VGRID: A Generic, Dynamic HDF5 Storage Model for Georeferenced, Grid Data

Chad A. Steed
Naval Research Laboratory
1005 Balch Blvd.
Stennis Space Center, MS 39529-5001
csteed@nrlssc.navy.mil

James E. Braud
The Naval Oceanographic Office
1003 Balch Blvd.
Stennis Space Center, MS 39529-5001
braudj@navo.navy.mil

Kim A. Koehler
Neptune Sciences, Inc.
4301 Pacific Hwy.
San Diego, CA 92110-3127
kim.koehler@navy.mil

Abstract-**This paper describes the Variable resolution GRID (VGRID) storage model designed to support the storage and retrieval of bathymetric data collected through the Precision Underwater MApping (PUMA) System using the Tactical Environmental Data Server (TEDS) and the Naval Oceanographic Office's (NAVOCEANO) Digital Bathymetric Data Base – Variable (DBDB-V) Resolution product. Sponsored by the Space and Naval Warfare Systems Command (SPAWAR, PMW-155), PUMA-TEDS represents a significant advancement in the collection and assimilation of environmental data at global, regional or local levels. Although VGRID has been developed for PUMA bathymetry, its generic implementation makes it suitable for use with any type of environmental data grid through the definition of a product specification.**

Built on NCSA's Hierarchical Data Format version 5 (HDF5), the VGRID model inherits the HDF5 file format and library implementation that is optimized for large-scale scientific data storage. The VGRID model provides a hierarchy of environmental storage objects: files, constituents, and grids. A VGRID file can contain VGRID constituents enabling multi-parameter data storage. VGRID constituents can contain VGRID grids that are identified by resolutions and have grid increments specified in arc minutes, meters, or polar stereographic grid units. The grid interface supports the storage of geographic, polar stereographic, Universal Transverse Mercator (UTM), and Universal Polar Stereographic (UPS) projected grids. Behind the scenes of the VGRID API, a tile scheme is applied to data written to the VGRID file. When VGRID grids are created, compression options can be set for all tiles created in the resolution. The VGRID tile scheme provides the framework for a robust tile caching mechanism, which minimizes the time required to read data from a VGRID file.

The VGRID API uses a "bounce" algorithm to search each resolution and extract the highest resolution data for a point query. In addition, three interpolation options are available for point queries: nearest neighbor, bilinear, and minimum curvature spline. The minimum curvature spline algorithm provides a "feathering" capability that effectively reduces the artifacts that often occur at the resolution boundaries of multiple resolution datasets.

To support the dynamic nature of the PUMA-TEDS system, the concept of co-existing supplemental and historical VGRID files has been developed to support near real-time enhancements to the principle database product. To preserve the generic storage model, the supplementary file concept is not included in the VGRID specification but is left for implementation at the product specification level. Investigation of the PUMA-TEDS DBDB-V model provides valuable insight into the dynamic possibilities of the VGRID file model.

I. INTRODUCTION

The Variable resolution GRID (VGRID) file format provides a generic, geo-spatial interface to the National Center for Supercomputing Applications' (NCSA) Hierarchical Data Format version 5 (HDF5). The development of VGRID is motivated by the need for a common grid file format to store and retrieve bathymetric data collected by the Precision Underwater MApping (PUMA) forward-looking sonar system using the Tactical Environmental Data Server (TEDS) and the Naval Oceanographic Office's (NAVOCEANO) Digital Bathymetric Data Base – Variable (DBDB-V) resolution product.

Sponsored by the Space and Naval Warfare Systems Command (SPAWAR, PMW-155), PUMA-TEDS represents a significant advancement in the ability of tactical platforms to collect and assimilate environmental data at global, regional or local levels of operation. In essence, PUMA-TEDS and other Through-The-Sensors (TTS) developments transform tactical platforms into capable survey vessels with near-immediate capabilities for analyzing collected data.

By harnessing the powerful grid storage features of HDF5, VGRID provides a hierarchy of storage objects for intuitive, geo-spatial data storage. As the top level of the object hierarchy, a VGRID file is created with the VGRID Application Programmer's Interface (API). Providing the next level of the hierarchy, VGRID constituents represent an intermediate level of abstraction between VGRID files and

0-7803-7534-3/$10.00©2002 IEEE

the data grids. Finally, VGRID grids are created under constituents and provide the data storage structure for the VGRID data model.

In order to improve access to file contents, VGRID provides several internal mechanisms that rely on the HDF5 "chunking" interface. Using the "chunking" mechanism, VGRID implements a sophisticated tiling scheme where the atomic unit of storage and retrieval is the tile structure. Other optimizations in the VGRID API, such as tile caching and tile coverage bitmaps, significantly enhance the efficiency of the data extraction function provided by the VGRID constituent interface. In addition, the VGRID grid interface offers compression options to provide efficient data storage and reduced file I/O.

In such dynamic projects such as PUMA-TEDS, the development of an efficient, comprehensive grid file format is crucial to the success of the system. Analysis of the role VGRID plays in the PUMA-TEDS system reinforces the suitability of the VGRID file format and API as a powerful, generic solution for environmental grid products, especially those that will be used in Through-The-Sensors architectures.

II. HDF5

Developed and maintained by the National Center for Supercomputing Applications (NCSA) at the University of Illinois at Urbana-Champaign, HDF5 is a data format specification with supporting library implementation that addresses both the limitations of the older HDF product, HDF4.x, and the current and anticipated requirements of modern systems and applications. In addition to the HDF 4.x improvements, HDF5 developers have built on the lessons learned by similar scientific data formats such as netCDF, PDB, AIO, and MPI-IO.

One of the most significant improvements introduced with HDF5 is the capability to store files larger than 2 gigabytes and an unlimited number of objects per file (HDF4.x is limited to 20,000 objects). HDF5 also provides a simpler, more comprehensive data model that is based on two basic data structures: a multi-dimensional array of record structures (dataset), and a grouping structure. The new HDF5 API is better engineered than its predecessor with improved support for parallel input / output, threads, and other requirements imposed by modern systems.

In the hierarchical data model imposed by HDF5, groups and datasets are the primary building blocks. A HDF5 group is an organizational structure that contains instances of zero or more groups or datasets while a HDF5 dataset is a multi-dimensional array of data elements. Both groups and datasets can be stored with supporting metadata. Similar to the UNIX interface for working with files and directories, HDF5 objects are referenced by their full (or absolute) path names through the HDF5 API and utilities.

To support custom metadata needs, HDF5 groups and datasets may have an associated attribute list. Used to describe the nature and/or the intended usage of a HDF5 object, attributes are small named datasets that are attached to primary datasets, groups or named datatypes. An attribute is composed of a value element that contains one or more data entries of the same datatype and a name element that provides a secondary means of referencing the attribute. When accessing attributes, they can be identified by name or by an index value [5].

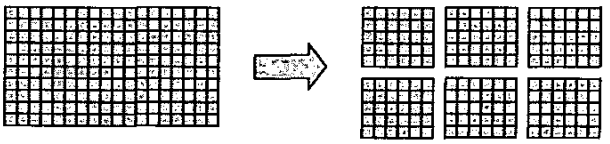

Fig. 1. HDF5 "Chunking" Feature

B. HDF5 Chunking

The HDF5 format offers several storage layout options for customizing the logical storage of a HDF5 dataset. The default storage layout, the contiguous option, forces data to be stored in the same linear fashion that it is organized in memory. For small amounts of data, the compact storage option is useful since data can be stored directly in the object header (compact storage is not yet supported in the HDF5 library).

Perhaps the most powerful storage option is HDF5's chunked layout option that partitions the dataset into equal-sized "chunks" that are stored separately in the file (see Figure 1). The chunking mechanism makes it possible to achieve good performance when accessing subsets of the dataset, even when the subset to be chosen is orthogonal to the normal storage order of the dataset. The blocked layout of a chunked dataset also makes it possible to compress large datasets and still achieve good performance when accessing subsets. With chunking enabled, the dimensions of a dataset can be efficiently extended in any direction and filters can be defined to operate on the chunks of datasets as they are read and before they are written to files. HDF5 provides a robust internal manager for the chunking mechanism where individual chunks are not allocated for a dataset until data is written to the chunk. Read operations in areas of the dataset where the chunk has not been allocated return a user-defined fill value [4].

B. HDF5 File Families

Because HDF5 files can become quite large, HDF5 provides a file family mechanism to split a HDF5 file address space across several smaller files, called file family members. This feature is particularly useful on systems that do not support files larger than 2 gigabytes. In HDF5 file families, each member of the family is the same logical size, though the size and disk storage reported by the file system listing tools may be substantially smaller. The name passed to the HDF5 file create or open function should include a printf(3c)-style integer format specifier that will be replaced with a family member number [4]. The first family member is numbered zero (0) and is created automatically when the file is created. New family members are added as needed by the file API.

C. HDF5 Filters

HDF5 allows chunked data to pass through user-defined filters on the way to or from the disk. As depicted in Figure 2, HDF5 filters operate on the blocks of a chunked dataset and can be arranged in a pipeline so output of one filter

becomes the input of the next filter. Each filter has a two-byte identification number allocated by NCSA and can also be passed application-defined integer resources to control its behavior.

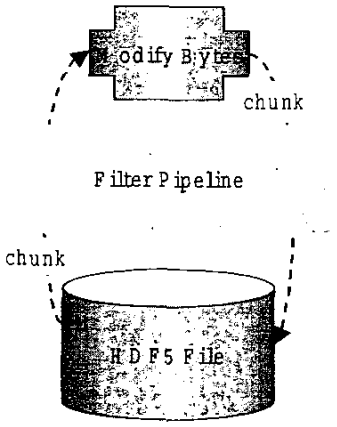

Fig. 2. HDF5 Filter Pipeline

Two types of filters can be applied to raw data I/O: permanent filters and transient filters. The permanent filter pipeline is defined when the dataset is created while the transient pipeline is defined for each I/O operation. During writes to the HDF5 dataset, the transient filters are applied first in the order defined and then the permanent filters are applied in the order defined. For a read operation, the opposite order is used: permanent filters in reverse order, then transient filters in reverse order. Each filter is bi-directional, handling both input and output to the file, and a flag is passed to the filter to indicate the direction. In either case, the filter reads a chunk of data from a buffer, usually performs some sort of transformation on the data, places the result in the same or new buffer, and returns the buffer pointer and size to the calling function [4].

III. VGRID OBJECTS

The VGRID file format is designed to provide a high-level, geo-spatial interface for HDF5 formatted files. Retaining HDF5's hierarchical structure, VGRID uses the HDF5 group and dataset objects as the primary building blocks for its three main components: files, constituents, and grids.

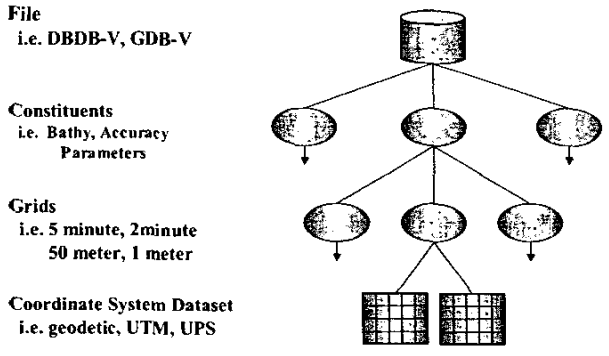

Fig. 3. VGRID Object Hierarchy

A. VGRID Files

As the root of the VGRID object hierarchy (see Figure 3), VGRID files are created through the VGRID API as either single HDF5 files or as HDF5 file families. In addition to the basic methods for creating, opening and closing a VGRID file, the VGRID API provides support methods for querying file information such as the properties for each constituent stored in a file.

If a VGRID file is successfully created or opened, the VGRID API returns a positive integer value that is used to reference the file in other API methods. Since the VGRID file identifier is actually a copy of the HDF5 identifier, it can also be used with HDF5 API functions to store product specific data outside the VGRID storage structure. For example, a custom header providing metadata for the file can be directly written to the VGRID file by using the HDF5 attribute interface with the VGRID/HDF5 file identifier. Although VGRID provides no direct means of accessing such custom data elements, product specific methods can be written to directly access the additional file data through the HDF5 interface.

B. VGRID Constituents

As members of VGRID file objects, VGRID constituents are designed to represent the individual elements of a data product as separate entities in a common file. The constituent interface provides an additional level of data abstraction to the VGRID grid organization that is inherited from VGRID's predecessor file format, DBDB-V version 4.0. In the new DBDB-V VGRID implementation, one constituent is created to store the seafloor depth measurements and two other constituents are created to store the error estimates for each depth (see Figure 4). Although the previous version of the DBDB-V file format provided the capability to store more than one data element for each grid cell, it lacked the expandability of the constituent interface. With the VGRID file format, the need to add a new data element to the file is easily accommodated by creating and populating a new file constituent. However, an equivalent expansion of the previous DBDB-V file format requires extensive changes to both the DBDB-V API and the file structure.

The constituent interface is also useful for separating different versions of the same data type into separate portions of a common file. Recalling the DBDB-V example above, the NAVOCEANO-certified ocean floor depths can be stored in one constituent while another constituent can be used to store real-time depth measurements from a survey vessel. By merging the data from each constituent, the void areas in the real-time dataset can be filled with the historical data to obtain a complete chart of an area of interest while onboard the survey vessel.

When a VGRID constituent is created using the VGRID API, a HDF5 group is created in the VGRID file and general metadata for the constituent is stored as HDF5 attributes. The VGRID constituent creation method accepts a constituent name and the NULL value flag that will be used for all data stored in the constituent. Similar to the VGRID file interface, the VGRID API returns the HDF5 group identifier for each constituent that is opened or created

successfully. Consequently, the constituent identifier can be used to fulfill custom data storage needs that are not specifically addressed by the VGRID API by directly accessing the file through the HDF5 API.

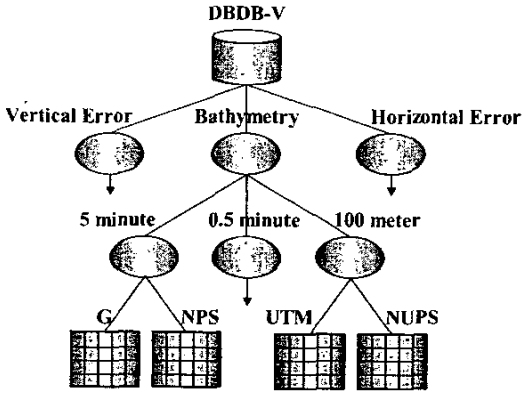

Fig. 4. NAVOCEANO's DBDB-V in VGRID

To provide access to the data stored in a constituent, the VGRID API provides a point extraction function that returns the best (meaning highest resolution) data value for a specific point of a constituent in the database. The finest grid resolution and coarsest grid resolution extraction properties can be adjusted to control the range of grids that will be searched for the requested point. Beginning with the finest grid resolution setting, the point extraction algorithm searches each grid for the requested point until it finds a non-NULL value or the coarsest grid has been searched. If a non-NULL point is found it is returned and the algorithm immediately ends the search but if no non-NULL data is found the algorithm returns the error code that indicates no data exists for the request point.

The VGRID point extraction method offers nearest neighbor, bilinear, and minimum curvature spline interpolation options. If the nearest neighbor interpolation option is chosen, the grid value closest to the requested point is returned along with the coordinates where the grid value is stored in the database. For the other interpolators, the coordinates for the point returned will be the coordinates of the requested point. The transition latitude extraction parameter can also be modified to control point extractions in the overlap zone in the round-earth grid system. A detailed explanation of the transition latitude parameter is given in the following section on the VGRID Grid interface.

C. VGRID Grids

As the low-level public data storage object (see Fig. 3), VGRID grids are created and stored under the constituent level of the object hierarchy. Distinguished by its resolution, each grid must be unique to the constituent in which it is stored. For example, if a 5-minute resolution grid is created for a VGRID file in a particular constituent, only one 5-minute grid is allowed in the constituent.

The grid resolution is quantified with one of three available units of measurement when it is created through the VGRID API: meters, minutes of arc, or polar stereographic grid units. The units of the grid increment are used to determine whether the flat-earth or round-earth grid system will be used to store data in the grid. Using minutes of arc or polar stereographic grid units enables a round-earth grid system, but using meter-based units for the grid resolution activates the flat-earth grid system.

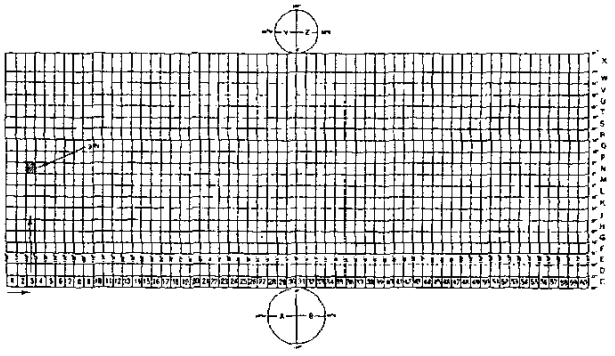

Fig. 5. UTM and UPS Zones for NIMA's MGRS

Based on the projection systems of the National Imaging and Mapping Agency's (NIMA) Military Grid Reference System (MGRS), the flat-earth grid system provides global data storage in a meters-based frame of reference [2]. In order to provide global coverage with minimal distortion, the flat-earth option uses two Universal Polar Stereographic (UPS) grids, one for each polar region, and a Universal Transverse Mercator (UTM) grid system from 84 N to 80 S (see Figure 5). The flat earth grid system stores one tiled grid for each of the 60 UTM zones and for each UPS polar region for a total of 62 data grids per flat-earth grid object [3]. Although VGRID does not impose restrictions on the use of the flat-earth grid system, it is best suited to large-scale datasets with regional coverage [9].

Based on the coordinate systems of the DBDB-V 4.x HDF5 file format, the round-earth grid system provides two polar stereographic grids, one for each polar region, and an equatorial grid that is based on geographic (latitude/longitude) coordinates. VGRID stores one tiled grid for the equatorial grid and each of the polar grids for a total of 3 grids per round-earth grid object. By default, the equatorial region of the round earth system extends from 72 N to 72 S and the polar regions extend from 64 N to 90 N and from 64 S to 90 S at the north and south poles, respectively (see Figure 6) [6].

Accessible through the constituent interface, the transition latitude property controls both the storage and extraction of data in the overlap region between the two polar zones and the equatorial zone. A transition latitude setting of 0, the default VGRID setting, will force the point extraction method to return a weighted value for points requested in the overlap area and makes it possible to store both equatorial and polar grid datasets to the overlap regions. The weighting method is executed by extracting a data value for the requested point from both the polar and equatorial grids. The weighted depth is computed by determining a weight factor for each depth based on the requested point's proximity to the 64 and 72 latitude zone limits. That is, if a requested point in the northern hemisphere is closer to the 64 N latitude limit of the

polar grid, the equatorial point is given more weight than the polar grid value in the weighted depth computation.

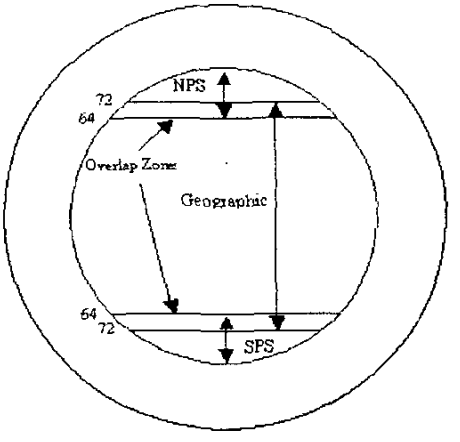

Fig. 6. DBDB-V Version 4.x Grid Model

If the transition latitude is set to 72, equatorial data values are retrieved for point extractions in the overlap region and only equatorial data can be written to the overlap region. Conversely, a transition latitude setting of 64 allows only polar grid extraction and ingest in the overlap area. Essentially, a transition latitude of either 64 or 72 disables the overlap feathering mechanism forcing the API to use polar in the former and equatorial data grids in the latter for point requests in the overlap region. In both of these cases, the transition latitude represents the cut-off latitude between the polar grids and the equatorial grids.

With some data types (such as sediment classification codes), interpolation and/or feathering does not make sense for data retrieval operations. In such cases the transition latitude should be set to either 64 or 72 as described above. Another option to consider is to disable the polar grid storage entirely by setting the transition latitude to 90. When the transition latitude is set to 90, only geographic data extractions and storage are allowed for the entire round-earth grid system. While point requests can still be made in any of the supported projections, including the flat-earth projections, only one geographic grid will be used to store data for the grid whose transition latitude is set to 90.

IV. VGRID COMPRESSION

As environmental data collection platforms continue to produce higher resolution data grids, the need for robust compression in scientific data formats will remain a major factor in the success/failure of a geo-spatial data format. The VGRID grid interface provides a compression option that may be one of three possible values: no compression, zlib "deflate" compression, or the geofilter compression methods. As its label suggestion, the no compression option stores "chunked" HDF5 datasets for each grid without any compression. This method is appropriate for small VGRID files or in situations where disk space is not a problem.

The zlib compression option is inherited from the HDF5 library through its inclusion of the zlib "deflate" filter. During the configuration of the HDF5 library, if the presence of the zlib library version 1.1.2 is discovered a filter is defined that is referenced as H5Z_FILTER_DEFLATE. H5Z_FILTER_DEFLATE is a symbolic reference to zlib, "a general-purpose, legally unencumbered, lossless data-compression library for use on virtually any computer hardware and operating system [7]." The zlib compression method, referred to as "deflate", operates on blocks of data where each block is compressed using a combination of the LZ77 [11] algorithm and Huffman coding producing typical compression ratios on the order of 2:1 to 5:1 [8]. Since this compression method has the potential for generating compressed data that is larger than the original, the H5Z_FLAG_OPTIONAL flag should be turned on so such cases can be handled gracefully by storing the original data instead of the compressed data [4]. If zlib compression is requested for a grid, the VGRID API specifies that the H5Z_FILTER_DEFLATE filter should to be used for each HDF5 dataset that is created in the dataset. The zlib compression option also accepts an aggression factor integer in the range 0 to 9 where 0 is the lowest level of compression and 9 is the highest level of compression.

An additional compression option has been incorporated into the VGRID API as a HDF5 filter. This compression option uses a modified delta encoding technique, run-length encoding and bit packing to yield better compression results than the zlib filter [1].

The geofilter compression algorithm generally takes advantage of the naturally occurring redundancy that exists between adjacent cells for many geophysical properties. That is, the value of the property of a geo-located cell that is immediately adjacent to another cell is likely to remain the same or change very little from the adjacent cell. In cases where there is a series of adjacent cells that do not change (flat areas of Bathymetry), a run length compression series is used such that a value indicating the number of continuous non-varying adjacent cells is stored. Where there are fluctuations in the value between adjacent cells, a delta length compression series is used such that only the difference in the values is stored for each cell in the series.

In the compression process, the bit storage needed for delta encoding versus the bit requirements of the generation of a new (delta or run length) series is continuously evaluated so that an optimum dynamic bit size for the deltas for each series is achieved. To ensure that only the values of adjacent cells are evaluated in the compression technique, the rows of the grid are processed in a "snake-like" fashion so that the last column of a row is immediately followed by the last column of the next row and each row is traversed in the opposite direction of the previous row. Before committing a compressed cell series to storage, the geofilter algorithm "looks ahead" to determine if it is more efficient to expand the number of delta bits needed to store a next cell's value or possibly continue to store repeat values in a delta series versus closing the current series and suffering the storage overhead of the creation of a new compression series.

With gridded geospatial data, there is often a need to handle specialized or flag values that may occur frequently in the grid but may violate the normal rules of redundancy that occur in the natural environment. There are two bathymetry

flag values that are used in the DBDB-V that if not handled by special provision would significantly degrade the performance of the geofilter compression technique. The two flags currently in use for DBDB-V choose values that are outside of normal acceptable ranges of bathymetric data to indicate values that are over land and/or NULL or missing data. The geofilter algorithm handles these flags by expanding the range of the delta storage domain to accommodate the flag values. For each gridded data stream that is processed, however, a predetermination of the existence of actual flag data is made so that no additional delta range is used if the flags do not occur in the data stream. The algorithm is designed to handle up to 4 sets of pre-determined flag occurrences in the data stream.

Because the optimum number of bits needed to store the length of the compression series (bits needed to store the maximum number of delta values or repeat values in a series) cannot be easily determined, the geofilter algorithm repeats the compression technique 7 times while cycling through the range of from 3 bits to 9 bits (max series of from 8 to 512 values respectively) for this value and selecting the case that yields the highest compression ratio. While the geofilter compression speed compares favorably against gzip and other compression techniques, this part of the geofilter process is a good candidate for multiprocessor vectorization since the multiple compressions could be done in parallel.

Finally, much of the good compression results achieved by geofilter can be attributed to the reasonable scaling of the data. In the case of bathymetry, it makes little sense in storing the mostly interpreted values to the 1000's of a meter or greater precision when the original collected data (where soundings may actually exist) is seldom even at meter accuracy. The floating-point values are scaled for each grid cell and are then converted to their integer equivalent. In most cases for DBDB-V, a 0.1-meter precision is still maintained by multiplying the values by 10 prior to conversion to integer. Proper scaling of the incoming data allows for smaller delta values and a higher potential for repeat values, which improves the efficiency of the compression. A higher ratio of compression could be achieved if a more realistic whole-meter precision were used.

It is important to note that once the compression method is assigned to a grid, the data is accessed externally in the same as a data grid with no compression. All compression and de-compression operations are applied at the lowest level of file I/O in the VGRID API, the chunk level. As a chunk is read from the HDF5 file, it is decompressed in such a manner that VGRID will only see the un-compressed data value. Conversely, when a chunk is written from the HDF5 file, it is compressed and the compressed bit stream is written to the disk inside the HDF5 low level API.

V. VGRID API SPECIAL FEATURES

The VGRID API is a comprehensive library written in the C programming language for creating and manipulating VGRID objects. The API includes a grid ingest interface and an access interface as well as basic methods for creating, opening, and closing objects. Several internal mechanisms are maintained in the VGRID API to reduce the time required to read the contents of a file.

A. VGRID Tile Scheme

Internal to the VGRID API, the square dimensioned tile is the atomic unit of storage in VGRID grids. The VGRID tile scheme is implemented using the HDF5 chunk storage option where the chunk size is defined when the grid creation function is called for round-earth or flat-earth grids. Use of the HDF5 chunking mechanism allows the VGRID API to access the HDF5 datasets of a VGRID grid in the same manner regardless of the dimensions of its tiles. As a result, the tile size (chunk size) can be modified to fine-tune the I/O efficiency of a grid.

B. VGRID Tile Cache

Since VGRID tiles are the smallest unit of storage in the VGRID grid, all extractions from a VGRID constituent require the reading of tiles from the grids stored in the constituent. In order to reduce the number of I/O operations required for area based extractions that use the VGRID point extraction interface, the VGRID API maintains an internal tile caching mechanism. As the API reads tiles from grids in a constituent group during point extractions, it stores the most recently read tiles in memory. Each time a tile read operation is requested in the VGRID API, the tile cache is first checked for the presence of the target tile. If the tile is in cache its contents can be accessed directly as opposed to having to read the tile from the HDF5 dataset. The tile cache has a maximum size that can be modified through the VGRID API initialized function. If a tile is to be inserted into the tile cache and there is not enough cache memory left to insert the tile, tiles are released from the back of the tile cache, the tiles that have been least frequently accessed, until enough free space is available for the new tile.

The fact that the VGRID API will read data by tiles from grids means the tile cache mechanism can be best utilized by reading data through the point extraction interface in chunks that are aligned with the tile dimensions of a particular grid. However, when several resolutions exist in a file, it is not always feasible that the resolution from which the extraction will be made is known before the extraction is made. In this case the maximum tile cache size can be modified to ensure that a full row of the finest resolution tiles for a particular area (or for the entire database) can be placed into the tile cache during a row first, scanline read from the VGRID constituent.

C. VGRID Coverage Bitmap

The VGRID API also maintains an internal coverage bitmap to record where tiles exist in the file and where they do not. Each grid in the VGRID file maintains its own coverage bitmap. The bitmap is implemented as a grid where each tile is represented as a single bit. If the bit is set to 0 the tile either exists or has not been checked. If a bit is set to 1, the API has accessed the tile and discovered that it does not exist. The bitmap mechanism allows the API to remember where it does not have data so that future tile read requests will quickly determine that the tile does not exist. This is necessary because every tile request using the HDF5

chunking will return data even if the chunk has never been written to. Similar to the theory behind HDF5 chunking, the bitmap is a dynamic structure that has a maximum size and allocates bitmap regions when a set bit request is made to that region. If the bitmap becomes larger than the maximum bitmap cache size, bitmap regions at the end of the bitmap region list are removed until sufficient space is available for the new bitmap region.

VI. VGRID in PUMA-TEDS

As an example, the VGRID file format and API is used as the common file format for the PUMA-TEDS system (see Figure 7). When the PUMA system produces a bathymetric data grid it writes the data to a VGRID file and transmits it to the local TEDS server. The TEDS server then archives the PUMA data file and passes the data grid to the DBDB-V ingest API. The DBDB-V ingest API is a product specific API built on the VGRID API. The DBDB-V ingest function accepts the VGRID file and uses it to update the DBDB-V supplemental partition. The DBDB-V supplemental partition is a dynamic VGRID file that is used to complement the historical OAML-certified DBDB-V that is stored in TEDS. As data is ingested into the supplemental VGRID file, the DBDB-V algorithm performs any required merging of data that the new data grid may overlap.

After ingesting the PUMA data updates, the TEDS can then access the DBDB-V supplemental and historical VGRID files as one file through the customized DBDB-V access API. The DBDB-V access routines extend the bounce algorithm of the VGRID point extraction to include the addition of the supplemental VGRID file. This bounce algorithm first searches the supplemental file for the requested data and then searches the VGRID file if non-NULL data is not found in the supplemental file. TEDS provides the mechanism to transfer individual PUMA update files or the entire supplemental partition off-board the host vessel [10].

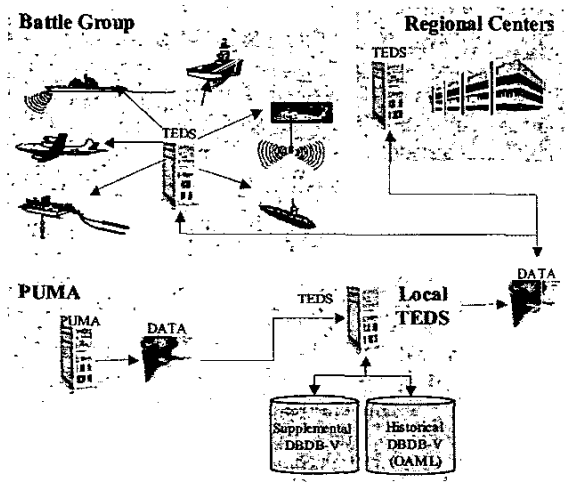

Fig. 7. PUMA-TEDS Data Flow

VII. CONCLUSION

Although VGRID has been designed to support the storage and retrieval of bathymetric data collected from the PUMA system, its generic model makes it suitable for use in any type of environmental data grid through a customized product specification. The current VGRID version inherits the powerful features of the HDF5 file format providing three basic geo-spatial objects: files, constituents, and grids. Tile-based mechanisms that operate behind the scenes in the VGRID API optimize access and storage operations for grid data. As the common file format in PUMA-TEDS, VGRID equips the system with advanced grid storage features that are attractive to most environmental data providers, such as NAVOCEANO.

In the future, NRL-SSC will investigate the inclusion of a triangulated irregular network (TIN) storage structure into VGRID to either replace or complement the current rectangular grid storage system. Such adaptive mesh structures are likely to become the next generation of grid storage that overcomes most of the deficiencies of rectangular, regularly spaced grids. In addition, NRL-SSC will explore the migration of VGRID to a pure geo-spatial database access layer that operates independent of the physical file storage format. The layered API hierarchy in VGRID lends itself to a modular file storage API that will allow the replacement of the HDF5 file format base with a more sophisticated Data Base Management System (DBMS). Incorporation of an advanced DBMS base will become increasingly important as more and more dynamic databases are introduced into two-way portal architectures such as TEDS.

The recent strides in low-cost, high-speed computer processing combined with major advances in communications bandwidth available to Navy ships, aircraft, and submarines, have made through-the-sensor oceanographic and atmospheric data collection feasible. This concept, which exploits the tactical dwell of sonar and radar systems, will result in the ability to collect, decimate, and store volumetric data for use in Fleet Tactical Decision Aids (TDAs). These data types include bathymetry (as noted in the PUMA effort), bottom sediment type (via fathometers), geoacoustic parameters such as scattering (via sonars), atmospheric refractivity (via radars), and many others. The VGRID effort will undoubtedly play a critical role in the development and fielding of these critical new technologies.

ACKNOWLEDGMENTS

This research was sponsored under Program Element No. 0603704N by the Oceanographer of the Navy (CNO N096) via SPAWAR PMW 155. This report and the VGRID file format are intended to support the PUMA-TEDS Through The Sensors program for near real-time assimilation of historical DBDB-V with PUMA bathymetry. The Naval Research Laboratory would like to thank the following individuals for their contribution to the production of this report and VGRID: CDR John Kusters (SPAWAR, PMW-155), Dave Kubik (ASTO), Jim Broughton (ASTO), Paul Stephens (NAVOCEANO), William Rankin (NAVOCEANO), Jerry Landrum (NRL-SSC, retired), Rick Bailey (ARL UT), Steve Lacker (ARL UT), David Wight (ARL UT), Keith Kelley (Anteon), Dave Parillo (Anteon), and Michelle McGregor (Lockheed Martin).

REFERENCES

[1] Braud, James E., John L. Breckenridge, James E. Current, Jerry L. Landrum. "Data Base Structure to Support the Production of the Digital Bathymetric Data Base," NRL Report (NORDA Report No. 236), Naval Ocean Research and Development Activity, Stennis Space Center, MS 39529, 1989.

[2] Defense Mapping Agency, "DMA Technical Manual 8358.1: Datums, Ellipsoids, Grids and Grid Reference Systems," Fairfax, VA, 20 Sept. 1990.

[3] Defense Mapping Agency, "DMA Technical Manual 8358.2: The Universal Grids: Universal Transverse Mercator (UTM) and Universal Polar Stereographic (UPS)," Fairfax, VA, Sept. 1989.

[4] HDF5 User's Guide. Retrieved May 16, 2002, from National Center for Supercomputing Applications, Hierarchical Data Format Web site: http://hdf.ncsa.uiuc.edu/HDF5/doc/H5.user.html

[5] Introduction to HDF5 Release 1.4. Retrieved May 16, 2002, from National Center for Supercomputing Applications, Hierarchical Data Format Web site: http://hdf.ncsa.uiuc.edu/HDF5/doc/H5.intro.html

[6] The Naval Oceanographic Office, "Data Base Description for Digital Bathymetric Data Base – Variable Resolution (DBDB-V) Version 3.0," July 2000.

[7] Roelofs Greg and Jean-loup Gailly. "zlib Home Page." Retrieved May 16, 2002, from the zlib Home Page Web site: http://www.gzip.org/zlib/

[8] Roelofs Greg and Jean-loup Gailly. "zlib Technical Details." Retrieved May 16, 2002, from the zlib Home Page Web site: http://www.gzip.org/zlib/zlib_tech.html

[9] Steed, Chad, James E. Braud. "A Flat Earth Model for DBDB-V," NRL/FR/7440--02-10,025, Naval Research Laboratory, Stennis Space Center, MS 39529, Submitted for publication 2002.

[10] Steed, Chad, Jerry Landrum, and Chris Moreau. "PUMA-TEDS Technical Execution Plan," NRL/FR/7440--02-10,003, Naval Research Laboratory, Stennis Space Center, MS 39529, June 2002.

[11] Ziv J., Lempel A., "A Universal Algorithm for Sequential Data Compression", IEEE Transactions on Information Theory, Vol. 23, No. 3, pp. 337-343.