Scientific Formats for Object-Relational Database Systems: A Study of Suitability and Performance

Shirley Cohen[1], Patrick Hurley[2], Karl W. Schulz[2], William L. Barth[2], and Brad Benton[3]

[1]Computer and Information Science
University of Pennsylvania
shirleyc@cis.upenn.edu

[2]Texas Advanced Computing Center
The University of Texas at Austin
{phurley,karl,bbarth}@tacc.utexas.edu

[3] IBM Corporation
brad.benton@us.ibm.com

ABSTRACT

Commercial database management systems (DBMSs) have historically seen very limited use within the scientific computing community. One reason for this absence is that previous database systems lacked support for the extensible data structures and performance features required within a high-performance computing context. However, database vendors have recently enhanced the functionality of their systems by adding object extensions to the relational engine. In principle, these extensions allow for the representation of a rich collection of scientific datatypes and common statistical operations. Utilizing these new extensions, this paper presents a study of the suitability of incorporating two popular scientific formats, NetCDF and HDF, into an object-relational system. To assess the performance of the database approach, a series of solution variables from a regional weather forecast model are used to build representative small, medium and large databases. Common statistical operations and array element queries are then performed using the object-relational database, and the execution timings are compared against native NetCDF and HDF operations.

1. Introduction

A common approach taken in early scientific computing algorithms was to utilize flat, sequential files for the handling of input and output data. While these sequential files were simple to access, they had performance drawbacks for storing and accessing large datasets [1]. In addition, the use of binary representations caused interoperability issues because of the differing byte-order assumed on "big-endian" and "little-endian" platforms. Consequently, the scientific demands for an efficient, binary independent, and extensible method for reading and writing scientific data led to the development of specific data-access libraries. Two popular libraries designed to address these needs are the Network Common Data Form (NetCDF) [2] and the Hierarchical Data Format (HDF) [3]. Both formats include data structures for supporting large datasets, multi-dimensional arrays, a variety of data types (integers, floating point numbers, strings), and methods to accommodate metadata. Both libraries also provide data access methods through an application programming interface that is available in several languages including Fortran, C, C++, and Java, which allow researchers to integrate NetCDF or HDF into their programs with relative ease.

While tools such as NetCDF and HDF offer a number of benefits over flat sequential files, the scientific community still frequently contends with the movement and storage of a large number of individual solution files. Indeed, researchers often output at least one solution file per simulation and then compute metrics by looping over all the files for a particular problem definition (e.g. multiple time-steps in a time-accurate physical simulation). One advantage of utilizing a database management system (DBMS) is that the entire solution set can be housed within one centralized location. Although this approach may seem to be an obvious solution, the relational model's lack of previous support for multi-dimensional arrays and poor performance on large, array-based datasets has limited its scientific applicability [4]. However, the emergence of object-relational database systems [5] has opened the door for including scientific datasets within a DBMS. Consequently, this paper explores an initial suitability study by considering the following questions:

- Is it possible to use an object-relational database as a storage structure for scientific datasets?
- If so, how does the data access performance using a relational engine compare to the performance of native NetCDF and HDF operations?

To examine these questions, several Oracle10g object-relational databases are derived using sample scientific datasets obtained from the Weather Research and Forecasting (WRF) Model. To construct the databases, existing SQL extensions are utilized to develop a physical schema that supports the storage and access of multi-

dimensional arrays. As such, the ideas introduced in this paper are generally applicable in the sense that they can be adapted to other object-relational DBMSs.

2. Scientific Data and the WRF Model

The WRF application is a community-based mesoscale weather prediction model originally developed at NCAR [6]. As a scientific application, the WRF model makes extensive use of the NetCDF format to define both input boundary conditions and output forecast solutions. In order to characterize a typical scientific workflow using WRF, a series of regional, 12-hour forecasts at a 40km resolution were generated spanning one month from July, 1995. The result of these simulations was the creation of over 400 NetCDF solution files which aggregate to provide approximately 5GB of time-dependent, gridded meteorological data. This sample WRF dataset was chosen for evaluation based on its heavy utilization of multi-dimensional array structures and native use of the NetCDF file format. In addition, it includes a modest number of solution files and within the context of climate modeling, these files are often post-processed in a serial fashion to derive specific statistical metrics (e.g., the average surface temperature at 1pm for the month).

Each individual solution file contains 77 floating point, multi-dimensional variables. However, a subset of three specific variables was identified for consideration in this study: accumulated precipitation, soil moisture, and temperature. These three variables are representative of the different array sizes and dimensionality present in the WRF output. Note that the precipitation variable is the smallest of the three and represents a two-dimensional spatial quantity which spans the lateral grid dimensions. In contrast, the temperature variable is the largest array and spans all three spatial dimensions (two lateral and one vertical). The soil moisture variable is sized in the middle and is dimensioned as a two-dimensional surface quantity for each of the modeled soil layers. In addition to the spatial dimensionality described for these variables, each variable is also a function of time. Consequently, the NetCDF representation for these variables defines the precipitation as a three-dimensional array, and the soil moisture and temperature variables as four-dimensional arrays.

3. Modeling Scientific Data

The three floating-point variables considered from the WRF dataset are defined as an array of values in which every element is associated with a unique time and space dimension. From the database perspective, the standard way of representing structures in a relational database is to use a star or snowflake schema [7] where each dimension is decomposed into its own table and is linked to the dependent variable using a foreign key relation. In an object-relational database, however, the dependent variable can be represented in a more natural manner. Rather than using a foreign key relation, the dependent variable can be stored with its dimensions using a multi-dimensional array column implemented as an abstract data type. The DBMS then keeps track of the mappings between the abstract data type and the other array dimensions that are implemented as regular table columns.

Commercial databases such as Oracle 10g, DB2 UDB 8, and SQL Server 2005 have adopted an object-relational data model that supports relational tables, abstract data types, and functions on abstract data types [8]. To simplify coding aspects, Oracle also provides two built-in array types that are commonly referred to as collections: Variable Arrays (Varray) and Nested Tables [9]. These two array types can be exploited to model scientific data. For the purpose of this study, Oracle 10g was chosen to store and operate on the multi-dimensional variables derived from the WRF dataset. Note that both types are stored in relational tables and can be used as a table column or an attribute of an object type. A Varray has fixed lower and upper bounds, similar to collection types from other programming languages. It stores elements in the order in which they are added, while a Nested Table does not preserve the insertion order. One additional difference is that Nested Tables may be indexed, while indexing a Varray is not permitted. Nested Tables have another advantage over Varrays in that they allow for inserts, updates, and deletes on individual elements. The Varray type does not allow such operations since Varray data is stored as a single, delimited piece of data within the database. To illustrate the hierarchical relationships between the Table and Varray types, and Table and Nested Table types, Figure 1 presents a schema diagram for the temperature variable. The syntax for the create type and table statements are presented in Figure 2. Note that the NUMBER(9,3) defines a floating-point number with a precision of 9 and a scale of 3. Except for the names used, the schema diagrams and the CREATE statements for soil moisture and precipitation are identical.

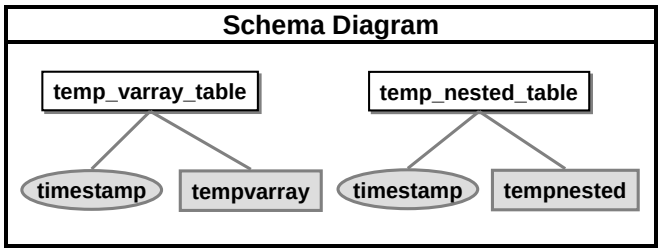

Figure 1: Schema diagram illustrating the hierarchical structure of the relationship types.

Varray

```
CREATE OR REPLACE TYPE tempvarray AS
VARRAY (1000000) OF NUMBER(9,3);

CREATE TABLE temp_varray_table(
timestamp date, data tempvarray);
```

Nested Table

```
CREATE OR REPLACE TYPE tempnested AS TABLE
OF NUMBER(9,3);

CREATE TABLE temp_nested_table(
TIMESTAMP date, data tempnested)
NESTED TABLE data STORE AS data_tab_temp;

CREATE INDEX temp_index ON
data_tab_temp(column_value);
```

Figure 2: Oracle CREATE type and table statements for the temperature variable.

4. Preparing Scientific Data

Scientific computing tasks can easily generate many gigabytes, if not terabytes, of data per run. Once generated, the data must be extracted from multiple files, cleaned, and transformed before it can be loaded into a database. Additional preprocessing steps may also be required depending on the application: e.g. checking integrity constraints, sorting, summarization, and building indices. This preprocessing step can be a time-consuming effort although batch loading utilities can greatly facilitate the process. In this study, Oracle's SQL*Loader utility was used to populate the object-relational schema described in the previous section with the temperature, soil moisture, and precipitation variables. Although the WRF dataset contained additional variables, we chose to load only the three variables of interest for convenience. To use the SQL*Loader utility, two input files were required: a control file that specified how data should be loaded into the database, and a data file, which specified the data to be loaded.

To accommodate performance measurements for different sized datasets, three unique databases were constructed using the 403 NetCDF files included in the WRF dataset. The databases are categorized into small, medium, and large sizes and their designation is based on the number of NetCDF files ingested. The large database includes variables from all 403 files, while the medium and small databases include 195 and 91 files, respectively. To illustrate the number of floating-point elements included in each of these datasets, Table 1 presents the total number of elements included within each of the three variable arrays for each dataset classification. The platform used to construct these Oracle databases was a uni-processor Linux server running RedHat 9 Enterprise with a 3.06GHz Xeon processor and 2GB of main memory. All data was stored on a local, ext3 file system. The 10g Enterprise Edition of Oracle was used (*Release 10.1.0.3.0*), and it was configured with the default 16MB buffer pool.

# of Array Elements			
Dataset	**Temp**	**Soil**	**Pressure**
Large	79,804,075	12,768,652	3,192,163
Medium	38,614,875	6,178,380	1,544595
Small	18,020,275	2,883,244	720,811

Table 1: Number of floating-point elements per variable stored in the small, medium, and large datasets.

To quantify the amount of time required to build the scientific databases, Table 2 presents the wall-clock time required to load each of the three meteorological variables for all three database sizes. Note that during the course of the loading process, several flaws were encountered with the SQL*Loader utility. Initially, none of the data was properly ingested due to an underscore character in the name of the Varray and Nested Table type object definition. Additionally, all records that were larger than 1MB were rejected, which in our case, constituted the majority of the records. Both problems prevented successful creation of the databases, and required several exchanges with Oracle Support. Based on these discussions, subsequent software patches were provided which eventually circumvented the errors. Note that the load times for the nested table were not inconsequential for the larger variables. In particular, it required over 30 minutes to load the temperature variable into the large database. In contrast, the same operation using Varrays required less than 7 minutes to complete.

Varray Load Times (mm:ss)			
Variable	**Database Size**		
	Large	**Medium**	**Small**
Temperature	06:17	02:51	01:21
Soil Moisture	00:43	00:18	00:08
Precipitation	00:12	00:05	00:02
Nested Table Load Times (mm:ss)			
Variable	**Database Size**		
	Large	**Medium**	**Small**
Temperature	31:41	15:11	07:19
Soil Moisture	04:56	02:18	01:02
Precipitation	01:08	00:29	00:06

Table 2: Wall-clock time required to load three WRF dataset variables into large, medium, and small databases.

5. Querying Scientific Data

Once the Varrays and Nested Tables were created and populated, queries were formulated using standard SQL to compute common statistical quantities of interest. To illustrate these queries, Figure 3 presents the syntax required to compute the sum, average, minimum, and maximum values for the temperature variable on Varrays using standard SQL-92 aggregation functions.

In addition to aggregate queries, application scientists are often interested in accessing a small subset of a particular array. Oracle provides no built-in methods for retrieving an element by its array index. To provide this capability, we defined two user-defined functions as presented in Figure 4. As input, the function *getIndexValueVarray* takes a Varray structure and an index and returns the value of the array element corresponding to the provided index. The function *getIndexValueNested* performs the same operation on a Nested Table. Once created, these functions can be invoked using standard SQL. Example syntax demonstrating the use of these functions is shown in Figure 5. These example queries apply a WHERE clause to identify records with a timestamp of "1995-07-01 00". The records that satisfy this filter are passed to the *getIndexValueVarray* and *getIndexValueNested* operators. The results from these operators are instances of the Varray and Nested Table abstract datatypes, respectively. Though their structures are fixed, these operators were written to handle any array variable stored in a Varray or Nested Table type. In addition to these examples, the object-relational model allows for the implementation of more complex analytical functions inside the DBMS.

6. Performance Study

The four aggregate queries and three index queries discussed in the previous section were run to evaluate the performance of Varrays and Nested Tables. The approach chosen to evaluate the database queries was based on an analysis of the execution plans generated by Oracle's cost based optimizer [10, 11]. Additionally, the I/O performance of each test was evaluated using *iostat*, a standard Linux utility for monitoring system I/O performance. For performance comparisons, NetCDF 3.6.0-p1, HDF 4.2r1, and HDF 5 were used. The data was originally in NetCDF format and converted into HDF format using an open-source utility [12].

All of the queries were run when the test machine was idle and the results are based on the average execution time of five successive runs. Note that one of the main goals during these tests was to minimize the caching effects on each measurement; this was accomplished by adhering to a strict set of testing sequences as presented in Figure 6. The purpose of unmounting the file system between each query was to flush out the operating system's buffer cache. The remount of the file system was followed by running a simple program which allocated and initialized a large portion of the host's memory in order to clear out the second-level cache. Using this approach, we were able to obtain consistent timings which minimize any file or memory related caching. Note, however, that in a production environment, the measured response times of repeated queries could be much lower than the average values presented herein due to caching benefits.

Varray Query

```
SELECT SUM(t2.COLUMN_VALUE)
FROM temp_varray t1, TABLE(t1.data) t2;

SELECT AVG(t2.COLUMN_VALUE)
FROM temp_varray t1, TABLE(t1.data) t2;

SELECT MIN(t2.COLUMN_VALUE)
FROM temp_varray t1, TABLE(t1.data) t2;

SELECT MAX(t2.COLUMN_VALUE)
FROM temp_varray t1, TABLE(t1.data) t2;
```

Figure 3: SQL query syntax to derive statistical metrics of the temperature variable using Varrays.

```
CREATE or REPLACE function
getIndexValueVarray (data VARRAY,
array_index number)
RETURN number is
BEGIN
RETURN data(array_index);
END;

CREATE or REPLACE function
getIndexValueNested (nestedtab typenested,
array_index number)
RETURN number is
BEGIN
RETURN nestedtab(array_index);
END;
```

Figure 4: User-defined functions to access individual array elements.

```
SELECT getIndexValueVarray(sn.data,
198025) FROM temp_varray sn
WHERE timestamp = to_date('1995-07-01 00',
'yyyy-mm-dd HH24');

SELECT getIndexValueNested(sn.data,
198025) FROM TEMP_NESTED sn
WHERE timestamp = to_date('1995-07-01 00',
'yyyy-mm-dd HH24');
```

Figure 5: Example SQL syntax to query individual array elements using Varrays and Nested Tables.

Oracle Database	Scientific File Formats
1. Shutdown Oracle 2. Unmount file system 3. Mount file system 4. Fill up memory 5. Startup Oracle 6. Flush buffer cache 7. Run query	1.Unmount file system 2.Mount file system 3.Fill up memory 4.Run Query

Figure 6: Testing sequences for the Oracle and scientific file format queries.

6.1 Aggregate Queries

Table 3 presents the measured query times for the small, medium, and large datasets. Note that four aggregate queries (sum, average, minimum, and maximum) and three index queries (first, middle, and last element) were run on each of the datasets. However, the results obtained for each dataset were almost identical for both the aggregate queries and the index queries. Due to this small variation, we thus include only one aggregate query (average) and one index query (middle) in the results.

Aggregate Query Timings (*in secs*)					
Large Dataset					
Variable	**NetCDF**	**HDF4**	**HDF5**	**Varray**	**Nested**
Temp.	16.56	67.11	20.99	1472.24	40.08
Soil	13.39	61.59	16.28	87.07	6.33
Press.	10.51	55.14	14.87	17.26	1.38
Medium Dataset					
Variable	**NetCDF**	**HDF4**	**HDF5**	**Varray**	**Nested**
Temp.	7.60	31.22	10.27	700.54	20.89
Soil	6.41	28.82	7.97	42.40	3.92
Press.	4.92	25.53	7.04	8.52	1.29
Small Dataset					
Variable	**NetCDF**	**HDF4**	**HDF5**	**Varray**	**Nested**
Temp.	3.48	14.20	4.64	327.48	10.21
Soil	2.97	13.01	3.63	19.11	2.17
Press.	2.18	11.58	3.16	4.11	0.62

Table 3: *Aggregate Query* - wall-clock measurements of the time required to compute an average value for the temperature, soil, and precipitation variables from small, medium, and large datasets.

For all three datasets, the HDF, NetCDF, and Nested Table methods outperformed Varrays on the large temperature and medium soil variables by as much as 90%. The main factor contributing to this performance difference is the use of an expensive nested-loop join [13] in which Oracle fetches a batch of timestamps from the outer Varray table before probing the inner Varray type. In contrast, the Nested type is directly accessible by the query processor, thus avoiding the cost of the join operation. For this reason, performing a full table scan operation on the Varrays is more expensive than on the Nested Tables.

Individual Index Query Timings (*in secs*)					
Large Dataset					
Variable	**NetCDF**	**HDF4**	**HDF5**	**Varray**	**Nested**
Temp.	0.016	0.010	0.020	0.170	40.180
Soil	0.014	0.010	0.005	0.090	6.370
Pressure	0.015	0.009	0.003	0.070	1.340
Medium Dataset					
Variable	**NetCDF**	**HDF4**	**HDF5**	**Varray**	**Nested**
Temp.	0.016	0.010	0.020	0.160	20.200
Soil	0.017	0.009	0.005	0.070	3.750
Pressure	0.015	0.009	0.003	0.060	1.200
Small Dataset					
Variable	**NetCDF**	**HDF4**	**HDF5**	**Varray**	**Nested**
Temp.	0.016	0.010	0.020	0.160	10.220
Soil	0.014	0.010	0.005	0.080	1.970
Pressure	0.015	0.009	0.003	0.050	0.070

Table 4: *Index Query* - wall-clock measurements of the time required to read the middle array value for the temperature, soil, and precipitation variables from small, medium, and large datasets.

The results also suggest that the database scales less than the file-based approach with the larger variables. For example, Nested Tables were more than 3 times faster on the small precipitation variable than the other methods, while the same queries on the large temperature variable took about twice as long as the netCDF and HDF queries. For the temperature variable queries, we note that Oracle was reading up to five times as much data from the disk as the file-based methods. The additional reads incurred by Oracle could be due to a less efficient data representation and a higher ratio of metadata per query.

On the precipitation variable, the database was transferring about the same amount of data as netCDF and over twice as much data as HDF. In this case, Oracle was able to coalesce its I/Os into larger and fewer transactions overall, which explains the faster execution times on the Nested Table queries with the precipitation variable.

6.2 Index Queries

For the index queries, a similar trend emerges in which both Varrays and Nested Tables performed worse than the file-based approaches as the number of elements increased per variable. In this case, however, the Nested queries performed significantly worse than the Varrays, by as much as 200%. The execution plans indicate that Nested Table queries require a full table scan to retrieve a single index. This explains why the Nested index numbers are similar to the aggregates. In total, Varray

index queries resulted in 5 fewer database blocks fetched from disk than the Nested queries. For the small precipitation variable, the Varrays had fewer I/O requests than both HDF methods and NetCDF. With the temperature variable, however, NetCDF required fewer I/O requests than the other methods. Consequently, the execution time correlates with the amount of data read.

7. Conclusion

This paper examined an object-relational DBMS through the prism of scientific data management. The extensibility of object-relational features such as abstract datatypes and user-defined functions were matched to the datatypes and operations commonly employed within the general scientific community. In particular, abstract data types were used to compare the performance of multi-dimensional array structures inside an object-relational DBMS versus native HDF and netCDF file access methods. These performance comparisons were done using Oracle 10g because of its inclusion of two convenient built-in array types: Varray and Nested Tables. We conclude that the database performed well as long as the number of elements per array was small, but did not scale as well as the scientific file-based methods when the number of elements grew. Nested Tables outperformed all the other methods on the small precipitation variable for the aggregate queries, but was slower than the scientific file formats on the larger temperature variable. Similarly, with the index queries, Varrays performed well overall for the smaller precipitation variable, but were less efficient with the larger temperature variable. These results suggest that for the schema considered, the support for array-based object-relational technology from a standard Oracle10g installation is at best mixed and there is substantial room for improving usability and performance. In addition, Oracle10g's storage structure and query processing can most likely fit the needs of small multi-dimensional arrays with fewer than 1 million elements as long as the user is willing to accept the burdens of installing the DBMS, designing and creating the schema, and loading the data into the database. Our overall experience with Oracle 10g indicates that none of these tasks are trivial (especially for a typical scientific user who likely lacks experience in DBMS administration).

The authors would like to note that the work reported here is a proof-of concept investigation which opens the door for further research. In particular, there may be additional tuning possibilities for Oracle10g to increase performance and other DBMSs may behave differently. Another issue to consider is how exactly should these scientific arrays interact with the SQL engine? Since arrays are implemented as abstract datatypes, they are not transparent to the query optimizer. Further work is likely needed to exploit statistics for object-relational DBMSs and produce higher quality query plans. Other issues to be resolved include developing techniques for parallelizing arrays and their operators so that they can scale to larger datasets.

8. Acknowledgements

We wish to thank Laura Timm and Chris Hempel for their help with the Oracle 10g installation. We also thank Tommy Minyard and John R. Boisseau for discussions on scientific data management.

9. References

[1] Gray, J., Liu, D., Nieto-Santisteban, M., Szalay, A., DeWitt, D., and Heber, G., *Scientific Data Management in the Coming Decade*. Microsoft Research Technical Report MSR-TR-2005-10 (2005).

[2] Rew, R. K., G. P. Davis, S. Emmerson, and H. Davies, "NetCDF User's Guide for C, An Interface for Data Access**",** Version 3, April 1997.

[3] Hierarchical Data Format (HDF): http://hdf.ncsa.uiuc.edu/hdf4.html

[4] Network Common Data Form – NetCDF is Not a Database Management System: http://my.unidata.ucar.edu/content/software/netcdf/docs/netcdf/Not-DBMS.html

[5] Stonebraker, M. and Brown, P., Object-Relational DBMSs: Tracking the Next Great Wave, 2nd edition. San Francisco: Morgan Kaufmann Publishers Inc., 1999.

[6] Michalakes, J., J. Dudhia, D. Gill, T. Henderson, J. Klemp, W. Skamarock, and W. Wang, "The Weather Research and Forecast Model: Software Architecture and Performance", Proceedings of the 11th ECMWF Workshop on the Use of High Performance Computing In Meteorology, October, 2004.

[7] Kim, Won. "Modern Database Systems: The Object Model, Interoperability and Beyond." ACM Press and Addison-Wesley, 1995.

[8] Krishnamurthy, V., Banerjee, Sandeepan, Nori, Anil. "Bringing Object-Relational Technology to Mainstream." SIGMOD Conference 1999: 513-514.

[9] Oracle Database. Application Developer's Guide – Object Relational Features 10g Release 1 (10.1). December 2003.

[10] Oracle Corporation. Cost Based Optimizer (CBO) Overview. Note: 10626.1. June 2002.

[11] Oracle Corporation. Cost Based Optimizer – Common Misconceptions and Issues. Note: 35934.1. April 2004.

[12] The NetCDF2HDF conversion utility: http://ioc.unesco.org/oceanteacher/resourcekit/M3/Converters/NetCDF2HDF/.

[13] Oracle Corporation. Interpreting Explain plan. Note 46234.1. October, 2003.