LEVEL II

Computer Corporation of America

575 Technology Square
Cambridge
Massachusetts 02139

617-491-3670

(12)

(6) The Enhancement of Datamodule I.

(9) FINAL TECHNICAL REPORT.

(12) 69

(11) 1980

Computer Corporation of America
575 Technology Square
Cambridge, Massachusetts 02139

(15) 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, of the Defense Advanced Research Projects Agency or the U.S. Government.

Table of Contents

1. Introduction

This report summarizes a project entitled 'The Enhancement of Datamodule 1.' The focus of this project is to study the Datacomputer's [MARILL and STERN] performance in its role as a datamodule in SDD-1 ~~[ROTHNIE et al]~~ 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. Some of the proposed enhancements would be subsequently implemented in the Datacomputer.

During the reporting period, the model of Datacomputer performance constructed previously during this project was used to analyze performance in the Advanced Command and Control Architecture Testbed (ACCAT). The Datacomputer was instrumented to produce extensive performance data at runtime. The results of this work have indicated which parts of the system are bottlenecks and this, in turn, is suggesting potential enhancements. In addition, the instrumented Datacomputer has been used in other performance work for ARPA.

The remainder of this report describes the study in detail. Section two gives an overview of the study methodology, section three describes the Datacomputer performance model, section four describes the measurement results and section five describes our enhancement recommendations.

2. Study Procedure

The study procedure we used 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 were:

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 all 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 was constructed earlier in this project and was used in the current analysis.

2. Measurements have been made in the ACCAT to obtain performance data for Datacomputer activities in a command and control environment.

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 to determine where the system bottlenecks lie.

5. Analysis of the data has led to a set of potential enhancements that will help Datacomputer performance both in the SDD-1 context and in the "stand-alone" context.

6. A version of the Datacomputer incorporating some of the potential enhancements has been produced. Use of this version of the Datacomputer has resulted in performance improvements of between 1.2 and 3.5 in a critical ACCAT application.

3. Model Definition

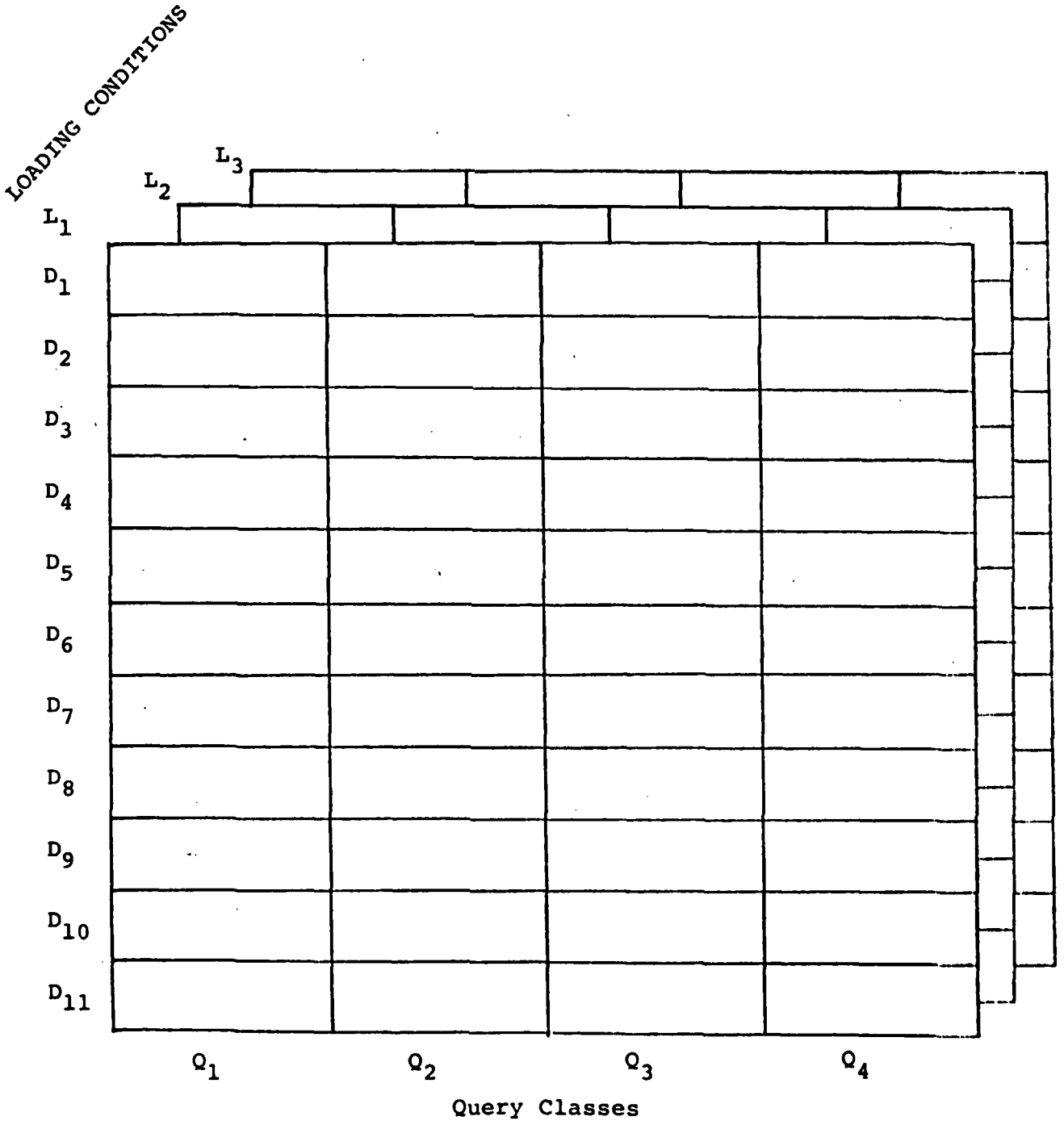
A Datacomputer performance model which represents the delay incurred by each component of the system for each of several types of queries and under each of several system loads has been defined. This model can be viewed as a 3-dimensional matrix of the sort pictured in figure 3.1. The first dimension 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.

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.

Datacomputer Model

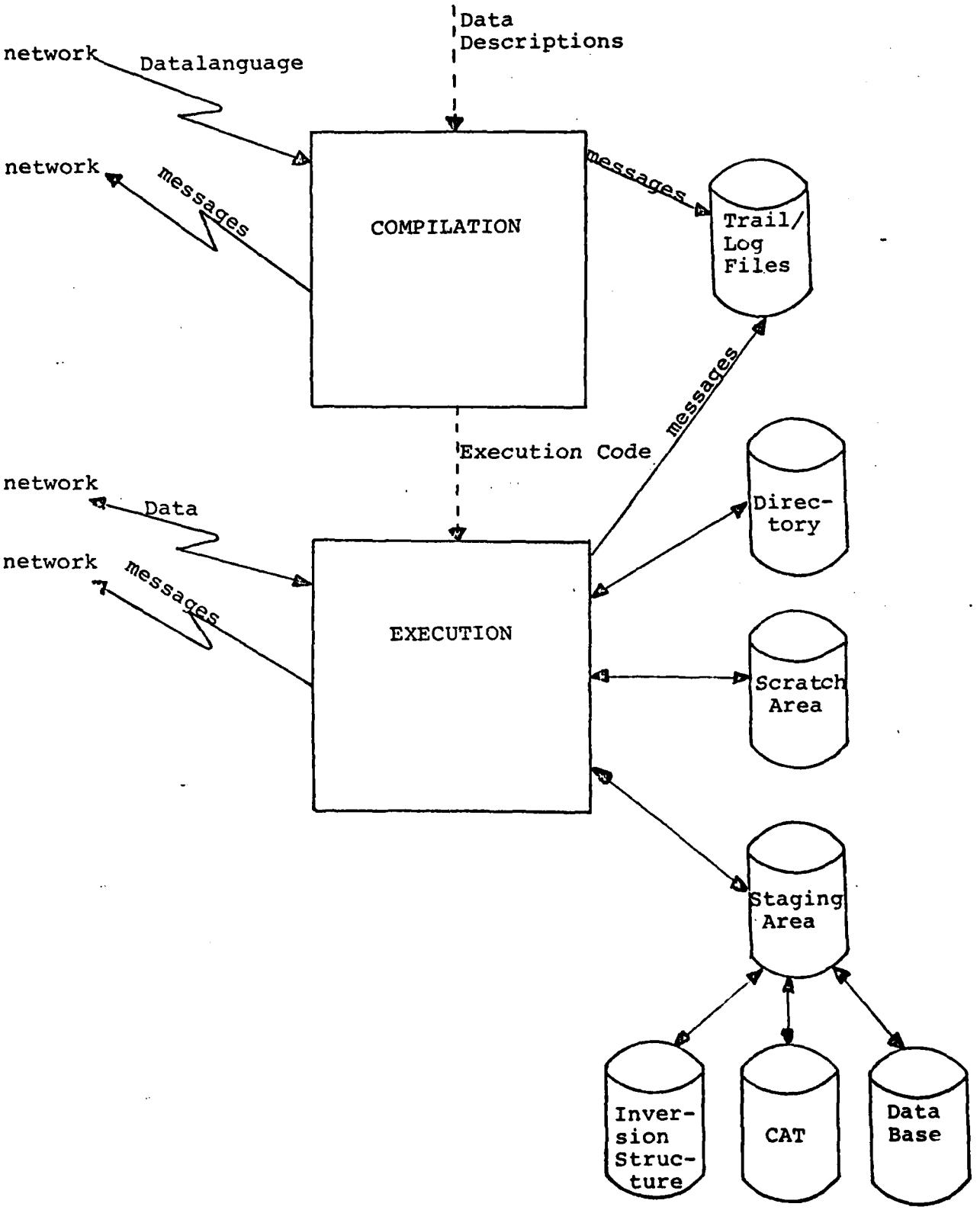
Figure 3.1



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.

Request Processing Components
Figure 3.2



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. 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.

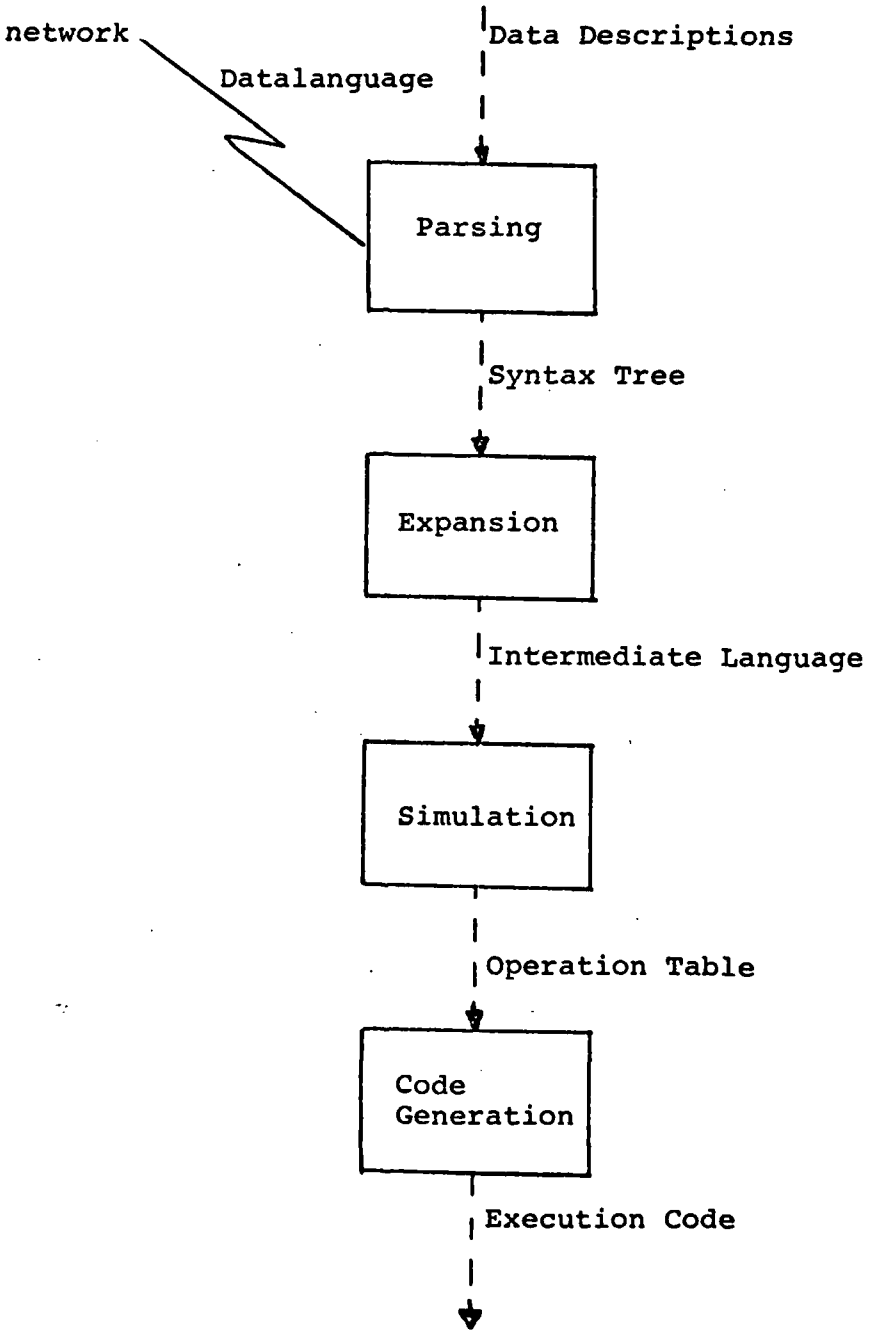
3.1.3 Compilation

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

Compilation Phase

Figure 3.3



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.

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

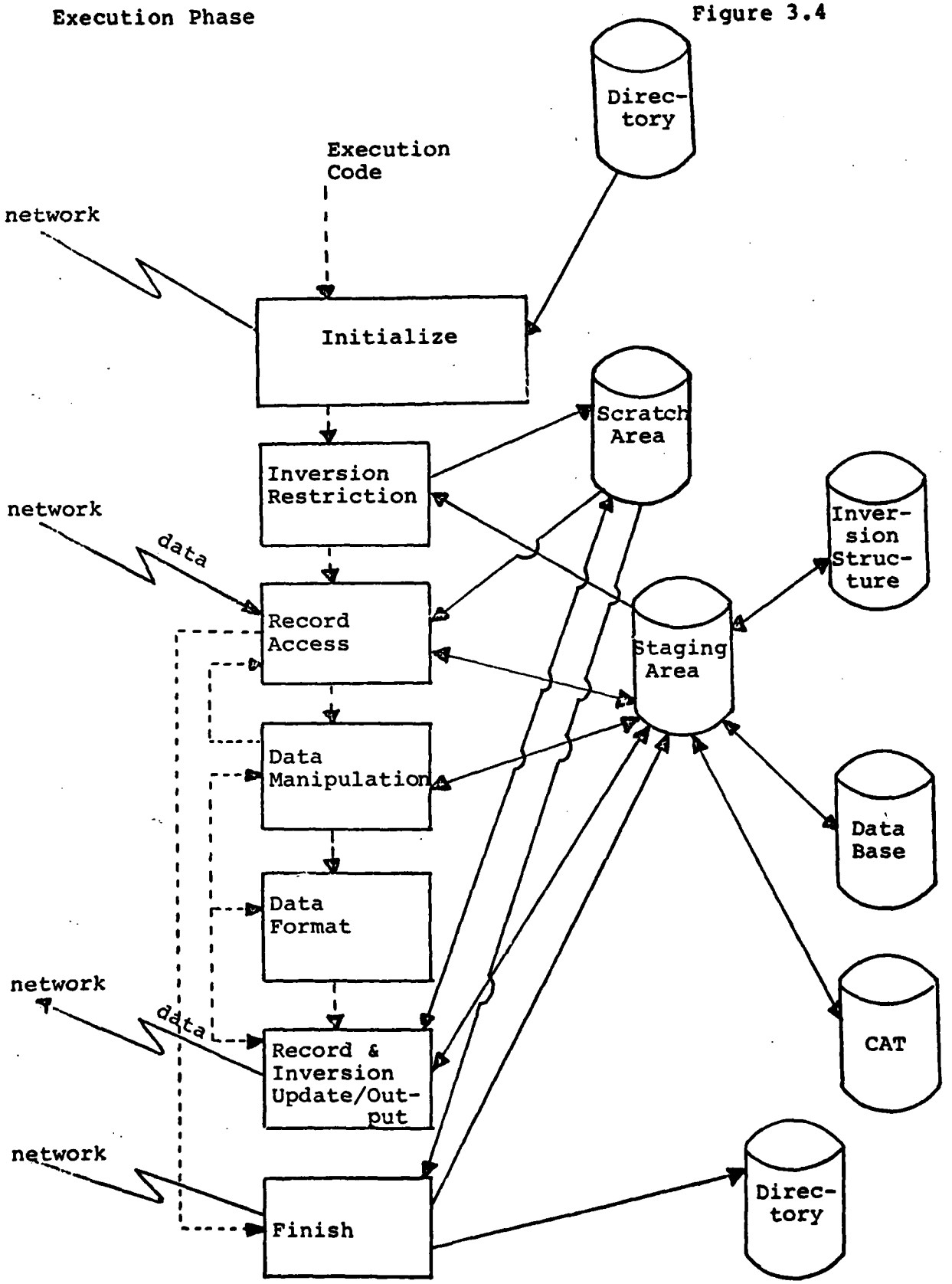
Code generation is the final step of compilation. Each entry in the operation table is converted into one or more instructions. The thrust of code generation is to eliminate as many tests and branches as possible from the final execution code. This is accomplished by performing all possible tests based 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

Execution Phase

Figure 3.4



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.

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

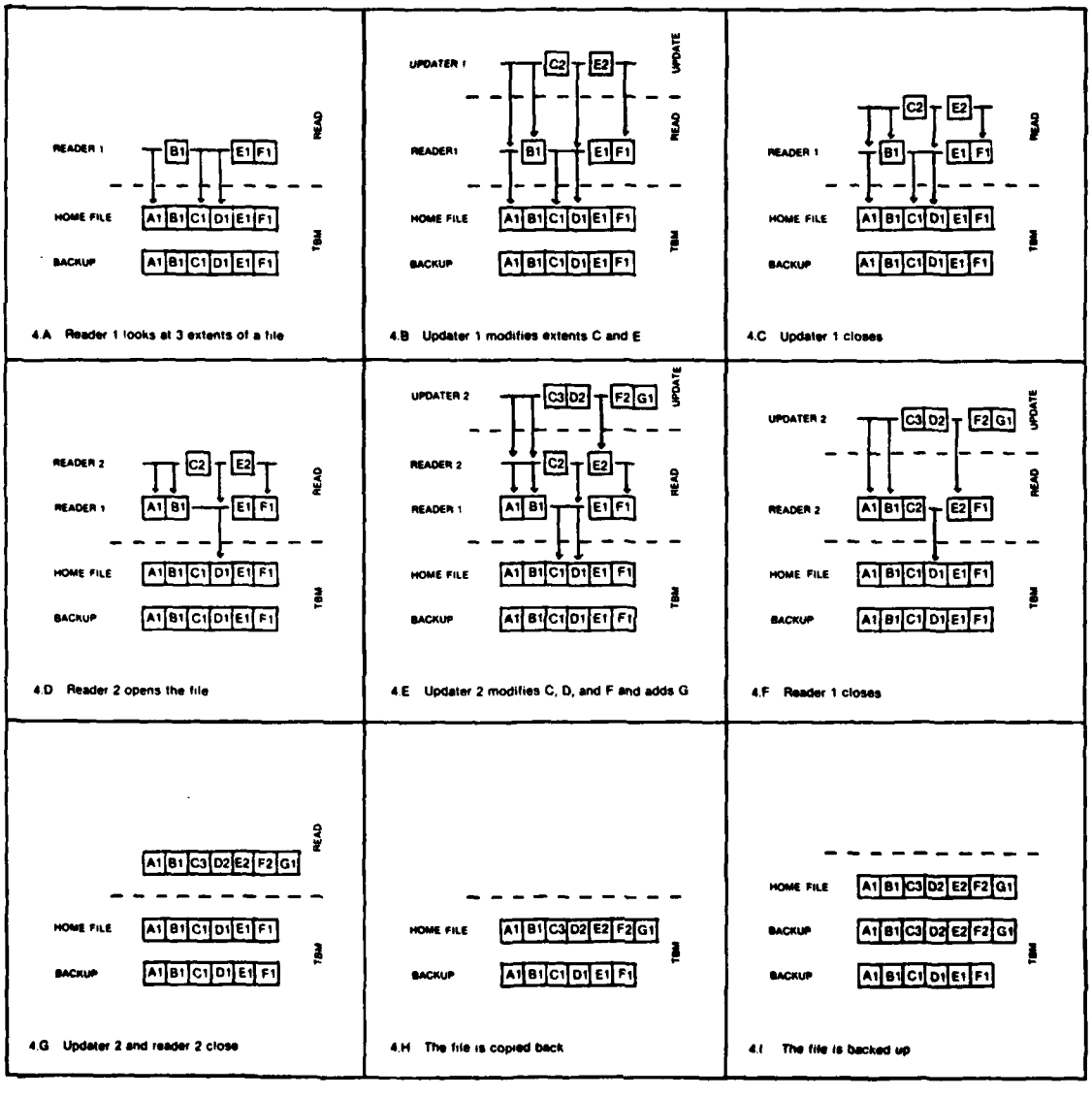
cylinder at a time disk I/O when moving an extent between secondary and tertiary storage or copying an extent from disk to disk.

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 (see figure 3.5).

Datacomputer Differential File Mechanism Figure 3.5



4.A Reader 1 looks at 3 extents of a file

4.B Updater 1 modifies extents C and E

4.C Updater 1 closes

4.D Reader 2 opens the file

4.E Updater 2 modifies C, D, and F and adds G

4.F Reader 1 closes

4.G Updater 2 and reader 2 close

4.H The file is copied back

4.I The file is backed up

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.

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

During the first phase of this project, performance measurements were carried out using queries on the Bluefile database. The results of these measurements have assisted us in modifying the selection of processing components in the model. The current model consists of 32 different components.

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. Also, some of the measured components overlap each other. This choice was made intentionally when it was desired to capture the cost of an entire large component and also the costs of small individual parts of this component. Specifically, the following components were selected for the vertical axis of the matrix:

1. SV Calls -- The Datacomputer is logically divided into two major components: RH and SV. RH is the request handler and provides the interface between the user program and a pseudo operating system, SV.

SV provides the capabilities typically found in an operating system such as an I/O interface, lock arbitration, memory management etc. Whenever RH calls a subroutine in SV it goes through a special calling mechanism. This calling mechanism provided a convenient place to monitor the time and resources used in SV. Many other components are subsumed in the SV calls component.

2. Error Calls -- Passing messages back to the user program (for errors, status, synchronization etc.) and maintaining audit trails and log files.

3. Wait Time -- Time spent waiting for new user input.

4. Syntax Analysis -- Lexical pass of compilation.

5. Context Recognition -- Parsing phase of compilation.

6. Precompile -- The expansion phase of compilation.

7. Compile -- The simulation phase of compilation.

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

9. RH Open Port -- Request handler part of opening a port for user data.

10. RH Clean Up -- Code executed between end of request or command and return to user.

11. IRFind -- Inversion set up.

12. Hunk Loop -- Inversion processing.

13. Net I/O -- Overhead in network I/O.

14. SV Open File -- Internally opening a file (happens at the beginning of every request before file can be used).

15. SDAX Open File -- That part of SV open dealing with building a map.

16. RH Open File -- Time spent in the actual Datalanguage OPEN command.

17. Page Read -- Datacomputer page reading subroutine.

18. Scratch Page I/O -- Writing and reading scratch files during inversion processing.

19. Making a Page Read Only -- For integrity/reliability reasons, directory pages are kept read only most of the time.

20. Device Mount -- Primarily consists of opening the TENEX/TOPS20 files containg the Datacomputers databases.

21. Buffer Allocation -- Manipulation of buffers in the Datacomputer.

22. Making Page Writeable -- When a read only page must be written, it is temporarily made writeable.

23. Tenex Read -- The TENEX/TOPS20 system call to read a page.

24. Buffer Swap -- Swapping the postitions of two buffers in core.

25. Page Write -- Datacomputer page write subroutine.

26. Tenex Write -- The TENEX/TOPS20 system call to write a page.

27. Create Node -- Creating a directory node.

28. Create File/Port -- Creating a file or a port.

29. Generated Code Execution -- Executing code produced by the compiler to answer the query (includes hunk loop, page read/write etc.).

30. RH Close File -- Executing the Datacomputer CLOSE command for files.

31. RH Close Port -- Executing the Datacompiter CLOSE command for ports.

32. SV Close File -- Executed at the end of every request using the file.

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

3.3 Query Classes

During the first phase of this work, measurements were primarily aimed at one set of queries. These were queries generated by Ladder on the Bluefile database. Some measurements were also made during WES games but these measurements were somewhat crude since we were unable to use the instrumented version of the Datacomputer.

The focus of the work reported herein is measurements using the instrumented Datacomputer on Ladder queries to the FC (Fleet Commander) database and WES updates. These measurements were conducted at NOSC by members of the CCA SDD-1/DM-1 Enhancement team. These two kinds of transactions were selected because of their obvious relevance to the Command and Control community.

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. Our previous measurements on the Bluefile gave us strong evidence to back up this bias. 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. In addition, a series of programs were written to tabulate the data in various different formats and compute average values where appropriate.

4.1 FC Measurements

The first set of measurements were made on LADDER queries to the FC database at NOSC. These queries had the following characteristics in common:

- The files were relatively large. Many of the files used had in excess of 10,000 records.
- Inverted access was used in most of the queries.

- All requests were retrievals.

A total of fifteen different queries were measured and analyzed under the three different loading conditions specified above. The results of the measurements indicate that the queries fall into three interesting broad catagories:

1. Queries that used an inverted retrieval to obtain a small number of records (typically one). Eight of the measured queries were of this type. These queries will be referred to as restrictive inverted queries for the rest of this report.

2. Queries that used inversion in a nested loop (requiring the inversion to be set up each time the loop was exectued). Three of the FC queries were of this type. These will be referred to as nested inverted queries.

3. The remaining queries were non-inverted retrievals that required linearly scanning files. These will be referred to as linear queries.

The results of these experiments are presented in figures 4.1 through 4.3. Since the expense of each component varies significantly for the different types of queries,

Percent Resource Utilization

Figure 4.1

Restrictive Inverted Queries

Component	Light Real/CPU/Pgflts	Medium Real/CPU/Pgflts	Heavy Real/CPU/Pgflts
SV calls	52 / 49 / 56	47 / 49 / 24	44 / 50 / 34
Error calls	4 / 5 / 2	3 / 5 / 1	2 / 5 / 3
Syntax analysis	13 / 3 / 2	9 / 3 / 2	11 / 4 / 7
Context recognition	2 / 1 / 3	2 / 1 / 4	1 / 1 / 3
Precompilation	6 / 5 / 9	8 / 5 / 18	10 / 7 / 18
Compilation	11 / 10 / 16	13 / 10 / 30	20 / 10 / 20
Instruction generation	16 / 21 / 8	18 / 22 / 13	10 / 20 / 13
RH open port	1 / 1 / 0	2 / 0 / 1	0 / 0 / 2
RH clean up	1 / 2 / 1	1 / 2 / 0	1 / 2 / 2
IR find	4 / 3 / 5	5 / 3 / 3	8 / 3 / 3
Hunk loop	3 / 3 / 14	3 / 3 / 5	2 / 4 / 4
Net I/O	12 / 3 / 0	8 / 3 / 0	6 / 3 / 1
SV open file	10 / 13 / 5	10 / 12 / 2	10 / 12 / 7
SDAX open file	6 / 8 / 2	6 / 7 / 0	2 / 7 / 2
RH open file	14 / 19 / 5	12 / 17 / 2	12 / 17 / 10
Page read	8 / 4 / 15	8 / 4 / 8	8 / 5 / 5
Scratch page I/O	3 / 3 / 31	2 / 2 / 8	1 / 2 / 4
Read-only page	6 / 10 / 0	6 / 10 / 0	1 / 10 / 0
Device mount	1 / 1 / 2	0 / 0 / 0	8 / 1 / 3
Buffer alloc/realloc	5 / 6 / 1	6 / 7 / 3	4 / 7 / 5
Writeable page	6 / 10 / 0	6 / 10 / 0	5 / 10 / 4
TENEX read	7 / 3 / 15	8 / 3 / 8	5 / 4 / 4
Buffer swap	0 / 0 / 0	0 / 0 / 0	0 / 0 / 0
Page write	0 / 0 / 0	0 / 0 / 0	0 / 0 / 0
TENEX write	0 / 0 / 0	0 / 0 / 0	0 / 0 / 0
Create node	0 / 0 / 0	0 / 0 / 0	0 / 0 / 0
Create file/port	0 / 0 / 0	0 / 0 / 0	0 / 0 / 0
Generated code execution	43 / 50 / 57	41 / 49 / 26	38 / 48 / 31
RH close file	4 / 7 / 0	4 / 7 / 0	3 / 6 / 0
RH close port	0 / 0 / 0	0 / 0 / 0	0 / 0 / 0
SV close file	3 / 5 / 0	3 / 5 / 0	3 / 5 / 0

Percent Resource Utilization
Figure 4.2

Nested Inverted Queries

Component	Light Real/CPU/Pgflts	Medium Real/CPU/Pgflts	Heavy Real/CPU/Pgflts
SV calls	87 / 85 / 91	89 / 85 / 94	80 / 85 / 78
Error calls	0 / 0 / 0	0 / 0 / 0	0 / 0 / 0
Syntax analysis	0 / 0 / 0	0 / 0 / 0	0 / 0 / 0
Context recognition	0 / 0 / 0	0 / 0 / 0	0 / 0 / 0
Precompilation	0 / 0 / 1	0 / 0 / 0	0 / 0 / 0
Compilation	0 / 0 / 2	0 / 0 / 1	0 / 0 / 0
Instruction generation	0 / 0 / 1	0 / 0 / 0	0 / 0 / 0
RH open port	0 / 0 / 0	0 / 0 / 0	0 / 0 / 0
RH clean up	0 / 0 / 0	0 / 0 / 0	0 / 0 / 0
IR find	31 / 31 / 27	35 / 32 / 37	32 / 31 / 30
Hunk loop	20 / 20 / 13	21 / 19 / 13	16 / 19 / 14
Net I/O	0 / 0 / 0	0 / 0 / 0	0 / 0 / 0
SVopen file	0 / 0 / 0	0 / 0 / 0	0 / 0 / 0
SDAXopen file	0 / 0 / 0	0 / 0 / 0	0 / 0 / 0
RHopen file	0 / 0 / 0	0 / 0 / 0	0 / 0 / 0
Page read	38 / 33 / 56	47 / 34 / 60	51 / 33 / 54
Scratch page I/O	9 / 8 / 50	10 / 7 / 47	6 / 8 / 21
Read-only page	0 / 0 / 0	0 / 0 / 0	0 / 0 / 0
Device mount	0 / 0 / 0	0 / 0 / 0	0 / 0 / 0
Buffer alloc/realloc	3 / 26 / 0	19 / 26 / 0	12 / 26 / 2
Writable page	0 / 0 / 0	0 / 0 / 0	0 / 0 / 0
TENEX read	33 / 28 / 56	43 / 29 / 60	42 / 28 / 45
Buffer swap	0 / 0 / 0	0 / 0 / 0	0 / 0 / 0
Page write	0 / 0 / 0	0 / 0 / 0	0 / 0 / 0
TENEX write	0 / 0 / 0	0 / 0 / 0	0 / 0 / 0
Create node	0 / 0 / 0	0 / 0 / 0	0 / 0 / 0
Create file/port	0 / 0 / 0	0 / 0 / 0	0 / 0 / 0
Generated code execution	98 / 98 / 92	98 / 98 / 96	97 / 98 / 97
RH close file	0 / 0 / 0	0 / 0 / 0	0 / 0 / 0
RH close port	0 / 0 / 0	0 / 0 / 0	0 / 0 / 0
SV close file	0 / 0 / 0	0 / 0 / 0	0 / 0 / 0

Percent Resource Utilization

Figure 4.3

Linear Queries

Component	Light Real/CPU/Pgflts	Medium Real/CPU/Pgflts	Heavy Real/CPU/Pgflts
SV calls	71 / 63 / 95	77 / 63 / 93	76 / 63 / 80
Error calls	0 / 0 / 0	0 / 0 / 0	0 / 0 / 0
Syntax analysis	0 / 0 / 0	0 / 0 / 0	0 / 0 / 0
Context recognition	0 / 0 / 0	0 / 0 / 0	0 / 0 / 0
Precompilation	0 / 0 / 0	0 / 0 / 0	3 / 0 / 3
Compilation	0 / 0 / 0	1 / 0 / 1	2 / 1 / 4
Instruction generation	1 / 2 / 0	1 / 2 / 0	1 / 2 / 2
RH open port	0 / 0 / 0	0 / 0 / 0	0 / 0 / 0
RH clean up	0 / 0 / 0	0 / 0 / 0	0 / 0 / 0
IR find	0 / 0 / 0	0 / 0 / 1	0 / 0 / 0
Hunk loop	0 / 0 / 0	0 / 0 / 1	0 / 0 / 0
Net I/O	1 / 0 / 0	0 / 0 / 0	0 / 0 / 0
SV open file	0 / 0 / 0	0 / 0 / 0	0 / 0 / 0
SDAX open file	0 / 0 / 0	0 / 0 / 0	0 / 0 / 0
RH open file	1 / 1 / 0	0 / 1 / 0	2 / 1 / 2
Page read	45 / 28 / 93	60 / 29 / 91	55 / 29 / 69
Scratch page I/O	0 / 0 / 0	0 / 0 / 0	0 / 0 / 0
Read-only page	0 / 0 / 0	0 / 0 / 0	1 / 0 / 0
Device mount	0 / 0 / 0	0 / 0 / 0	0 / 0 / 0
Buffer alloc/realloc	7 / 11 / 0	5 / 11 / 0	4 / 10 / 1
Writeable page	0 / 0 / 0	0 / 0 / 0	0 / 0 / 2
TENEX read	41 / 22 / 93	56 / 22 / 91	49 / 22 / 66
Buffer swap	3 / 5 / 0	2 / 5 / 0	1 / 5 / 0
Page write	0 / 0 / 0	0 / 0 / 0	0 / 0 / 0
TENEX write	0 / 0 / 0	0 / 0 / 0	0 / 0 / 0
Create node	0 / 0 / 0	0 / 0 / 0	0 / 0 / 0
Create file/port	0 / 0 / 0	0 / 0 / 0	0 / 0 / 0
Generated code execution	93 / 94 / 97	93 / 94 / 94	88 / 94 / 85
RH close file	0 / 0 / 1	0 / 0 / 0	2 / 0 / 2
RH close port	0 / 0 / 0	0 / 0 / 0	0 / 0 / 0
SV close file	0 / 0 / 0	0 / 0 / 0	2 / 0 / 2

three different tables are shown. In the tables, the rows represent different processing components in the Datacomputer. These correspond to the components described previously although some were omitted since they do not occur at all in these queries. Each major column of the table represents a loading level. The numbers separated by slashes indicate the percent of real time, cpu time and page faults attributed to that component. It is important to remember that components do overlap each other so it is not unreasonable for the percentages to add up to more than 100%.

The figures indicate that the bulk of resource utilization occurs in different components for the different kinds of queries. The restrictive inverted queries spend a large percent of their time in overhead. In fact, 78% of the time is associated with compilation, network I/O and opening and closing files. The nested inverted queries spend a great deal of time actually processing data. This includes a large amount of time setting up and processing inversions (IR find and Hunk loop). A full 98% of the time is spent in generated code which includes inversion and data processing. The linear queries exhibit similar behavior in that most of the time (93%) is spent running generated code. The difference is that the time is divided between reading data pages and examining the data as opposed to processing inversions.

Based on measurments made and reported in our previous technical report [CCA], we know that a page read on TENEX/TOPS-20 takes approximately 14 ms cpu time. This is obviously an important factor in restrictive inverted queries and nested inverted queries. Clearly, anything that can be done to reduce the cost of a page fault or reduce the number of page reads would be a desirable enhancement.

The results from the first kind of query indicate that the overhead of compilation and file opening and closing are prime candidates for enhancement. Our Bluefile measurements demonstrated that a significant part of the time opening and closing files was spent making directory pages read-only and writeable.

The previous set of figures do not give any indication of either how much actual time is spent in the various components or how many times each component is actually invoked. Figures 4.4 through 4.6 are resource usage tables for one query in each group. They are included to give a more concrete idea of the meaning of the percentages.

In the figures, the numbers in square brackets represent the number of times the particular component was invoked during the query. The numbers under the "Real" and "CPU"

Resource Utilization
Figure 4.4

Restrictive Inverted Retrievals (light load)

Component	Calls	Real<%>	CPU<%>	Pgflts<%>
SV calls	[61]	4197 <54>	/ 2360 <55>	/ 47 <29>
Error calls	[8]	304 < 3>	/ 201 < 4>	/ 1 < 0>
Syntax analysis	[1]	785 <10>	/ 174 < 4>	/ 2 < 1>
Context recognition	[1]	134 < 1>	/ 84 < 1>	/ 3 < 1>
Precompilation	[1]	736 < 9>	/ 272 < 6>	/ 21 <13>
Compilation	[1]	856 <11>	/ 466 <10>	/ 47 <29>
Instruction generation	[1]	1082 <14>	/ 774 <18>	/ 32 <20>
RH open port	[1]	295 < 3>	/ 97 < 2>	/ 4 < 2>
RH clean up	[1]	104 < 1>	/ 102 < 2>	/ 0 < 0>
IR find	[2]	318 < 4>	/ 123 < 2>	/ 5 < 3>
Hunk loop	[2]	305 < 3>	/ 136 < 3>	/ 8 < 5>
Net I/O	[12]	684 < 8>	/ 98 < 2>	/ 0 < 0>
SV open file	[2]	1565 <20>	/ 1024 <23>	/ 18 <11>
SDAX open file	[2]	990 <12>	/ 635 <14>	/ 9 < 5>
RH open file	[2]	1782 <23>	/ 1209 <28>	/ 19 <11>
Page read	[8]	535 < 7>	/ 153 < 3>	/ 10 < 6>
Scratch page I/O	[12]	213 < 2>	/ 141 < 3>	/ 11 < 6>
Read-only page	[19]	583 < 7>	/ 575 <13>	/ 0 < 0>
Device mount	[6]	471 < 6>	/ 309 < 7>	/ 7 < 4>
Buffer alloc/realloc	[96]	405 < 5>	/ 261 < 6>	/ 6 < 3>
Writeable page	[19]	553 < 7>	/ 550 <12>	/ 0 < 0>
TENEX read	[8]	483 < 6>	/ 121 < 2>	/ 9 < 5>
Buffer swap	[4]	14 < 0>	/ 14 < 0>	/ 0 < 0>
Generated code execution	[1]	3918 <51>	/ 2375 <55>	/ 54 <33>
RH close file	[2]	387 < 5>	/ 301 < 7>	/ 1 < 0>
RH close port	[1]	38 < 0>	/ 36 < 0>	/ 0 < 0>
SV close file	[2]	336 < 4>	/ 250 < 5>	/ 1 < 0>
Total Request	[1]	7635	/ 4267	/ 159

Resource Utilization

Figure 4.5

Nested Inverted Queries (light load)

Component	Calls	Real<%>	CPU<%>	Pgflts<%>
SV calls	[35867]	723594 <87>	/ 601445 <86>	/ 1864 <98>
Error calls	[8]	269 < 0>	/ 164 < 0>	/ 5 < 0>
WAIT TIME	[1]	3994 < 0>	/ 76 < 0>	/ 0 < 0>
Syntax analysis	[1]	571 < 0>	/ 120 < 0>	/ 0 < 0>
Context recognition	[1]	135 < 0>	/ 134 < 0>	/ 0 < 0>
Precompilation	[1]	215 < 0>	/ 206 < 0>	/ 0 < 0>
Compilation	[1]	638 < 0>	/ 622 < 0>	/ 2 < 0>
Instruction generation	[1]	1756 < 0>	/ 1585 < 0>	/ 2 < 0>
RH open port	[1]	103 < 0>	/ 43 < 0>	/ 2 < 0>
RH clean up	[1]	267 < 0>	/ 120 < 0>	/ 9 < 0>
IR find	[7448]	384655 <46>	/ 324444 <46>	/ 1017 <53>
Hunk loop	[3724]	28850 < 3>	/ 25358 < 3>	/ 1 < 0>
Net I/O	[11]	611 < 0>	/ 93 < 0>	/ 3 < 0>
SV open file	[2]	603 < 0>	/ 540 < 0>	/ 1 < 0>
SDAX open file	[2]	397 < 0>	/ 335 < 0>	/ 1 < 0>
RH open file	[2]	1412 < 0>	/ 884 < 0>	/ 5 < 0>
Page read	[12380]	351726 <42>	/ 258851 <37>	/ 1840 <97>
Read-only page	[15]	470 < 0>	/ 456 < 0>	/ 1 < 0>
Device mount	[3]	9 < 0>	/ 8 < 0>	/ 0 < 0>
Buffer alloc/realloc	[73336]	227168 <27>	/ 211002 <30>	/ 11 < 0>
Writeable page	[15]	466 < 0>	/ 457 < 0>	/ 0 < 0>
TENEX read	[12380]	310046 <37>	/ 219323 <31>	/ 1840 <97>
Buffer swap	[811]	3363 < 0>	/ 2515 < 0>	/ 0 < 0>
Generated code execution	[1]	820930 <99>	/ 690140 <99>	/ 1873 <99>
RH close file	[2]	523 < 0>	/ 340 < 0>	/ 13 < 0>
RH close port	[1]	288 < 0>	/ 60 < 0>	/ 9 < 0>
SV close file	[2]	402 < 0>	/ 278 < 0>	/ 9 < 0>
Total request	[1]	824538	/ 692953	/ 1886

Resource Utilization

Figure 4.6

Linear Queries (light load)

Component	Calls	Real<%>	CPU<%>	Pgflts<%>
SV calls	[1136]	54108 <72> /	34752 <65> /	1133 <91>
Error calls	[9]	293 < 0> /	189 < 0> /	4 < 0>
Syntax analysis	[1]	1147 < 1> /	167 < 0> /	2 < 0>
Context recognition	[1]	170 < 0> /	91 < 0> /	4 < 0>
Precompilation	[1]	708 < 0> /	269 < 0> /	24 < 1>
Compilation	[1]	1695 < 2> /	636 < 1> /	39 < 3>
Instruction generation	[1]	1539 < 2> /	1228 < 2> /	21 < 1>
RH open port	[1]	120 < 0> /	53 < 0> /	2 < 0>
RH clean up	[1]	204 < 0> /	114 < 0> /	5 < 0>
Net I/O	[14]	1135 < 1> /	130 < 0> /	3 < 0>
SV open file	[2]	816 < 1> /	539 < 1> /	8 < 0>
SDAX open file	[2]	377 < 0> /	316 < 0> /	4 < 0>
RH open file	[2]	1144 < 1> /	810 < 1> /	10 < 0>
Page read	[1107]	33448 <44> /	16089 <30> /	1107 <89>
Read-only page	[15]	466 < 0> /	455 < 0> /	1 < 0>
Device mount	[3]	30 < 0> /	11 < 0> /	1 < 0>
Buffer alloc/realloc	[4402]	6615 < 8> /	6138 <11> /	6 < 0>
Writeable page	[15]	456 < 0> /	453 < 0> /	0 < 0>
TENEX read	[1107]	29692 <39> /	12363 <23> /	1107 <89>
Buffer swap	[1076]	3288 < 4> /	3270 < 6> /	0 < 0>
Generated code execution	[1]	69144 <92> /	50262 <95> /	1141 <92>
RH close file	[2]	584 < 0> /	341 < 0> /	13 < 1>
RH close port	[1]	141 < 0> /	52 < 0> /	5 < 0>
SV close file	[2]	420 < 0> /	279 < 0> /	9 < 0>
Total request	[1]	74646 /	52805 /	1236

columns indicate the number of milliseconds spent in that particular component and the numbers in the "Pagflts" column represent the number of page faults taken in that component. The numbers enclosed in angle brackets indicate the percentage of the resource used by the component. All of these tables correspond to queries running in the lightly loaded situation.

The main conclusion that can be drawn from these results is that performance is dependent on two factors: overhead and processing time. Overhead is the critical factor in relatively fast transactions and processing time is important in transactions that reference large quantities of data. The processing time component is also very much influenced by the cost of page accesses in TENEX.

4.2 WES Updates

WES is a war games program in use in the ACCAT. It can be instructed to enter data into the Datacomputer as the game proceeds. This usage of the Datacomputer is significantly different from the usage previously described in the following ways:

1. The files are very small.

2. Transactions include OPENs, CLOSEs, MODEs, UPDATEs and APPENDs.

3. None of the files are inverted.

4. All requests are precompiled.

The precompiled requests that were measured were:

1. TWESASSIGN - This request copies data from the game into a temporary Datacomputer file. The file is initially empty.

2. LUPDTPOS - Updates the position of ships and aircraft if they are already in the database.

3. LHIST - Add records for previously unreported ship and aircraft to the trackhist file.

4. LPOSAPPEND - Append to the position file all records not found in the LUPDTPOS request.

5. LUNITUPDATE - Update the unit file.

6. LCONT - Update the contact file.

7. LCASR - Update the casualty file and the readiness file.

A large quantity of data was collected during WES runs under light and heavy system loading conditions. The results of these measurements are summarized in figures 4.7 through 4.14.

Resource Utilization in WES — Figure 4.7

Request: UNITUPDATE Load: LIGHT

Component	Calls	Real<%>	CPU<%>	Pgflts<%>
Services calls	[326]	29689<76> /	23481<74> /	113<94>
Error calls	[29]	1760< 4> /	1627< 5> /	6< 5>
Waiting time	[6]	6416<16> /	292< 0> /	0< 0>
Syntax analysis	[6]	13006<33> /	8061<25> /	69<57>
Instruction generation	[6]	6463<16> /	6356<20> /	0< 0>
RH cleanup	[6]	254< 0> /	246< 0> /	0< 0>
Network I/O	[24]	522< 1> /	514< 1> /	0< 0>
Services Open File	[18]	5934<15> /	5064<15> /	2< 1>
SDAX Open File	[18]	3750< 9> /	3105< 9> /	2< 1>
RH Open File	[12]	6015<15> /	5349<16> /	14<11>
Page Read	[98]	5403<13> /	2000< 6> /	80<66>
Read Only Page	[234]	9042<23> /	8623<27> /	0< 0>
Device Mount	[27]	59< 0> /	59< 0> /	0< 0>
Buffer Alloc/Realloc	[332]	1043< 2> /	1003< 3> /	0< 0>
Writeable Page	[234]	9057<23> /	8168<25> /	1< 0>
Tenex Read	[102]	4913<12> /	1086< 3> /	88<73>
Buffer Swap	[38]	123< 0> /	123< 0> /	0< 0>
Page Write	[12]	2845< 7> /	2305< 7> /	20<16>
Tenex Write	[24]	352< 0> /	281< 0> /	12<10>
Generated Code Execution	[6]	18771<48> /	16871<53> /	51<42>
RH Close File	[12]	11571<29> /	10685<33> /	25<20>
Services Close File	[18]	10751<27> /	9971<31> /	25<20>
Total Request	[6]	38664 /	31702 /	120

Resource Utilization in WES

Figure 4.8

Request: POSAPPEND Load: LIGHT

Component	Calls	Real<%>	CPU<%>	Pgflts<%>
Services calls	[504]	56936<75> /	44628<72> /	203<98>
Error calls	[36]	1703< 2> /	1556< 2> /	2< 0>
Waiting time	[6]	12< 0> /	12< 0> /	0< 0>
Syntax analysis	[6]	16732<22> /	9346<15> /	101<49>
Instruction generation	[6]	15177<20> /	14304<23> /	0< 0>
RH cleanup	[6]	495< 0> /	316< 0> /	2< 0>
Network I/O	[24]	252< 0> /	152< 0> /	0< 0>
Services Open File	[24]	7801<10> /	6459<10> /	0< 0>
SDAX Open File	[24]	5005< 6> /	4050< 6> /	0< 0>
RH Open File	[18]	8668<11> /	7250<11> /	1< 0>
Page Read	[132]	7092< 9> /	2585< 4> /	91<44>
Read Only Page	[438]	18693<24> /	16153<26> /	0< 0>
Device Mount	[30]	73< 0> /	73< 0> /	0< 0>
Buffer Alloc/Realloc	[376]	2193< 2> /	1225< 1> /	0< 0>
Writeable Page	[438]	17121<22> /	15218<24> /	0< 0>
Tenex Read	[204]	7506< 9> /	2128< 3> /	107<51>
Buffer Swap	[20]	65< 0> /	65< 0> /	0< 0>
Page Write	[48]	14596<19> /	12329<19> /	100<48>
Tenex Write	[132]	2051< 2> /	1754< 2> /	84<40>
Generated Code Execution	[6]	42896<56> /	37671<60> /	103<50>
RH Close File	[18]	32806<43> /	29188<47> /	102<49>
Services Close File	[24]	21992<29> /	19826<32> /	28<13>
Total Request	[6]	75491 /	61825 /	206

Resource Utilization in WES
Figure 4.9

Request: CASR Load: LIGHT

Component	Calls	Real<%>	CPU<%>	Pgflts<%>
Services calls	[331]	36819<79> /	27721<77> /	149<94>
Error calls	[30]	1654< 3> /	1615< 4> /	2< 1>
Waiting time	[6]	9027<19> /	539< 1> /	1< 0>
Syntax analysis	[6]	13997<30> /	8271<23> /	72<45>
Instruction generation	[6]	7310<15> /	6284<17> /	0< 0>
RH cleanup	[6]	251< 0> /	246< 0> /	1< 0>
Network I/O	[24]	448< 0> /	443< 1> /	0< 0>
Services Open File	[18]	6689<14> /	5228<14> /	1< 0>
SDAX Open File	[18]	4439< 9> /	3300< 9> /	1< 0>
RH Open File	[12]	6090<13> /	5042<14> /	6< 3>
Page Read	[79]	5001<10> /	1455< 4> /	59<37>
Read Only Page	[270]	11396<24> /	10049<28> /	0< 0>
Device Mount	[24]	63< 0> /	63< 0> /	0< 0>
Buffer Alloc/Realloc	[270]	891< 1> /	885< 2> /	0< 0>
Writeable Page	[270]	10759<23> /	9543<26> /	1< 0>
Tenex Read	[115]	6524<14> /	1254< 3> /	95<60>
Buffer Swap	[14]	48< 0> /	47< 0> /	0< 0>
Page Write	[24]	8273<17> /	6331<17> /	78<49>
Tenex Write	[66]	1077< 2> /	1034< 2> /	42<26>
Generated Code Execution	[6]	24376<52> /	20815<58> /	84<53>
RH Close File	[12]	17730<38> /	15225<42> /	78<49>
Services Close File	[18]	11771<25> /	10737<29> /	23<14>
Total Request	[6]	46142 /	35809 /	157

Resource Utilization in WES

Figure 4.10

Request: CONT Load: LIGHT

Component	Calls	Real<%>	CPU<%>	Pgflts<%>
Services calls	[326]	35304<77> /	27809<75> /	154<95>
Error calls	[30]	1586< 3> /	1513< 4> /	3< 1>
Waiting time	[6]	47< 0> /	46< 0> /	0< 0>
Syntax analysis	[6]	11717<25> /	8092<21> /	75<46>
Instruction generation	[6]	7751<17> /	7184<19> /	0< 0>
RH cleanup	[6]	399< 0> /	395< 1> /	0< 0>
Network I/O	[24]	361< 0> /	330< 0> /	0< 0>
Services Open File	[18]	5594<12> /	5186<14> /	2< 1>
SDAX Open File	[18]	3643< 8> /	3307< 8> /	2< 1>
RH Open File	[12]	6012<13> /	5124<13> /	2< 1>
Page Read	[74]	3945< 8> /	1302< 3> /	54<33>
Read Only Page	[270]	10515<23> /	9950<26> /	0< 0>
Device Mount	[24]	50< 0> /	51< 0> /	0< 0>
Buffer Alloc/Realloc	[260]	985< 2> /	983< 2> /	0< 0>
Writeable Page	[270]	10158<22> /	9412<25> /	1< 0>
Tenex Read	[110]	5710<12> /	1169< 3> /	95<58>
Buffer Swap	[14]	51< 0> /	51< 0> /	0< 0>
Page Write	[24]	8845<19> /	6363<17> /	83<51>
Tenex Write	[66]	1017< 2> /	959< 2> /	42<25>
Generated Code Execution	[6]	25435<55> /	21173<57> /	87<53>
RH Close File	[12]	18847<41> /	15517<41> /	85<52>
Services Close File	[18]	12581<27> /	11143<30> /	25<15>
Total Request	[6]	45476 /	37016 /	162

Resource Utilization in WES
Figure 4.11

Request: UNITUPDATE Load: HEAVY

Component	Calls	Real<%>	CPU<%>	Pgflts<%>
Services calls	[111]	35389<59> /	8550<73> /	244<59>
Error calls	[9]	4879< 8> /	572< 4> /	36< 8>
Waiting time	[2]	4619< 7> /	173< 1> /	14< 3>
Syntax analysis	[2]	21083<35> /	3071<26> /	165<40>
Instruction generation	[2]	9975<16> /	2224<18> /	64<15>
RH cleanup	[2]	1931< 3> /	145< 1> /	13< 3>
Network I/O	[8]	1907< 3> /	197< 1> /	12< 2>
Services Open File	[6]	8937<15> /	1905<16> /	54<13>
SDAX Open File	[6]	5895< 9> /	1149< 9> /	40< 9>
RH Open File	[4]	11246<18> /	2108<18> /	81<19>
Page Read	[35]	8116<13> /	974< 8> /	77<18>
Read Only Page	[78]	3794< 6> /	2956<25> /	1< 0>
Device Mount	[11]	408< 0> /	31< 0> /	4< 0>
Buffer Alloc/Realloc	[120]	2068< 3> /	362< 3> /	17< 4>
Writeable Page	[78]	4106< 6> /	2851<24> /	13< 3>
Tenex Read	[35]	6054<10> /	471< 4> /	57<13>
Buffer Swap	[15]	49< 0> /	49< 0> /	0< 0>
Page Write	[4]	3596< 6> /	646< 5> /	18< 4>
Tenex Write	[8]	271< 0> /	110< 0> /	8< 1>
Generated Code Execution	[2]	24096<40> /	6182<52> /	156<37>
RH Close File	[4]	10832<18> /	3576<30> /	60<14>
Services Close File	[6]	10331<17> /	3390<28> /	56<13>
Total Request	[2]	59260 /	11706 /	412

Resource Utilization in WES

Figure 4.12

Request: POSAPPEND Load: HEAVY

Component Component	Calls Calls	Real<%> Real<%>	CPU<%> CPU<%>	Pgflts<%> Pgflts<%>
Services calls	[172]	61556<69> /	16297<72> /	370<69>
Error calls	[12]	8570< 9> /	744< 3> /	62<11>
Waiting time	[2]	5< 0> /	5< 0> /	0< 0>
Syntax analysis	[2]	22651<25> /	3546<15> /	173<32>
Instruction generation	[2]	12553<14> /	4801<21> /	64<11>
RH cleanup	[2]	2054< 2> /	128< 0> /	17< 3>
Network I/O	[8]	1495< 1> /	71< 0> /	12< 2>
Services Open File	[8]	7086< 8> /	2336<10> /	44< 8>
SDAX Open File	[8]	4949< 5> /	1431< 6> /	31< 5>
RH Open File	[6]	11161<12> /	2615<11> /	66<12>
Page Read	[48]	9847<11> /	1019< 4> /	78<14>
Read Only Page	[146]	7338< 8> /	5598<24> /	2< 0>
Device Mount	[10]	374< 0> /	33< 0> /	4< 0>
Buffer Alloc/Realloc	[144]	3187< 3> /	455< 2> /	25< 4>
Writeable Page	[146]	9940<11> /	5368<23> /	23< 4>
Tenex Read	[72]	12071<13> /	921< 4> /	98<18>
Buffer Swap	[12]	38< 0> /	37< 0> /	0< 0>
Page Write	[16]	14986<16> /	4472<19> /	94<17>
Tenex Write	[44]	1146< 1> /	621< 2> /	35< 6>
Generated Code Execution	[2]	49599<56> /	14072<62> /	268<50>
RH Close File	[6]	33584<37> /	10488<46> /	182<33>
Services Close File	[8]	20462<23> /	7132<31> /	87<16>
Total Request	[2]	88465 /	22632 /	536

Resource Utilization in WES

Figure 4.13

Request: CASR Load: HEAVY

Component	Calls	Real<%>	CPU<%>	Pgflts<%>
Services calls	[112]	36577<62> /	9892<76> /	249<59>
Error calls	[10]	6990<11> /	625< 4> /	51<12>
Waiting time	[2]	5059< 8> /	236< 1> /	20< 4>
Syntax analysis	[2]	21339<36> /	3045<23> /	156<37>
Instruction generation	[2]	10391<17> /	2124<16> /	64<15>
RH cleanup	[2]	1288< 2> /	137< 1> /	13< 3>
Network I/O	[8]	967< 1> /	105< 0> /	9< 2>
Services Open File	[6]	5709< 9> /	1818<14> /	35< 8>
SDAX Open File	[6]	3892< 6> /	1137< 8> /	25< 5>
RH Open File	[4]	7384<12> /	1754<13> /	54<12>
Page Read	[28]	7218<12> /	608< 4> /	53<12>
Read Only Page	[90]	4018< 6> /	3423<26> /	1< 0>
Device Mount	[8]	336< 0> /	26< 0> /	3< 0>
Buffer Alloc/Realloc	[98]	1905< 3> /	322< 2> /	17< 4>
Writeable Page	[90]	3795< 6> /	3246<25> /	4< 0>
Tenex Read	[40]	6465<10> /	492< 3> /	57<13>
Buffer Swap	[7]	24< 0> /	24< 0> /	0< 0>
Page Write	[8]	5151< 8> /	2105<16> /	51<12>
Tenex Write	[22]	457< 0> /	295< 2> /	19< 4>
Generated Code Execution	[2]	24317<41> /	7488<57> /	172<40>
RH Close File	[4]	15507<26> /	5329<41> /	107<25>
Services Close File	[6]	11446<19> /	3856<29> /	58<13>
Total Request	[2]	58932 /	12931 /	420

Resource Utilization in WES

Figure 4.14

Request: CONT Load: HEAVY

Component	Calls	Real<%>	CPU<%>	Pgflts<%>
Services calls	[111]	37581<60> /	10003<73> /	236<59>
Error calls	[10]	6349<10> /	717< 5> /	46<11>
Waiting time	[2]	775< 1> /	34< 0> /	10< 2>
Syntax analysis	[2]	19796<32> /	2996<22> /	138<34>
Instruction generation	[2]	9554<15> /	2536<18> /	64<16>
RH cleanup	[2]	1538< 2> /	143< 1> /	16< 4>
Network I/O	[8]	1244< 2> /	207< 1> /	11< 2>
Services Open File	[6]	6526<10> /	1878<13> /	38< 9>
SDAX Open File	[6]	5091< 8> /	1209< 8> /	31< 7>
RH Open File	[4]	8848<14> /	1894<14> /	63<15>
Page Read	[27]	4697< 7> /	549< 4> /	48<12>
Read Only Page	[90]	5079< 8> /	3425<25> /	1< 0>
Device Mount	[8]	119< 0> /	22< 0> /	1< 0>
Buffer Alloc/Realloc	[96]	1865< 3> /	280< 2> /	17< 4>
Writeable Page	[90]	4789< 7> /	3288<24> /	14< 3>
Tenex Read	[39]	6132< 9> /	454< 3> /	57<14>
Buffer Swap	[7]	24< 0> /	24< 0> /	0< 0>
Page Write	[8]	7778<12> /	2191<16> /	51<12>
Tenex Write	[22]	769< 1> /	333< 2> /	17< 4>
Generated Code Execution	[2]	30163<48> /	7775<57> /	173<43>
RH Close File	[4]	19527<31> /	5461<40> /	101<25>
Services Close File	[6]	12261<19> /	3919<28> /	52<13>
Total Request	[2]	61779 /	13527 /	398

This data leads to a number of conclusions concerning Datacomputer usage and performance in the WES environment:

1. Due to the small size of the files used by WES, overhead assumes an even larger role in resource utilization than in the FC requests.

2. One large overhead component deals with disk I/O. Reading pages, writing pages, and changing the mode of pages consume on the order of 30-50% of the CPU and real time during typical WES requests.

3. Even though the requests are precompiled, syntax analyisis and converting the precompiled request into code consume around 40% of the real and CPU time in many cases.

4. Net I/O and error calls account for significant amounts of real time in requests that do little disk I/O such as MODEs and TWESASSIGN.

5. Enhancement Recommendations

The data obtained from these FC experiments and WES experiments and also the previous Bluefile data has been analyzed. The results of this analysis indicate that the major factor limiting Datacomputer performance is overhead. In most cases more time is spent in start up and compilation of a request than in actual execution. In addition, for requests that access large amounts of data, the cpu overhead incurred on each I/O operation becomes significant.

5.1 Enhancement Alternatives

Following is a list of potential enhancements aimed at reducing request overhead. As shown in the following section several of these enhancements have already been implemented.

1. Dedicated Datacomputer disk -- The Datacomputer's file system can be removed from the Tenex/Tops-20 file system. This will entail dedicating one disk

drive to the Datacomputer. Since the Datacomputer already maintains its own file system, it currently treats Tenex data files as logical disks. This imposes a large quantity of additional CPU overhead in dealing with data files. We estimate that giving the Datacomputer a dedicated disk could cut the CPU time for disk I/O operations in half. This enhancement would help FC retrievals primarily.

2. Removal of Directory Page Mode Changes -- This enhancement involves trading off performance against reliability. In the WES requests, a large amount of both CPU and real time was spent changing the mode of pages from read-only to writeable and back. This is done to keep the directory pages protected as much as possible. Experience has shown that the Datacomputer has reached a degree of reliability such that it is exceedingly unlikely that any code will "accidentally" write a directory page. In fact, the current system would get a pager interrupt if it attempted to write a page when it was read-only. This interrupt has never occurred because of trying to write into a directory page. Our measurements have shown that changing the mode of a page is very expensive in Tenex/Tops-20. It is obvious from the WES data

that this activity consumes a significant percent of the CPU and real time used by the Datacomputer. A dramatic decrease in the expense of these operations could yield a speed-up factor as high as two.

3. Inversion Improvements -- The Datacomputer's inverted file mechanism was designed to handle inversion of very large files. As a result, active inversion information is always kept in temporary disk files. This adds unnecessary overhead in many cases where the inversion is small. This enhancement entails modifying the inversion algorithms to substitute in-core tables for disk files whenever possible. This would improve performance of inverted retrievals against small databases like BLUEFILE.

4. Simplification of Buffer System -- The Datacomputer employs a rather complex buffer management scheme that includes numerous runtime checks to insure that buffers are properly allocated/deallocated. This includes making operating system calls to set the access mode of unallocated buffers to "no access". This was found to be an expensive operation. Although these checks are useful in a

test environment, they add unnecessary overhead during normal system operation. Modifying the buffer system so that these run time checks can be disabled will improve performance of small requests and provide a significant improvement in performance of the OPEN command.

5. Improving Precompilation -- The precompilation facility allows a user to partially compile a Datalanguage request and store it away in the Datacomputer. It can then be repeatedly invoked and executed with substantial savings of the compilation cost normally associated with a request. Although much of the compilation process is skipped when a precompiled request is executed, the code generation step must still be performed. This enhancement would expand the precompilation process to enable the generated code to be stored. This would yield a large performance improvement in the WES application which makes extensive use of precompiled requests.

6. Reduce Page Load Sensitivity -- Our analysis has indicated that Datacomputer performance is very sensitive to page loading. Several things can be done to reduce this sensitivity. First, the size

of the Datacomputer core image can be reduced by removing unnecessary subroutines. A number of Datacomputer features including file groups, mass memory interface and variable length strings are not being utilized in the ACCAT. These features could be removed without impacting Datacomputer usage in the ACCAT. Second, subroutines can be rearranged in the core image to improve locality of reference. By loading subroutines that are frequently executed together near each other in the core image, the number of page faults incurred during execution can be reduced.

5.2 Enhancement Implementation

At the end of July 1979, a new version of the Datacomputer was installed at NOSC. It included the following enhancments:

1. Dedicated Datacomputer disk -- The Datacomputer was modified to access a dedicated disk outside of the TENEX file system. This enhancement also requires modification of the TENEX operating system to provide the Datacomputer with direct access to the

disk. These modifications will be made by NOSC personnel. When this occurs, the new Datacomputer will be able to be tested.

2. Removal of Directory Page Mode Changes -- The directory pages mapped into the Datacomputer's core image are kept in read/write mode and changing their modes has been removed. As the WES performance comparisons (figures 5.1 through 5.8) indicate, this change has resulted in speedup factors of between 1.5 and 3.5.

3. Simplification of Buffer System -- This change has simplified the Datacomputer's buffer allocation and deallocation algorithm. The performance data indicates that this has decreased the time spent in the buffer system.

4. Increase in Table Sizes -- Although this change did not have an immediate performance affect, it will provide for long-term improvements. NOSC personnel have complained that the Datacomputer cannot handle very large Datalanguage request due to table size limits. This has forced them to split their requests on a number of occasions. The table size increase will permit larger requests to be written with an accompanying performance enhancement.

Figures 5.1 through 5.8 summarize the results of these enhancements to the WES updates.

Resource Utilization in WES (Enhanced) Figure 5.1

Request: UNITUPDATE Load: LIGHT

Component	Calls	Real<%>	CPU<%>	Pgflts<%>
Services calls	[324]	9033<50> /	6813<45> /	71<95>
Error calls	[29]	1640< 9> /	1500<10> /	0< 0>
Waiting time	[6]	10869<61> /	512< 3> /	0< 0>
Syntax analysis	[6]	4141<23> /	3084<20> /	41<55>
Instruction generation	[6]	6890<38> /	6546<43> /	0< 0>
RH cleanup	[6]	207< 1> /	206< 1> /	0< 0>
Network I/O	[24]	553< 3> /	454< 3> /	0< 0>
Services Open File	[18]	1025< 5> /	984< 6> /	1< 1>
SDAX Open File	[18]	767< 4> /	728< 4> /	1< 1>
RH Open File	[12]	2684<15> /	1929<12> /	9<12>
Page Read	[96]	3284<18> /	1428< 9> /	54<72>
Read Only Page	[234]	271< 1> /	272< 1> /	0< 0>
Device Mount	[27]	66< 0> /	66< 0> /	0< 0>
Buffer Alloc/Realloc	[324]	454< 2> /	450< 3> /	0< 0>
Writeable Page	[234]	256< 1> /	256< 1> /	0< 0>
Tenex Read	[100]	3098<17> /	1111< 7> /	56<75>
Buffer Swap	[36]	141< 0> /	142< 0> /	0< 0>
Page Write	[12]	861< 4> /	695< 4> /	14<18>
Tenex Write	[24]	286< 1> /	282< 1> /	12<16>
Generated Code Execution	[6]	6302<35> /	4940<33> /	33<44>
RH Close File	[12]	2500<14> /	2268<15> /	16<21>
Services Close File	[18]	2168<12> /	1950<13> /	16<21>
Total Request	[6]	17794 /	14946 /	74

Resource Utilization in WES (Enhanced) Figure 5.2

Request: POSAPPEND Load: LIGHT

Component	Calls	Real<%>	CPU<%>	Pgflts<%>
Services calls	[496]	17256<47> /	13014<44> /	168<98>
Error calls	[36]	2073< 5> /	1780< 6> /	5< 2>
Waiting time	[6]	12< 0> /	12< 0> /	0< 0>
Syntax analysis	[6]	5755<15> /	3816<12> /	77<45>
Instruction generation	[6]	16184<44> /	14240<48> /	0< 0>
RH cleanup	[6]	476< 1> /	246< 0> /	2< 1>
Network I/O	[24]	215< 0> /	214< 0> /	0< 0>
Services Open File	[24]	1279< 3> /	1173< 3> /	0< 0>
SDAX Open File	[24]	827< 2> /	746< 2> /	0< 0>
RH Open File	[18]	2566< 7> /	2097< 7> /	0< 0>
Page Read	[124]	3471< 9> /	1661< 5> /	75<44>
Read Only Page	[438]	597< 1> /	519< 1> /	0< 0>
Device Mount	[30]	69< 0> /	67< 0> /	0< 0>
Buffer Alloc/Realloc	[344]	661< 1> /	521< 1> /	0< 0>
Writeable Page	[438]	506< 1> /	488< 1> /	0< 0>
Tenex Read	[196]	4272<11> /	2133< 7> /	81<47>
Buffer Swap	[12]	80< 0> /	80< 0> /	0< 0>
Page Write	[48]	5710<15> /	4206<14> /	90<52>
Tenex Write	[132]	2122< 5> /	1599< 5> /	84<49>
Generated Code Execution	[6]	13599<37> /	11021<37> /	91<53>
RH Close File	[18]	10053<27> /	8157<27> /	91<53>
Services Close File	[24]	5386<14> /	4586<15> /	25<14>
Total Request	[6]	36374 /	29555 /	170

Resource Utilization in WES (Enhanced) Figure 5.3

Request: CASR Load: LIGHT

Component	Calls	Real<%>	CPU<%>	Pgflts<%>
Services calls	[329]	11293<56> /	8069<50> /	114<97>
Error calls	[30]	1817< 9> /	1582< 9> /	0< 0>
Waiting time	[6]	10731<53> /	480< 2> /	0< 0>
Syntax analysis	[6]	4670<23> /	3084<19> /	50<42>
Instruction generation	[6]	6764<34> /	6502<40> /	0< 0>
RH cleanup	[6]	201< 1> /	198< 1> /	0< 0>
Network I/O	[24]	687< 3> /	494< 3> /	0< 0>
Services Open File	[18]	960< 4> /	935< 5> /	0< 0>
SDAX Open File	[18]	628< 3> /	607< 3> /	0< 0>
RH Open File	[12]	1781< 8> /	1569< 9> /	4< 3>
Page Read	[77]	2722<13> /	1062< 6> /	49<41>
Read Only Page	[270]	322< 1> /	314< 1> /	0< 0>
Device Mount	[24]	59< 0> /	60< 0> /	0< 0>
Buffer Alloc/Realloc	[262]	394< 1> /	387< 2> /	0< 0>
Writeable Page	[270]	290< 1> /	290< 1> /	0< 0>
Tenex Read	[113]	4143<20> /	1240< 7> /	70<59>
Buffer Swap	[12]	58< 0> /	57< 0> /	0< 0>
Page Write	[24]	3597<18> /	2258<14> /	63<53>
Tenex Write	[66]	913< 4> /	879< 5> /	42<35>
Generated Code Execution	[6]	7869<39> /	6159<38> /	67<57>
RH Close File	[12]	5768<29> /	4273<26> /	63<53>
Services Close File	[18]	2709<13> /	2339<14> /	17<14>
Total Request	[6]	19874 /	16124 /	117

Resource Utilization in WES (Enhanced) Figure 5.4

Request: CONT Load: LIGHT

Component	Calls	Real<%>	CPU<%>	Pgflts<%>
Services calls	[324]	11064<50> /	8363<47> /	115<97>
Error calls	[30]	1895< 8> /	1802<10> /	6< 5>
Waiting time	[6]	65< 0> /	65< 0> /	0< 0>
Syntax analysis	[6]	4570<20> /	3110<17> /	50<42>
Instruction generation	[6]	8775<39> /	7644<42> /	0< 0>
RH cleanup	[6]	216< 0> /	201< 1> /	0< 0>
Network I/O	[24]	493< 2> /	467< 2> /	0< 0>
Services Open File	[18]	1112< 5> /	1046< 5> /	0< 0>
SDAX Open File	[18]	883< 4> /	817< 4> /	0< 0>
RH Open File	[12]	1902< 8> /	1727< 9> /	0< 0>
Page Read	[72]	2116< 9> /	1001< 5> /	45<38>
Read Only Page	[270]	317< 1> /	316< 1> /	0< 0>
Device Mount	[24]	48< 0> /	48< 0> /	0< 0>
Buffer Alloc/Realloc	[252]	436< 1> /	405< 2> /	0< 0>
Writeable Page	[270]	317< 1> /	287< 1> /	0< 0>
Tenex Read	[108]	3283<14> /	1200< 6> /	71<60>
Buffer Swap	[12]	56< 0> /	55< 0> /	0< 0>
Page Write	[24]	3292<14> /	2129<11> /	68<57>
Tenex Write	[66]	973< 4> /	857< 4> /	42<35>
Generated Code Execution	[6]	8095<36> /	6511<36> /	68<57>
RH Close File	[12]	5841<26> /	4451<25> /	68<57>
Services Close File	[18]	3263<14> /	2674<15> /	20<16>
Total Request	[6]	22004 /	17784 /	118

Resource Utilization in WES (Enhanced) Figure 5.5

Request: UNITUPDATE Load: HEAVY

Component	Calls	Real<%>	CPU<%>	Pgflts<%>
Services calls	[111]	11825<58> /	2431<45> /	123<50>
Error calls	[9]	1020< 5> /	496< 9> /	13< 5>
Waiting time	[2]	8763<43> /	150< 2> /	15< 6>
Syntax analysis	[2]	10192<50> /	1198<22> /	124<51>
Instruction generation	[2]	5080<25> /	2174<41> /	60<24>
RH cleanup	[2]	336< 1> /	75< 1> /	5< 2>
Network I/O	[8]	322< 1> /	174< 3> /	2< 0>
Services Open File	[6]	942< 4> /	330< 6> /	9< 3>
SDAX Open File	[6]	869< 4> /	257< 4> /	9< 3>
RH Open File	[4]	1821< 8> /	644<12> /	17< 6>
Page Read	[35]	3718<18> /	678<12> /	54<22>
Read Only Page	[78]	97< 0> /	89< 1> /	0< 0>
Device Mount	[11]	21< 0> /	21< 0> /	0< 0>
Buffer Alloc/Realloc	[120]	328< 1> /	171< 3> /	3< 1>
Writeable Page	[78]	136< 0> /	82< 1> /	2< 0>
Tenex Read	[35]	3559<17> /	514< 9> /	52<21>
Buffer Swap	[15]	57< 0> /	57< 1> /	0< 0>
Page Write	[4]	612< 3> /	189< 3> /	12< 4>
Tenex Write	[8]	175< 0> /	102< 1> /	6< 2>
Generated Code Execution	[2]	4617<22> /	1817<34> /	54<22>
RH Close File	[4]	1785< 8> /	713<13> /	30<12>
Services Close File	[6]	1680< 8> /	608<11> /	30<12>
Total Request	[2]	20264 /	5302 /	243

Resource Utilization in WES (Enhanced) Figure 5.6

Request: POSAPPEND Load: HEAVY

Component	Calls	Real<%>	CPU<%>	Pgflts<%>
Services calls	[172]	55978<74> /	5326<46> /	429<72>
Error calls	[12]	6525< 8> /	653< 5> /	69<11>
Waiting time	[2]	5< 0> /	4< 0> /	0< 0>
Syntax analysis	[2]	10764<14> /	1543<13> /	117<19>
Instruction generation	[2]	9893<13> /	4753<41> /	51< 8>
RH cleanup	[2]	1170< 1> /	124< 1> /	16< 2>
Network I/O	[8]	429< 0> /	76< 0> /	7< 1>
Services Open File	[8]	3277< 4> /	537< 4> /	61<10>
SDAX Open File	[8]	2690< 3> /	420< 3> /	47< 7>
RH Open File	[6]	12533<16> /	967< 8> /	138<23>
Page Read	[48]	5725< 7> /	836< 7> /	77<13>
Read Only Page	[146]	486< 0> /	179< 1> /	6< 1>
Device Mount	[10]	82< 0> /	23< 0> /	2< 0>
Buffer Alloc/Realloc	[144]	4506< 5> /	216< 1> /	10< 1>
Writeable Page	[146]	844< 1> /	196< 1> /	9< 1>
Tenex Read	[72]	20589<27> /	1004< 8> /	110<18>
Buffer Swap	[12]	214< 0> /	50< 0> /	4< 0>
Page Write	[16]	21453<28> /	1618<14> /	126<21>
Tenex Write	[44]	1388< 1> /	558< 4> /	38< 6>
Generated Code Execution	[2]	53142<70> /	4859<42> /	395<66>
RH Close File	[6]	35070<46> /	3041<26> /	223<37>
Services Close File	[8]	9299<12> /	1664<14> /	97<16>
Total Request	[2]	75615 /	11394 /	590

Resource Utilization in WES (Enhanced) Figure 5.7

Request: CASR Load: HEAVY

Component	Calls	Real<%>	CPU<%>	Pgflts<%>
Services calls	[112]	23314<69> /	3077<51> /	205<60>
Error calls	[10]	15035<44> /	609<10> /	37<10>
Waiting time	[2]	3794<11> /	138< 2> /	17< 5>
Syntax analysis	[2]	11194<33> /	1314<22> /	173<51>
Instruction generation	[2]	6673<19> /	2125<35> /	60<17>
RH cleanup	[2]	609< 1> /	157< 2> /	8< 2>
Network I/O	[8]	1048< 3> /	158< 2> /	4< 1>
Services Open File	[6]	1121< 3> /	333< 5> /	21< 6>
SDAX Open File	[6]	809< 2> /	252< 4> /	15< 4>
RH Open File	[4]	1681< 5> /	483< 8> /	18< 5>
Page Read	[28]	6048<18> /	491< 8> /	49<14>
Read Only Page	[90]	100< 0> /	101< 1> /	0< 0>
Device Mount	[8]	117< 0> /	24< 0> /	3< 0>
Buffer Alloc/Realloc	[98]	466< 1> /	200< 3> /	5< 1>
Writeable Page	[90]	241< 0> /	99< 1> /	2< 0>
Tenex Read	[40]	4806<14> /	544< 9> /	61<18>
Buffer Swap	[7]	204< 0> /	28< 0> /	2< 0>
Page Write	[8]	2838< 8> /	715<12> /	41<12>
Tenex Write	[22]	351< 1> /	265< 4> /	16< 4>
Generated Code Execution	[2]	15005<44> /	2289<38> /	96<28>
RH Close File	[4]	12424<37> /	1449<24> /	73<21>
Services Close File	[6]	9995<29> /	841<14> /	41<12>
Total Request	[2]	33520 /	5924 /	337

Resource Utilization in WES (Enhanced) Figure 5.8

Request: CONT Load: HEAVY

Component	Calls	Real<%>	CPU<%>	Pgflts<%>
Services calls	[111]	23407<54> /	3273<48> /	238<56>
Error calls	[10]	9607<22> /	740<10> /	56<13>
Waiting time	[2]	337< 0> /	34< 0> /	10< 2>
Syntax analysis	[2]	6405<14> /	1226<18> /	111<26>
Instruction generation	[2]	6817<15> /	2558<38> /	69<16>
RH cleanup	[2]	6271<14> /	149< 2> /	18< 4>
Network I/O	[8]	510< 1> /	200< 2> /	10< 2>
Services Open File	[6]	9789<22> /	499< 7> /	63<15>
SDAX Open File	[6]	8455<19> /	398< 5> /	49<11>
RH Open File	[4]	13447<31> /	735<10> /	103<24>
Page Read	[27]	2933< 6> /	388< 5> /	43<10>
Read Only Page	[90]	1133< 2> /	105< 1> /	1< 0>
Device Mount	[8]	113< 0> /	17< 0> /	1< 0>
Buffer Alloc/Realloc	[96]	484< 1> /	68< 1> /	6< 1>
Writeable Page	[90]	286< 0> /	101< 1> /	4< 0>
Tenex Read	[39]	4023< 9> /	500< 7> /	53<12>
Buffer Swap	[7]	331< 0> /	34< 0> /	3< 0>
Page Write	[8]	3186< 7> /	765<11> /	42<10>
Tenex Write	[22]	370< 0> /	273< 4> /	16< 3>
Generated Code Execution	[2]	22631<52> /	2706<40> /	207<49>
RH Close File	[4]	7505<17> /	1634<24> /	88<20>
Services Close File	[6]	4522<10> /	1023<15> /	50<11>
Total Request	[2]	42847 /	6728 /	420

6. Conclusions

The instrumentation of the Datacomputer for this project and the programs to analyze the resulting data have proved very useful in determining the processing bottlenecks in the Datacomputer. This analysis technique was also utilized when the Datacomputer was being considered by the Army in the Fort Bragg experiments. The tools enabled CCA to produce results in a rapid fashion to questions about Datacomputer performance.

The results of the experiments performed on WES requests and the FC database in the ACCAT indicate that there are few enhancements beyond the kind related to the reduction of overhead that would further improve Datacomputer performance in the ACCAT. However, careful analysis of the final results of the measurement did yield a substantial improvement for a relatively small investment of resources.

References

[CCA]
Computer Corporation of America, A Distributed Database Management System for Command and Control Applications: Semi-Annual Technical Report 3, Technical Report No. CCA-78-10, Computer Corporation of America, 575 Technology Square, Cambridge, Massachusetts 02139.

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

[ROTHNIE et al.]
J. B. Rothnie, Jr., P. A. Bernstein, S. Fox, N. Goodman M. Hammer, T. A. Landers, C. Reeve, D. Shipman, E. Wong "SDD-1: A System for Distributed Databases", ACM Transactions on Database Systems, Vol. 5, No. 1 (March 1980), pp. 1-17.