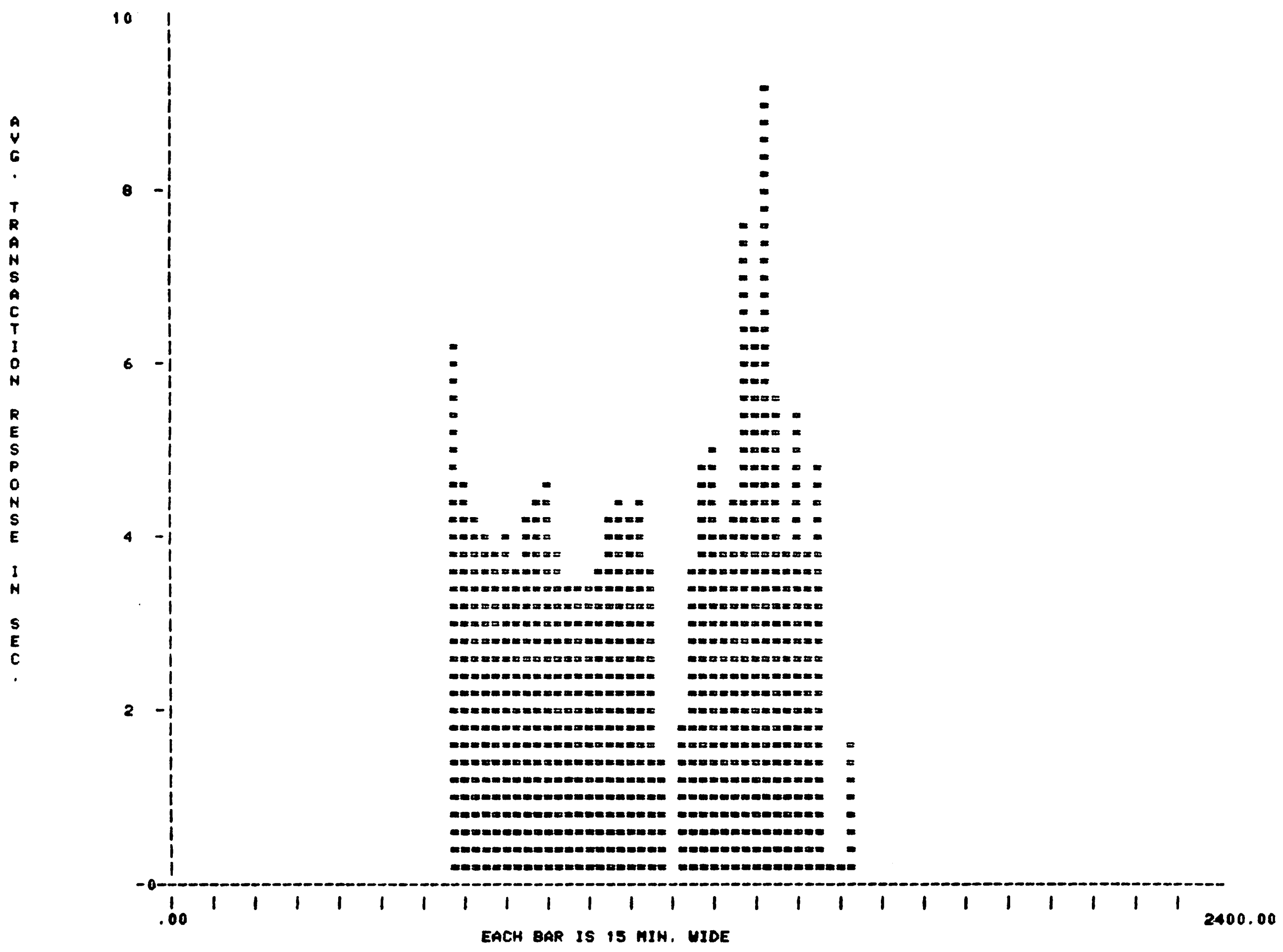
MAXIMUM VALUE: 1272.0 MIN IS : .0 SCALE FACTOR: 40.0 AVG : 37.06 Y AXIS MAX: 2000 TOTAL # IN ALL CELLS: 3706

Figure 14



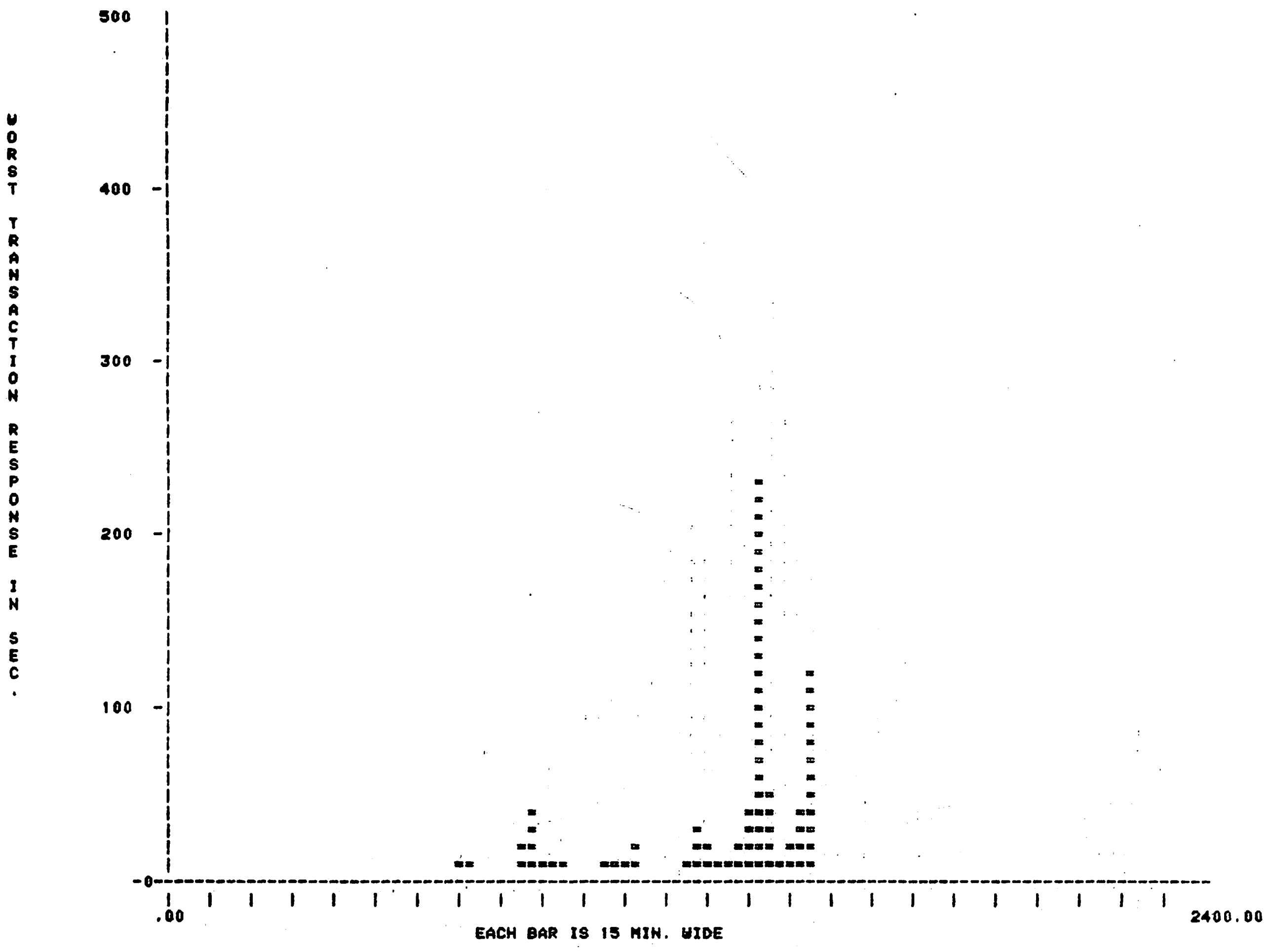
MAXIMUM VALUE: 1538.0 MIN IS : 0 SCALE FACTOR: 40.0 AVG : 37.06 Y AXIS MAX: 2000 TOTAL # IN ALL CELLS: 3706

Figure 15



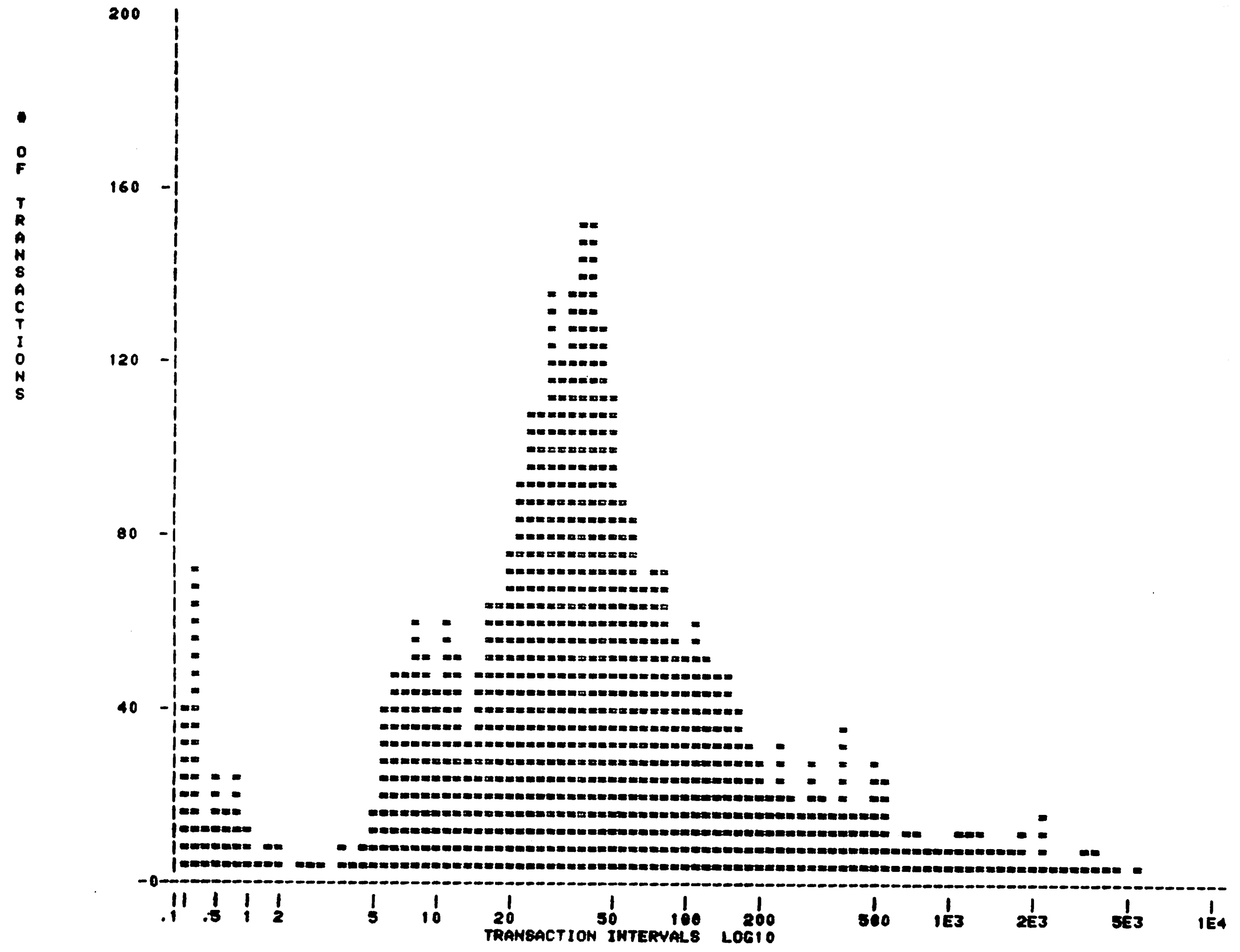
```
MAXIMUM VALUE:     9.4 MIN IS :     .0 SCALE FACTOR:     .2 AVG :     1.60 Y AXIS MAX:   10 TOTAL # IN ALL CELLS:   160
```

Figure 16



```
MAXIMUM VALUE:   235.4 MIN IS :      .0 SCALE FACTOR:    10.0 AVG :      9.87 Y AXIS MAX:  500 TOTAL # IN ALL CELLS:    987
```

Figure 17



MAXIMUM VALUE: 155.0 MIN IS : .0 SCALE FACTOR: 4.0 AVG : 36.25 Y AXIS MAX: 200 TOTAL # IN ALL CELLS: 3625

Figure 18

```
*****************************************************************
*     NUMBER OF RECORDS PROCESS         22112                   *
*     PUTS, DELETES, UPDATES            11186                   *
*     DBBEGINS & DBENDS                  7415                   *
*     AVERAGE TRANSACTION TIME           1.60                   *
*                STD DEVIATION           4.57                   *
*                                                               *
*     AVERAGE TRANSACTION INTERVAL     182.04                   *
*            STD DEVIATION             645.68                   *
*                                                               *
*     AVERAGE BLOCK LENGTH               1.49                   *
*            STD DEVIATION               2.08                   *
*     # OF LOGICAL BLOCKS                3625                   *
*****************************************************************
```

Figure 19

```
DATA-SET(BASE)         #UPDATES  #DELETES  #PUTS    CAPACITY  ENTRIES  PERCENT FULL
------------------------------------------------------------------------------------

CROSSREFJ05                   0         8       7     50036     41355        82.65
OPTION-DETL                   0         0       1       987       369        37.39
SERV-DETL                   210       120     129     30984     21705        70.05
UTIL-DETL                   759        45     532     30990      7443        24.02
EQUIP-DETL                  704      2015    2021     34986     27363        78.21
NOMCL-DETL                  399         1      28      5004      3970        79.34
USER-DETL                  3732         0       0      1704      1140        66.90
NOMH-DETL                    12         1      13      1512       697        46.10
SPEC-DETL                   229         5      48     34986     16389        46.84
USEWITH-XREF                  0        10     106      1014       101         9.96
WHAREHOUSELOC                 0        24      27      5031      2518        50.05
```

Figure 20