The Enhancement of
Datamodule I

TECHNICAL REPORT I

APPROVED FOR PUBLIC RELEASE
DISTRIBUTION UNLIMITED

The Enhancement of Datamodule I

TECHNICAL REPORT I

August 1, 1978 to October 15, 1978

Computer Corporation of America
575 Technology Square
Cambridge, Massachusetts 02139

This research was supported by the Defense Advanced Research Project Agency of the Department of Defense and was monitored by the Naval Electronic System Command under Contract No. N00039-78-C-0443, ARPA Order No. 3175, Ammendment 20. The views and conclusions contained in this document are those of the authors and should not be interpreted as necessarily representing the official policies, either expressed or implied, or the Defense Advanced Research Projects Agency or the U.S. Government.

TABLE OF CONTENTS

1. Introduction

This report summarizes the first two and one half months of a project entitled "The Enhancement of Datamodule 1". The focus of this project is to study the Datacomputer's performance in its role as a datamodule in SDD-1 [ROTHNIE and GOODMAN] and as a DBMS in other command and control applications. The goal of this study is to produce a set of potential performance enhancements and an analysis of their expected effect on overall Datacomputer performance.

During the reporting period, a model of Datacomputer performance was constructed and measurements of current system performance were made. The results of this initial work have already indicated that parts of the system are bottlenecks and this, in turn, is suggesting potential enhancements to be analyzed further along in the study.

2. Study Procedure

The study procedure we are using is a formal analytic process aimed at identifying request processing bottlenecks, suggesting techniques to relieve these bottlenecks, and evaluating the cost-effectiveness of the suggested improvement techniques. The study procedure steps being followed are:

1. Model building -- determining a simple picture of the way the Datacomputer performs.

2. Requirements Analysis -- determining the performance requirements of the command and control community.

3. Measurement -- calibrating the model with the performance parameters of the current Datacomputer.

4. Sensitivity Analysis -- determining where the system bottlenecks are using the calibrated model and the requirements analysis data.

5. Enhancement option generation -- proposing candidate actions for improving performance.

6. Enhancement option analysis -- determining the effects on the model of adopting each performance enhancement option.

7. Evaluation and recommendation -- recommending a specific set of actions to be taken based on the cost, flexibility, and effectiveness of each option.

During the reporting period, progress has been made in the first four steps of the study procedure.

1. A model of the Datacomputer which represents the delay incurred by different components of the Datacomputer under various classes of query traffic and various system loads has been constructed.

2. Interactions have taken place between CCA and NOSC as to Datacomputer performance requirements and typical command and control database traffic.

3. A version of the Datacomputer has been modified to produce extensive performance data during request processing. Performance measurements have been made using this data.

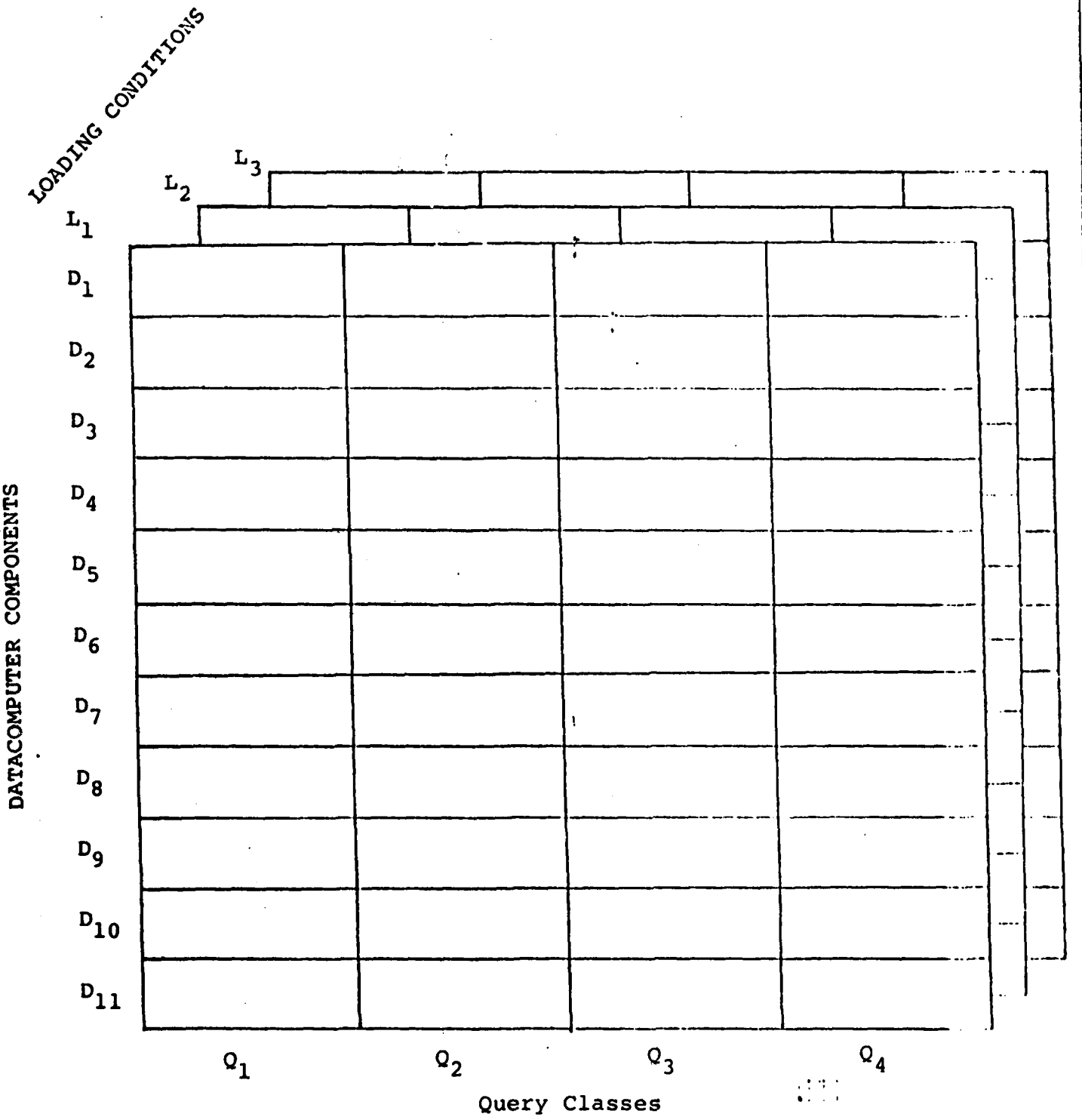
4. The data from the measurements have been used to populate the model and begin to discover where the system bottlenecks lie.

3. Model Definition

A Datacomputer performance model which represents the delay incurred by each component of the system for each of several types of queries and under each of several system loads has been defined. This model can be viewed as a 3-dimensional matrix of the sort pictured in figure 3.1. The first dimension represents major processing components of the Datacomputer, the second dimension corresponds to different classes of queries, and the third dimension represents different system loading conditions. An initial version of this model has been constructed.

Datacomputer Model

Figure 3.1



3.1 Processing Components

The first axis to parameterize is the Datacomputer processing component axis. The components to be represented must be meaningful in terms of the resources used and the part they play in request processing. In order to make this selection, an understanding of the processing steps invoked by the Datacomputer while handling a request is required. The next few sections describe these steps.

3.1.1 Overview of Datacomputer Processing Steps

In any database management system, a request must go through a language evaluation phase followed by execution of the database operations requested. The Datacomputer's data management functions are invoked by requests in a high level language called Datalanguage. Datalanguage has facilities for storing, retrieving and updating data. Data is stored by the Datacomputer in files whose descriptions are maintained in a system directory. Data is transmitted to or from the Datacomputer through ports

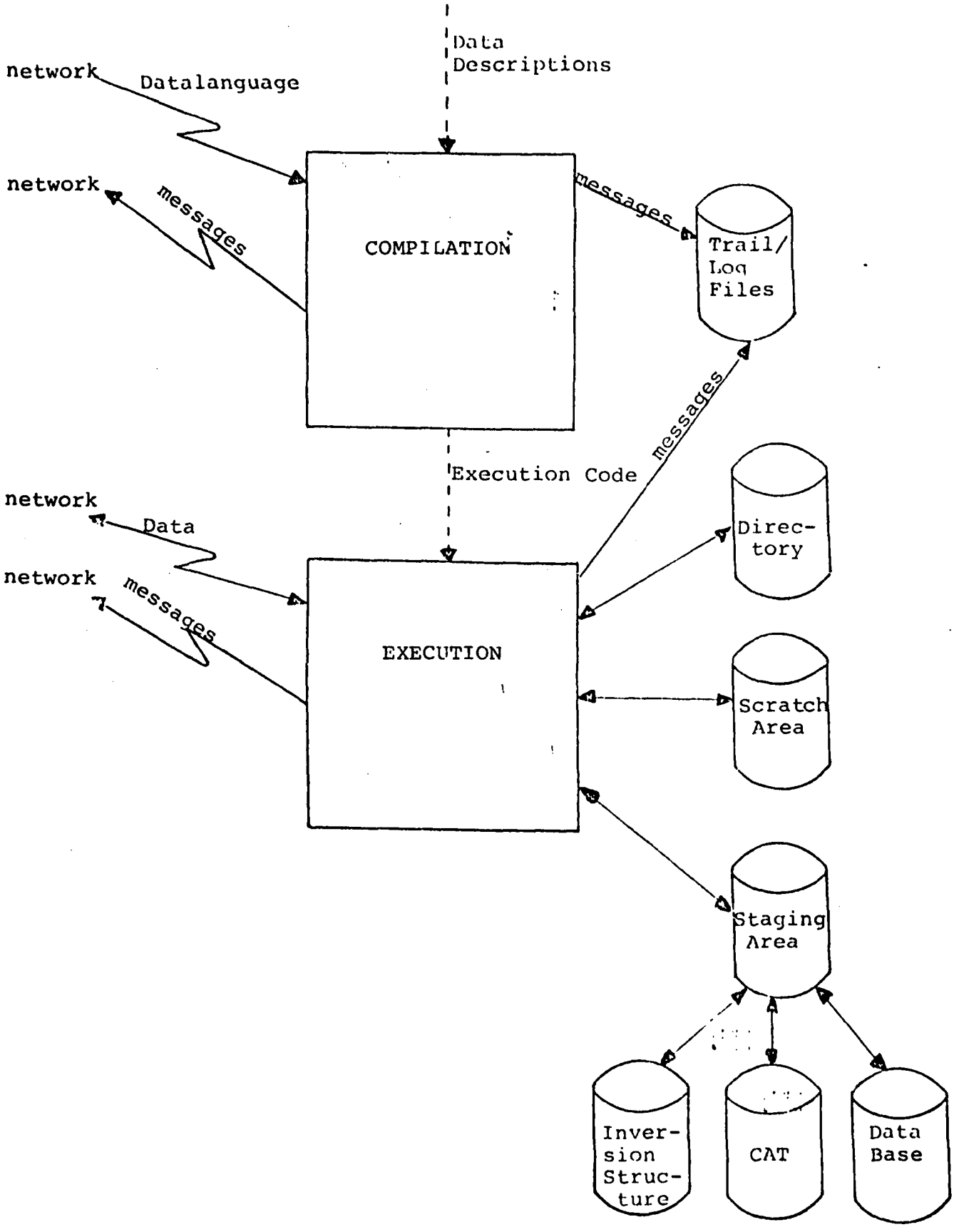
whose descriptions are also maintained in the system directory. All requests, regardless of the operations to be performed, go through a compilation phase followed by execution as illustrated in figure 3.2. In the following sections, these phases are detailed with emphasis on I/O and other areas of potential delay. Also to follow is a section on Datacomputer/user-job communications which impact all requests.

3.1.2 Communications

Although all database management systems communicate with the end user, the Datacomputer is different in that the typical user of this system is a program running at some other site on the Arpanet. All interaction between the user program and the Datacomputer -- sending Datalanguage requests, receiving Datacomputer responses, and transferring data outside the Datacomputer -- takes place over the network. Communication delays incurred over the network tend to be much greater than those incurred on single-site systems. Since the end user is another process, the Datacomputer performs synchronization, error reporting, prompting for actions etc. during the compilation and execution phases via network messages.

Request Processing Components

Figure 3.2



Each message is formatted with a prefix code and human-readable text. The prefix code is designed to be machine-readable so that the user process can make decisions based on the message without having to parse a human-readable string. The human-readable portion of the messages is designed to provide the end user with a more detailed description of what is going on.

A record of all dialog in the users' sessions is kept in audit trails and log files which the Datacomputer maintains in the TENEX/TOPS20 file system. These files are produced primarily for system maintenance purposes. If the Datacomputer exhibits unexpected behavior, the audit trails and log files may be used to determine the exact sequence of events that led to the problem situation. Although these files do not require network communication, they do incur additional formatting and local I/O overhead.

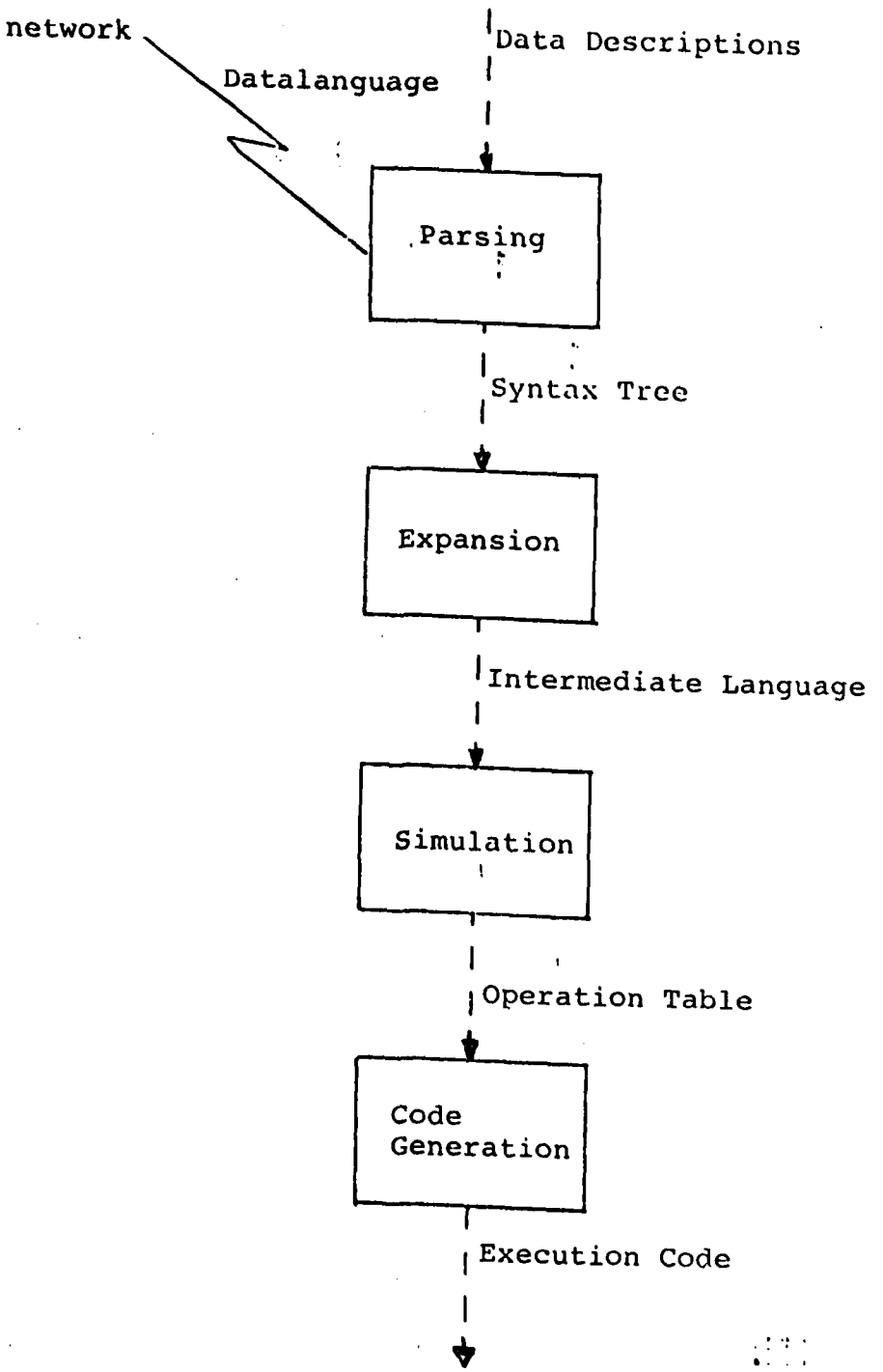
3.1.3 Compilation

The Datacomputer was designed to handle large files. Thus the compilation phase of the system was designed with an optimization phase which reduces the number of instructions executed for each record. While this strategy yields overall savings when processing large files, it may in fact slow down the processing of requests involving small files, if the time to perform the optimization exceeds the savings gained.

The process of compilation proceeds in four steps as illustrated in figure 3.3. The parsing step reads in the Datalanguage over the network. The expansion step adds loop control and inversion handling to the requested database operations. The simulation step optimizes the operations to be performed at each point based on operations which have already been performed. The code generation step selectively generates instructions to perform the operations, eliminating all tests and branches based on information known at compilation time. These steps are elaborated below.

Compilation Phase

Figure 3.3



3.1.3.1 Parsing

Syntactic and semantic analysis is performed as Datalanguage is parsed and a syntax tree is built. The data descriptions for all files and ports involved in the request are utilized during semantic analysis. These descriptions are accessed from the directory system at the time the file/port is opened and remain in core until it is closed. The resulting syntax tree is an unambiguous internal representation of the request.

3.1.3.2 Expansion

The expansion step of compilation analyzes the syntax tree and produces an internal structure named intermediate language. The two most important functions of the expansion step are to add looping and inversion handling to the request.

- The looping structure which will process a file or port on a record by record basis is embedded within intermediate language

- All boolean qualifications on files are analyzed in the light of inverted fields. The qualification is broken into two booleans, one of which only references inverted fields and one which cannot utilize inversion. The 'inverted' boolean is used at execution time to restrict the records which will be accessed within the loop on the file. The 'non-inverted' boolean is then applied to all records accessed within the loop.
- All assignments to inverted fields are expanded to update the inversion structure as well as the database itself.

3.1.3.3 Simulation

Central to the simulation process is optimization of execution code. During this phase, intermediate language is transformed into a series of atomic operations which are entered into a table. The optimization at this level is accomplished by simulating the runtime environment and using the results of this simulation to produce the most efficient sequences of atomic operations to handle each intermediate language operation. Minimization of the operations to skip from one field in a record to the next

is an example of the kind of optimizations produced in this phase. Obviously these optimizations produce the biggest payoffs when the code is executed many times or in other words when large numbers of records are being processed.

3.1.3.4 Code Generation

Code generation is the final step of compilation. Each entry in the operation table is converted into one or more instructions. The thrust of code generation is to eliminate as many tests and branches as possible from the final execution code. This is accomplished by performing all possible tests based on information known at compile time, either generated by the simulation step or based on field descriptions, and generating appropriate code based on the result of the tests. This eliminates the need to perform the identical tests at execution time for every record when the results of the tests will not vary from record to record.

3.1.4 Execution

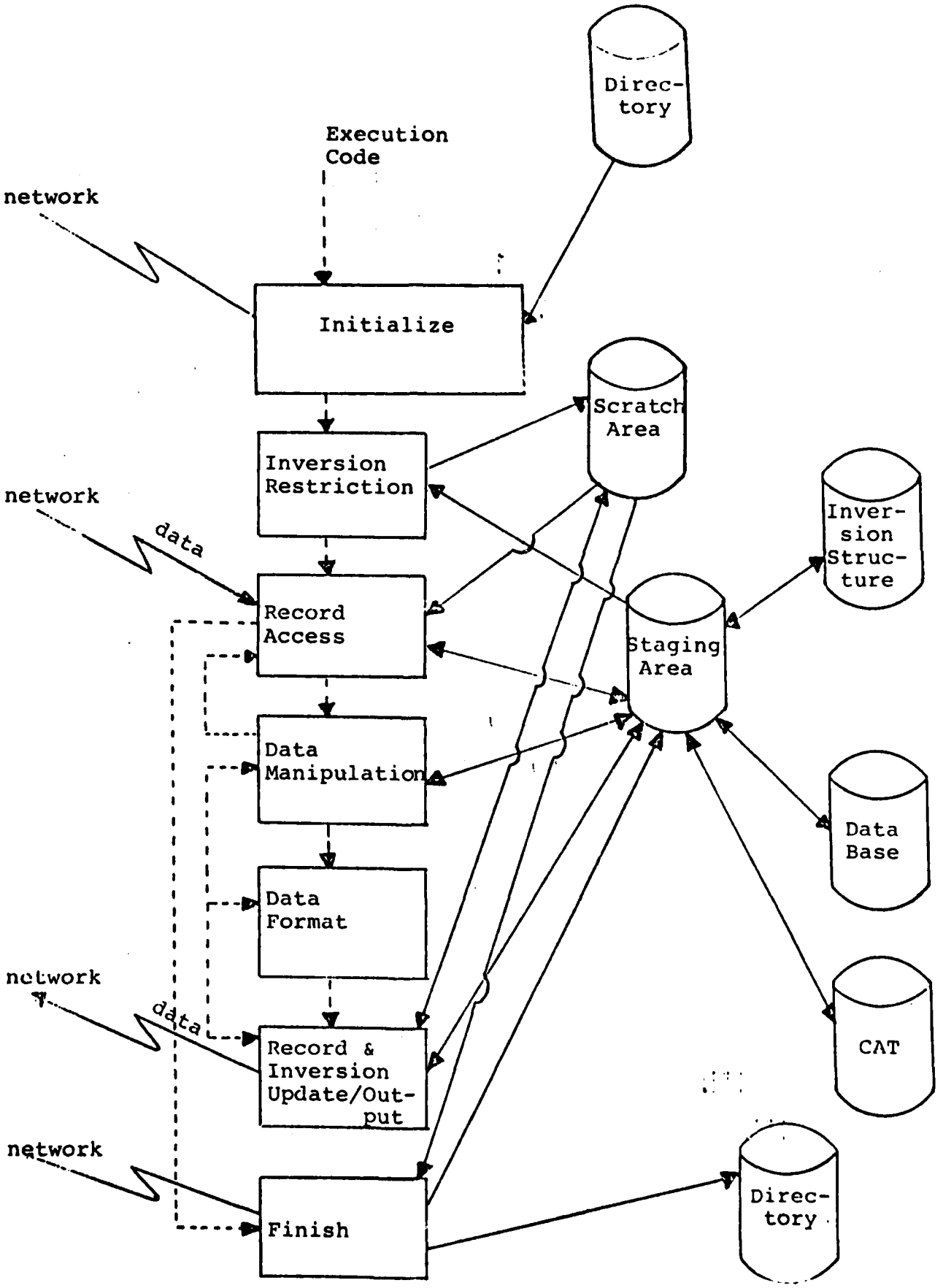
Although the execution phase of request processing varies with the mode of operation, each file and port referenced in the request proceeds through seven basic steps as illustrated in figure 3.4. The first step initializes files and ports for data reading and writing. The second step restricts file access based on inversion. The third through sixth steps comprise the looping mechanism and perform the requested database operations on each record. The seventh step performs the termination operations.

3.1.5 Storage Interface

The effect of data I/O throughout the execution phase of request processing is directly related to the system's storage techniques and the supporting hardware. This section describes the Datacomputer's storage interface in general terms and points out areas where systems using different hardware may vary.

Execution Phase

Figure 3.4



The Datacomputer maintains a system of maps that describe the location, possibly on several storage devices, of various versions of different parts of files. Files are divided into sections. Each section is of logically contiguous storage but may be several physically non-contiguous storage extents.

For efficiency, the physical extents in which sections of data are stored are made as large as can conveniently be handled by the hardware configuration. If the hardware supports it, this makes it possible to do track and cylinder at a time disk I/O when moving an extent between secondary and tertiary storage or copying an extent from disk to disk.

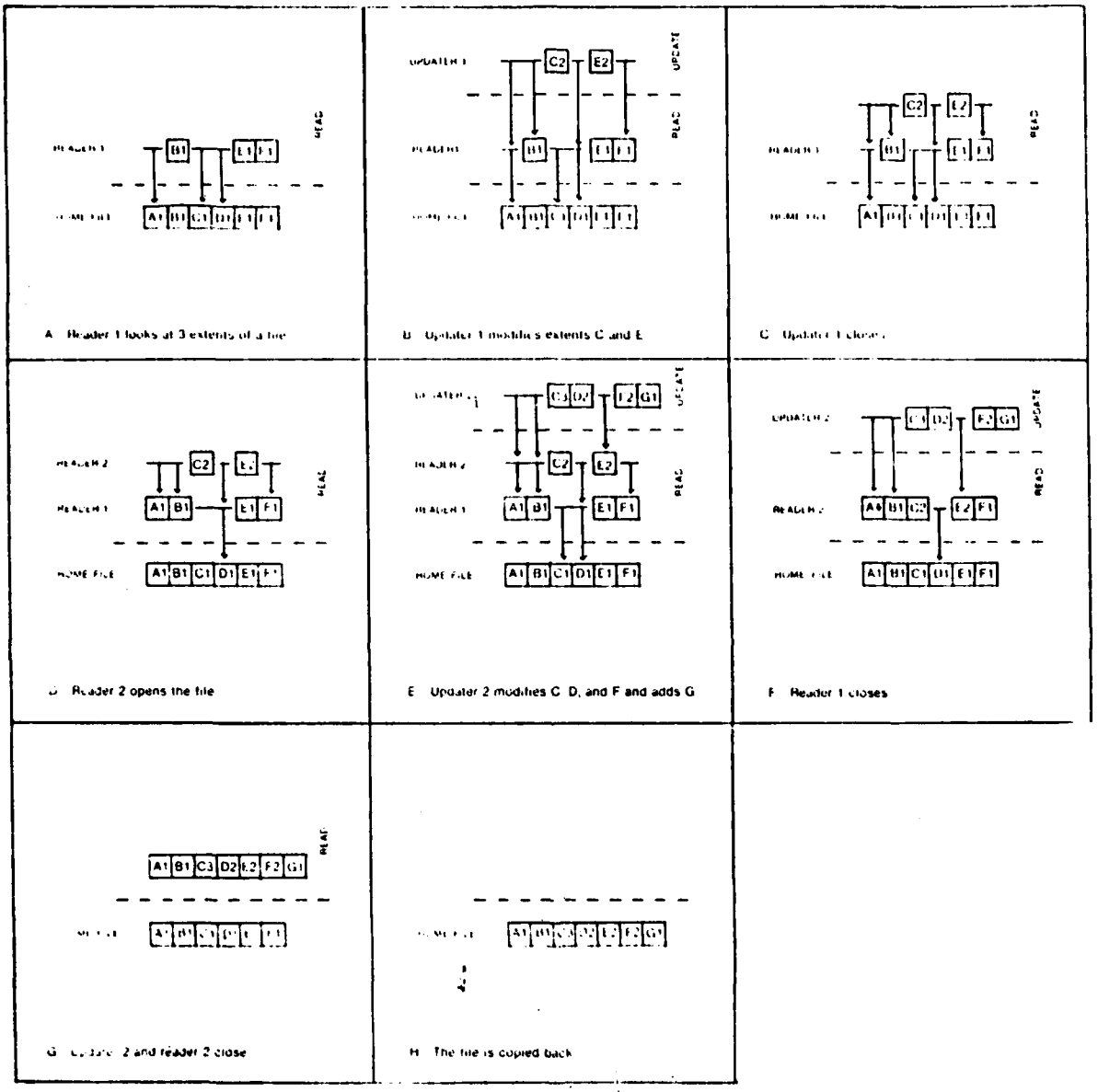
The Datacomputer provides for multiple readers and one updater of a file with each reader guaranteed to see a consistent version of the file. When an updater of a file modifies an extent, the change is made to a new copy of that extent and thus requires copying data. Any reader coming along in the meantime sees the file as it was before it was opened for modification. If a serious error occurs during the update, the modified extents will be discarded and the file left unchanged for consistency.

If an update is successful, when the file is released by the updater the modified extents are logically merged into

the file. A new reader will see the modified file, but if the file is still being accessed by an old reader, the unmodified file is preserved.

All this is done with chains of maps that are maintained by the Datacomputer. There can be at most one update map to modified extents followed by one or more read-only maps and then the 'home' file map which is the map of all the file in its original quiescent state. Maps other than the home file map may be incomplete and any missing parts are found by searching down the map chain [EASTLAKE and FOX] (see figure 3.5). Certain physical volumes of secondary storage are used exclusively for active extents of files for readers and writers as described above. In contrast, the home file map may point to an area on a volume of tertiary storage depending on the system hardware configuration. Copying referenced extents from the home file to secondary storage is referred to as staging the data. Although figures 3.3 and 3.4 show that all data is staged, actually the data that is staged is configuration dependent. On configurations not having a tertiary store, non-modified data is read directly from the home file. Modified data is written to the staging area and is eventually copied back to the home file. Modified data is read from the staging area until the copy back process takes place.

Datacomputer Differential File Mechanism Figure 3.5



Datacomputer Differential File Mechanism Showing Different Maps For Readers and Updater

When disk space for active file extents gets crowded, modified data must in general be copied back to the home file and unmodified data can be discarded. On configurations having a tertiary store, a background task within the Datacomputer periodically awakens and, if secondary storage is getting sufficiently tight, attempts to discard or copy back data to make more room. On configurations not having a tertiary store, modified data is copied back when no jobs are using the file.

3.2 Selection of Components in the Model

In order to map the actual processing components of the Datacomputer into the model, some initial experiments were performed. These experiments were intended to indicate which components were likely candidates for processing bottlenecks. The results of these initial experiments indicated that for the kinds of databases used in SDD-1 and other command and control activities, the cost of running execution code was negligible and that overhead was very expensive. These results in conjunction with our understanding of the Datacomputer's processing steps led us to define twelve interesting processing components for the model.

Whereas some of these components correspond to those described above, others are not related to the kind of component described above, but are components that permeate the entire request processing activity. Specifically, the following components were selected for the axis of the matrix:

1. Printouts -- This refers to the sending of messages to the user program over the network.

2. Context -- This corresponds roughly to the parsing phase of compilation.

3. Precompile -- The expansion phase of compilation.

4. Compile -- The simulation phase of compilation.

5. Instruction Generation -- The instruction generation part of compilation.

6. Overhead -- Unaccountable time lost to system overhead.

7. Inversion -- Inversion processing.

8. Page Read -- Reading a page of the database.

9. Scratch Read -- Reading a page from a scratch file built during request processing.

10. Read Only Page -- Making a page read-only (mainly entails making a system call to do this).

11. Writable Page -- Making a page writable.

12. Open Buffer -- The Datacomputer operation of allocating a buffer.

The reasons for this selection of components will become more apparent in the measurements section of this report.

3.3 Query Classes

Four classes of query traffic were selected for use in the model based on SDD-1 usage and Datacomputer usage in the ACCAT. The query classes selected are:

1. Bluefile Retrievals -- The bluefile [NELC] is a small sanitized version of a real command and control database which exists on the Arpanet. Queries on the bluefile were generated using LADDER [SACERDOTI]. The most extensive measurements were made on these.

2. WES Updates -- WES is a war games simulator that produces update transactions for the Datacomputer at ACCAT. Some measurements were made on the Datacomputer running WES updates.

3. FC Database Retrievals -- The FC database is the classified fleet commander's database at the ACCAT. LADDER retrievals on the FC database were used as another query class.

4. SDD-1 Retrievals -- Finally, Datacomputer queries produced by SDD-1 were included.

3.4 System Loading

Based on previous experience and interactions with NOSC, we decided to emphasize paging as the important loading factor in the model. Some preliminary experiments were conducted to determine if the CPU time resource was an interesting parameter. It turned out that most components experienced delays almost directly proportional to CPU load. On the contrary, delays incurred by paging appear to vary much more widely. Three levels of paging activity were chosen for the model:

- light -- essentially no page contention from other jobs on the system.

- medium -- a moderate amount of paging produced by one page-bound competing job on an otherwise empty system.

- heavy -- a large amount of paging produced by two such jobs.

Higher levels of paging were investigated but did not produce dramatically different results so they were left out of the model.

4. Measurements

In order to make measurements of Datacomputer performance, a version of the Datacomputer was modified to produce statistical output at each significant step of processing. These statistics included CPU utilization, real time elapsed and page faults for each step. This modified Datacomputer was used to analyze performance on the bluefile queries. Extensive experiments were conducted on thirteen such queries (each accessing two files and one port) and a few of the experiments were run on one hundred queries as further verification of the results on the smaller sample. Several characteristics of these queries and the database are important to the analysis of the results:

- The files are short (200 or fewer records per file).
- Each file has many inverted fields.
- All the requests are retrievals.

- All files were open and stayed open during the tests. No OPENs or CLOSEs were executed.

4.1 CPU Time Measurements

The first set of results, as illustrated by figure 4.1, show the CPU time used by different parts of the Datacomputer. For expository purposes, the twelve processing components were first collapsed into the following four:

1. Services -- The services section constitutes a pseudo operating system for the Datacomputer. It provides the basic functions of a traditional operating system in a form which is maximally convenient for the request processing parts of the Datacomputer. Time spent in the services section can be viewed as Datacomputer system overhead. It was selected as a component because as the measurements below show, a large percent of the CPU time is spent there.

2. Inversion Processing -- This corresponds to time spent processing inversions during the execution phase.

3. Compilation -- This is the entire compilation phase as described in section 3.1.3.

4. Other -- Time unaccounted for elsewhere. This includes executing the compiled code.

As shown in figure 4.1, services utilized about two-thirds of the time; thus it was subdivided, as illustrated in in figure 4.2, in order to better understand where the time was spent. The subdivisions in general correspond to components in the model described above. The one component which requires further explanation is opening and closing files and ports. Even though the files and ports are already open (as far as the user is concerned), whenever they are used in requests, several actions must take place within the Datacomputer. These include:

- allocation of I/O buffers,
- data staging, and
- directory updating to indicate the last referenced date.

Division of CPU Utilization — Figure 4.1

Component	Percent of CPU time
Services	62
Inversion Proc.	15
Compilation	20
Other	3
TOTAL	100

Services CPU Breakdown — Figure 4.2

Component	Percent of CPU time
Page reads	22
Scratch file reads	19
Opening files/ports*	8
Closing files*	4
Listening on network	6
Buffer calls	3

*these refer to internal opening and closing, not the OPEN and CLOSE commands

From figure 4.2, it is obvious that page reads are very expensive. In order to get a better handle on why this is the case, an experiment was conducted comparing the CCA Datacomputer and a standard TENEX Datacomputer (like the one in the ACCAT). Unlike the TENEX Datacomputer, the one at CCA maintains its own disks separate from those on TENEX. Therefore, it was possible to compare the time to do a page read on the two systems to determine how much of that time was TENEX overhead. Figure 4.3 shows the

results of this test. The page reading activity is divided into three steps in this figure. The first step allocates a buffer in core to hold the contents of the page. The second step is physically causing the page to be read in to the buffer. The third component is miscellaneous additional processing. The difference between the two is accounted for by the additional CPU time required by the TENEX operating system to read the page. Obviously this is an expensive TENEX operation.

Comparison of TENEX & CCA-TENEX page reads Figure 4.3

TENEX Page Read

Component	CPU time in ms
Buffer manipulation	3.5
Reading Page	9.0
Other	1.5
TOTAL	14.0

CCA-TENEX Page Read

Component	CPU time in ms
Buffer manipulation	3.5
Reading Page	2.0
Other	1.5
TOTAL	7.0

Internally opening and closing files also incurs a significant time cost in these experiments. Further investigation indicated that a major reason for this cost

was changing the access mode of Datacomputer directory pages. In general, these pages are kept read-only for reliability reasons. However, whenever they are accessed, the Datacomputer updates the "last read" slot in the page. This process entails making the page writable, writing the date and time and then making it read-only again. Experiments indicated that about 75% of the time opening and closing files was spent in changing the modes of pages. This seems to be an area where reliability could be traded off against performance.

4.2 System Load Experiments

The next set of experiments measured Datacomputer performance at different levels of paging activity. Three levels of paging were established as described earlier. The results are summarized in figures 4.4 and 4.5. Figure 4.4 indicates the average amount of CPU time, elapsed time and page faults for the bluefile queries. Figure 4.5 indicates clearly which processing components are most sensitive to paging activity. The numbers in parenthesis indicate the factor by which the value increased over the lightly loaded case.

Paging Activity Variation — Figure 4.4

	Light	Moderate	Heavy
CPU(ms)	11441	11669	12703
Real(ms)	20000	49470(2.5)*	80919(4.0)
Page faults	197	363(1.9)	571(2.9)

*numbers in parenthesis indicate the factor of increase over light.

Paging activity on a component basis — Figure 4.5
Elapsed time in ms

	Light	Moderate	Heavy
Printouts	229	1309(6.0)	2193(9.9)
Context	128	1071(8.0)	1504(12.0)
Precompile	551	3868(6.5)	5628(12.0)
Compile	950	5372(5.5)	3697(3.5)
Inst. Gen.	1426	5060(3.5)	7176(5.0)
Overhead	2078	6900(3.3)	10627(5.0)
Inversion	7731	14782(2.0)	19560(2.6)
Page read	4914	11775(2.4)	20541(4.4)
Scratch read	3429	4923(1.4)	7217(2.1)
Read Only pg.	597	526(.9)	526(.9)
Writable pg.	599	928(1.6)	2414(4.1)
Open Buffer	112	1188(10.6)	1165(10.6)

Initial analysis of the measurements made on the LADDER queries indicates that overhead both in TENEX and in the Datacomputer is a very significant for queries on the small files used.

4.3 WES Experiments

The WES transactions which we examined had different characteristics than the LADDER queries:

- The files were very small.
- All the requests involved appending or updating to files.
- The files were not inverted.
- Each request accessed at most 400 records.
- Each request accessed at most 20 data pages.
- The requests were all precompiled.

Since the WES data had tc be produced on the ACCAT Datacomputer, we were not able obtain as much measurement data as we did with the LADDER queries. If more data is required during the study, a trip will be made to NOSC to perform experiments using our modified Datacomputer. From our discussions with NOSC personnel, it is apparent that although the Datacomputer performs correctly in the WES environment, its execution speed is significantly below the desired level.

WES operates by running a series of requests every cycle through the simulation. Seven different precompiled requests are run each time. Figure 4.6 lists the requests and what they accomplish; figure 4.7 shows CPU and elapsed time for the compile and run phases of each request. The compile phase in this case includes reading the precompiled request and finishing the compilation. The runtime part includes essentially everything else involved in the request. These results indicate that even the small amount of time required to setup a precompiled request is significant when we are dealing with requests on such small files. Once again overhead is a very important cost factor.

WES Requests — Figure 4.6

TRANS	transfer records into temporary file
UPDATE	update position of ships and aircraft if they are already in the database
HIST	add records for new ships and aircraft to tracl.hist file
CONT	update the contact file
CASR	update the casualty file and the readiness file
POSAPPEND	append to the position file all records not found during the update phase
UNITUPDATE	update the unit file

WES Requests CPU time and elapsed time (expressed in ms) — Figure 4.7

	Compile CPU / ELAPSED	Runtime CPU / ELAPSED
TRANS	1727 / 2799	2792 / 28898
UPDATE	3400 / 7085	5827 / 8824
HIST	3550 / 6166	4906 / 10669
CONT	2200 / 4150	2620 / 6203
CASR	3550 / 3968	4085 / 8594
POSAPPEND	3650 / 10211	4628 / 8027
UNITUPDATE	3100 / 11032	2530 / 4067

Some interesting observations can be made from studying this data on the WES updates. Even though the requests are precompiled, a significant cost is still involved in the compilation phase. This includes reading the precompiled request into core and executing the final instruction generation phase. The first step involves 1

second of cpu time, about 15 page reads, 3 directory page reads and 14 directory mode changes. Mode changes are quite expensive (from our previous experiments) and take approximately 45 ms each for a total of .63 seconds. The instruction generation step is mainly cpu intensive taking on the order of 2-3 ms of cpu time.

The primary expense in actual execution of WES requests is in overhead. The overhead is mostly internal opening and closing files and moving data to the staging area. Once again mode changing is an important factor. In some cases 75 mode changes are made. At 45 ms each, this adds up to about 3 seconds of cpu time.

4.4 Measurements on the FC Database and SDD-1 Usage

Currently we do not have as much data for these query classes as we had for the others. A small number of queries were run on the FC database and some statistics were collected. In most cases the results corresponded to the results on bluefile queries. There were a few cases where linear scans of large numbers of records were required and these produced results where execution time was a much more important factor.

The data from some runs of SDD-1 have just recently been produced and has not as yet been fully analyzed. However, initial observations indicate that the trend is similar to that for bluefile queries.

5. Preliminary Conclusions

From the analysis of the data we have obtained so far, some initial conclusions can already be drawn. These include:

1. Overhead in the Datacomputer and the operating system is a very significant factor. This fact indicates that optimization toward smaller files is desirable. Also trading off reliability with performance could cut down some of the system overhead.

2. The Datacomputer is very sensitive to paging activity. One way of cutting down on paging is to decrease the working set size. Two approaches to this are possible: restructure the system's core image to increase locality of reference and remove parts of the system that are not used in the SDD-1/command and control environment. Paging experiments have been conducted which indicate that the number of page faults could be cut approximately in half by loading all subroutines near to the other subroutines which call them.

3. An inversion mechanism tailored more towards small files would cut down substantially on the number of scratch reads performed. Currently the Datacomputer builds its inversion structure in scratch files during execution. This technique was implemented because, in general, the information will not fit in core. However, with small files in-core inversion processing is possible and would be much more efficient.

6. Future Work

Future activities in this project will focus on proposing and analyzing specific Datacomputer enhancements. In order to do this, additional measurements may be required. Complete analysis of the SDD-1 results must also be done to verify the initial analysis of these results. In addition, enhancements which affect performance in indirect ways will be considered. These include expanding the number of files and/or ports that can be open at a time and increasing the amount of space available for compiling large requests. There are obvious tradeoffs involved in these kinds of enhancements that will have to be considered.

REFERENCES

[EASTLAKE and FOX]
Eastlake, D.E. and Fox, S.A. "Integrity and Recovery in the Datacomputer Machine for Very Large Databases',Submitted to National Computer Conference, 1978.

[MARILL and STERN]
Marill, T. and Stern,D.H. "The Datacomputer: A Network Utility", Proceedings AFIPS National Computer Conference, AFIPS Press, Vol. 44, 1975.

[NELC]
NELC, "A Relational Model for an A7 Sea Commander's Tactical Data Base", Project Scientist: Garrison Brown, SEI, October 29, 1976.

[ROTHNIE and GOODMAN]
Rothnie, J.B.; and Goodman, N. "An Overview of the Preliminary Design of SDD-1: A System for Distributed Databases", 1977 Berkeley Workshop on Distributed Data Management and Computer Networks, Lawrence Berkeley Laboratory, University of California, Berkeley California, May 1977. (Also available from Computer Corporation of America, 575 Technology Square, Cambridge Massachusetts 02139, as Technical Report No. CCA-77-04).