7C.2 EXPERIENCE WITH AN ENHANCED NETCDF DATA MODEL AND INTERFACE FOR SCIENTIFIC DATA ACCESS

Edward Hartnett*, and R. K. Rew
UCAR, Boulder, CO

1 INTRODUCTION TO NETCDF AND THE NETCDF-4 PROJECT

The purpose of the Network Common Data Form (netCDF) interface is to support the creation, efficient access, and sharing of data in a form that is self-describing, portable, compact, extendible, and archivable. Version 3 of netCDF is widely used in atmospheric and ocean sciences due to its simplicity. NetCDF version 4 has been designed to address limitations of netCDF version 3 while preserving useful forms of compatibility with existing application software and data archives.

In version 4.0 (out in beta release at the time of this writing) netCDF adds the use of HDF5, another popular data format and set of libraries, as a storage layer. Many of the advanced features supported by the HDF5 format become available to netCDF users with the 4.0 release, including an expanded data model, compression, chunking, parallel I/O, multiple unlimited dimensions, groups, and user defined types.

The netCDF-4 release fully maintains backward format and code compatibility. That is, all existing netCDF files remain readable and writable, and existing netCDF applications may upgrade to netCDF-4 without source code changes or loss of functionality. NetCDF-4 adds a new HDF5-based binary format to formats already available.

NetCDF-4 represents an addition, not a replacement or successor format. Some users will use this new format for the extra features it provides or for performance benefits. Others will continue to use the existing netCDF binary formats.

1.1 Introduction to NetCDF

NetCDF consists of:

- a conceptual data model
- a set of binary data formats
- a set of APIs for C/Fortran/Java

*Corresponding author address: Ed Hartnett, Unidata/UCAR, PO Box 3000, Boulder, CO 80307, email: ed@unidata.ucar.edu. The National Science Foundation is Unidata's primary sponsor.

The NetCDF-4.0 release expands all three of these aspects of netCDF. In this paper (and netCDF documentation) the word "classic" is used to refer to the data model and binary format prior to netCDF-4, and "netCDF-4" is used to refer to the new data model, format, and API additions.

1.1.1 The Data Model

By "data model" we mean the way scientific data is conceptually modeled with a set of objects, operations, and rules that determine how the data is represented and accessed.

The classic model of netCDF represents data as a set of multi-dimensional arrays, with sharable dimensions, and additional metadata attached to individual arrays or the entire file. In netCDF terminology, the data arrays are **variables**, which may share **dimensions**, and may have attached **attributes**. Attributes may also be attached to the file as a whole. One dimension may be of unlimited length, so data may be efficiently appended to variables along that dimension. Variables and attributes have one of six primitive data types: char, byte, short, int, float, or double.

NetCDF-4 expands this model to include elements from the HDF5 data model, including hierarchical grouping, additional primitive data types, and user defined data types. (See figure 1).

The new data model is a superset of the existing data model. With the addition of a nameless "root group" in every netCDF file, the classic model fits within the netCDF-4 model.

1.1.2 The Binary Formats

By "binary formats" we mean the layout of bytes on the disk.

NetCDF-4.0 supports three binary data formats:

1. classic – the original netCDF binary data format
2. 64-bit offset – the variant format which allows for much larger data files
3. netCDF-4 – the HDF5-based format, with netCDF-specific constraints and conventions.

Additionally there is one "virtual" format: netCDF-4

classic model. This format is obtained by passing the classic model flag when creating the netCDF-4 data file. Such a file will use the netCDF-4 format restricted to the classic netCDF data model. Such files can be accessed by existing programs that are linked to the netCDF-4 library.

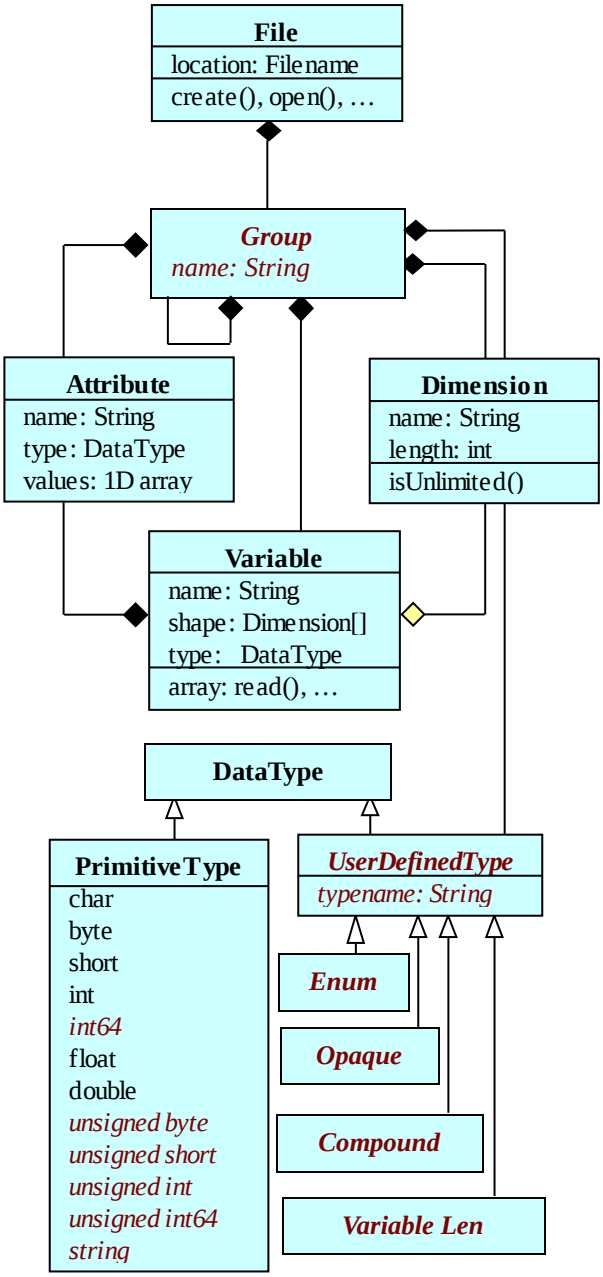

Figure 1: The netCDF-4 data model. Objects in black are in the netCDF classic data model. Objects in red have been added to the classic model in netCDF-4.

1.1.3 The Programming APIs and Libraries

By "programming APIs and Libraries" we mean the software that makes netCDF available in various computer programming languages.

The language APIs are implemented in two distinct core libraries: the original C library and the independent Java library. The Fortran and C++ APIs call the C library functions. All other APIs not in a Java environment are based on the C library.

NetCDF-4 has been fully implemented in the C library; implementation in the Java library is underway.

1.2 NetCDF Library Architecture

NetCDF-4 is a straightforward extension of the existing netCDF code base. All netCDF classic and 64-bit offset functions are handled by the netCDF-3 core library, with an additional netCDF-4 library handling netCDF-4 files.

The netCDF-4.0 release includes the entire code base of the current 3.6.2 release, plus changes since that release. A 3.x variant of the release will continue to be available to build only the netCDF 3.x release. The 3.x release is equivalent to the 4.x release without the --enable-netcdf-4 option to configure.

1.3 Getting, Building, and Using NetCDF

The netCDF-4 library is available from the netCDF web site as a beta release. It depends on the zlib library and the HDF5-1.8.0 library (both are also available from the netCDF web site). HDF5 must be built to use zlib (by specifying the --with-zlib option to HDF5's configure script).

As of the time of this writing, the HDF5-1.8.0 release is available in beta release. Since netCDF-4 relies on HDF5 1.8.0, it must stay in beta release until the full HDF5 1.8.0 release.

NetCDF is built by running the configure script, and then running make to build the library. There are many configure options; the --help option provides the full list. The following options are likely to be of interest to many users:

- --enable-netcdf-4 – without this option the latest code from the version 3.6.x series is built.
- --with-hdf5=/location – specifies the location of the HDF5 1.8.0 library.
- --enable-benchmarks – builds (and tests) the benchmarking program bm_file.c, used to gather performance data for the graphs in this paper.
- --enable-parallel-tests – turns on parallel tests if a parallel build is taking place. Note that tests

are run with mpiexec.

For parallel I/O builds, the MPI library must be available on the target platform, and HDF5 must be built with the --enable-parallel option to configure. NetCDF will automatically detect an HDF5 build for parallel and will build with parallel I/O capabilities in that case.

Users should always run "make check" and should report any build or test problems to:

support-netcdf@unidata.ucar.edu.

1.4 Running the Benchmarks

The program used to benchmark netCDF is bm_file.c, in the nc_test4 directory of the netCDF distribution. The program is run from the command line, and outputs the results of a benchmarking run in a comma separated value (csv) format suitable for use in a spreadsheet for analysis of the benchmarking results.

The bm_file.c program may be used on any netCDF file that conforms to the classic model. The program copies any netCDF file into another netCDF file of user-specified binary format. That is, it can convert a netCDF classic format file to netCDF-4, and vice versa. The program uses the gettimeofday() function to time reads and writes.

The data are read and written one slice at a time, with the slice being a user-specified fraction of the extent of the slowest varying dimension, and the full extent for all other dimensions. (An exception is made for 1-dimensional variables — the slice is one-tenth of the maximum size).

Most of the benchmarks in this paper were run with gridded radar 2D data on an x686 Linux workstation. The sample datasets used are available on the Unidata FTP site: **ftp://ftp.unidata.ucar.edu/pub/netcdf/sample_data**

The benchmarks are run by scripts in the nc_test4 directory of the netCDF distribution. The scripts clear the system's file caches by running a platform-specific script in the nc_test4 directory named clear_cache.sh.

2 EARLY EXPERIENCE WITH NETCDF-4

2.1 Experiences Reported by Users

We thank the netCDF-4 user community for their input and contributions. Below we list early experiences of some groups with netCDF-4.0.

2.2.1 NCO Experience

The netCDF Operators, or NCO, are a suite of file operators that manipulate netCDF data files.

The NCO package, when built from source, may be built with netCDF-4 to support reading and writing netCDF-4 data in the classic netCDF model. NCO has an extensive set of tests that run successfully for netCDF-4 files, with the exception of some tests relating to the renaming of netCDF variables. NCO can also handle the new netCDF-4 primitive numeric types.

The NCO tools can reduce the size of netCDF files by scaling the data. This is a lossy compression scheme in which the data are reduced in precision and scaled. NCO also supports the creation of netCDF-4 files with lossless zlib compression. NCO developers report that a compression of 30% to 60% is typical on scientific data with zlib compression, and up to 80% when scaling and compression are combined; see Zender (2007).

2.2.2 Writing Subsections of Large Variables

A dramatic example of the benefits of chunking storage has been provided by a user writing 64x64x32 subsections of 256x256x64 variables in modeling code. With netCDF-3, this involves writing 64x64 non-contiguous blocks of 32 values each, and the I/O portion of the resulting program runs at a rate of only 0.39 MB/s. With the use of chunking to change non-contiguous writes into contiguous writes, an I/O speed of 51 MB/s is achieved, making the I/O portion of the code run in 27 seconds instead of 56 minutes.

3 RECOMMENDATIONS FOR NETCDF USERS

With the development and imminent release of netCDF-4, a richer but more complex data model will be available. Some of the new features in netCDF-4 provide better ways to represent observational data, new ways to represent metadata, and ways to make data more self-describing for communities outside the traditional atmospheric, ocean, and climate modeling communities. The backward compatibility with existing data and software will allow an incremental transition, using only features that provide clear benefits. In some cases, there are significant performance benefits to using existing software with the new format or to writing data using the classic data model and new format for performance benefits of programs that will later access the data.

3.1 Using NetCDF-4 with the Classic Data Model

NetCDF-4 brings many new features to users within the classic netCDF model. By confining themselves to the classic model, data producers ensure that their data files can be read by any existing netCDF software which has been relinked with the netCDF-4 library.

For example, the use of a compound type in a file requires the netCDF-4 data model, but reading compressed data does not.

One advantage of only using features that conform to the classic data model is that existing code that reads, analyzes, or visualizes the data will continue to work. No code changes are needed for such programs, and they can transparently use netCDF-4 features such as large file and object sizes, compression, control of endianness, reading chunked data, and parallel I/O, without modification of existing code.

For example, data producers can use zlib compression when writing out data files. Since this is transparent to the reader, the programs that read the data do not need to be modified to expand the data. That happens without any help from the reader.

In many cases, users may wish to use netCDF-4 data files without adding any of the model-expanding features. As a convenience netCDF-4 includes the CLASSIC_MODEL flag. When a file is created with this flag, the rules of the classic netCDF model are strictly enforced in that file. This remains a property of the file, and the file may never contain user-defined types, groups, or any other objects that are not part of the classic netCDF data model.

3.1.1 Large File and Object Size

NetCDF-4 files may contain larger objects than classic netCDF or even 64-bit offset netCDF files. For example, variables that do not use the unlimited dimension cannot be larger than about 4 GiBytes in 64-bit offset netCDF files, but there is no such limit with netCDF-4 files on 64-bit platforms.

3.1.2 Compression and Shuffle Filters

NetCDF-4 uses the zlib library to allow data to be compressed and uncompressed as it is written and read. The data writer must set the appropriate flags, and the data will be compressed as it is written. Data readers do not have to be aware that the data are compressed, because the expansion of the data as it read is completely transparent.

The shuffle filter does not compress the data, but may assist with the compression of integer data. The shuffle algorithm changes the byte order in the data stream; when used with integers that are all close together, this results in a better compression ratio. There is no benefit from using the shuffle filter without also using compression.

Data compression and shuffling may be set on a per-variable basis. That is, the zlib compression flag (from 0, no compression, to 9, maximum compression) can be set independently for each variable. In our tests we notice that setting the deflate higher than one takes more time, but has little benefit.

3.1.3 Control of Endianness

In netCDF classic format files (and 64-bit offset format files), numeric data are stored in big-endian format. On little-endian platforms, netCDF is converted to big-endian when the data are written, and converted back to little-endian when read from the file.

In netCDF-4 files, the user has direct control over the endianness of the each data variable. The default is to write the data in the native endianness of the machine. This is useful in cases where the data are to be read on the same machine, or machines of similar architecture.

However, in some cases the data may be produced on a machine of one native endianness, and read on a machine of the other endianness. In these cases, the data writer may wish to optimize for the reader by explicitly setting the endianness of the variable.

In our tests, the endianness of the data only affected read rates significantly when disk caches were in full use, and the data were read from the disk cache. In this case, data with a native endianness were read noticeably faster. However, when disks caches were cleared, the endianness of the data does not affect the read rate much. Apparently the disk speed is slow enough without caching that the CPU has plenty of time to swap the bytes of the data while waiting for the disk. When the data are available in cache, the I/O rate is much faster, and then the cost of the byte swapping becomes noticeable.

For high-performance applications in which netCDF file reading is a bottleneck and access patterns allow disk caching to be used effectively, users should consider writing variables in the file with the endianness of the target platform. Higher-performance disk systems may also serve the data fast enough for its endianness to matter.

3.1.4 Chunking

NetCDF-4 files may be written as chunked data, each chunk representing a multidimensional tile of the same size. That is, the data are written as chunks of a given size, specified by the user when the variable is created and before any data is written. Compressed variables must be chunked, and each chunk is compressed or uncompressed independently.

Chunking has important performance ramifications. Both file size and I/O rates are affected by chunk sizes, and choosing very small chunk sizes can be disastrous for performance. The following graph shows the file sizes of the radar 2D sample data for a variety of chunk sizes.

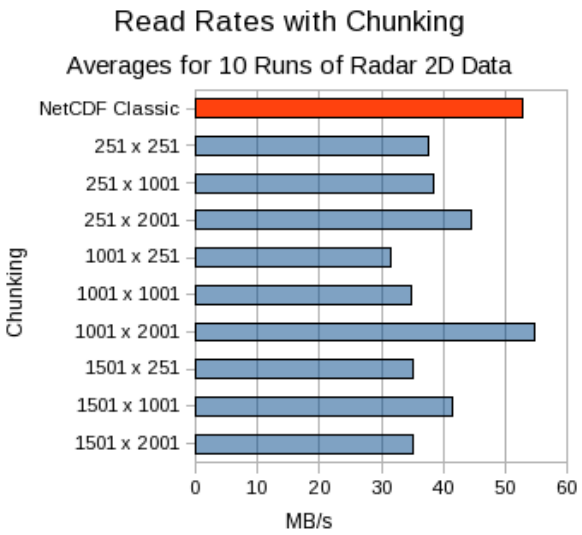

Figure 2: The read rate in MB/s for reading the same data, stored in netCDF-4 files of varying chunk size. As can be seen, the chunk sizes can have a large effect on the read performance. In this case, using 1001 x 2001 chunk sizes is the only setting that performed better than netCDF classic (shown in red).

Chunk sizes should be chosen to yield an amount of data that can be comfortably handled by disk buffers. Chunk sizes that are too small or too large result in poor performance or overly large data files. Since compression and expansion work on individual chunks, specifying too large a chunk size may cause a large portion of a file to be uncompressed when reading only a small subset of the data.

One heuristic for data providers to use is square chunks about one megabyte in size. Chunk sizes should also be chosen so that a whole number multiple of the chunk completely fills the dimension.

Users will also experience better performance by using contiguous storage for variables of fixed size, if data are accessed sequentially.

3.1.5 Parallel I/O

NetCDF-4 supports parallel I/O on platforms that support MPI (the Message Passing library). Parallel I/O in netCDF-4 only works on netCDF-4 data files. Users who wish to user parallel I/O with classic or 64-bit offset netCDF files must use some other solution such as pnetCDF, see Li (2003).

NetCDF-4 users may use special functions to open or create files, to which they can write data in parallel, and from which they can read data in parallel. Parallel data reads can result in significant performance improvements in some high-performance computing applications. Equivalent wrapper functions for the Fortran APIs are provided in the netCDF distribution.

Recent testing on TeraGrid machines showed clear performance gains with parallel I/O, on parallel file systems with low processor counts.

3.2 Using New Features of the NetCDF-4 Data Model

The NetCDF-4 data model supports the Unidata Common Data Model, an ongoing effort to identify a common subset of netCDF, HDF5, and OPeNDAP data model features to improve interoperability for data providers and users.

It is not necessary to use new model features to benefit from using netCDF-4, but they allow a more complete representation of the intent of data providers and of the meaning in complex data structures, sometimes with performance benefits. However, once these features are used, it is no longer possible to read the data with existing netCDF-3 software, even relinked to the netCDF-4 software library. The reading software must be modified to handle the additional model constructs. Since these features are only being introduced in netCDF-4, it will take some time before netCDF applications can be modified to handle the new features. Thus users are cautioned against using these features until software that will be used to access the data has been adapted to handle them.

3.2.1 Multiple Unlimited Dimensions

In the netCDF classic model, only one dimension can be unlimited. This restriction is a direct result of the netCDF classic binary format, which appends slices along the unlimited dimension to the end of the file as the user writes them.

Data chunking allows data to be added in a more flexible way. Data slices are not simply appended to the end of the file — they may be added as new chunks along more than one dimension of a multidimensional variable. This allows data to grow along any number of unlimited dimensions. For example, a dataset representing observations at an indeterminate number of stations and times may allow data to be added for both new times and new observing stations, without copying or restructuring data already in the dataset.

3.2.2 Hierarchical Groups

NetCDF-4 allows variables, dimensions, and global attributes to be organized into hierarchical groups, like the directories and sub-directories of a computer file system. Every netCDF file contains an unnamed root group. If no subgroups are created, all operations take place in the root group. Thus the netCDF classic model is compatible with the group concept — in a netCDF classic file, there is *only* a root group.

In netCDF-4 files, named subgroups can be created and nested to any level. The groups in a file each have their own set of variables, dimensions, group attributes, subgroups, and user-defined types. In a file with multiple

groups, there may be more than one variable or dimension with the same name, since each group provides a scope for names. Groups also provide another way to use multiple unlimited dimensions, because each group may have its own unlimited dimensions.

Potential uses for groups include applications that require:

- containers to "factor out" common information, such as for regions, grids, or model ensembles
- organization for a large number of variables
- multiple name scopes, so that data objects in different groups may use the same names
- large-granularity nested hierarchies
- containers for storing tightly coupled data, for example instrument calibrations or metadata for a specific standard

3.2.3 New Primitive Types

NetCDF-4 makes a distinction between primitive types and user-defined types. Primitive types are those which are intrinsic to netCDF — they are the built-in data types. The netCDF classic data model includes six primitive types: a character type, three integer types, and two floating point types. In netCDF-4, unsigned and 64-bit integer types are added, as is a new string type.

Users should take care using the unsigned integer types, especially the unsigned 64-bit integer type. Neither Fortran 90 nor Java has a 64-bit unsigned type. (They can read such data, but only see it as signed 64-bit integers).

The string type allows compact storage of arrays of variable length string values. Strings are a special case of the variable length arrays described below, but they are built-in types, which do not have to be defined by the user. As with other variable length types, special care must be taken in using C-based interfaces to free storage that the library allocates when string data are read.

3.2.4 User Defined Types

The netCDF-4 data model makes available several kinds of user-defined types. Each type has a name and a definition. Named types are contained in groups, but may be referenced in type definitions in other groups. Both variables and attributes may be declared to be of user-defined types, which allows a natural extension of conventions that require some variable attributes to be of the same type as the associated variable.

User-defined types can express important metadata relationships, provide programming convenience, and represent nested data structures. Specific uses include:

- Representing vector quantities like wind
- Bundling multiple in situ observations together (profiles, soundings)
- Modeling relational database tables
- Representing C structures portably

User defined types are constructed with calls to newly added functions. User-defined types may be nested inside other user-defined types, and the nesting may be arbitrarily deep. (This may make the reading program arbitrarily complex, if not done with care).

The four classes of user-defined types are:

- compound types – like structures in C, these types consist of other types, grouped together by the user. Compound types can contain any of the other user defined types, as well as any primitive type.
- variable length types – allow efficient storage and retrieval of ragged arrays of data. Variable length types are built around a base type, that is, they are variable length arrays of a certain type. That type may be any primitive or user-defined type, including another variable length type.
- enum types – like enumerations in C, these are integer types with an associated symbolic name, stored as an integer in the data file.
- opaque types – named types of user-defined size, which will be accessed atomically by the netCDF-4 libraries.

With little experience using user-defined types, conventions are not yet established for their use. A draft white paper, “Developing Conventions for NetCDF-4,” Rew (2007) is available from the netCDF web site discussing issues in developing CF conventions for netCDF-4. The white paper recommends use of the classic netCDF data model with the netCDF-4 format to obtain benefits for both writers and readers, without breaking backwards compatibility for existing applications that read netCDF data.

4 FUTURE PLANS FOR NETCDF

NetCDF-4.0 will be released a short time after the official HDF5-1.8.0 release. The most recent netCDF-4.0 beta release is available on the netCDF home page, as is the daily snapshot release, which is built every night by the automatic build/test system.

Work has begun on the next versions of netCDF, 3.7 and 4.1. These versions will add the ability to work as an OPeNDAP client. This will allow netCDF users to access files remotely, connecting to OPeNDAP servers over the Internet.

Performance measurement and improvements will continue, as part of continued netCDF-4 development. As an open-software project, we happily accept user contributed experience and code. If you have interesting measurements of netCDF-3 versus netCDF-4

performance on real applications, please send email to the netCDF support address:

support-netcdf@unidata.ucar.edu.

ACKNOWLEDGMENTS

NetCDF was developed and is maintained at Unidata, funded primarily by the National Science Foundation, and managed by the University Corporation for Atmospheric Research (UCAR) Office of Programs (UOP).

NetCDF-4 was developed with a grant from the NASA Earth Science Technology Office under NASA award AIST-02-0071.

We would also like to acknowledge collaboration and development efforts of Mike Folk, Quincey Koziol, Robert E. McGrath, Elena Pourmal, and Muqun (Kent) Yang at The HDF5 Group, and Brian Kelly and Mike Schmidt at Unidata.

REFERENCES

HDF5 I/O Performance HDF5 I/O Performance, HDF and HDF-EOS Workshop VI, December 5, 2002, http://hdfeos.org/workshops/ws06/presentations/Pourmal/HDF5_IO_Perf.pdf

Li, J., W. Liao, A. Choudhary, R. Ross, R. Thakur, W. Gropp, R. Latham, A. Siegel, B. Gallagher, M. Zingale, 2003: Parallel netCDF: A High-Performance ScientificI/O Interface. SC2003, Phoenix, Arizona, ACM.

Zender, C., NCO Users Guide

http://nco.sourceforge.net/nco.html

Russ Rew*, Ed Hartnett, and John Caron, 2006: NETCDF-4: SOFTWARE IMPLEMENTING AN ENHANCED DATA MODEL FOR THE GEOSCIENCES, AMS

Rew, R., 2007, Developing Conventions for NetCDF-4, http://www.unidata.ucar.edu/software/netcdf/docs/nc4-conventions.html