16.3 **NETWORKED SYNCHRONIZATION OF NETCDF DATASETS**

Joe Sirott[1], L.C. Sun[2], Donald W. Denbo[3]
[1]Sirott and Associates, Seattle, WA
[2]National Oceanographic Data Center, Silver Spring, MD
[3]Joint Institute for the Study of the Atmosphere and Ocean, University of Washington, Seattle

1. INTRODUCTION

Scientists in the geophysical data community now universally expect to have remote access to scientific datasets. Many of these datasets are constantly changing as new information is gathered or generated. The most commonly used protocols for accessing these updated datasets are ftp, http, and OPeNDAP. All of these protocols assume that the client has uninterrupted, high speed access to the Internet.

In today's world of mobile computing it can no longer be assumed that clients will have uninterrupted high speed network access. Even if a client has high speed access, network latency in wireless networks can be much greater than in traditional wired connections.

An efficient alternative to the traditional network based protocols for accessing datasets is to mirror the remote dataset on a local machine. As new data is added to the remote dataset, the client dataset is synchronized with the remote dataset. Only changes to the remote dataset are sent across the network.

In this paper we describe cdfsync, an extension to the open source rsync synchronization program that is optimized for efficient mirroring of datasets archived in Unidata's netCDF format over a high latency, low bandwidth network link.

2. THE RSYNC ALGORITHM

Suppose we have a remote file A, a local file B and we wish to synchronize B with A. The rsync algorithm contains the following steps (from Tridgell (2003)):

1. Divide B into a set of non-overlapping fixed size blocks of size S.
2. Calculate two checksums on each of the blocks from B: a weak 32 bit checksum (see below) and a strong 128-bit MD4 checksum.
3. Send the checksums and S to the remote host.
4. On the remote host, calculate weak 32 bit checksums on A of *all* blocks of length S at any offset.
5. Create a 16-bit hash of the 32-bit checksums from 4).
6. Find all blocks of length S in the hash table for A that match the checksums sent for B. If there is a match, verify the match by calculating the strong 128-bit MD4 checksum on the block and comparing it to the value sent from the local client.
7. Send a sequence of instructions from the remote host to the local host for constructing a copy of A. The instruction is either a reference to a local block of B or literal data. Literal data is only sent if there are no blocks of A which match blocks of B.

The weak rolling checksum has the property that it is very inexpensive to incrementally compute, but there is a much greater likelihood of a hash collision.

3. CDFSYNC EXTENSIONS TO RSYNC

3.1 Changes to the Rsync Algorithm

A netCDF file consists of a header block containing file metadata followed by blocks containing data from netCDF variables. We take advantage of this block structured format to speed up file synchronization. Steps 1) and 4) of the algorithm are changed as follows:

1. Divide B into a set of non-overlapping variable sized blocks. The first block corresponds to the netCDF header, and each following block contains the data for a given netCDF variable. Blocks do not cross variable boundaries, and the block size depends on whether the file contains a record dimension and the size of the variable.
4. On the remote host, divide A into a set of blocks in the same way that B was divided. Calculate weak 32 bit checksums on these blocks.

This reduces the hash search time on the remote host and also optimizes the block size (since most netCDF file changes tend to occur as additions to the "record" dimension).

3.2 Compressed File Lists

In-situ atmospheric and ocean data are commonly distributed as large collections of small netCDF files. For example, the Global Temperature Salinity Profile Program (GTSPP) at the National Oceanographic Data Center contains 2.3 million 15 KB netCDF files. Although updates to the datasets often only involve changes to several files, the synchronization program must transfer a list of all the files in the dataset from the client to the host. This transfer time can dominate the

* *Corresponding author address:* Joe Sirott, 7600 Sand Point Wy NE, Seattle, WA 98115; email: Joe.Sirott@noaa.gov

time for synchronization. We use LZ77 compression on the file list to reduce the transfer time.

3.3 In-place Updates

Rsync updates local files by writing the old and new data to a temporary file and, once the transfer from the remote host is complete, renaming the temporary file to the name of the original local file. This can be quite inefficient for small changes to large files. Cdfsync offers an in-place option that instead causes changes to be written directly to the local file.

In-place updates are implemented in a three stage process. First, a delta file is created that contains two types of instructions: block copies that move data from the original file to a new place in the file (a **copy** command) and literal data commands (an **add** command) that add new data to the original file. Block copies can contain cyclic dependencies: for example, if blocks 1 and 2 are swapped then block 1 depends on block 2 while block 2 depends on block 1. A naïve in-place update would result in a corrupt file as block 1 would first be replaced by block 2, overwriting the original block 1 data that is required by the new block 2.

In the second stage, a graph of block dependencies is created. Each edge of the graph represents a dependency, and each node contains a **copy** command. Cycles in the graph are detected and broken by replacing one of the edges in a cycle with a link to a new node with a **save** command. The parent node of the new node has a **load** command prepended to it.

Finally, the commands in the graph are executed by emitting the instructions in a depth-first traversal of the graph, followed by all of the literal data commands. In our example the following commands would be emitted:

save 1
copy 2->1
load 1
copy 1->2

4. RESULTS

The netCDF optimization changes to the rsync algorithm resulted in about a 2x speedup in checksum calculation and comparison with standard rsync using the default block size.

Compression of file lists resulted in a speedup of over 4x when synchronizing a file list containing about 1.45 million netCDF files with no changes over a 256 kbs link.

An in-place update of a 1.5 GB netCDF file where a single record was appended to the remote data file was about 2x faster in cdfsync than in rsync.

5. ACKNOWLEDGEMENTS

This publication was funded by NOAA's High Performance and Computing (HPCC) program.

6. REFERENCES

Tridgell, A and Mackerras, P, 2003: The rsync algorithm
http://samba.anu.edu.au/rsync/tech_report/

Unidata,2004:
http://www.unidata.ucar.edu/packages/netcdf