THE DEVELOPMENT AND IMPLEMENTATION OF A PERFORMANCE DATABASE SERVER

Brian Howard LaRose

Computer Science Department

CS-93-195 August 1993

The Development and Implementation of a Performance Database Server

A Thesis

Presented for the

Master of Science Degree

The University of Tennessee, Knoxville

Brian Howard LaRose

August 1993

Dedication

To John and Bess Dunning, Ed and Melba LaRose and Mom, Dad and Sherri.

Acknowledgements

I am indebted to my loving wife Ginger for her understanding and her support throughout my work on this and other projects. Her support is constant and strong and without it I would not succeed. I thank Jack Dongarra for keeping this project pointed in the right direction. I thank Michael Greene for being my friend through this effort and keeping me pointed in the right direction. I thank God for blessing me with the ability and the desire to continue achieving my goals. I would also like to extend my deepest thanks for my advisor, Michael Berry. His support and ideas along with his personal touch have meant much to me and continue to be an inspiration to my work. I am here but for the support and love of my parents. Their efforts to put their children first have set a model for my life for that I am thankful.

This research was supported by NASA under grant number NAG–5–2083, and by ARPA under grant number DAAL03–92–G–0284.

Abstract

The process of gathering, archiving, and distributing computer benchmark data is a cumbersome task usually performed by computer users and vendors with little coordination. Most importantly, there is no publicly-available central depository of performance data for all ranges of machines: supercomputers to personal computers. We present an Internet-accessible performance database server (PDS) which can be used to extract current benchmark data and literature. As an extension to the X-Windows-based user interface (Xnetlib) to the Netlib archival system, PDS provides an on-line catalog of public-domain computer benchmarks such as the Linpack Benchmark, Perfect Benchmarks, and the Genesis benchmarks. PDS does not reformat or present the benchmark data in any way which conflicts with the original methodology of any particular benchmark, and is thereby devoid of any subjective interpretations of machine performance. We feel that all branches (academic and industrial) of the general computing community can use this facility to archive performance metrics and make them readily available to the public. PDS can provide a more manageable approach to the development and support of a large dynamic database of published performance metrics.

Contents

List of Figures

Chapter 1

Overview of Benchmarking

1.1 Introduction

Given the current rate of change in computer technology, high-performance machines purchased today may well be obsolete within 5 years, if not before. The new advancements often lead to solving larger problems in smaller amounts of time. In order to monitor progress in such problems, we may attempt to compare the performance of various machines quantitatively with *benchmarks*. Although benchmarking has become very popular because of the diversity and competition in the computer hardware business, there are only a limited number of libraries or archive sites for benchmark data. The real need for a central repository of performance metrics is becoming a major concern.

1.2 Definition of Benchmarking

The term *benchmark* first appeared in the 1840's and is derived from surveying studies. A benchmark was an elevation mark on a permanent object to be used as a reference in topological surveys [Webs83]. This term became a standard in the surveys of North America, but within the context of the computer performance, the term has come to find new meaning. The Random House Dictionary [Rand87] defines a benchmark as "an established point of reference against which computers or programs can be measured in tests comparing their performance". For an objective comparison of machines, benchmark numbers tend to become permanent marks by which others can be measured. Machines are most often compared by reviewing their performance on a number of well-defined programs or benchmarks [Hock91].

A benchmark code is a program designed to be run on an architecture and produce a relative measure of its execution. Measures of execution are often scalar values, such as: the number of instructions executed compared with some predefined control system run, the number of instructions completed in a fixed time interval, the time required to execute a certain set of instructions, or many other metrics. Scientific benchmarks typically involve program segments that are floating-point intensive [BeCL91], so that in addition to measuring and reporting execution (elapsed or CPU) times, scientific benchmarks often report the rates at which floating-point operations were performed in Mflops/sec.[1]

Performance metrics can also be multi-valued functions [Hock92]. The Perfect Benchmarks [Berr89] produce 13 distinct values for each run. Furthermore, the Perfect methodology allows four modes of runs each of which produce 13 values. Perfect can be run to measure baseline wall-clock time, or baseline cpu (Mflops/sec) and users can also optimize the codes to obtain optimized wall-clock time or optimized

[1] Millions of floating-point operations per second.

Machine	ADM base wall-clock time	ARC2D base wall-clock time	BDNA base wall-clock time
Cray Y-MP/C90-1	13.941 sec	4.825 sec	5.289 sec
Fujitsu VP2600/10	17.240 sec	2.860 sec	3.600 sec

Figure 1.1: Sample of Perfect data.

cpu (Mflops/sec). Hence, this leads to a variety of numbers to interpret and rank order machines. Lets consider a few of the Perfect Benchmarks [Berr89] to illustrate this problem. As shown in Figure 1.1 the baseline wall-clock time for ADM when run on the Cray Y-MP/C90-1 is faster than that of the Fujitsu VP2600/10 configuration. However, if you compare ARC2D and the BDNA baseline wall-clock times for these machines, you can see that the Fujitsu is the faster machine. This might appear to be an inconsistency in the data, but it is not. This is a good example of different machine configurations performing differently on application-based benchmarks. It could be that the ADM program exhibits some characteristic which the Cray Y-MP/C90-1 can perform faster than the Fujitsu, however, ARC2D and BDNA may exhibit some characteristics which the Fujitsu can perform faster than the Cray. This is a common occurrence when comparing performance metrics. While some computer vendors concentrate on building faster floating-point processors, others will build faster integer processors, and some try to achieve an even balance. Perhaps the best indicator of expected target system performance on a certain application is to run the specific application on the target architecture. Because porting the code to the target is not always feasible, benchmarks exist to attempt to describe general system performance.

1.3 Complexity of Benchmarks

In the changing world of computer technology, there are hundreds of different architectures available. Subsequently, there are millions of application programs that have been written for solving problems arising in both academic and industrial settings. Programmers are typically concerned about the performance of a given computer when running their specific application. Many users want a computer that will perform well for their general application type, but do not necessarily care about other applications. This, of course, leads to different views on which system parameters are important. Thus, benchmarks tend to evolve from individual applications which may not stress all features of a given architecture.

Just as there are hundreds of different vendors in the industry, there are often several compilers for each vendor's machine. There are compilers which optimize, vectorize, and parallelize, and while such compilers are necessary, they complicate the amount of performance data one may accumulate. In theory, every benchmark code should be run with every compiler version on every available machine in order to completely cover all the possibilities. This, however, is simply not possible, and we instead make generalizations about performance based on available data.

The number of combinations of architectures, machines, applications, compilers, benchmarks, and performance metrics is staggering. In this study, we have accumulated over 3 megabytes of performance data in ASCII format in just over 4 months. The need for standardized benchmark methodologies and measures of performance will grow as the amount of data becomes overwhelming.

In future benchmarking efforts, there must be an attempt to describe the composition of the benchmarks. Running some specific codes which return a number for a machine does not imply a survey of general machine performance. These numbers instead describe the performance of the machine on the algorithm or application class, not all usage patterns. The composition of the benchmark codes is a very important matter which is often overlooked. Users who read read performance data extrapolate rank orderings from specific benchmark usage and imply general system performance rankings. The first statement in the Linpack Report [Dong88] addresses this concern: "The timing information presented here should in no way be used to judge the overall performance of a computer system. The results reflect only one problem area: solving dense systems of equations." This statement clearly lays out the intent of the benchmark and its characteristic usage; however, in practice this statement is rarely applied.

The Author of the Bonnie Benchmark Disk Suite [Bray93], Tim Bray of the University of Waterloo, states his intentions in the Bonnie introduction. "The author wishes to go on record that he feels this is relatively low-quality information, and that as presented it probably misrepresents, in some fashion, the performance of every manufacturer named herein. For heaven's sake, don't use this as a guide in a procurement exercise, and if you do, don't say I told you to".

Chapter 2

Benchmarking today

2.1 Benchmark Organizations

Because of the diversity of computing and the numerous metrics of system performance, there are very few comprehensive public domain (free) benchmark results. A few businesses, such as the Transaction Processing Council (TPC), have evolved to distribute performance metrics at cost. Other than these organizations, broad ranging metrics are not widely available. Although the major hardware vendors release metrics to the public, it is often better that a company publishes only the best numbers for a particular code and restrict the release of other intermediate numbers. As Weicker describes in [Weic91], small benchmarks typically make a machine look better, and therefore are more popular with marketing departments.

The Standard Performance Evaluation Corporation (SPEC) was formed to try to bridge the gaps in performance metrics by specifying a standard set of metrics that will be used to describe system performance. SPEC is a non-profit organization whose common goal is to provide the industry with a realistic yardstick to measure the performance of advanced computer systems through the education of the user community [Dixi90]. Some of the SPEC membership includes: Apple, AT&T, NCR, Bull, Compaq, Control Data, Data General, DEC, Fujitsu, Hal Computer, Hewlett-Packard, IBM, Intel, Intergraph, MIPS, Motorola, NeXT, Prime, Siemens Nixdorf, Silicon Graphics, Solbourne, Sun, and Unisys [Unie89].

As SPEC has gained acceptance both through usage and recognition by vendors and customers, it has become a requested metric. A good portion of the questions posted to the Usenet news group *comp.benchmarks* are related to SPEC and SPEC numbers. Many potential customers who need *SPECmarks* [Unie89] request them from the hardware vendors themselves. This seems a bit like asking the grocer *how fresh is your fruit?*

In the past few years a movement to develop and release public domain performance data over the Internet has gained some momentum. Internet news groups such as *comp.benchmarks* and *comp.parallel* have provided valuable postings and related discussions of performance data. In the 1980's the National Institute of Standards and Technology (NIST) attempted to collect and make performance data available through *ftp* over the Internet. Recently some Internet *ftp* sites have been established; yet the amount of performance data tends still to be sparse, and the interfaces are quite limited.

2.2 Presentation Standards

Currently there are as many presentation formats as there are benchmarks. Linpack [Dong88] presents 3 numbers: n=100 Mflops/sec, n=1000 Mflops/sec and theoretical peak performance numbers. The first two numbers reflect the solution of a dense linear system of equations of order 100 and 1000 respectively. Theoretical peak performance is defined as the absolute upper bound on performance, which is enforced by the architecture limits.

The Perfect Benchmarks [Berr89] are in a very different format than Linpack. Refer to Figure 1.1 for a small sample of some Perfect data. Perfect is a set of 13 scientific and engineering application programs which are individually run and measured for elapsed CPU time and Mflops/sec. Each program in the Perfect Benchmarks can be run in optimized or unoptimized mode, and hence, 52 separate numbers can be produced for each machine configuration. Perfect and Linpack are just examples of the diverse presentation formats that one may encounter in the benchmark field. Nonetheless, these presentation formats are derived from the very nature of the benchmarks themselves, and should be preserved.

Chapter 3

Taxonomy of Benchmarks

3.1 Classification by Type

Although there are numerous performance metrics, we may classify them [BeCL91] into four major categories: synthetic, kernel, algorithm, and application.

Synthetic Benchmarks.

Synthetic benchmarks are not representative of any real computation, rather, they exercise various basic machine functions. IOZone, a package written by Bill Norcott[1], primarily tests disk throughput by *stress testing* the reading and writing of very large data files [Norc92]. Dhrystones and Whetstones [Weic91] are examples of once-popular synthetic benchmarks which are rarely used today. Dhrystones were designed to stress integer performance and use many string operations. The use of Whetstones is on the decline because they prevent vectorization and various compiler-based optimizations.

[1] norcott_bill@tandem.com

Kernel-based Benchmarks.

Kernel-based benchmarks contain sections or *kernels* of a sample application code. A large library of routines with many different functions may be characterized by a small code sample. An example might be a loop that is processed millions of times in the application. The Livermore Loops [McMa86] are representative of this type of benchmark. These benchmark programs contain intensive floating-point operations, and typically stress a single functional unit of the hardware.

Algorithm Benchmarks.

Algorithm-based benchmarks are implementations of well-defined algorithms that vary slightly over different platforms. Algorithms that have been optimized are implemented in that optimized format. Because of the large number of operations typically processed, however, small variations in specific implementations or machine-specific calls can be masked in the long run. Thus, barring new optimizations or radically new approaches, these benchmarks give consistent measures of performance over various platforms and implementations. Examples of algorithm benchmarks are LINPACK [HoPa87], Slalom [Gust91a], and the NAS Fortran kernels [BaBa85].

Application Benchmarks.

Application benchmarks may be complete samples of engineering, scientific or business applications. These applications typically stress several functional groups of the hardware. The Perfect Benchmarks [Berr89] are examples of application benchmarks. This benchmark suite comprises 13 different scientific and engineering applications that can be run in predefined configurations. Such benchmarks are especially interesting to scientists whose research may closely resemble that modeled by the benchmarks. Application benchmarks are the closest performance estimation to actually running a real engineering application on candidate hardware.

3.2 Usage of Popular Benchmarks

Benchmarks are usually targeted at a certain operating environment. There is an attempt to describe operating environments and to address performance concerns within such environments. Some benchmarks are indicative of workstation or personal computer (PC) environments, while others are intended for mainframes or supercomputers. The Dhrystones, for example, were designed to test integer performance, spend significant time in string operations, and are therefore considered more representative of a workstation environment. The NAS parallel benchmarks [Bail91], on the other hand, are intensively parallel and considered representative of a multi-processor system environment.

Peripheral devices and local area networks (LANs), however, generally lack adequate benchmarks. Two notable exceptions are the IOZone benchmark and the Bonnie Benchmark Disk suite, which both could be classified as a *peripheral* benchmarks since they test the disk performance and I/O bandwidth. Given the the growing interest in distributed programming environments such as PVM [BDGM91] and Linda [CaGe92], we anticipate benchmarks for homogeneous and heterogeneous networks of machines in the near future. Figure 3.1 illustrates the typcial usage of several popular benchmark suites.

Benchmark Name	Target Machines: Workstations/ Personal Computers	Supercomputers/ Parallel Computers	Peripheral Devices
Linpack	**X**	**X**	
Perfect		**X**	
SPEC	**X**		
IOZone	**X**		**X**
Dhrystones	**X**		
Livermore Loops	**X**	**X**	
Slalom	**X**	**X**	
Flops	**X**		
Whetstones	**X**	**X**	
NAS Parallel		**X**	

Figure 3.1: Classification by benchmark usage.

Chapter 4

Motivation

4.1 Introduction

Given the current evolution of computer hardware technology, computer vendors are consistently producing more advanced versions of current machines as well as introducing new architectures which can cause leaps in system performance. This seemingly *exponential* growth in machine performance is accompanied by new varieties of computer benchmarks to track this growth [Gust91a]. Until recently, serial benchmarks ([Weic91]) have been the primary available measures of processor performance. With the advent of parallel benchmarks ([Bail91], [Gust91a]), the complexity of benchmark acquisition and presentation will certainly increase. Recent meetings of the Parallel Benchmark Working Group (PBWG), led by Professor Roger Hockney at the University of Southampton, England, have led to proposals on benchmark methodologies, benchmark standards and classifications. These proposal drafts should provide an open forum for discussions on the desemination of performance metrics.

For example, classifications of parallel benchmarks may be based on communication characteristics, processor utilization and load balancing, data layout or mappings, and even parallel I/O constructs. Hence, the number of variations of a *single* parallel benchmark program can be large. The ability to store, organize, and disseminate credible computer benchmark data is of paramount importance if we are to categorize the performance of computers ranging from laptop computers (e.g., Apple Powerbook) to massively parallel machines (e.g., CM-5).

The Performance Database Server (PDS) developed at the University of Tennessee and Oak Ridge National Laboratory is an initial attempt at performance data management. This on-line database of computer benchmarks is specifically designed to provide easy maintenance, data security, and data integrity in the benchmark information contained in a *dynamic* performance database.

4.2 Initial PDS goals

The primary goal of this research is the development of a performance database. Our objective is the accumulation, classification, and distribution of benchmark data acquired from industry and academia. The database server will be centrally located on the Internet and will serve data to the remote sites. The distribution of the data is more important than the format or details of the distribution. Thus, users should be able to access specific data meaningful to their application or environment. Engineers, for example, should be able to view performance metrics from machines which were running *typical* engineering applications. Users interested in certain machines can view data on only those machines, if they so choose. PDS can serve performance data to clients in whatever format they wish from a central database.

Therefore, a database was built which contains performance metrics from many published sources. Initially, we could only include public domain data which had been published, such as the Perfect Benchmarks [Berr89], the Linpack benchmark [Dong88] and the NAS Parallel benchmarks [BaBa85].

The PDS interface was designed to provide users with easy access to a massive amount of on-line benchmark metrics. The access has to be general enough to allow the users to access data in whatever format they wish. The *friendly* interface which we desired could be easily accomplished by writing a X-windows graphical user interface to the database.

We designed a Graphical User Interface (GUI) for the database using a *client-server* [BiNe84] model for network computing. The database is centralized in one location for easy management and update, but the GUI client can be distributed over many sites. The interface would give the appearance that the data was local to the user.

The Xnetlib tool [DoRW93] developed at Oak Ridge National Laboratory and the University of Tennessee is a GUI socket-based retrieval tool to allow access to a set of public domain software packages which are available. Xnetlib is a *point-click* driven interface into a software database. In this way, Xnetlib met our PDS functionality requirements perfectly, and was an existing client-server interface. Subsequently, our performance database client interface became an extension of Xnetlib. This certainly facilitated our implementation of the PDS client and should proliferate its use among Netlib users.

Chapter 5

The Design of a Performance Database

5.1 Design Factors

Several issues need to be addressed in building a performance database. The foremost issue is the organization of the data. Due to the complexity and volume of the data involved, storage should occur in some data management mechanism. A database management system (DBMS) will help in managing the data and the various presentation formats associated with the benchmarks.[1]

It seems logical for PDS to organize data in the DBMS according to the benchmarks themselves: a Linpack table, a Perfect Benchmarks table, etc. Separate tables are required due to the drastic differences in the data formats. It would be nearly

[1] At the inception of the PDS project, the Xnetlib server did not feature a underlying DBMS, but since the development of PDS, Netlib has adopted a database format.

impossible to *force* these numerous presentation formats to adhere to a single presentation standard just for the sake of storage and reporting. Individual tables preserve the display characteristics of each benchmark but still allow users to query all tables for various machines.

Parsing the data into these tables is routine, except that a custom parser must be written for each benchmark. This parser must feature regular-expression matching and can be written in any high-level language, thus *perl* [WaSc90] appears to be a good choice. In the parsing process, building a standard format ASCII file from a *raw* datafile would ease migration of the data into the database.

5.2 Acquisition of benchmark data

We initially construct the database, with entries from a few popular benchmarks. The Linpack and Perfect Benchmarks are widely-accepted public domain benchmarks. Interest in these benchmarks stem from the fact that (*i*) they contain a large spanning set of data, (*ii*) the results are freely distributed, and (*iii*) the benchmark programs are representative of many scientific applications. We will use a flexible design format so that incorporation of new and variant data sets will be simple.

The functionality required by PDS is not very different from that of a standard database application. The difference lies in the user interface. Financial databases, for example, typically involve specific queries like

$$\text{EXTRACT ROW ACCT\_NO} = \text{R103049} ,$$

in which data points are usually discrete and the user is very familiar with the data. The user, in this case, knows exactly what account number to extract, and the format of retrieved data in response to queries. With our performance database, however, we would expect the contrary: the user does not really know (i) what kind of data is available, (ii) how to request/extract the data, and (iii) what form to expect the returned data to be in. These assumptions are based on the current lack of coordination in (public-domain) benchmark management. The number of benchmarks in use continues to rise with no standard format for presenting them. The number of performance-literate users is increasing, but not at a rate sufficient to expect proper queries from the performance database. Often, users simply want to know the best-performing machines for a particular benchmark. Hence, a simple rank-ordering of the *rows* of machines according to a specific benchmark *column* may be sufficient for a general user.

5.3 Client Function Goals

We initially want the client to have a menu of options which are available. This menu-driven format will reduce the complexity of the user interface. The main PDS user interface menu should include:

(1) the ability to extract specific machine and benchmark combinations that are of interest,

(2) the ability to search on multiple keywords across the entire dataset,

(3) the ability to view cross-referenced papers and bibliographic information about the benchmark itself, and

(4) the capacity to build up results from queries and store them locally.

We include (3) in the above list to address the concern of proliferating numbers without any benchmark methodology information. PDS provides abstracts and complete papers related to benchmarks and thereby provides a needed educational resource without risking improper interpretation of retrieved benchmark data.

This client interface design accounted for three different kinds of users: *naive, beginners* and *expert*. Features were incorporated so that any of the three potential users could find relevant information quickly. For example, we anticipated that users may want to utilize upper or lower case keywords in the search process. As a result, we have designed all searches to be case insensitive. We considered that users might want to alias families of machines when they know the terminology. We also considered that a user might not know anything about benchmarking, and they might want to obtain references or articles. We considered that some users might want to view *specific* data and others might want to view 500 lines of performance data. Some users might want to view only the top performing machines and nothing else, while others might simply want to see what is available in PDS. The interface that we have designed will easily facilitate all the above functions at a minimal cost to the new user and minimal penalty to the expert.

Chapter 6

The Specifications of the PDS Implementation

In this chapter, we describe the specifications of the PDS tool developed and maintained at the University of Tennessee and the Oak Ridge National Laboratory.

6.1 Choice of DBMS

Benchmark data is represented in a database format using a Relational DataBase (RDB) query language tool [Hobb91] developed by Walter Hobbe from Rand Corporation. This database query language is sufficient for our needs and permits easy database maintenance. Based on the *perl* language [WaSc90], RDB uses Unix [1] pipes to run entirely in memory. We have converted raw *ASCII* performance data into RDB format using only a few perl commands. Additionally, RDB provides several report

[1] Unix is a trademark of Unix System Laboratories.

features that help standardize the presentation of the performance data.

The RDB implementation specifies that database tables are defined using a schema of the form:

01	Computer	35
02	OS/Compiler	45
03	N=100	7N
04	N=1000	7N
05	Peak	7N.

The first column is the field number, the second is the field label, and the third is the field type, in (type-size) format. The default form is ASCII, and N denotes numeric data so that the *Peak* entry, for example, is a size 7 numeric field.

The contents of the data files themselves are expected to be in some regular grammar, usually a space separated columnar format. A perl script takes a description of the columns and builds a tab-separated file. The schema is converted to a tab-separated file using the RDB command *headchg*. The data files are appended to the end of the schema file, and the resulting flat tab-separated file becomes the rdb format table. RDB uses the schema in the header to process the file. An example from the linpack.rdb is provided below.

```
Computer        OS/Compiler     N=100   N=1000  Peak
35      45      7N      7N      7N
CRAY Y-MP C90 (16 proc. 4.2 ns) CF77 5.0 -Zp -Wd-e68    479     9715    15238
CRAY Y-MP C90 (8 proc. 4.2 ns)  CF77 5.0 -Zp -Wd-e68    468     5994    7619
CRAY Y-MP C90 (4 proc. 4.2 ns)  CF77 5.0 -Zp -Wd-e68    388     3272    3810
CRAY Y-MP C90 (2 proc. 4.2 ns)  CF77 5.0 -Zp -Wd-e68    387     1709    1905
```

This rdb table may then be searched for query matching. An example query for a Linpack Benchmark with `N=100 number equal to 388` is given below.

```
thud> cat linpack.rdb | row N=100 eq 388 | ptbl
Computer                          OS/Compiler          N=100   N=1000   Peak
--------------------------------  -------------------  -------  -------  ----
CRAY Y-MP C90 (4 proc. 4.2 ns)    CF77 5.0 -Zp -Wd-e68  388     3272     3810
```

6.2 Xnetlib Server (nlrexecd)

An implementation of the Xnetlib server [DoRW93], *nlrexecd* , is currently maintained at the University of Tennessee and Oak Ridge National Laboratory. However there were some additional Xnetlib features needed by PDS, so that, several major features were added to an extension of the Xnetlib server. The modified server simply acts as a database engine for retrieving and distributing data to the clients. We note that the users of PDS will not normally see or access this interface, in that it is strictly designed for the client interface which the user directly controls. Figure 6.1 shows the full range of available nlrexecd services which are available to the client and other users. Notice that twelve of the services are native to nlrexecd, and the three services prefixed with *performance-* are the extensions for PDS.

The first feature added was the ability to obtain performance results on a per machine, per request basis. This feature called *performance-query*, allows the user to make individual queries to the performance database tables. The query operations

```
thud> telnet performance.cs.utk.edu 5555
Trying...
Connected to AUSTIN.CS.UTK.EDU.
Escape character is '^]'.
larose@cs.utk.edu
list-services
xxxx
who
keyword-or
keyword-and
keyword-and
keyword-lsi
keyword-literal
keyword-literal-case
keyword
file-tag
file-get
file-get-dep
index-get
performance-query
performance-or-search
performance-and-search
list-services
Connection closed by foreign host.
thud>
```

Figure 6.1: A telnet session showing the full capabilities of the server.

are typical database operations, and one query returns the complete list of all rows which match the search string. A sample query is given below.

```
performance/linpack.rdb row N=100 eq 388
```

Because Berkeley sockets are the underlying transfer mode, the Unix tool *telnet* is used. This tool allows varying ports to be accessed for connections to different socket-based services. In Figure 6.2 a *telnet* connection is made to the server socket and the **bold** text is sent. The user's electronic mail address, when available, is used for Netlib records.[2] The service requested in this example, **performance-query**, is followed by

[2]Netlib tries to keep records of all users of packages so Netlib may contact them with updates, newsletters, and other electronic mailings.

the arguments to that service, the table name **performance/linpack.rdb** followed by the parameters to the search **row N=100 eq 388**. The query displayed in Figure 6.2 returns the entries in the Linpack table of performance data which have N=100 numbers equal to 388 (in this case only 1 machine, a 4 processor Cray Y-MP).

```
thud> telnet performance.cs.utk.edu 5555
Trying...
Connected to AUSTIN.CS.UTK.EDU.
Escape character is '^]'.
larose@cs.utk.edu
performance-query
performance/linpack.rdb row N=100 eq 388
Computer                        OS/Compiler    N=100  N=1000   Peak
----------------------------------------------------------------
CRAY Y-MP (4 proc. 4.2 ns) CF77 5.0 -Zp  388    3272      3810
Connection closed.
thud>
```

Figure 6.2: A telnet session accessing the server's query feature.

The second server feature required is the literal search of the database. This feature provides a literal string matching over the performance database. To permit a boolean-based search capability, the search feature is split into two separate services: **performance-or-search** and **performance-and-search**. These searches are invoked with multiple keywords to allow the user the capability of performing a full spectrum of searches ranging from very general to very specific. The user specifies the keywords along with `or`, `and`. The results are returned in an ASCII format via the socket connection.

In Figure 6.3, the connection is opened to the server socket for a **performance-or-search**. The service name is followed by the arguments to that service, the table name **performance/flops.rdb** followed by the parameters to the search **SGI**

mips. The user then obtains the results of the query, and the connection to **performance.cs.utk.edu** is closed.

```
thud> telnet performance.cs.utk.edu 5555
Trying...
Connected to AUSTIN.CS.UTK.EDU.
Escape character is '^]'.
larose@cs.utk.edu
performance-or-search
performance/flops.rdb SGI mips
VENDOR        COMPILER         CPU-TYPE  CLOCK  MFLOPS   NOTES
-----------------------------------------------------------------
MIPS RC6380   cc3.0 -O3 -mips2   R6000    60.0   17.4     4
SGI 4D/310    Irix 4.0.1, cc-O3  R3000    ---    9.3525   2
SGI 4D/420    one processor, ccR3000  40.0  18.1572  3
SGI IRIS 4D/25 one processor, ccR3000  20.0  6.0395  7
Connection closed.
thud>
```

Figure 6.3: A telnet session accessing the server's search-or feature.

In Figure 6.4, the user requests the performance-and-search, and specifies the arguments to that service, the table name **performance/linpack.rdb** followed by the parameters to the search **IBM 6000 550**. The user then obtains the IBM RS/6000 550 results and the connection to **performance.cs.utk.edu** is closed.

6.3 Client-server interface

Within PDS, the database manager runs only on the server, and the *clients* communicate via Berkeley sockets [BiNe84] to attach to the server and access the database. This socket-based functionality was provided by the pre-existing Xnetlib tool [DoRW93] and was extended to provide support for the performance data. As previously shown, the server port is available to the client to serve data from the database. The client

```
thud> telnet performance.cs.utk.edu 5555
Trying...
Connected to AUSTIN.CS.UTK.EDU.
Escape character is '^]'.
larose@cs.utk.edu
performance-and-search
performance/linpack.rdb rios 6000 550
Computer                    OS/Compiler   N=100  N=1000  Peak
-------------------------------------------------------------
IBM RISC Sys/6000-550 (42 MHz) v2.2.1      26     70      84
Connection closed.
thud>
```

Figure 6.4: A telnet session accessing the server's search-and feature.

opens a socket to the available port on the server and submits a query string. The *perl rdb* tools then access the database using the query string returning typical query results. The output socket is opened, read and then channels the output into the X client. Figure 6.5 illustrates the client-server interactions.

6.4 Client development

After the server was developed, the client was built to interact with the server. The development of the performance client option of the Xnetlib client has coincided with the continuing development of Xnetlib version 3.3. We decided to develop the PDS as a compile time option to Xnetlib. Defining the object before the compile with the **-DPERFORMANCE** option invokes a Unix *make* which produces Xnetlib with PDS.

The Xnetlib client is an X windows interface that obtains data via sockets from the server. The client is a view-only tool which provides the user a window into the database, and prohibits data modifications. The `Performance` button under the

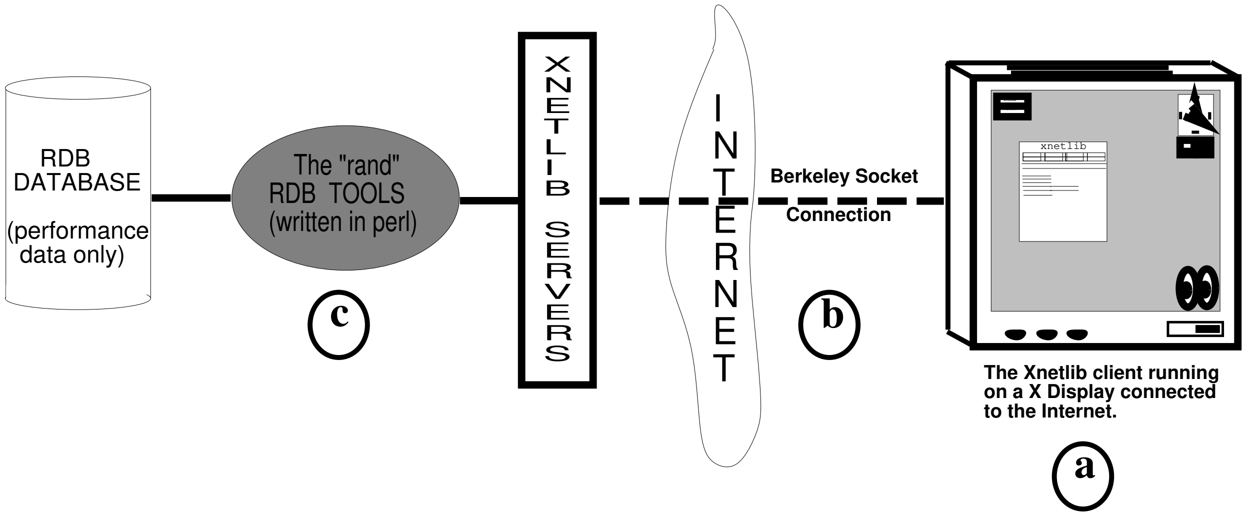


Figure 6.5: The PDS client-server interface: the X workstation (a) communicates over the Internet via Berkeley socket connection (b) to the Xnetlib server, which queries the database using rdb tools (c), and returns benchmark data via the socket connection.

main Xnetlib menu will allow the user to access the PDS interface. At this point users will have numerous menu options to access the various PDS functions. In the following sections, we will describe the functions as outlined in the design process. Users familiar with Xnetlib 3.0 will have an easy transition in using the Xnetlib performance client.

A top level menu with major functions is provided using list widgets under X11/Athena [Pete92]. The options `Rank Ordering`, `Browse`, `Search`, `Papers & Notes`, and `Bibliographic` appear as main buttons under the PDS main menu.

6.5 Rank Ordering

The first goal was to allow the user the ability to view rank orderings of the performance data online. This rank ordering is a sorted output of the performance data. There is no interpretation of the data, simply a report of the data sorted on relevant fields. This goal could be accomplished while still allowing the user to see on-demand current rank orderings. The database maintenance tools will update the information as the reports are published, and the user simply receives the most current information. In this way, the PDS can provide immediate information on the fastest machines available. The rank is implemented with a variant of the Athena text widget [Pete92]. The user selects the icons of interest, and the data is displayed within the window. The download of data is transparent to the user, and occurs only on demand.

6.6 Browse

The second goal was to allow the user the ability to browse through performance data on-line. To address this need, a list of available vendors and benchmarks is kept in the database and downloaded to the client upon startup. The Athena form widget contains an appropriate number of box widgets representing the vendors and the benchmarks (see Figure 7.4). We note that as the database manager changes the master database, the clients across the internet will add buttons as needed. After selecting the requested items, a socket to the server is opened, and the query-matched results downloaded/displayed.

6.7 Search

An Athena form widget contains the search window details [Pete92]. The buttons needed for field entry, search flags, clearing the search field and the display window are all action-based buttons. Within PDS, the search option will operate in a literal sense. It will search for the requested string over the entire performance database and return whatever hits are found. The search feature does allow boolean searches over multiple keywords. Using this ability, users may build up a query using logical **and**, **or** with the keywords. Because of the many names used to describe today's architectures, aliasing is an important feature. The alias *rios*, for example, is associated with the IBM RS/6000 series workstation family. It is also important that users be able to query the database multiple times, and build up an entire collection of performance information. Accumulating search results is accomplished by adding the display text to a text widget. This permits a user, who might want to compare two machines directly, to search for both machines separately, yet have both sets of search results on the same screen.

Chapter 7

Xnetlib 3.3 with the PDS Performance Extension

7.1 Features

Figure 7.1 displays the entry window to the performance extension (client PDS). The user obtains this window after clicking on the Xnetlib `Performance` button and can then select any of the available PDS features for extracting/viewing data. `Rank Ordering` will allow the user to view a sorted list of machines which have been ranked according to a relevant performance metric. The `Rank Ordering` and `Papers` features are similar in nature as both are icon-driven data access paths.

Figure 7.3 shows the rank ordering of results when the user selects an icon from Figure 7.2. This rank-ordered list for machines from the `flops` is displayed when the icon `flops benchmark` is selected. This information is downloaded to the local machine on demand thus making efficient use of network bandwidth.

Figure 7.4 illustrates the PDS browse facility. Here the user selects the vendor(s) and benchmark(s) of interest and then selects `Process` to query the database. The client then opens a socket connection to the server, and using the query language (rdb) remotely queries the database.

The format of the result of the query is shown in the Figure 7.5. Notice the column headings which will vary with each benchmark. Results are displayed via an ASCII widget [Pete92] with scrollbars when needed.

The PDS keyword search facility is demonstrated in Figure 7.6. This feature allows keyword searches into the database. Literal searches are case insensitive and do allow a moderate amount of aliasing. Notice that multiple keywords are permitted, and that a boolean flag is provided for complex queries. The user has the option of entering vendor names, machine aliases, benchmark names and specific strings; thus complicated boolean-based keyword searches are possible.

The results of the search from Figure 7.6 are displayed in Figure 7.7. Here the alias `rios` matched the IBM RS/6000 series and the `Linpack` and `Perfect` terms limited the search to the Linpack and Perfect benchmarks only. Due to the complexity of the Perfect Benchmarks [Berr89], results were separated into 4 database tables (scrollbars can be used to view the entire table).

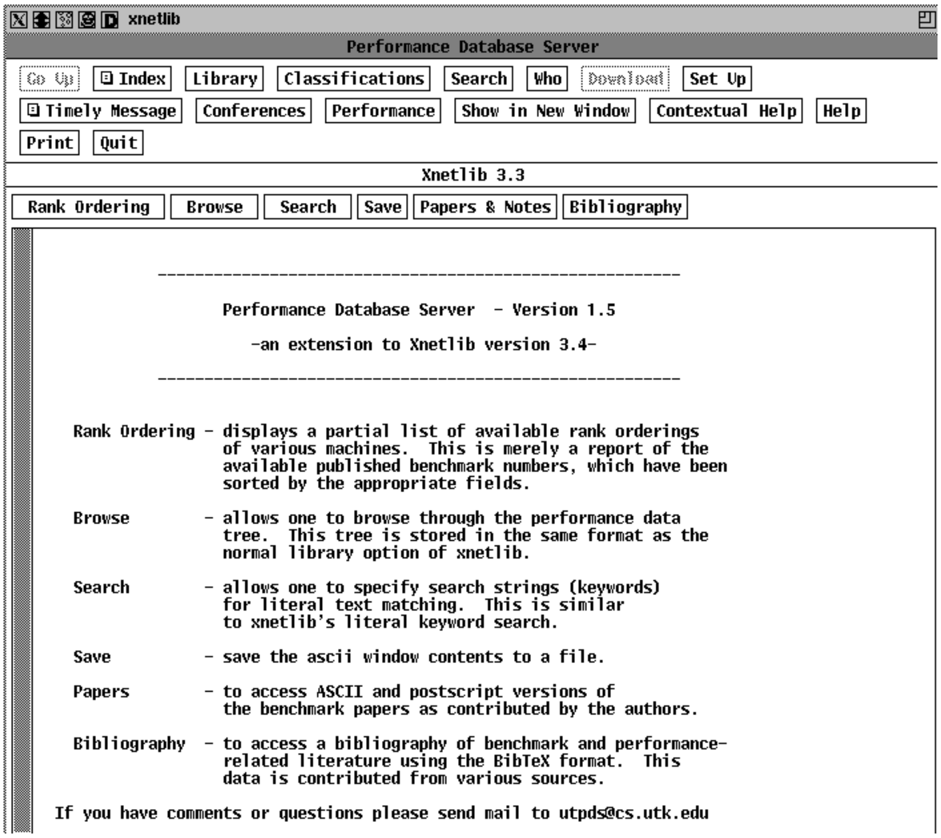


Figure 7.1: Entry level screen for the Performance Database Server.

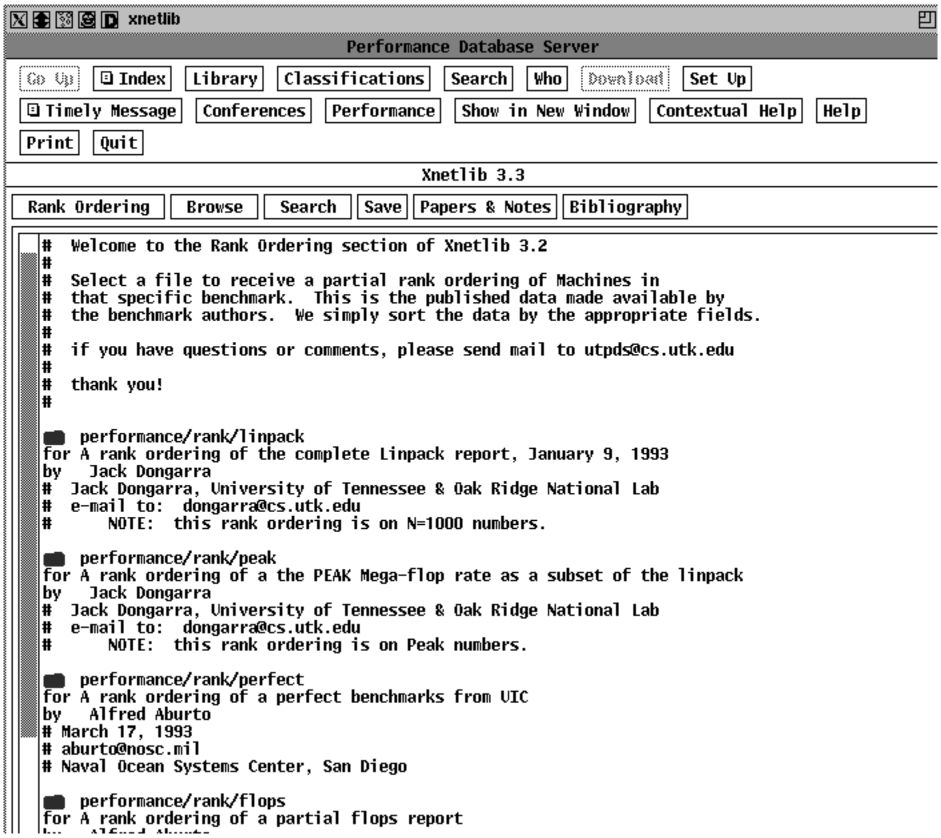
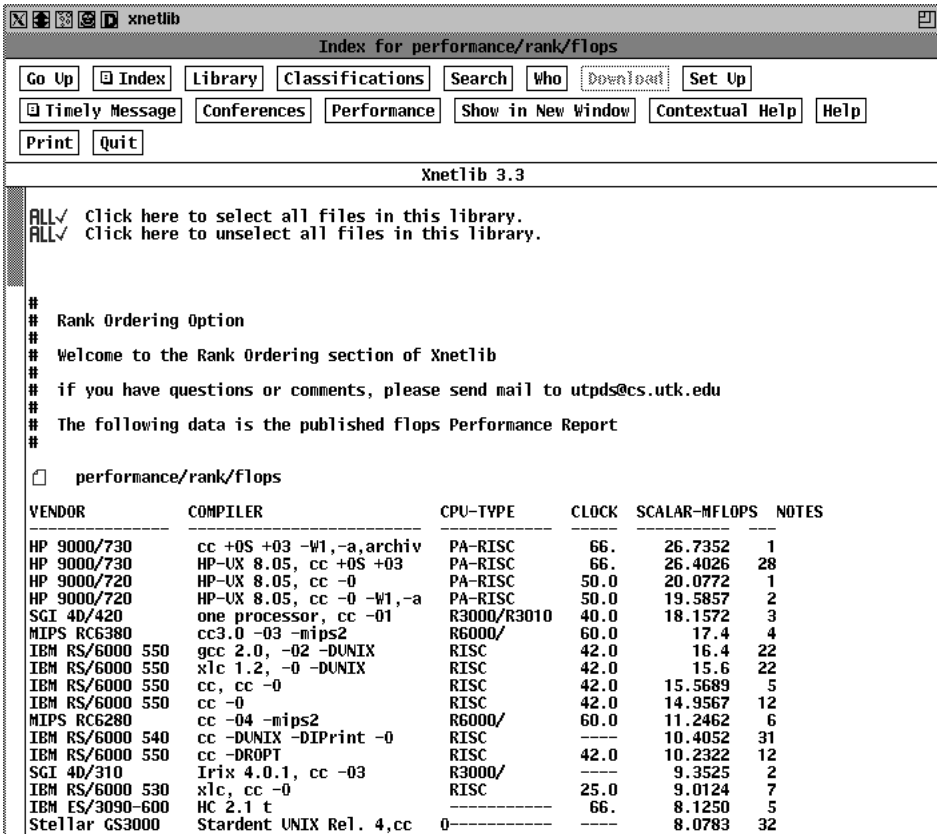


Figure 7.2: A sample rank ordering of some available machines.


xnetlib

Index for performance/rank/flops

Go Up | Index | Library | Classifications | Search | Who | Download | Set Up

Timely Message | Conferences | Performance | Show in New Window | Contextual Help | Help

Print | Quit

Xnetlib 3.3

ALL✓ Click here to select all files in this library.
ALL✓ Click here to unselect all files in this library.

```
#
#  Rank Ordering Option
#
#  Welcome to the Rank Ordering section of Xnetlib
#
#  if you have questions or comments, please send mail to utpds@cs.utk.edu
#
#  The following data is the published flops Performance Report
#
```

performance/rank/flops

VENDOR	COMPILER	CPU-TYPE	CLOCK	SCALAR-MFLOPS	NOTES
HP 9000/730	cc +0S +03 -W1,-a,archiv	PA-RISC	66.	26.7352	1
HP 9000/730	HP-UX 8.05, cc +0S +03	PA-RISC	66.	26.4026	28
HP 9000/720	HP-UX 8.05, cc -0	PA-RISC	50.0	20.0772	1
HP 9000/720	HP-UX 8.05, cc -0 -W1,-a	PA-RISC	50.0	19.5857	2
SGI 4D/420	one processor, cc -01	R3000/R3010	40.0	18.1572	3
MIPS RC6380	cc3.0 -03 -mips2	R6000/	60.0	17.4	4
IBM RS/6000 550	gcc 2.0, -02 -DUNIX	RISC	42.0	16.4	22
IBM RS/6000 550	xlc 1.2, -0 -DUNIX	RISC	42.0	15.6	22
IBM RS/6000 550	cc, cc -0	RISC	42.0	15.5689	5
IBM RS/6000 550	cc -0	RISC	42.0	14.9567	12
MIPS RC6280	cc -04 -mips2	R6000/	60.0	11.2462	6
IBM RS/6000 540	cc -DUNIX -DIPrint -0	RISC	----	10.4052	31
IBM RS/6000 550	cc -DROPT	RISC	42.0	10.2322	12
SGI 4D/310	Irix 4.0.1, cc -03	R3000/	----	9.3525	2
IBM RS/6000 530	xlc, cc -0	RISC	25.0	9.0124	7
IBM ES/3090-600	HC 2.1 t	-----------	66.	8.1250	5
Stellar GS3000	Stardent UNIX Rel. 4,cc	0-----------	----	8.0783	32

Figure 7.3: A result from selecting the `flops` icon from Figure 9 .

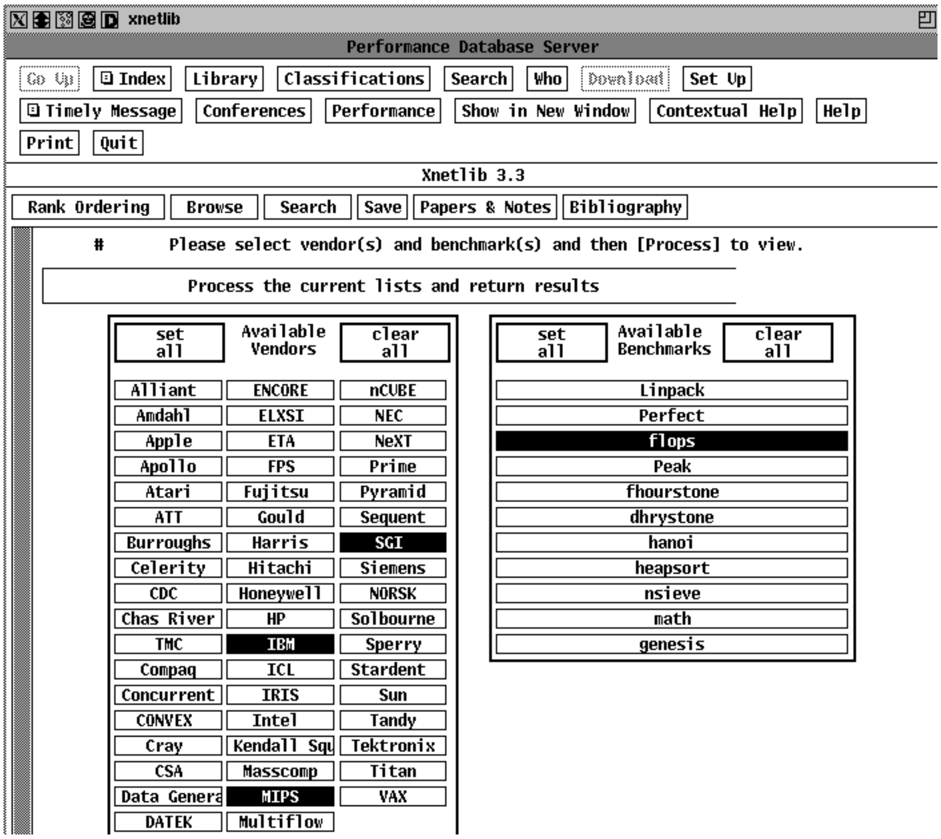


Figure 7.4: The Browse feature allowing specific machine/benchmark selections.

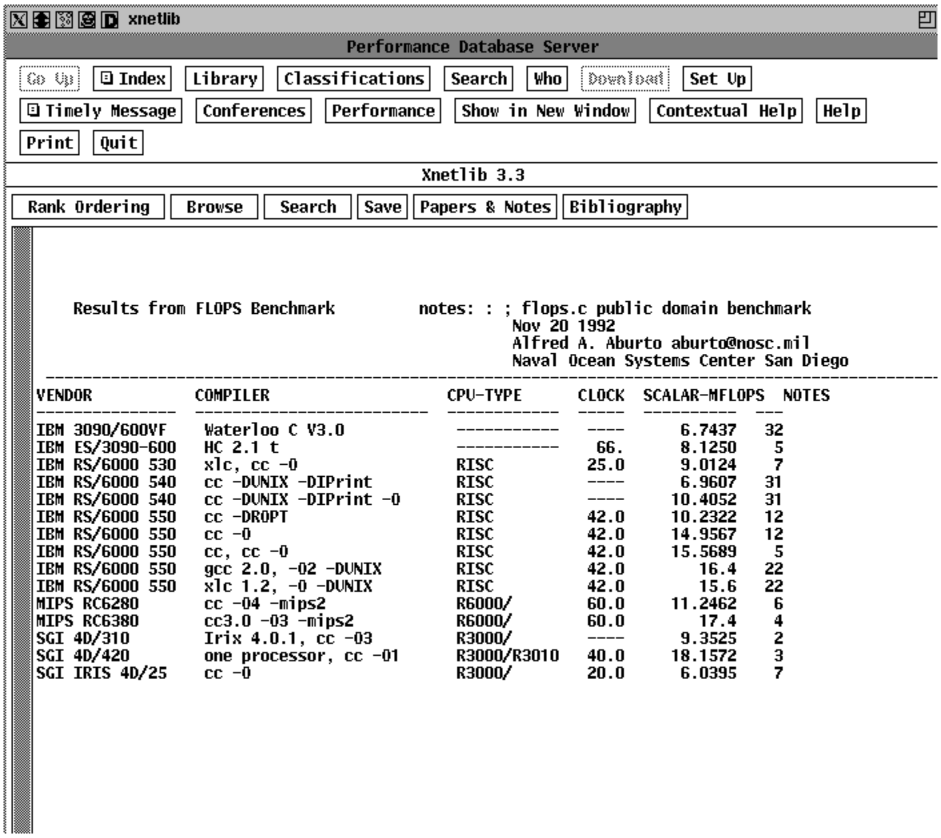


Figure 7.5: The Browse feature returns results into a scrollable window.

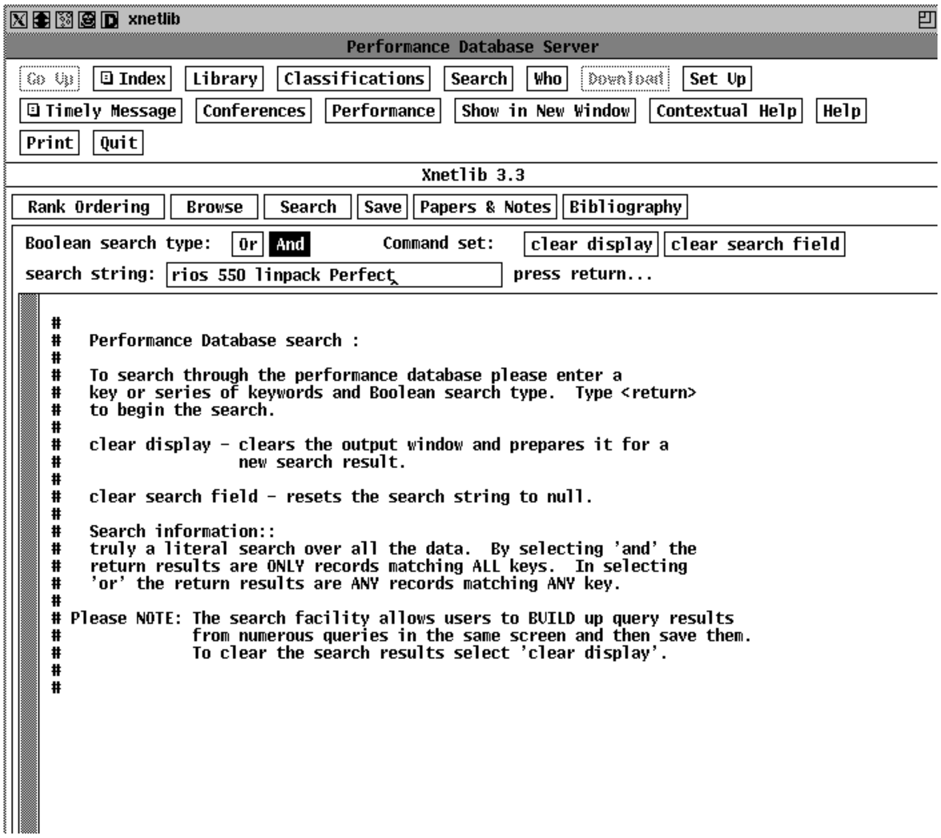


Figure 7.6: The Search feature allowing multiple keyword searches.

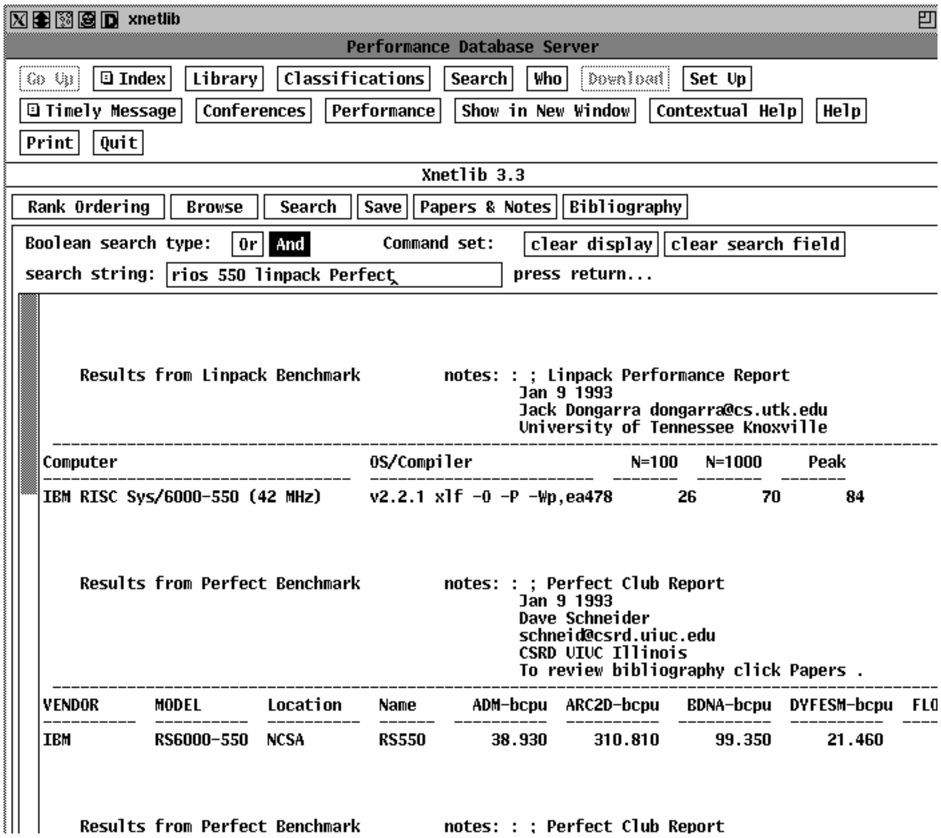


Figure 7.7: The Search feature returns results into a scrollable window.

7.2 Simple Queries

The following performance-related questions were posted to the Internet news group *comp.benchmarks* by various users. Figure 7.8 lists a few examples of usage patterns for potential users of PDS. The left-hand column describes the type of query requested, while the right-hand column describes how to achieve the desired results. Search descriptions are provided with the assumption that the user has already invoked Xnetlib and has selected `Performance`. [1]

To view the best performing machines in the Linpack Benchmark	[Rank Ordering] + [performance/rank/linpack]
To view the SGI results from the hanoi benchmark	[Browse] + [SGI] + [hanoi] + [Process ..]
To search the database for Linpack results on Titan	[Search] + **type "titan linpack"** [[Return]]
To save the current window contents to a local file	[Save] + **type filename** [[Return]]

Figure 7.8: Simple queries for PDS client.

[1] The symbol [Performance] represents clicking the button or icon, and the symbol [[Return]] denotes pressing the return key.

7.3 Actual Internet Queries

Now that we have showed the usage patterns and things that you can do with PDS, we would like to demonstrate it's usefulness answering actual questions posted in the Internet news group *comp.benchmarks*. For the query below, we illustrate a corresponding PDS search description and results in Figures 7.9 and 7.10, respectively.

```
Roger Uzun from CTS Network Services, El Cajon, CA. posted asking:
"Of the common RISC processors, like Sparc, Sparc2, R3000, R4000,
how do they rate compared to a 68040 or 80486 system?"
```

Compare the database results for Sparc r3000 r4000 68040 80486 chips	$\boxed{\text{Search}}$ + **type "Sparc r3000 r4000 680404 80486"** + $\boxed{\boxed{\text{Return}}}$

Figure 7.9: Actions to produce comparison of Sparc r3000 r4000 680404 80486 chips.

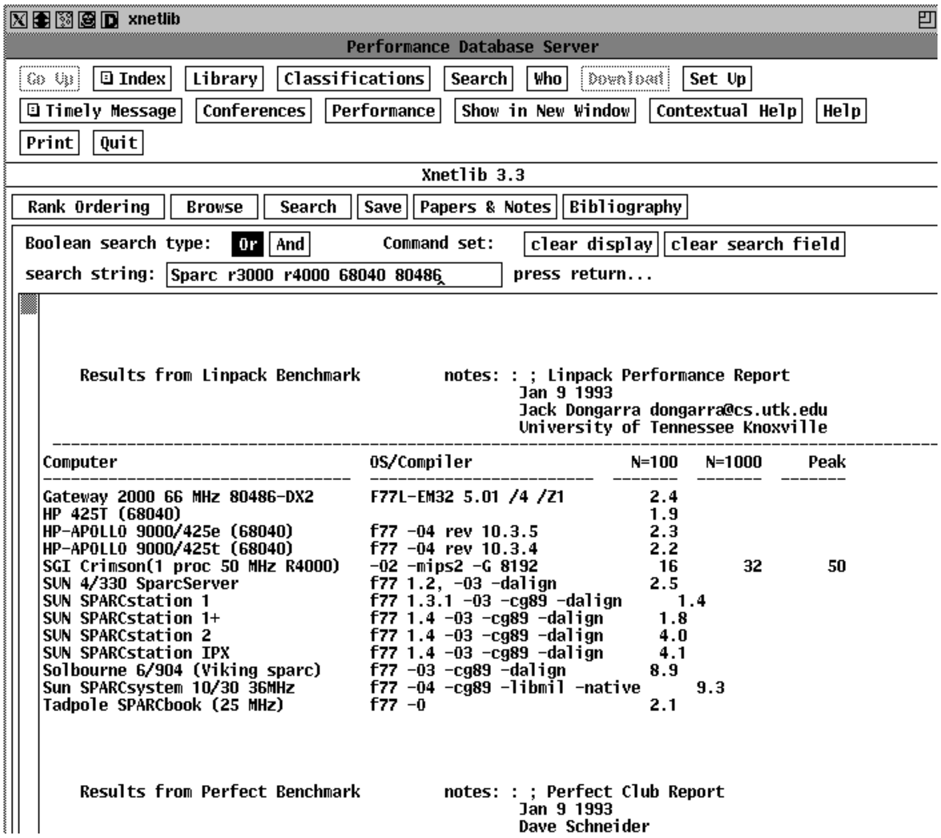


Figure 7.10: Comparison of Sparc r3000 r4000 680404 80486 chips .

The following benchmark question was posted by Erik Hoel from University of Maryland, College Park, MD:

```
"We are looking for benchmark numbers on the Cray Y/MP-8 for scalar (i.e.,
non-vectorizable) operations. Ideally, we would like to be able to relate
the scalar performance of this machine with current state-of-the-art
microprocessors for general purpose computation."
```

We illustrate the actions required to answer this question in Figure 7.11.

To view the state of the art machines on the linpack	$\boxed{\text{Rank Ordering}}$ + $\boxed{\text{performance/rank/linpack}}$

Figure 7.11: Actions to view rank ordering of Linpack results.

The following question was posted by David Vickers (vickersd@gecko.ee.byu.edu) at Brigham Young University, Provo UT USA:

```
" I am working on a research paper on parallel processing.  I would
appreciate any benchmarks anyone could send me.....
My main focus is on how well parallel processing speeds
up a specific machine (SGI 4D/410 vs SGI 4D/480, 1 processor Cray vs
16 processor Cray)."
```

We illustrate the actions required to answer this question in Figure 7.12.

To compare SGI models with Cray models	$\boxed{\text{Search}}$ + **type "SGI Cray"** $\boxed{\boxed{\text{Return}}}$

Figure 7.12: Actions to compare SGI and Cray results.

The following question was posted by Alkiviadis Vazacopoulos <av0h+@andrew.cmu.edu> at Carnegie Mellon, Pittsburgh, PA:

```
"I would like to obtain the latest version of the paper
of Jack J. Dongarra :  Performance of Various Computers using Standard
Linear Equations Software. "
```

We illustrate the actions required to answer this question in Figure 7.13.

To view the Dongarra Linpack report online	$\boxed{\text{Papers \& Notes}}$ + $\boxed{\text{performance/papers/linpack}}$

Figure 7.13: Actions to view the Linpack report online.

The following was posted by Anshu Aggarwal at Boston University:

```
Hi, I was wondering what benchmarking information is available for
the CM-5 other than the Linpack numbers?
```

We illustrate the actions required to answer this question in Figure 7.14.

To view CM-5 numbers	$\boxed{\text{Search}}$ + **type "CM-5"** $\boxed{\boxed{\text{Return}}}$

Figure 7.14: Actions to view numerous CM-5 results.

Chapter 8

Conclusion

8.1 Summary

The Performance Database Server (PDS) provides an on-line catalog of available performance metrics. This X-windows based tool provides a user-friendly interface with multiple views into a dynamic set of data. PDS allows direct performance comparisons of machines using popular benchmarks. It is anticipated that PDS will serve a valuable role in the dissemination of performance information and standardization of benchmark presentation formats.

8.2 Availability

To receive the Xnetlib client package to a Unix system send the electronic mail message *send xnetlib.shar from xnetlib* to `netlib@netlib.ornl.gov`. You can the *unshar* the file and compile it by following the included directions. Once compiled, this shar file will enable you the full functionality of Xnetlib along with the latest PDS client tool. If you have questions or comments about PDS you may send electronic mail to `utpds@cs.utk.edu`. Information concerning the acquisition/inclusion of additional benchmark data is certainly welcome.

8.3 Future work

The PDS project continues to evolve. In fact, PDS has several new features which need to be developed, and as always, the data needs to be frequently updated in order to track performance trends.

8.3.1 Addition of Graphical Interfaces

A major problem with the current performance interface is the deluge of data that a user is confronted with. A simple query can produce more data than the user can comprehend without a great deal of expertise. Because of the flood of performance data that we have uncovered during the course of this project, it is important that we work on ways to better display this massive amount of data for the user. A graphical interface which could display a graph of data, either bar chart or simple plots, would reduce the user strain and improve the tool significantly. A spreadsheet-based display so that users can easily extract subsets of performance data with little effort is anticipated.

8.3.2 Tool development

We also anticipate adding a compare feature to the client interface. The compare option will allow users to make head-to-head comparisons of machines. Using a spreadsheet, appropriate benchmarks and vendors may be searched so that data fills the spreadsheet and may be display in a variety of forms for the user. This direct comparison feature will be a powerful tool for users who want to quickly view a very small subset of data.

8.4 Updating Bibliography

The process of gathering papers and references relating to performance metrics, parallelizing compilers, benchmarks and machine architectures will never cease. It is important that the bibliography be kept up to date.

Bibliography

Bibliography

[BaBa85] D. Bailey and J. Barton. The NAS Kernel Benchmark Program. NASA Ames Technical Memorandum 86711, 1985.

[Bail91] D. Bailey et al. The NAS Parallel Benchmarks. NAS Systems Division, RNR–91–002, January 1991.

[BDGM91] A Beguelin, J Dongarra, G Geist, R Manchek and V Sunderam. Solving Computational Grand Challenges using a Network of Supercomputers. *Proceedings of the Fifth SIAM Conference on Parallel Processing*, Philadelphia, PA, SIAM, 1991.

[Berr89] M. Berry et al. The Perfect Club Benchmarks: Effective Performance Evaluation of Supercomputers. *International Journal of Supercomputer Applications*, 3(3):5–40, Fall 1989.

[BeCL91] M. Berry, G. Cybenko and J. Larson. Scientific Benchmark Characterizations. *Parallel Computing*, 17:1173–1194, 1991.

[BeDL93] Michael W. Berry, Jack J. Dongarra and Brian H. LaRose. PDS: A Performance Database Server. Submitted to *Supercomputing '93*, Portland, OR., November, 1993.

[BiNe84] Andrew D. Birrell and Bruce Jay Nelson. Implementing Remote Procedure Calls. *ACM Transactions on Computer Systems*, 2(1), February 1984.

[Bray93] Tim Bray. Personal Correspondence, May 1993.

[CaGe92] N. Carriero and D. Gelernter. Supercomputing Out of Recycled Garbage: Preliminary Experience with Piranha. *Proceedings of the ACM International Conference on Supercomputing*, ACM Press, 1992.

[Dixi90] K. M. Dixit. Speculations: Defining the SPEC Benchmark. *SunTech Journal*, Vol 1, January 1990.

[DoRW93] Jack J. Dongarra, T. Rowan, R. Wade. Software Distribution Using XNETLIB. *Communications of the ACM*, To appear, 1993.

[Dong88] Jack Dongarra. Performance of Various Computers Using Standard Linear Equations Software in a FORTRAN Environment. *Computer Science Department Technical Report CS–89–85,* University of Tennessee, March, 1990.

[Gust90] J. Gustafson et. al. The Design of a Scalable, Fixed-Time Computer Benchmark. *Tech Report IS–5049/UC–32.* Ames Lab, Iowa State University, 1990.

[Gust91a] J. Gustafson et. al. SLALOM: The First Scalable Supercomputer Benchmark. *Journal of Parallel and Distributed Computing*, 12:388–401, August 1991.

[Gust91b] John Gustafson et al. Slalom Update: The Race Continues. *Supercomputing Review*, 56–61, March 1991.

[Hobb91] W. V. Hobbs. *RDB: A Relational Database Management System.* Rand Corporation, December 1991.

[Hock91] Roger Hockney. Performance Parameters and Benchmarking of Supercomputers. *Parallel Computing*, 17:1111–1130, December 1991.

[Hock92] Roger Hockney. A Framework for Benchmark Performance Analysis. *Supercomputer*, March 1992.

[HoPa87] E. N. Houstis, T. S. Papatheodorou and C. D. Polychronopolous. The LINPACK Benchmark: An Explanation. *Supercomputing*, (Springer Lecture Notes on Computer Science) 297:456–474, 1987.

[McMa86] Frank McMahon. The Livermore Fortran Kernels: A Test of the Numerical Performance Range. *Technical Report UCRL-53745*, Lawrence Livermore Lab, Livermore CA., 1986.

[Norc92] Bill Norcott. Personel Correspondence, October, 1992.

[Pete92] Chris Peterson. *Athena Widget Set – C Language Interface.* X Version 11, Release 5, MIT X Consortium, MIT Press, Boston, MA., 1992.

[Rand87] *Random House Dictionary of the English Language,* Second edition, unabridged. Random House Publishing Co., New York, 1987.

[Unie89] Joseph Uniejewski. SPEC Benchmark Suite: Designed for Today's Advanced Systems. *SPEC Newsletter*, 1(1), Fall 1989.

[WaSc90] L. Wall and R. Schwartz. *Programming in perl*. O'Reilly and Associates, Inc., Sebastopol, CA., 1990.

[Webs83] *Webster's Ninth New Collegiate Dictionary*. Merriam–Webster, Inc, New York, 1983.

[Weic91] Reinhold P. Weicker. A Detailed Look at Some Popular Benchmarks. *Parallel Computing*, 17:1153–1172, December 1991.

Appendix

PDS Quick Reference Guide

When compiling Xnetlib 3.3 to include the PDS client please be sure to use the `-DPERFORMANCE` flag in the `Imakefile` at compile time. This will include all the PDS client's source in creating the object. You will then have access to both current Xnetlib and current PDS X-windows interface. If you do not see the `Performance` button under the main Xnetlib menu, then the performance section was not compiled correctly. Please re-edit the `Imakefile` and verify that the `-DPERFORMANCE` is not commented out. After checking Imakefile, type "make clean" to remove the old objects, and `xmkmf` to remake the Makefile. Then remake the client with the Unix command `make`. If you are having trouble building the client, please contact your system administrator for assistance. Send mail to `utpds@cs.utk.edu` for further assistance with the installation.

To access the PDS database from the Xnetlib client, select the `Performance` button with your left mouse button. After doing this, the six buttons of the Performance Extension will popup. Select the button of your choice to gain access to the online database. A brief description of the available functions is given below:

- Rank Ordering

```
Rank Ordering - displays a partial list of available rank orderings
                of various machines.  This is merely a report of the
                available published benchmark numbers, which has been
                sorted by the appropriate fields.
```

- Browse

```
Browse        - allows one to browse through the performance data
```

```
                tree.  This tree is stored in the same format as the
                normal library option of Xnetlib.
```

- Search

```
Search        - allows one to specify search strings (keywords)
                for literal text matching.  This is similar
                to Xnetlib's literal keyword search.
```

- Save

```
Save          - save the ascii window contents to a file.
```

- Papers

```
Papers & Notes - to access ASCII and postscript versions of
                 the benchmark papers and notes as contributed
                 by the authors.
```

- Bibliography

```
Bibliography  - to access a bibliography of cross references
                to various benchmarks.
```

Vita

Brian Howard LaRose was born in Newport News, Virginia on April 25, 1967. He graduated from Farragut High School in Knoxville in 1985. After matriculating to the University of Tennessee, he received his Bachelors of Science in May of 1989. In September, 1989, he entered the graduate program in Computer Science. While a graduate student, he has worked as a Graduate Teaching Assistant in the Computer Science Department, and a Graduate Research Assistant for Jack Dongarra in the Innovative Computing Laboratory at UTK. In the spring of 1993, he took a position with Hewlett-Packard Company as a system engineer. He and his wife Ginger moved to Georgia in April 1993. Brian's interests include spending time with his friends, camping, playing basketball, antique autos, fishing, street rods and hiking.