A High Performance System for Viewing, Querying, and Retrieval of Geospatial Data Distributed Across a Beowulf Cluster

Mahdi Abdelguerfi[1], Venkata Mahadevan[1], Nicolas Challier[1], Maik Flanagin[1], Kevin Shaw[2*], and Jay Ratcliff[3]

[1] Department of Computer Science, University of New Orleans
New Orleans, Louisiana 70148, USA
Voice (504) 280-7076
{mahdi, venkata, nchallie, maik}@cs.uno.edu
http://www.cs.uno.edu
[2] Naval Research Laboratory, Stennis Space Center
Mississippi 39529, USA
shaw@nrlssc.navy.mil
[3] U.S. Army Corps of Engineers
New Orleans, Louisiana 70118, USA
jay.j.ratcliff@mvn02.usace.army.mil

Abstract. The size of many geospatial databases has grown exponentially in recent years. This increase in size brings with it an increased requirement for additional CPU and I/O resources to handle the querying and retrieval of this data. A number of proprietary systems could be ideally suited for such tasks, but are impractical in many situations because of their high cost. On the other hand, Beowulf clusters have gained popularity for providing such resources in a cost-effective manner. In this paper, we present a system that uses the compute nodes of a Beowulf cluster to store fragments of a large geospatial database and allows for the seamless viewing, querying, and retrieval of desired geospatial data in a parallel fashion i.e. utilizing the compute and I/O resources of multiple nodes in the cluster. Experimental results are provided to quantify the performance of the system and ascertain its feasibility versus traditional GIS architectures. Keywords: Parallel and distributed computing, Geospatial database, Beowulf cluster.

1 Introduction

Geospatial data is constantly being gathered by a variety of sensors, satellites, and other devices. The quantity of data collected by these units is staggering – NASA's Earth Observing System (EOS), for example, generates 1 terabyte of data on a daily basis [1]. The design and implementation of GIS databases have attracted considerable interest and research efforts in recent years [18, 19] and the performance prob-

* The work of Kevin Shaw was partially funded by the National Guard Bureau with program manager Major Mike Thomas.

lems associated with large databases have been widely documented by researchers for many years and several techniques have been devised to cope with this. One of the techniques that have gained popularity in recent years is parallel processing of geospatial database operations. Indeed, geospatial database operations are often time-consuming and can involve a large amount of data, so they can generally benefit from parallel processing [2, 3, 4].

There are 3 broad classes of parallel machines: shared memory systems, shared disk systems, and shared nothing systems. In a shared memory architecture, every processor has direct access to the memory of every other processor. In other words, there is only one physical memory address space for the entire system. A shared disk architecture, as its name suggests, provides shared access to the disks in the system from any of the processors via the communication network. Memory in this architecture is managed locally at each processor. Finally, in a shared nothing architecture, each processor has its own local memory and disk – nothing is shared and all communication between the processors is accomplished via the communication network. The shared nothing architecture is the most widely used design for building systems to support high performance databases, primarily due to its relatively low cost and flexible design.

Beowulf compute clusters are probably the best-known example of shared nothing machines in existence today. A number of factors can be attributed to this including the emergence of relatively inexpensive but powerful off-the-shelf desktop computers, fast interconnect networks such as Fast Ethernet and Gigabit Ethernet, and the rise of the GNU/Linux operating system. A compute cluster, for all intents and purposes, can be defined as a "Pile-of-PCs" [5] interconnected via some sort of network. Each node in the cluster usually has its own processor, memory, and optionally an I/O device such as a disk. However, it is important to note that a cluster is not simply a network of workstations (NOWs) – in a cluster, the compute nodes are delegated only for cluster usage and nodes typically have a dedicated, “cluster-only” interconnect linking them. In the Beowulf paradigm, all of the compute nodes (also called slave nodes) are isolated on a high-speed private network that is not directly visible to the outside world. A single computer called the “master” or “head-end” provides a single entry point to the cluster from the external network. This machine is sometimes referred to as the login or submit node. Essentially, the master node is a system with 2 network interfaces – one connected to the private Beowulf network and the other connected to the regular LAN. Users of the cluster will typically log in only to the master node. From here, they can spawn processes that will execute on the slave nodes. Beowulf clusters are typically applicable to any area of research where a speedup in program execution time is possible by splitting a large job into several sub-tasks that can run concurrently on the compute nodes. This capability provides an intriguing setting for evaluating the use of such a cluster for hosting large GIS databases, where there is an ever-increasing need for greater computing power to process and query the massive amounts of information being stored.

Partitioned parallelism is the main source of parallelism in a parallel shared-nothing system [16]. This is achieved through the declustering of the spatial data across the different computer nodes. The spatial database operation is then run locally on each node. Of course, the clustering scheme should distribute the data evenly across the computer nodes, and the design of the parallel spatial operation should be

such that each node uses only the data fragment stored locally. Data clustering techniques implemented in our system includes hashing, round robin, spatial tiling [17], Hilbert space-filling Curve [14], and R*-tree based [13].

The work presented in this paper attempts to improve the performance aspects of geospatial querying as they pertain to a large geospatial database distributed across the compute nodes of a Beowulf cluster and is organized as follows. In Section 2 we describe the fundamental design of our Beowulf cluster and the environment in which geospatial data is stored and organized. Section 3 describes the functionality and design of the viewer used to query and retrieve geospatial data from the cluster. Section 4 discusses the test data selected for use in the process of evaluating overall system performance and explains the methodology used to experimentally gauge geospatial querying performance. An analysis of the performance results is provided in Section 5. In Section 6, we conclude with a discussion of the results and suggestions for future work.

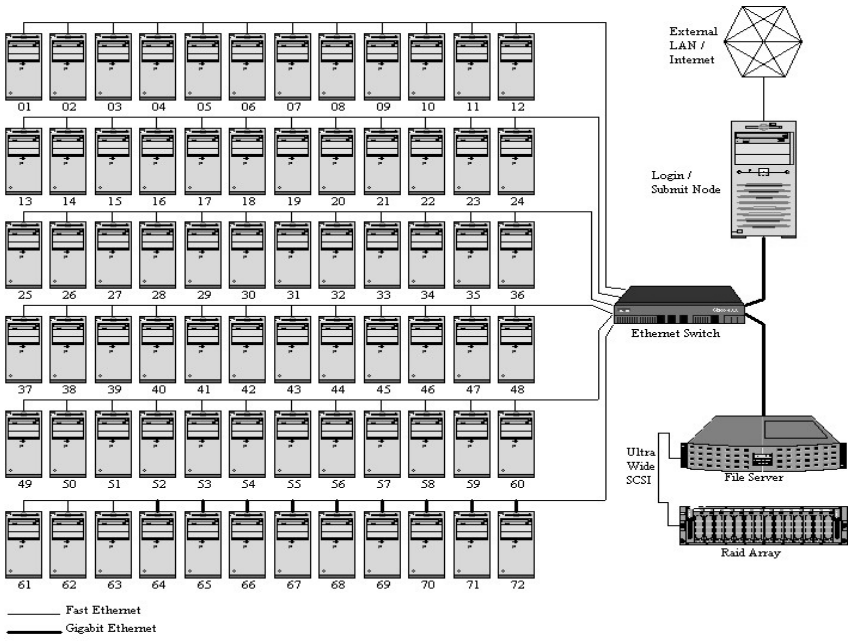

Fig. 1. Overview of 72-node Beowulf Cluster

2 Overview of System Organization

The Beowulf cluster housed at the Department of Computer Science at the University of New Orleans consists of 72 compute nodes, 1 login/submit node, and 1 file

server. 63 of the slave nodes are 2.2 GHz Intel Pentium IV systems with 1 GB of memory, 20 GB of local disk storage, and Fast Ethernet networking; the remaining 9 nodes are 2.4 GHz Intel Pentium IV systems with 1GB of memory, 20 GB of local disk storage, and Gigabit Ethernet networking. The file server is a dual 1.4 GHz SMP Intel Xeon system with 2 GB of memory, 500 GB of RAID-1 disk storage, and Gigabit Ethernet networking. Last, but not least, is the master node which is a dual 2.2 GHz SMP Intel Xeon system with 2 GB of memory, 300GB of disk storage, and 2 Ethernet interfaces. The interface that links the cluster with the private Beowulf network is Gigabit Ethernet; the external interface is 100BaseTX. All of the systems in the cluster are networked together by a Cisco Catalyst 4000 series switch with 10/100/1000 auto-sensing ports and a 12-Gbps backplane. Redhat Linux 7.3 is the operating system for all nodes in the cluster and the Warewulf Clustering System [6] is used to provide support utilities for cluster monitoring and maintenance. This cluster was recently benchmarked using HPL 1.0, a portable, freely available implementation of the standard High Performance Computing Linpack Benchmark. HPL solves a random, dense linear system in double precision (64 bits) arithmetic on distributed memory computers [7]. Benchmarking yielded a "theoretical peak" performance of approximately 63 Gigaflops. The amount of raw computing capacity provided is therefore quite substantial. Figure 1 provides a broad overview of the cluster architecture.

In order to store geospatial data and issue geospatial queries on it, a DBMS that provides spatial extensions and support for geospatial querying is required. For this purpose, the PostgreSQL object-relational DBMS [8] was selected. PostgreSQL is a good choice for use on a GNU/Linux system (such as our Beowulf cluster) because of the robust, native support that it provides on this platform. PostgreSQL also has a spatial extension called PostGIS [9] that follows the OpenGIS Consortium's "Simple Features Specification for SQL", a proposed specification to define a standard SQL schema that supports storage, retrieval, query, and update of simple geospatial feature collections via the ODBC API [10]. Each of the slave nodes in the cluster executes the PostgreSQL server engine and has its own database instance stored on its local disk (the file server is not used because of the potential performance penalty associated with a large number of nodes accessing a shared filesystem simultaneously). Geospatial data is distributed as evenly as possible across the slave nodes; depending upon the number of slave nodes participating in the system, each node may hold more or less data in its local database. That is, the data with id k = (feature-id mod n) is stored on node k where n is the total number of nodes. The entire data set is organized by feature id, feature type, and feature geometry. The feature id is simply an integer that uniquely identifies each feature in the database. Since it is the primary key, data (features) is split among the nodes based on the number of unique features comprising the entire data set i.e. there is no replication or duplication of data. Another advantage of this method is that if we know the id of a feature, we can easily locate the node in which the feature is stored. The feature type may be one of 7 different types specified by the "Simple Features" specification of the OpenGIS Consortium [10]. These include point, linestring, polygon, multipoint, multilinestring, multipolygon, and geometrycollection. Finally, the feature geometry is the set of geospatial coordinates comprising a particular feature. It is represented in the database as human readable text and the coordinates may be either 2-dimensional or 3-dimensional.

3 Functionality and Design of Data Viewing, Querying, and Retrieval Component

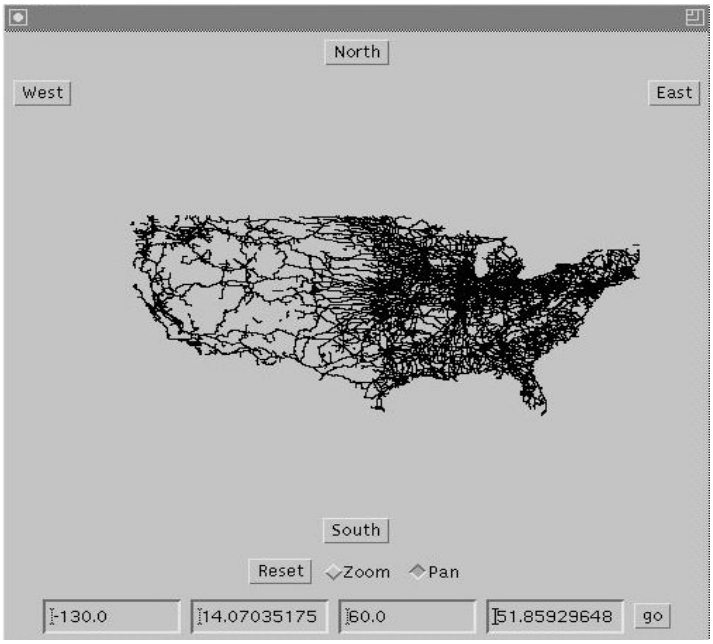

Fig. 2. User Interface for Viewing and Querying Geospatial Databases

The system we have implemented is designed to exploit the parallel nature of the Beowulf cluster in order to provide fast response times when performing geospatial queries on a large data set. In particular, the "shared-nothing" parallel I/O and memory architecture of the Beowulf cluster is expected to provide a measurable performance gain in geospatial querying performance and data retrieval operations versus a single processor architecture. The functionality of the system can be broken down into 3 parts, namely: (i) geospatial query / search (ii) retrieval / download and (iii) visualization / display of data. Each of these steps is further described in detail below.

Several types of geospatial queries are supported by the system's user interface that we implemented (see Figure 2 for a screen capture). These include area of interest (bounding box) queries, range / distance queries, and combination queries. The user interface itself runs on the master node and is responsible for spawning a separate thread to query each node in the system for relevant data encompassed within the geospatial query i.e. each query is executed in parallel across the selected nodes in the cluster that contain fragments of the geospatial database. This design makes good use of the independent disk and memory subsystems of each slave node. Once a node has finished searching its data fragment, the results (if any) are returned to the master node for visualization. Figure 3 succinctly depicts the entire process. In addition to the

above functionality, the user interface also supports distributed panning of a large map – if the user pans the view and the new map data has not yet been retrieved, it is queried for on the fly by the interface.

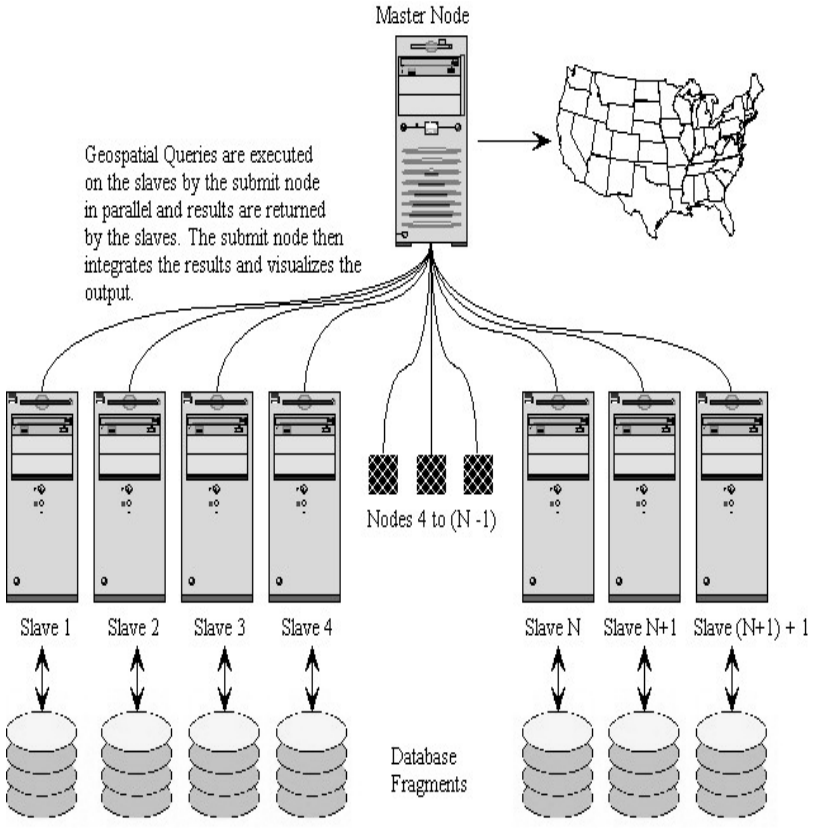

Fig. 3. Putting It All Together

4 Description of Test Data and Benchmarking Methodology

A large geospatial data set was obtained from the Bureau of Transportation Statistics (BTS): the 2002 National Transportation Data Hydrographic Features. The hydrographic features are a state-by-state database of both important and navigable water features creating a single, nationwide hydrography network containing named arcs and polygons [11]. The spatial domain of this data set is West: -179.252851, East: 179.830622, North: 71.441056 and South: 17.831509. The size of this data set is approximately 310 MB in compressed ESRI Shapefile format. When imported into a PostgreSQL database, the size is about 300 MB with 517537 unique features. The ge-

ometry of the features in this data set is represented as object type *multilinestring* from the OpenGIS Specification aforementioned. Due to the large number of unique features present in the data set and the sufficient degree of complexity present in each feature, distribution of the data across multiple nodes is expected to be beneficial.

The first step in the benchmarking process was to load the test data into the various database instances dispersed across the cluster nodes. This was accomplished as follows: a system script executes at the master node which batch loads fragments of the total data set into each database instance in the cluster – the number of fragments of course depends on the total number of slave nodes participating in the system. Once the test data has been loaded, queries can be executed on the slave nodes. The geospatial queries are similar to standard Structured Query Language (SQL) queries but provide a richer set of functions for dealing with geospatial data. For example, it is possible to express queries such as "return all features within 100 units of a particular point" using the PostGIS extensions to PostgreSQL. The following queries were selected for use in our evaluations:

Distance Query:

```
select gid, the_geom from [table_name]
where distance (the_geom, GeometryFromText(
`POINT(X Y)`, -1)) < [distance];
```

Window Query:

```
select gid, the_geom from [table_name]
where the_geom && GeometryFromText(
`BOX3D (X1 Y1, X2 Y2)`::box3d, -1);
```

The first query specifies that the feature id and feature geometry that lie within a particular distance from a point should be returned; the second query is a bounding box query – it simply specifies that all features that lie with the specified bounding box should be returned. It is conceivable that the distance function will be a computationally intensive operation within the queries because a separate distance calculation is required for finding the distance between the specified point and each point in the table (in other words, a separate distance calculation for each row of the table is needed).

The queries were tested under 3 different cluster configurations: 8 slave nodes, 16 slave nodes, and 32 slave nodes. The reference system was the master node which holds a complete copy of the data set – it was benchmarked separately. The benchmarking process itself works as follows: the master process on the master node executes each query concurrently on each of the slaves and forks a sub-process to keep track of the time it takes to retrieve the desired data. Each slave processes the query independently and returns the results to the master. When the last slave in the group has finished processing, the master process combines the results and its timing subprocess computes the time in seconds that the entire operation took to complete. In the next section, we will compare and contrast the benchmark results obtained under different cluster configurations as well as those of the reference system.

5 Performance Results

Table 1 summarizes the results for the bounding box query obtained from benchmarking using 8, 16, and 32 compute nodes (the references system is also included for comparison purposes). Each benchmark was run several times for all configurations; only averages are reported. In addition, the time reported is the total time it took from the start of the query to the visualization of the final output.

Table 1. Summary of results for bounding box (-95:30:10:10)

Number of nodes	Time (in seconds)
1 (reference system)	942.096
8	25.272
16	24.203
32	23.999

The results presented in the above table show that there is certainly a tangible performance gain that is associated with distributing and querying the data set across multiple nodes. Even with only 8 nodes, the query executed in less than a minute versus about 15.7 minutes for a single processor (the reference system). With more than 8 nodes, the difference is less marked, although there is a slight improvement with 16 and 32 nodes. This is probably attributable to the size of the data set versus the increase in communication overhead when more nodes are involved in the querying process. That is, network communication overhead caused by a large number of nodes returning query results to the master negates any processing gains achieved by having more processors and disks in the overall system. However, a larger data set may warrant the use of an even greater number of nodes.

Table 2 summarizes the results for the distance query obtained from benchmarking using 8, 16, and 32 compute nodes. As in the previous table, the time reported is the total time it took from the start of the query to the visualization of the final output.

Table 2. Summary of results for distance query (-100:35) distance < 5

Number of nodes	Time (in seconds)
1 (reference system)	846.126
8	17.741
16	16.453
32	15.729

The results for the distance query are in line with what was achieved with the bounding box query – a sharp difference in performance between the parallel executions of the queries versus those of the single processor execution. Once again, however, there was only a slight improvement in performance when the number of nodes was doubled or even quadrupled. It is our conjecture that this is due to the size of the

data set used – we expect that a larger data set will benefit more from additional nodes.

6 Conclusions and Future Work

A scheme for viewing, querying, and retrieving geospatial data distributed across the compute nodes of a Beowulf cluster was discussed in this paper. The performance of this scheme under various cluster configurations was benchmarked, and the results certainly advocate using a compute cluster for mining geospatial data – large data sets such as the hydrographic data can benefit greatly from a reduction in query processing time when distributed across multiple nodes. However, it is expected that such a cluster will really make its mark when handling even larger data sets, on the order of tens of gigabytes or even terabytes of data, volumes that are simply unsustainable on even the most powerful non-parallel systems.

Current work in progress includes the parallel implementation of multi-scan queries such as the computationally expensive spatial join operation [12]. Towards this end, a semi-dynamic clustering scheme in which one of the two relations (usually the largest of the two) involved in the spatial join is declustered using an R*-tree [13] and the tree leaves placement policy is based on the well-known Hilbert curve [14]. This new approach has the advantage of dramatically reducing the replication of tuples among the computer nodes. Preliminary experiments indicate that this newly devised algorithm compares well with existing ones such as the parallel clone spatial join algorithm described in [15]. Underway is the development of an analytical model that will permit the study of the characteristics of the devised parallel join algorithm.

There is still a great deal of work that can be done in addition to that presented in this paper. In particular, we would like to make the system we have presented more modular in order to accommodate different types of data storage back-ends and visualization capabilities. Methods to handle both 2D and 3D visualization in parallel are of particular interest to us. Also, we are looking into adaptive clustering (adaptive to the size and type of data) and load balancing techniques to enhance the scalability of the system. In addition, we are interested in implementing a high-level web services application that would use this system as its back-end for providing rapid access to large data sets via a web browser interface.

Acknowledgements

We would like to thank the developers of the wonderful GeoTools Java GIS Toolkit (http://www.geotools.org) for their product, which helped considerably in developing the viewing components of the user interface.

References

1. Shashi Shekhar, Sanjay Chawla, "Spatial Databases: A Tour", Prentice Hall, 2002
2. Hector Garcia-Molina, Jeffrey D. Ullman, and Jennifer Widom, "Database System Implementation", Prentice Hall, 2000
3. Mahdi Abdelguerfi, Kam-Fai Wong (Eds.), "Parallel Database Techniques", IEEE Computer Society, 1998
4. Mahdi Abdelguerfi, Simon Lavington (Eds.), "Emerging Trends in Database and Knowledge-Base Machines", IEEE Computer Society, 1995
5. Daniel Ridge, Donald Becker, Phillip Merkey, Thomas Sterling, "Beowulf: Harnessing the Power of Parallelism in a Pile-of-PCs", Goddard Space Flight Center, CACR Caltech, Proceedings IEEE Aerospace, 1997
6. Greg Kurtzer, "The Warewulf Clustering System" http://www.runlevelzero.net/greg/warewulf/stuff/lbnl-pres/
7. Jack J. Dongara, Piotr Luszczek, Antoine Petitet, "The LINPACK Benchmark: Past, Present, and Future", 2001
8. John Worsley, Joshua Drake, Andrew Brookins (Ed.), Michael Holloway (Ed.), "Practical PostgreSQL", Command Prompt Inc., 2002
9. Refractions Research Inc., "PostGIS Manual" http://postgis.refractions.net/docs/, Refractions Research Inc., 2003
10. OpenGIS Consortium Inc., "OpenGIS Simple Features Specification for SQL", OpenGIS Consortium Inc., 1999
11. Bureau of Transportation Statistics (BTS), "2002 National Transportation Atlas Data Hydrographic Features" http://www.bts.gov/gis/download_sites/ntad02/metadata/hydrolin.htm, BTS, 2002
12. Geospatial Database Queries in a Parallel Environment, August 2003, http://www.cs.uno.edu/Facilities/vcrl/RapportV3.doc
13. Beckmann, N., Kreigel, R., Seeger, B., The R*-tree: An Efficient and Robust Access method for Points and rectangles, Proceedings of the ACM SIGMOD International Conference on management of Data, pp.322-331, 1990
14. Moon, B., Jagadish, H., Faloutsos, C., and Saltz, J.H., Analysis of the Clustering Properties of Hilbert Space-filling Curve, Technical Report UMIACS-TR-96-20, CS Department.
15. Patel, J., DeWitt, D., Clone Join and Shadow Join: Two Parallel Spatial Join Algorithms, ACM GIS Conference, Washington D.C., 2000.
16. DeWitt, D., Gray, J., Parallel Database Systems: The future of Database Processing or a passing Fad? Communications of the ACM, June, 1992.
17. Patel, J., *et al.,* Building A Scalable GeoSpatial Database System: Technology, Implementation, and Evaluation , SIGMOD 1997.
18. R. Wilson, M. Cobb, F. McCreedy, R. Ladner, D. Olivier, T. Lovitt, K. Shaw, F. Petry, M. Abdelguerfi, "Geographical Data Interchange Using XML-Enabled Technology within the GIDB™ System", invited in edited manuscript: A B. Chaudhri (ed), *XML Data Management,* John Wiley & Sons, 2003.
19. Digital Mapping, Charting and Geodesy Analysis Program (DMAP) Team, http://dmap.nrlssc.navy.mil