NASA-CR-193132

Arachnid Document 6.0

CDM Analysis

June 10, 1993

Sponsored by

NASA/JPL
under the
Small Business Innovation Research Program
Contract NAS7-1156

Produced by:

MIMD Systems, Inc.
1301 Shoreway Road
Belmont, CA 94002

ABSTRACT

The C Data Manager (CDM) is an advanced tool for creating an object-oriented database and for processing queries related to objects stored in that database. The CDM source code was purchased and will be modified over the course of the Arachnid project. In this report, the modified CDM is referred to as MCDM.

Using MCDM, a detailed series of experiments was designed and conducted on a Sun Sparcstation. The primary results and analysis of the CDM experiment are provided in this report. The experiments involved creating the Long-form Faint Source Catalog (LFSC) database, and then analyzing it with respect to: (1) the relationships between the volume of data and the time required to create a database; (2) the storage requirements of the database files; and (3) the properties of query algorithms.

The effort focused on defining, implementing, and analyzing seven experimental scenarios:

1. Find all sources by right ascension, RA;
2. Find all sources by declination, DEC;
3. Find all sources in the right ascension interval (RA_1, RA_2);
4. Find all sources in the declination interval (DEC_1, DEC_2);
5. Find all sources in the rectangle defined by (RA_1, RA_2, DEC_1, DEC_2);
6. Find all sources that meet certain compound conditions; and
7. Analyze a variety of query algorithms.

Throughout this document, the numerical results obtained from these scenarios are reported; conclusions are presented at the end of the document.

TABLE OF CONTENTS

1.0 INTRODUCTION

The C Data Manager (CDM), supplied by Database Technologies (Brookline, Massachusetts) consists of approximately 100,000 lines of C language source code for creating databases and conducting queries. Arachnid employs this system as a database engine to create a complete object-oriented system for querying massive astronomical databases.

The CDM source code is undergoing modification and augmentation for the Arachnid project in order to form the basis of a graphical system by which users can define and execute complex queries on the Long-form Faint Source Catalog (LFSC) database or on other large-scale databases. The system that is resulting from the on-going modifications is referred to as MCDM.

The motivation for conducting the CDM experiment was to:

1. Determine the baseline efficiency of CDM;

2. Improve the efficiency of CDM on complex queries;

3. Develop and test various query algorithms;

4. Reduce the response time by developing optimal versions of the algorithms; and

5. Estimate the efficiency of CDM in the context of Arachnid.

Section 2 of this report contains a brief description of CDM. Then, Sections 3 through 5 discuss the experimental scenarios and present the experimental results. The experiments reported herein focus on: (1) obtaining data on the time required to create the LFSC; (2) exploring the relationships between the volume of data and the database creation time; (3) examining the storage requirements for database files; and (4) determining the execution times for various ways of carrying out certain queries on the database. Although most of these queries are "simple," it is vital that they be done as efficiently as possible.

The queries used in conducting the experiments are as follows:

1. Find all sources by right ascension, RA;

2. Find all sources by declination, DEC;
3. Find all sources in the right ascension interval (RA_1, RA_2);
4. Find all sources in the declination interval (DEC_1, DEC_2);
5. Find all sources in the rectangle defined by (RA_1, RA_2, DEC_1, DEC_2);
6. Find all sources that meet certain compound conditions; and
7. Analyze a variety of query algorithms.

The experimental results are listed in numerous figures and tables. The conclusions derived from conducting the experiments are presented in Section 6.

2.0 CDM OVERVIEW

CDM is an advanced tool for creating, querying, and maintaining object-oriented databases. CDM supports both C and C++, and runs in a multi-user environment on most commercial UNIX platforms, as well as in a single-user mode on PCs running DOS.

One of CDM's primary design features is that it employs a B-tree approach to provide two data access methods: (1) an Indexed Sequential Access Method (ISAM); and (2) a network-type approach. CDM also has features that allow inheritance of object structure, information hiding, and dynamic arrays. As part of the Arachnid project, modifications to the CDM source code were made to provide programmers with capabilities to create, copy, delete, structure, and modify large static databases, such as the LFSC.

CDM's high-level interface is built on a file manager, which hides all file management details from programmers. In turn, the file manager provides fast and flexible access to data, as well as compact data storage. The file manager uses a small set of standard C run-time functions for manipulation of data in both files and RAM.

In CDM, data is stored as objects, with each object containing one or more variable-length attributes. Key attributes contain search keys, data attributes, and one relationship attribute per object (which is used to maintain relationships). Dynamic arrays conveniently accomplish the task of passing attributes. Objects are searched by keys or accessed by relationships.

CDM is often used in CAD/CAM, artificial intelligence, text processing, and graphics applications.

2.1 Basic Functionality

2.1.1 Types of Objects

The database schema is the set of type definitions in which each object type describes the type and number of attributes in an object. Each object type is assigned a unique number for future reference and is declared at run-time by the CDM function *DefType()*.*

* CDM functions are shown in italics, while variables are placed in bold text.

To read the type definition of an existing object type, CDM provides *ReadType()*. CDM also allows dynamic database schemas, whereby an object type definition can be changed (with all existing objects of this type automatically converted to the new format) via the use of *ChgType()*.

2.1.2 Entity-Relationship and Network Model

In addition to keys, objects may be referenced by an object ID, which is a unique number assigned to an object when it is created. Object IDs are used to implement networks of objects and to relate objects to each other. CDM provides a complete interface for the network database model; for example, it supports many-to-many relationships -- a feature not available in relational databases. Function *UpdateRel()* is used to add/update or delete one-to-many relationships between the objects. The network search is given by the function *FindRel()*, which finds all objects related to a given object.

2.1.3 Dynamic Arrays

In standard C, the size of a regular array is determined by a declaration in the C source. Therefore, a programmer must estimate the amount of storage required by a program at run-time. CDM supports dynamic arrays, which are allocated in blocks, as needed.

Dynamic arrays are maintained by special macros that add, insert, delete, and move elements. The elements may be of any C type, including complicated structures. All elements are located in contiguous memory and use dynamic arrays to handle variable size attributes.

2.1.4 Object Access

CDM provides four functions that provide sequential access to objects of a given type by a given key.

- o *FirstObj()* locates the first object;
- o *LastObj()* locates the last object;
- o *PrevObj()* locates the previous object; and
- o *NextObj()* locates the next object.

In all cases, the object selected becomes the current object, whose contents can then be obtained by using *GetCur()*. An object can be deleted via *DeleteObj()*.

In CDM, random access to an object is achieved via *FindMatch()*, which finds all objects with a specified set of key values.

CDM limits a program to having one database open at any one time. The commands *OpenCDM()* and *CloseCDM()* open (and create if necessary) and close the database, respectively. *SaveCDM()* incrementally saves the database; *CheckCDM()* checks the consistency of the database on disk; and *RevertCDM()* lets the programmer retrieve the last version that was saved on disk. .

2.2 Data File Types

CDM creates and maintains four database files on the disk for each database:

1. A data file, which carries an extension of *.dat*, stores all objects on disk. The file is based on a proprietary format and is specially designed to handle variable length objects efficiently.

2. An index file contains a B-tree with indexes and pointers to corresponding objects in data file. Index files carry an extension of *.idx*.

3. A type definition file contains descriptions of object types defined by a program. These files carry an extension of *.def*.

4. An information file contains information about the data file. Information files have an extension of *.inf*.

CDM automatically reclaims storage space when objects are deleted. In order to reduce disk access time, recently referenced data is kept in dynamically allocated buffers. Virtual memory algorithms are used to accomplish data transfer to and from disk files.

2.3 Database Creation

Creating a database is a two-step process. First, the programmer must define the database schema; this design, in fact, is the single most important factor in determining the operational efficiency that the resulting system will achieve. An analogous situation arises in the design of the database schema for implementation of a Relational DBMS. In that case, a design that is normalized (i.e., placed in first, second, third -- but not necessarily fourth -- normal form) is virtually guaranteed to achieve a higher level of performance than is a database that is not so constructed. Although object-oriented theory has yet to provide a universally accepted classification that is parallel to the forms of normality that are provided by relational theory, a design that is not well-planned can easily lead to poor performance.

The second step in the database creation process is to input the data, a process that is normally done electronically from a file or a series of files. For a massive database, this process can be time consuming, largely because it requires a considerable amount of effort to verify the data.

A CDM data model is designed by the programmer via the data schema; a data structure is selected by a well-defined syntax in the application language (e.g., a specification of key vs. non_key, or unique_key vs. non_unique_key). The data structure is then created in CDM by using the following functions:

```
int DefType (rec_def, atr_def, def_data)
REC_DEF *rec_def;
ATR_DEF atr_def[];
Handle def_data[];
```

This function receives type information given by two structures, *REC_DEF* and *ATR_DEF*, which are defined in the (required) CDM header files. *REC_DEF* specifies the number of attributes for the type, while *ATR_DEF* contains information on each attribute. The argument *def_data* is an optional array of handles to a dynamic array containing additional information on attributes (e.g., their names). Function *DefType* returns a type ID, which can later be used to refer to the type.

Once the data structure is defined, the database can be created by the following CDM function.

```
int OpenCDM (create, prefix)
int create;
char *prefix;
```

Here, **create** is a flag that indicates whether the program wants to open or open/create the file, and **prefix** is the name of the database file. If, for example, **prefix** is set equal to "FSC", then the four CDM database files would be:

- o FSC.dat;
- o FSC.idx;
- o FSC.def; and
- o FSC.inf.

The contents of these files were described in the previous section.

2.4 List of Functions

Listed below are CDM's primary functions. The reader is referred to C Data Manager User's Guide[1] for a complete description of these functions.

Database Manipulation

OpenCDM	Open/Create database
SaveCDM	Save database
CloseCDM	Close database
RevertCDM	Revert database
CheckCDM	Check database consistency

Entity Relationship

UpdateRel	Add/delete one-many relationships
FindRel	Find related objects

Object Manipulation

FirstObj	Find first object of given type
LastObj	Find last object of given type
NextObj	Find next object of given type
PrevObj	Find previous object of given type
AddObj	Create new object
DelmatchObj	Delete all objects with matching key
FindMatch	Find object with matching key
GetObj	Read object attributes
UpdObj	Update object
DeleteObj	Delete object
GetCur	Read type definition

Type Definition

DefType	Define new type
ChgType	Change type definition and convert all object of this type
ReadType	Read type definition

Dynamic Arrays

NEW_AR	Create new dynamic array
ADD_ELS	Add elements to dynamic array
INS_ELS	Insert elements in dynamic array
FRE_AR	Free dynamic array
MOV_ELS	Move elements
DEL_ELS	Delete elements
EL	Value of an element in dynamic array
PEL	Pointer to an element in dynamic array
RESET_AR	Reset dynamic array
SIZE_EL	Size of element in dynamic array

3.0 DATABASE CREATION

3.1 Testbed Database

A data tape containing the LFSC database was supplied to MIMD Systems by the Infrared Processing and Analysis Center (IPAC) at NASA/JPL.[2,3] The database, which was approximately 240 MB in size, was then transferred to a SUN Sparcstation, and a disk file containing 30 MB of this data was created. It was this sub-set of the entire LFSC database that was used as the testbed for conducting CDM experiments.

The testbed database contained 21,846 sources, along with their associated attributes. Although the testbed database was substantially smaller than the databases on which NASA will eventually apply Arachnid, it was still large enough to conduct representative experiments and obtain results that can be accurately extrapolated to larger databases.

3.2 Database Creation

The format of the LFSC database was described in a preceding document.[4] This format was maintained in the creation of the testbed database.

The CDM database creation procedures were tested for databases of 20 different sizes and the creation times were recorded. Table 1 presents the creation time (T_c) for each of these sizes, while Figure 1 presents the results graphically. Following the experiments with the incremented database sizes, the full testbed was created. The time required was 9,300 seconds, and was consistent with the earlier results.

A detailed analysis of the experimental data is shown in Table 2. The analysis shows that the time CDM requires to create a database is:

$$T_c \approx n \cdot \lg(n)$$

where n is the number of sources. This dependence is "reasonable" for databases of the size used in this study. It also indicates that the full LFSC catalog, which contains approximately 173,000 sources, will require about 26 hours to be put into CDM format. (This time would be substantially less if a more powerful Sparcstation were to be used.) However, given that such a process only occurs once, the time to create the database is acceptable. Finally, it is important to note that no algorithm for creating a database with an index file can be better then the theoretical rate of O[n·lg(n)].

Number of Sources	Time (sec)
1000	236
2000	494
3000	762
4000	1050
5000	1353
6000	1714
7000	2054
8000	2411
9000	2870
10000	3292
11000	3762
12000	4149
13000	4609
14000	5076
15000	5571
16000	5988
17000	6524
18000	7081
19000	7641
20000	8241

Table 1

Relationship between the number of sources and the time required for database creation

No. of sources (n_i)	No. of sources (n_j)	$n_i \cdot lg(n_i)/n_j \cdot lg(n_j)$	Ratio of creation times (t_i/t_j)
10000	5000	2.162	2.433
15000	5000	3.387	4.117
15000	10000	1.566	1.692
20000	15000	1.373	1.474

Table 2

Database creation time scaling

Relationship Between the Number of Sources and the Time Required for Database Creation

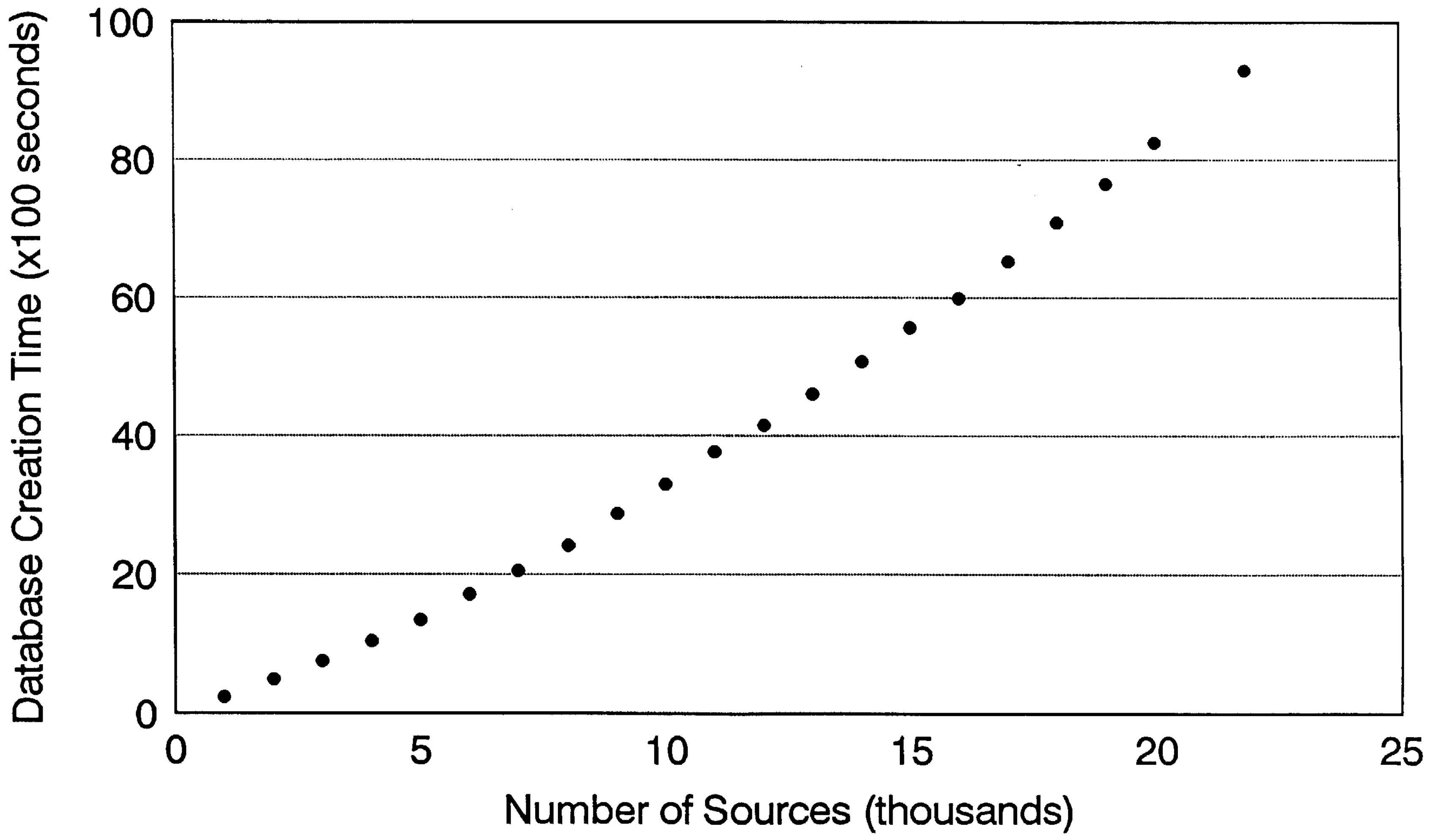

Figure 1

3.3 Database Storage Requirements

The disk space required by CDM to create the testbed database was 36.5 MB, approximately 120% of the size of the testbed LFSC database (30 MB). The size of the index file was found to be approximately linear with respect to the number of sources, while the sizes of the type and information files are insignificant. Consequently, the complete LFSC database should require approximately 290 MB for storage as a CDM database.

File	Name	Memory Space (bytes)
data file	fsc.dat	29,253,632
index file	fsc.idx	7,234,048
type file	fsc.def	1,024
information file	fsc.inf	57,138
Total		36,545,842

Table 3

Disk space required for database storage

4.0 DATA RETRIEVAL

CDM supports ISAM, random search, and relative attribute retrieval. These access methods are discussed in this section.

4.1 Sequential Access

CDM has functions to find and access objects sequentially: *Firstobj()* and *Lastobj()* locate the first and last object of a given type by a given key, respectively. Examples of their usage are presented below:

```
int Firstobj (obj-type, ind-num, key-hdl, key-val)
int LastObj (obj-tpe, ind_num, key_hdl, key_val)
int obj-type; specifies the object type
int ind-num; contains the index number
Handle key_hdl; handle to the dynamic array storing the key value
char *key_val; storage for the key value
```

For all functions, if the specified object is found, then it becomes a current object. The functions *Nextobj()* and *Prevobj()* can be used to find the next and previous objects of current type and index number, respectively.

4.2 Random Search

In CDM, a random search is an indexed search, which is accomplished by the function *FindMatch()*. An example of its usage is:

```
int Find Match (obj_type, find_which, ind_num, key_hdl, key_val, obj_set)
int obj_type; specifies the object type
int find_which; specifies the condition (e.g., "=", "≤", "≥")
int ind_num; contains the index number by attribute order
Handle key_hdl; is the search value in the dynamic array
char *key_val; is the search value in the regular array
REC_ID **object; is the returned set of IDs for the objects found
```

In practice, the random search by a given key value and an operand (e.g., "=," "≤," "≥") is not convenient for interval searches, such as $l_i \leq key \leq l_j$. The Arachnid project team modified the CDM source code to handle interval searches more efficiently. In MCDM, the interval random search can be implemented directly by calling a function with the interval boundary as its arguments.

4.3 Object Access

CDM supports an extremely fast and convenient function to access related objects: *FindRel*. The format for using *FindRel* is shown below.

```
int FindRel (rel_type, obj-type, rel_atr, recs, off)
int rel_type; specifies relationship type
int object_type; specifies object type of related objects to be found
Handle rel_atr; obtains relationship attribute of the source
REC_ID **recs; obtains dynamic array to return related object IDs
unsigned int*off, obtains the offset in the relationship attribute
    where the relationship is found.
```

5.0 LFSC DATA QUERIES

LFSC data queries are designed to meet the requirements set forth in Report #5[4]. Such queries involve attribute retrieval for the following cases:

1. Sources by name;
2. Sources by RA and DEC;
3. Sources by flux density (*fnu_**);
4. Sources by ratios;
5. Sources by galactic latitude (*glat*);
6. Sources in an area; and
7. By algebraic condition.

5.1 Sources within a Specified Region

Arachnid allows sources to be found within the boundaries of two geometric shapes: an ellipse and a rectangle. To find sources within an ellipse, the user must specify the ellipse's axes and orientation. To find sources in a rectangle, the user must specify the rectangle's boundaries, which are put in the form (RA_1, RA_2) and (DEC_1, DEC_2).

We considered three ways to execute a query to find all sources in a rectangle, where the rectangle's conditions are defined as $RA_1 \leq RA \leq RA_2$ and $DEC_1 \leq DEC \leq DEC_2$.

5.1.1 Select Optimal Condition Query

The first method analyzed was the selection of one of the four limits:

$RA_1 \leq RA$
$RA \leq RA_2$
$DEC_1 \leq DEC$
$DEC \leq DEC_2$

CDM should select one of the limits so that the response time is minimal. After being queried by CDM for one limit, an ID set is provided. The attributes of the object with that ID can be taken and examined. Usually this choice is undesirable, but if the proper condition is selected, then there may be only a few returned objects. In practice, retrieval of the source ID by one condition is rapid. (Table 29 presents the results of a study of such query times.) However, taking the sources' attributes by ID and then comparing the conditions and the attributes is slow in both absolute and relative terms, as shown in Table 30.

5.1.2 Interval Query and ID Set Comparison

The second method examined was a two-interval query by MCDM. The first query is according to the condition:

$$RA_1 \leq RA \leq RA_2,$$

while the second query is specified by the condition:

$$DEC_1 \leq DEC \leq DEC_2.$$

Two ID sets are provided, with the resulting set intersection providing the IDs of the query. The experimental results show that executing a query by intervals is faster by MCDM, and sorting IDs and then comparing the intersection of two ID sets is very fast. Table 4 presents a comparison of these methods.

A discussion of the algorithm for finding the intersection of two sets is necessary. The ID set provided by MCDM (or by CDM) is in the order of the key values. The two ID sets from the interval (RA_1, RA_2) and (DEC_1, DEC_2) query by MCDM are ordered according to RA and DEC separately. Clearly, this order index has no relevance for comparing the intersection of the two sets; the two order indexes have different types of attributes. Furthermore, the ID is the pointer to the address of the response object; it is not the true key value itself. Consequently, the ID set is not ordered for the comparison for intersection.

The algorithm for finding the intersection of two non-ordered sets divides into three alternative cases. The first alternative is to compare the IDs sequentially. In this case, the number of operations is $O[n \cdot m]$, where one set contains m IDs and the other set contains n IDs. The second alternative is to take one set in order (e.g., the set containing the m IDs). Here, the number of operations is $O[n \cdot lg(m)]$. The third alternative is that both sets are already ordered and the operations are reduced to O[n+m]. Note that the sorting order for a set of n elements requires a number of operations at least of $O[n \cdot lg(n)]$.

Box Number	Responses	Method #1 Time (sec) Maximum	Method #1 Time (sec) Minimum	Method #2 Time (sec)	Method #3 Time (sec)
1	77	820	3	5	29
2	0	832	<1	19	19
3	108	701	14	24	26
4	3329	601	57	16	53
5	1184	473	184	26	72
6	4384	698	162	27	102
7	3101	482	169	26	125
8	2533	815	136	19	150
9	1446	810	98	15	63
10	774	654	90	15	67
11	21846	828	169	35	155

Table 4

Query Times

These three alternatives have computational complexities of $O[n \cdot m]$, $O[(n+m) \cdot lg(m)]$, and $O[(n+m)+n \cdot lg(n)+m \cdot lg(m)]$, respectively. The difference in computational complexity between the second and third approaches is:

$$d = (m + n) \cdot [lg(n) - lg(m)] > 0$$

Usually m and n satisfy the inequality:

$$(n + m) \cdot lg(m) \ll (n \cdot m)$$

Sample computations show that the sorting and comparison times are faster than taking the attributes of an object by ID and examining the conditions; hence, the first two alternatives are good for sorting and comparison.

5.1.3 Select Optimal Interval Query

An optimal interval may be selected from two intervals, (RA_1, RA_2) and (DEC_1, DEC_2), by considering their lengths. The idea is first to take an optimal interval, for example (RA_1, RA_2), and then to perform the query, finding all sources in this interval by MCDM and obtaining an ID set. Then, the attributes of the object with the ID can be taken and examined for other conditions, with the appropriate objects kept.

Unfortunately, the sources are not uniformly distributed in the domain. However, files could be created that give the approximate density of sources for any range of either RA or DEC. Without such a modification this alternative has to examine the attributes of the object -- a process that is generally slow. In fact, the numerical results show that this method is slower than the second alternative, above. (See Tables 5 through 15.)

5.2 Compound Queries

A typical query on a catalog is a compound query. For example, find all sources that meet the criteria:

(fnu_*)/(fnu_ x) > u,

(fnu_▲) > v, and

| glat | > w.

In general, a compound query is one in which all sources that meet a set of m conditions are found. Such conditions are denoted by: C_1, C_2,..., C_m.

There are three alternatives:

1. Retrieve all sources that meet the condition

   (fnu_*)/(fnu_x) > v,

   where v is a given value.

2. Retrieve all sources that meet the condition

   (fnu_▲) > u,

   where u is a given value.

METHOD 1

Condition	RA ≥ 0.0	RA ≤ 10.0	DEC ≥ -90	DEC ≤ -80
Responses	21846	3618	21846	337
Query Time (sec)	40	4	34	1
Sources in Box	77	77	77	77
Attribute Compare Time (sec)	264	22	786	2
Total Time (sec)	304	26	820	3

METHOD 2 **METHOD 3**

Condition	0≤RA≤10	-90≤DEC≤-80	Condition	0≤RA≤10
Responses	3618	337	Responses	3618
Query Time (sec)	5	<1	Query Time (sec)	4
Sorting Time	<1	<1	Attribute Compare Time (sec)	25
ID Compare Time (sec)	<1			
Sources in Box	77		Sources in Box	77
Total Time (sec)	5		Total Time (sec)	29

Table 5

Query Time for (RA_1, RA_2, DEC_1, DEC_2) = (0, 10, -90, -80)

METHOD 1

Condition	RA $\geq$ 160	RA $\leq$ 180	DEC $\geq$ 80	DEC $\leq$ 90
Responses	0	21846	302	21846
Query Time (sec)	<1	29	1	37
Sources in Box	0	0	0	0
Attribute Compare Time (sec)	0	160	11	795
Total Time (sec)	<1	189	12	832

METHOD 2

Condition	160$\leq$RA$\leq$180	80$\leq$DEC$\leq$90
Responses	0	302
Query Time (sec)	18	<1
Sorting Time	<1	<1
ID Compare Time (sec)	<1	
Sources in Box	0	
Total Time (sec)	19	

METHOD 3

Condition	160$\leq$RA$\leq$180
Responses	0
Query Time (sec)	18
Attribute Compare Time (sec)	<1
Sources in Box	0
Total Time (sec)	19

Table 6

Query Time for (RA_1, RA_2, DEC_1, DEC_2) = (160, 180, 80, 90)

METHOD 1

Condition	RA ≥ 50	RA ≤ 60	DEC ≥ 30	DEC ≤ 40
Responses	1065	21846	5831	18319
Query Time (sec)	1	28	13	32
Sources in Box	108	108	108	108
Attribute Compare Time (sec)	13	156	213	669
Total Time (sec)	14	184	226	701

METHOD 2

Condition	$50 \leq RA \leq 60$	$30 \leq DEC \leq 40$
Responses	1065	2304
Query Time (sec)	19	5
Sorting Time	<1	<1
ID Compare Time (sec)	<1	
Sources in Box	108	
Total Time (sec)	24	

METHOD 3

Condition	$50 \leq RA \leq 60$
Responses	1065
Query Time (sec)	19
Attribute Compare Time (sec)	7
Sources in Box	108
Total Time (sec)	26

Table 7

Query Time for (RA_1, RA_2, DEC_1, DEC_2) = (50, 60, 30, 40)

METHOD 1

Condition	RA ≥ 0	RA ≤ 20	DEC ≥ -30	DEC ≤ 30
Responses	21846	7446	15753	16015
Query Time (sec)	40	10	28	29
Sources in Box	3329	3329	3329	3329
Attribute Compare Time (sec)	261	41	570	581
Total Time (sec)	301	57	598	610

METHOD 2			METHOD 3	
Condition	0≤RA≤20	-30≤DEC≤30	Condition	0≤RA≤20
Responses	7446	9922	Responses	7446
Query Time (sec)	6	9	Query Time (sec)	6
Sorting Time	<1	<1	Attribute Compare Time (sec)	47
ID Compare Time (sec)	<1			
Sources in Box	3329		Sources in Box	3329
Total Time (sec)	16		Total Time (sec)	53

Table 8

Query Time for (RA_1, RA_2, DEC_1, DEC_2) = (0, 20, -30, 30)

METHOD 1

Condition	$RA \ge 20$	$RA \le 60$	$DEC \ge -10$	$DEC \le 0$
Responses	14400	21846	12359	11176
Query Time (sec)	13	29	23	21
Sources in Box	1184	1184	1184	1184
Attribute Compare Time (sec)	171	144	450	406
Total Time (sec)	184	173	473	427

METHOD 2

Condition	$20 \le RA \le 60$	$-10 \le DEC \le 0$
Responses	14400	1689
Query Time (sec)	18	7
Sorting Time	<1	<1
ID Compare Time (sec)	<1	
Sources in Box	1184	
Total Time (sec)	26	

METHOD 3

Condition	$-10 \le DEC \le 0$
Responses	1689
Query Time (sec)	6
Attribute Compare Time (sec)	66
Sources in Box	1184
Total Time (sec)	72

Table 9

Query Time for $(RA_1, RA_2, DEC_1, DEC_2) = (20, 60, -10, 0)$

METHOD 1

Condition	RA ≥ 20	RA ≤ 50	DEC ≥ 0	DEC ≤ 40
Responses	14400	20781	10670	18319
Query Time (sec)	13	27	21	33
Sources in Box	4384	4384	4384	4384
Attribute Compare Time (sec)	174	135	386	665
Total Time (sec)	187	162	407	698

METHOD 2

Condition	20≤RA≤50	0≤DEC≤40
Responses	13335	7143
Query Time (sec)	18	8
Sorting Time	<1	<1
ID Compare Time (sec)	<1	
Sources in Box	4384	
Total Time (sec)	27	

METHOD 3

Condition	20≤RA≤50
Responses	13335
Query Time (sec)	18
Attribute Compare Time (sec)	84
Sources in Box	4384
Total Time (sec)	102

Table 10

Query Time for $(RA_1, RA_2, DEC_1, DEC_2) = (20, 50, 0, 40)$

METHOD 1

Condition	RA ≥ 0	RA ≤ 60	DEC ≥ -10	DEC ≤ 10
Responses	21846	21846	12359	12588
Query Time (sec)	40	29	23	24
Sources in Box	3101	3101	3101	3101
Attribute Compare Time (sec)	264	140	451	458
Total Time (sec)	304	169	474	482

METHOD 2 **METHOD 3**

Condition	0≤RA≤60	-10≤DEC≤10	Condition	-10≤DEC≤10
Responses	21846	3101	Responses	3101
Query Time (sec)	17	7	Query Time (sec)	6
Sorting Time	<1	<1	Attribute Compare Time (sec)	119
ID Compare Time (sec)	<1			
Sources in Box	3101		Sources in Box	3101
Total Time (sec)	26		Total Time (sec)	125

Table 11

Query Time for (RA_1, RA_2, DEC_1, DEC_2) = (0, 60, -10, 10)

METHOD 1

Condition	RA ≥ 10	RA ≤ 70	DEC ≥ 40	DEC ≤ 80
Responses	18228	21846	3527	21544
Query Time (sec)	26	28	8	37
Sources in Box	2533	2533	2533	2533
Attribute Compare Time (sec)	218	152	128	278
Total Time (sec)	244	180	136	815

METHOD 2

Condition	10≤RA≤70	40≤DEC≤80
Responses	18228	3225
Query Time (sec)	16	3
Sorting Time	<1	<1
ID Compare Time (sec)	<1	
Sources in Box	2533	
Total Time (sec)	19	

METHOD 3

Condition	10≤RA≤70
Responses	18228
Query Time (sec)	16
Attribute Compare Time (sec)	134
Sources in Box	2533
Total Time (sec)	150

Table 12

Query Time for (RA_1, RA_2, DEC_1, DEC_2) = (10, 70, 40, 80)

METHOD 1

Condition	RA ≥ 10	RA ≤ 30	DEC ≥ -80	DEC ≤ -40
Responses	18228	11628	21509	4532
Query Time (sec)	27	15	35	10
Sources in Box	1446	1446	1446	1446
Attribute Compare Time (sec)	219	83	775	154
Total Time (sec)	246	98	810	164

METHOD 2

Condition	10≤RA≤30	-80≤DEC≤-40
Responses	8010	4195
Query Time (sec)	11	3
Sorting Time	<1	<1
ID Compare Time (sec)	<1	
Sources in Box	1446	
Total Time (sec)	15	

METHOD 3

Condition	10≤RA≤30
Responses	8010
Query Time (sec)	11
Attribute Compare Time (sec)	52
Sources in Box	1446
Total Time (sec)	63

Table 13

Query Time for $(RA_1, RA_2, DEC_1, DEC_2) = (10, 30, -80, -40)$

METHOD 1

Condition	RA ≥ 0	RA ≤ 30	DEC ≥ -40	DEC ≤ -30
Responses	21846	11628	17314	6093
Query Time (sec)	41	16	30	13
Sources in Box	774	774	774	774
Attribute Compare Time (sec)	263	74	624	223
Total Time (sec)	304	90	654	236

METHOD 2

Condition	0≤RA≤30	-40≤DEC≤-30
Responses	11628	1561
Query Time (sec)	10	3
Sorting Time	<1	<1
ID Compare Time (sec)	<1	
Sources in Box	774	
Total Time (sec)	15	

METHOD 3

Condition	-40≤DEC≤-30
Responses	1561
Query Time (sec)	3
Attribute Compare Time (sec)	64
Sources in Box	774
Total Time (sec)	67

Table 14

Query Time for $(RA_1, RA_2, DEC_1, DEC_2) = (0, 30, -40, -30)$

METHOD 1

Condition	RA ≥ 0	RA ≤ 360	DEC ≥ -90	DEC ≤ 90
Responses	21846	21846	21846	21846
Query Time (sec)	41	29	37	38
Sources in Box	21846	21846	21846	21846
Attribute Compare Time (sec)	256	140	791	783
Total Time (sec)	306	169	828	821

METHOD 2

Condition	$0 \leq RA \leq 360$	$-90 \leq DEC \leq 90$
Responses	21846	21846
Query Time (sec)	18	15
Sorting Time	<1	<1
ID Compare Time (sec)	<1	
Sources in Box	21846	
Total Time (sec)	35	

METHOD 3

Condition	$0 \leq RA \leq 360$
Responses	21846
Query Time (sec)	18
Attribute Compare Time (sec)	137
Sources in Box	21846
Total Time (sec)	155

Table 15

Query Time for (RA_1, RA_2, DEC_1, DEC_2) = (0, 360, -90, 90)

3. Retrieve all sources that meet the above two conditions and get two ID sets. Sort and compare the ID sets; then perform the comparison for the intersection of two ID sets and examine *glat*. The responses for the retrieval that meet the three conditions are the "winners."

For the general case, compound retrievals will be required to meet additional conditions. When the volume of response IDs is smaller, the examination of attributions will be less burdensome.

The numerical results of this experiment, which are summarized in Table 28, show that alternative 3 is best.

5.3 Experimental Results

The query experiment focused on the rectangle query and the compound query.

5.3.1 Query Sources in a Rectangle

Eleven combinations of (RA_1, RA_2, DEC_1, DEC_2) were evaluated:

1. (0, 10, -90, -80)
2. (160, 180, 80, 90)
3. (50, 60, 30, 40)
4. (0, 20, -20, 30)
5. (20, 60, -10, 0)
6. (20, 50, 2, 40)
7. (0, 60, -10, 10)
8. (10, 70, 40, 80)
9. (10, 30, -80, -40)
10. (0, 30, -40, -30)

11. (0, 36, -90, 90)

The query experiment contained ten items:

1. Find all sources by RA
2. Find all sources by DEC
3. Find all sources in the right ascension interval (RA_1, RA_2)
4. Find all sources in the declination interval (DEC_1, DEC_2)
5. Find all sources in the rectangle (RA_1, RA_2, DEC_1, DEC_2)
6. Find all sources by conditions
7. Find all sources by optimal interval query and ID set comparison
8. Find all sources by two interval and comparison of two ID sets is made
9. Examine ID set intersections
10. Examine ID set orders

The experimental results were shown in Tables 5 through 15.

A general comparison of query efficiency for the three algorithms was previously shown in Table 4, while the percent of time for comparison between attribution is shown in Table 16.

5.3.2 Compound Query Experiment

The compound query experiment was designed to find sources that meet three conditions that were shown in Section 5.2:

fnu_12/fnu_25 > u,

fnu_60 > v, and

| glat | > w

where u, v, and w are given at random in the domains of interest.

Box Number	Method 1 Mean Value Attribute Comparison	Method 2 ID Comparison	Method 3 Attribute Comparison
1	84	<3	86
2	89	<3	<3
3	92	<3	27
4	89	<6	89
5	92	<4	91
6	91	<4	82
7	89	<4	95
8	90	<3	89
9	91	<7	94
10	89	<7	96
11	90	<3	88

Table 16

Percent of Query Time

The values u, v, and w were selected to be:

1. (0.5, 1.0, 30.0)
2. (3.7, 0.0, 20.0)
3. (0.0, 0.6, 10.0)
4. (0.7, 0.0, 30.0)
5. (0.0, 0.0, 0.0)
6. (0.5, 0.1, 90.0)
7. (1.0, 0.2, 70.0)
8. (1.5, 0.3, 50.0)
9. (2.0, 0.35, 45.0)
10. (2.5, 0.4, 42.0)
11. (3.0, 0.5, 40.0)

The numerical results are presented in Tables 17 through 27. The experiment showed that method 3 typically outperformed both methods 1 and 2. The reason is that the query time to sort by one condition and the comparison between two ID sets for the intersection of the two sets is fast. The experimental data and statistical data show that for each attribute, the | glat | examination time was 0.02 seconds, as is demonstrated by the results of Table 29. The method that used three ID sets for comparison is not good even when there are only a few hundred candidates. For a larger number, the method becomes increasingly less efficient.

Method 1 and method 2 are the same in principle. Their efficiencies in practice depend upon the volume of responses for each condition. Generally, method 1 and method 2 are much worse than method 3 (see Table 28).

The query for getting the ID of an object is fast using CDM or MCDM (see Table 30). Looking for some attributes of the object ID and then making a comparison is slower than getting the two ID sets and then making the comparison between those sets (see Table 29).

	Method 1	Method 2	Method 3	
Query Condition	fnu_12/fnu_25>0.5	fnu_60>1.0	fnu_12/fnu_25>0.5	fnu_60>1.0
Responses	20066	1312	20066	1312
Query Time	33	3	33	2
Meet Condition	fnu_60>1.0	fnu_12/fnu_25>0.5	ID- intersection set	
Responses	818	818	818	
Meet Condition	abs (glat) > 30.0	abs (glat) > 30.0	abs (glat) > 30.0	
Responses	568	568	568	
Total Time	740	61	69	

Table 17

Compound Query -- 1 of 11

	Method 1	Method 2	Method 3	
Query Condition	fnu_12/fnu_25>3.7	fnu_60>0.0	fnu_12/fnu_25>3.7	fnu_60>0.0
Responses	2358	21846	2358	21846
Query Time	4	36	5	33
Meet Condition	fnu_60>0.0	fnu_12/fnu_25>3.7	ID-intersection	
Responses	2358	2358	2358	
Meet Condition	abs (glat) > 20.0	abs (glat) > 20.0	abs (glat) >20.0	
Responses	1768	1768	1768	
Total Time	108	823	106	

Table 18

Compound Query -- 2 of 11

	Method 1	Method 2	Method 3	
Query Condition	fnu_12/fnu_25>0.0	fnu_60>0.6	fnu_12/fnu_25>0.0	fnu_60>0.6
Responses	21846	2461	21846	2461
Query Time	36	6	36	2
Meet Condition	fnu_60>0.6	fnu_12/fnu_25>0.0	ID-intersection set	
Responses	2461	2461	2461	
Meet Condition	abs (glat) > 10.0	abs (glat) > 10.0	abs (glat) > 10.0	
Responses	2461	2461	2461	
Total Time	824	119	106	

Table 19

Compound Query -- 3 of 11

	Method 1	Method 2	Method 3	
Query Condition	fnu_12/fnu_25>0.7	fnu_60>0.0	fnu_12/fnu_25>0.7	fnu_60>0.0
Responses	17280	21846	17280	21846
Query Time	30	36	31	23
Meet Condition	fnu_60>0.0	fnu_12/fnu_25>0.7	ID-intersection set	
Responses	17280	17280	17280	
Meet Condition	abs (glat) > 30.0	abs (glat) >30.0	abs (glat) > 30.0	
Responses	11822	11822	11822	
Total Time	840	1001	255	

Table 20

Compound Query -- 4 of 11

	Method 1	Method 2	Method 3	
Query Condition	fnu_12/fnu_25>0.0	fnu_60>0.0	fnu_12/fnu_25>0.0	fnu_60>0.0
Responses	21846	21846	21846	21846
Query Time	36	36	35	21
Meet Condition	fnu_60>0.0	fnu_12/fnu_25>0.0	ID-intersection set	
Responses	21846	21846	21846	
Meet Condition	abs (glat) > 0.0	abs (glat) > 0.0	abs (glat) > 0.0	
Responses	21846	21846	21846	
Total Time	1063	1065	293	

Table 21

Compound Query -- 5 of 11

	Method 1	Method 2	Method 3	
Query Condition	fnu_12/fnu_25>0.5	fnu_60>0.1	fnu_12/fnu_25>0.5	fnu_60>0.1
Responses	20066	20868	20066	20868
Query Time	32	36	34	21
Meet Condition	fnu_60>0.1	fnu_12/fnu_25>0.5	ID-intersection set	
Responses	19088	19088	19088	
Meet Condition	abs (glat) > 90.0	abs (glat) > 90.0	abs (glat) > 90.0	
Responses	0	0	0	
Total Time	962	994	280	

Table 22

Compound Query -- 6 of 11

	Method 1	Method 2	Method 3	
Query Condition	fnu_12/fnu_25>1.0	fnu_60>0.2	fnu_12/fnu_25>1.0	fnu_60>0.2
Responses	13262	12734	13262	12734
Query Time	25	24	25	16
Meet Condition	fnu_60>0.2	fnu_12/fnu_25>1.0	ID-intersection set	
Responses	4987	4987	4987	
Meet Condition	abs (glat) > 70.0	abs (glat) > 70.0	abs (glat) > 70.0	
Responses	706	706	706	
Total Time	550	529	125	

Table 23

Compound Query -- 7 of 11

	Method 1	Method 2	Method 3	
Query Condition	fnu_12/fnu_25>1.5	fnu_60>0.3	fnu_12/fnu_25>1.5	fnu_60>0.3
Responses	9549	6833	9549	6833
Query Time	18	14	19	10
Meet Condition	fnu_60>0.3	fnu_12/fnu_25>1.5	ID-intersection set	
Responses	971	971	971	
Meet Condition	abs (glat) > 50.0	abs (glat) > 50.0	abs (glat) > 50.0	
Responses	278	278	278	
Total Time	366	268	68	

Table 24

Compound Query -- 8 of 11

	Method 1	Method 2	Method 3	
Query Condition	fnu_12/fnu_25>2.0	fnu_60>0.35	fnu_12/fnu_25>2.0	fnu_60>0.35
Responses	6943	5351	6943	5351
Query Time	13	12	15	9
Meet Condition	fnu_60>0.35	fnu_12/fnu_25>2.0	ID-intersection set	
Responses	494	494	494	
Meet Condition	abs (glat) > 45.0	abs (glat) > 45.0	abs (glat) > 45.0	
Responses	157	157	157	
Total Time	261	210	45	

Table 25

Compound Query -- 9 of 11

	Method 1	Method 2	Method 3	
Query Condition	fnu_12/fnu_25>2.5	fnu_60>0.4	fnu_12/fnu_25>2.5	fnu_60>0.4
Responses	5088	4332	5088	4332
Query Time	11	9	11	7
Meet Condition	fnu_60>0.4	fnu_12/fnu_25>2.5	ID-intersection set	
Responses	278	278	278	
Meet Condition	abs (glat) > 42.0	abs (glat) > 42.0	abs (glat) > 42.0	
Responses	115	115	115	
Total Time	196	168	31	

Table 26

Compound Query -- 10 of 11

	Method 1	Method 2	Method 3	
Query Condition	fnu_12/fnu_25>3.0	fnu_60>0.5	fnu_12/fnu_25>3.0	fnu_60>0.5
Responses	3856	3175	3856	3175
Query Time	8	7	9	6
Meet Condition	fnu_60>0.5	fnu_12/fnu_25>3.0	ID-intersection set	
Responses	141	141	141	
Meet Condition	abs (glat) > 40.0	abs (glat) > 40.0	abs (glat) > 40.0	
Responses	66	66	66	
Total Time	148	122	20	

Table 27

Compound Query -- 11 of 11

No.	Responses	Method 1 Time (sec)	Method 2 Time (sec)	Method 3 Time (sec)
1	568	740	61	69
2	1768	108	823	106
3	2461	824	119	106
4	11822	840	1001	255
5	21846	1063	1065	293
6	0	962	994	280
7	706	550	529	125
8	278	366	268	68
9	157	261	210	45
10	115	196	168	31
11	66	148	122	20

Table 28

Compound Query Time

No	RA ≥ (sec/ID)	RA ≤ (sec/ID)	DEC ≥ (sec/ID)	DEC ≤ (sec/ID)	(RA_1,RA_2) (sec/ID)	(DEC_1,DEC_2) (sec/ID)	fnu_▲/fnu_x > (sec/ID)	fnu_* > (sec/ID)
1	0.0018	0.0011	0.0016	0.0030	0.0014	0.0029	0.0016	0.0023
2		0.0013	0.0033	0.0017			0.0017	0.0016
3	0.0010	0.0013	0.0022	0.0017	0.0178	0.0022	0.0016	0.0024
4	0.0018	0.0013	0.0018	0.0018	0.0008	0.0009	0.0017	0.0016
5	0.0009	0.0013	0.0019	0.0019	0.0013	0.0041	0.0016	0.0016
6	0.0009	0.0013	0.0020	0.0018	0.0013	0.0011	0.0016	0.0017
7	0.0018	0.0013	0.0019	0.0019	0.0008	0.0023	0.0019	0.0019
8	0.0014	0.0013	0.0023	0.0017	0.0009	0.0009	0.0019	0.0020
9	0.0015	0.0013	0.0016	0.0022	0.0014	0.0007	0.0019	0.0022
10	0.0019	0.0014	0.0017	0.0021	0.0009	0.0019	0.0022	0.0021
11	0.0019	0.0013	0.0017	0.0017	0.0008	0.0007	0.0020	0.0022
Mean	0.0015	0.0013	0.0019	0.0020	0.0027	0.0017	0.0013	0.0020
Total mean 0.0018								

Table 29

ID-based Query time

Time						
No.	RA & DEC	RA & DEC	RA & DEC	RA & DEC	DEC	Glat
1	0.0121	0.0061	0.0360	0.0059	0.0069	0.0415
2		0.0073	0.0364	0.0347		0.0288
3	0.0122	0.0071	0.0365	0.0365	0.0066	0.0276
4	0.0119	0.0055	0.0362	0.0363	0.0063	0.0116
5	0.0118	0.0066	0.0364	0.0363	0.0390	0.0108
6	0.0121	0.0065	0.0362	0.0363	0.0363	0.0119
7	0.0121	0.0064	0.0365	0.0364	0.0384	0.0168
8	0.0119	0.0070	0.0363	0.0361	0.0074	0.0402
9	0.0120	0.0071	0.0360	0.0340	0.0065	0.0425
10	0.0120	0.0063	0.0360	0.0366	0.0410	0.0468
11	0.0117	0.0064	0.0362	0.0358	0.0079	0.0355
Mean	0.0120	0.0066	0.0362	0.0332	0.0167	0.285
Total weighted mean 0.0214						

Table 30

Time for obtaining attributes and examining conditions (seconds/attribute)

6.0 CONCLUSIONS

The experiments reported upon in this document demonstrated the practical efficiency of the methods used in CDM and MCDM. Following are the conclusions from the experiments; these conclusions will be re-examined and refined as Arachnid evolves over the course of the project.

1. The creation time for the complete LFSC database can be estimated from the empirical relationship between the number of sources and the creation times for subsets of the database, as shown in Tables 1 and 2. The approximate time for creating the full LFSC database (containing 173,000 sources) is estimated to be 26 hours. If a more powerful SUN Sparcstation were used, then this time would be reduced substantially. However, since the database will be created only once, its creation time is a relatively minor concern.

2. The storage space required for database files is approximately 120% of the size of the initial database. For LFSC, the raw data is approximately 240 MB, so that the Arachnid LFSC database will be about 290 MB.

3. Retrievals by CDM (and MCDM) for queries with one condition (e.g., "=", "<", ">") are fast: the mean time is 0.0018 seconds/ID. The sorting and comparing of ID sets is also fast.

   However, taking the attributes of objects by ID and then testing them against specified conditions is slower, having a meantime of 0.0214 second/attribute. In practice, the efficiency and speed of this approach are dependent on the relationship between the ID and the attribute, as well as on the current environment.

   The experimental data shows that between 84% and 96% of query overhead is for comparisons with one condition. The time for ID sorting and comparison with two ID sets is not over 7% of the query time. In Table 16, the percent of query time for comparing attributes is shown.

4. The CDM (and MCDM) query times are reasonable, but not extremely fast. CDM can be used as the basis of Arachnid if the best algorithms are selected. For example, the ID query should be tested for multiple conditions. For a massive database, the ID query times will increase only marginally over the times reported here, because ID query by B-tree index grows as O[lg(n)]. Consequently, the overhead will not be substantial; moreover, the use of parallel algorithms will speed-up the non-index operations.

APPENDIX A

REFERENCES

1. C Data Manager User's Guide, Database Technologies, March 15, 1990.

2. Proposal from the Infrared Processing and Analysis Center, NASA/JPL, 1992.

3. IRAS Faint Source Survey, Infrared Processing and Analysis Center, NASA/JPL, 1992

4. Arachnid Database Retrieval Algorithms and Resource Allocation, MIMD Systems, Inc. under contract NAS7-1156, January, 1993.

NASA
National Aeronautics and Space Administration

Report Documentation Page

1. Report No.	2. Government Accession No.	3. Recipient's Catalog No.
4. Title and Subtitle A Survey of Object-Oriented Database Management Systems: Technical Report #6		5. Report Date 6/10/93 6. Performing Organization Code
7. Author(s) Robert E. Larson Paul L. McEntire John G. O'Reilly		8. Performing Organization Report No. 10. Work Unit No.
9. Performing Organization Name and Address MIMD Systems, Inc. 1301 Shoreway Road, #200 Belmont, CA 94002		11. Contract or Grant No. NAS7-1156
12. Sponsoring Agency Name and Address NASA Resident Office - JPL 4800 Oak Grove Drive Pasadena, CA 91109		13. Type of Report and Period Covered Technical Report #6 14. Sponsoring Agency Code

15. Supplementary Notes

16. Abstract

Using C Data Manager (CDM), a detailed series of experiments was designed and conducted on a Sun Sparcstation. The primary results and analysis of the experiment are provided in this report. The experiments involved creating the Long-form Faint Source Catalog (LFSC) database, and then analyzing it with respect to: (1) the relationships between the volume of data and the time required to create a database; (2) the storage requirements of the database files; and (3) the properties of query algorithms.

The effort focused on defining, implementing, and analyzing seven experimental scenarios: (1) find all sources by right ascension, RA; (2) find all sources by declination, DEC; (3) find all sources in the right ascension interval (RA_1, RA_2); (4) find all sources in the declination interval (DEC_1, DEC_2); (5) find all sources in the rectangle defined by (RA_1, RA_2, DEC_1, DEC_2); (6) find all sources that meet certain compound conditions; and (7) analyze a variety of query algorithms.

17. Key Words (Suggested by Author(s)) Object-Oriented DBMS Parallel Computers	18. Distribution Statement Unclassified/Unlimited

19. Security Classif. (of this report) Unclassified	20. Security Classif. (of this page) Unclassified	21. No. of pages 53	22. Price

NASA FORM 1626 OCT 86