A Distributed Scientific Data Archive Using the Web, XML and SQL/MED

Mark Papiani Jasmin L. Wason Alistair N. Dunlop Denis A. Nicole

Department of Electronics and Computer Science,
University of Southampton, Southampton, SO17 1BJ, UK.
{mp|jlw98r|dan}@ecs.soton.ac.uk

Abstract

We have developed a web-based architecture and user interface for fast storage, searching and retrieval of large, distributed, files resulting from scientific simulations. We demonstrate that the new DATALINK type defined in the draft SQL Management of External Data Standard can help to overcome problems associated with limited bandwidth when trying to archive large files using the web. We also show that separating the user interface specification from the user interface processing can provide a number of advantages. We provide a tool to generate automatically a default user interface specification, in the form of an XML document, for a given database. This facilitates deployment of our system by users with little web or database development experience. The XML document can be customised to change the appearance of the interface.

1. Introduction

We have been working with the UK Turbulence Consortium [1] to provide an architecture for archiving and manipulating the results of numerical simulations. One of the objectives of the consortium is to improve collaboration between groups working on turbulence by providing a mechanism for dissemination of data to members of the turbulence modelling community. The consortium is now running simulations on larger grid sizes than has previously been possible, using the United Kingdom's new national scientific supercomputing resource[1]. One complete simulation, comprising perhaps one hundred timesteps, requires a total storage capacity of some hundreds of gigabytes. This necessitates new web-based mechanisms for storage, searching and retrieval of multi-gigabyte datasets that are generated for each timestep in a simulation. In particular, an architecture is required that can minimise bandwidth usage whilst performing these tasks.

The Caltech Workshop on Interfaces to Scientific Data Archives [2] identified an urgent need for infrastructures that could manage and federate active libraries of scientific data. The workshop found that whilst databases were much more effective than flat files for storage and management purposes, trade-offs existed as the granularity of the data objects increased in size. If a database is being created to manage metadata describing scientific results, then ideally the database should also be used to store the actual scientific result data in a unified way. However for large output files it becomes costly and inefficient to store the data as binary large objects (BLOBS) within the database.

We describe a solution that uses an implementation of the new SQL:1999 (formerly known as SQL3, see for example [3]) DATALINK type, defined in *SQL Management of External Data (SQL/MED)* [4], to provide database management of scientific metadata and large, distributed result files simultaneously with integrity. We apply this technology to the web, by providing a user-interface to securely manage large files in a distributed scientific archive, despite limited bandwidth.

A database table containing an attribute defined as a DATALINK type can store a URL that points to a file on a remote machine. Once a URL has been entered into the database, software running on the remote machine ensures that the file is treated as if it was actually stored in the database, in terms of security, integrity, recovery and transaction consistency. We use this mechanism to allow large result files to be distributed across the web.

Our system generates a user interface, from a specification defined in Extensible Markup Language (XML) [5], to a database that supports DATALINKs. We have created an XML Document Type Definition (DTD) to define the structure of the XML file.

[1] A 576 processor Cray T3E-1200E situated at the University of Manchester, which forms part of the *Computer Services for Academic Research (CSAR)* service run on behalf of the UK research Councils. http://www.csar.cfs.ac.uk/

Our architecture provides the following features for scientific data archiving:

1. *The system can be accessed by users of the scientific archive, who may have little or no database or web development expertise*. Users are presented with a dynamically generated HTML query form that provides a search interface akin to Query by Example (QBE) [6]. We generate this interface automatically from an XML user interface specification file (XUIS). The XUIS specifies the web interface to a particular object-relational database (which may contain BLOBS and DATALINK types). This file is constructed automatically using metadata extracted from the database catalogue but can be customised to provide specialised features in the interface.
2. *The default interface specification adds a novel data browsing facility to maintain a web-based feel.* Contrary to Manber's statement that finding ways to include browsing in even relational databases would be a great step [7], we show that one simple way to browse relational databases is to follow relationships between tables, implied by referential integrity constraints defined in the database catalogue. We use this principle to provide hypertext links in displayed results that access related information.
3. *Large result files can be archived at (or close to) the point where they are generated.* For the UK Turbulence Consortium, this means that files can be archived at the Manchester site on a local machine that is connected via a high-speed link to the supercomputer. By entering the URLs of these files into a DATALINK column of a remote database (via a web-based interface to the remote database), database security and integrity features can then be applied to the files. An alternative to this, which achieves similar database security and integrity for result files, is to use a web interface to upload a file across the Internet and then store it as a BLOB in a centralised archive at the new location. However, this alternative is not feasible for large files due to limited Internet bandwidth. Even if a file can be transferred to a centralised site, additional processing cost is incurred (which is not present with the DATALINK mechanism) when loading the file as a BLOB type into the database.
4. Because simulation results are stored in unmodified files, *existing post-processing applications, that use standard file I/O techniques, can be applied to the files without having to rewrite the applications*. An alternative would be to modify applications to first access result objects from a database but this would be very undesirable for many scientific users who often apply post-processing codes written in FORTRAN.
5. The Caltech workshop [2] recommended 'cheap supercomputers for archives'. The report suggests that research is necessary to establish whether high-performance computing resources, built from commodity components, are viable for data-intensive applications (as well as compute-intensive applications as has been shown to be the case in for example, the Beowulf project [8]). *We are using our architecture to build a large scientific archive from commodity components, with many distributed machines acting as file servers for a single database.* Security, backup and integrity of the file servers can be managed using SQL/MED. This arrangement can provide high performance in the following areas:
   - Data can be distributed so that it is physically located closest to intensive usage.
   - Data distribution can reduce access bottlenecks at individual sites.
   - Each machine provides a distributed processing capability that allows multiple datasets to be post-processed simultaneously. Suitable user-directed post-processing, such as array slicing and visualisation, can significantly reduce the amount of data that needs to be shipped back to the user. These post-processing codes can be associated with remote data files using the XUIS.
6. The XUIS could be customised to define access to autonomous databases in a loosely coupled federated database system (FDBS)[2].

The rest of this paper is structured as follows. Section 2 begins with a high level description of our architecture. We next describe how we use XML to specify user interfaces. We then describe the default user interface our system provides for a scientific data archive, including searching and browsing capabilities. We explain the functionality that SQL/MED brings to our architecture. Section 3 describes future directions for this research, including post-processing of the distributed result files and extending the system to a federated database architecture. Finally we draw some conclusions from our work.

2. System Architecture and User Interface

This section starts with a high level view of our architecture. We then describe how XML is used to specify the functionality of the user interface. We show how our system supports two kinds of information retrieval, searching and browsing. We illustrate four different types of browsing links that our system can include automatically in web pages displaying query results. 'DATALINK browsing' is particularly important in our architecture for managing large, distributed scientific data files. This section ends, therefore, with an overview of the DATALINK type defined in the draft SQL/MED standard.

[2] We use the definition of 'loosely coupled FDBS' by Sheth and Larson [9] to mean that there is no central administration in terms of creation of, and access to, the data.

2.1 System Architecture

The architecture of our system is shown in Figure 1. It consists of a database server host (located at Southampton University) and a number of file server hosts that may be located anywhere on the Internet. All of these hosts have an installed web server to allow HTTP (or HTTPS – HTTP plus Secure Socket Layer) communications directly from a web browser client.

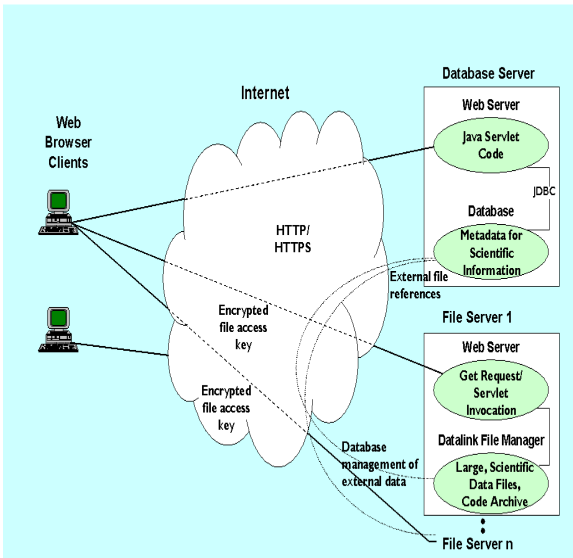

Figure 1: System architecture.

A user of our system initially connects to the web server on the database server host. The URL of our system invokes a Java servlet [10] program. The first time the servlet is invoked it builds the database user interface from the XUIS. Thereafter separate threads within the servlet process handle requests from multiple web browser clients. Each user is first presented with a login screen. Once a user has been verified, interaction with the database is possible via HTML pages that are dynamically generated by our servlet code. These pages consist of HTML forms, Javascript and hypertext links.

The database server stores metadata describing the scientific information such as, simulation titles, descriptions and authors. This data is stored locally in the database and is accessed by our servlet code using Java Database Connectivity (JDBC) [11]. The data is represented by tables with attributes defined as standard SQL-types, BLOB types, or CLOB (Character Large object) types. The latter types are used in our system to store *small* image/video files, executable code binaries or *small* ASCII files containing source code or descriptive material for the turbulence simulations.

For scientific data archiving, an essential feature of our interface is the novel use of remote file severs which store files referenced by attributes defined as DATALINK SQL-types. These file servers manage the *large* files associated with simulations, which have been archived where they were generated. When the result of a database access yields a DATALINK value, our interface presents this to the user as a hypertext link that can be used to download the referenced file from a file server. The URL contains an encrypted key that is prefixed to the required file name. This key is verified by DATALINK file manager code (running on the file server host) which intercepts attempts to access any files controlled by the remote database. Without this key, files cannot be accessed on the file server host, either via the locally installed web sever or directly from the file system by a locally connected user.

We have limited our discussion here to the distributed nature of our architecture, which consists of a central database that can also manage external files on remote hosts. Section 3 describes an extension to the architecture to allow FDBS access.

2.2 XML Specification of the User Interface

Our system is started by initialising the Java servlet code with an XUIS. This initialisation can take several seconds but it is a one-off cost that is not repeated during subsequent requests from users. A default XUIS can be created prior to system initialisation using a tool that we provide. This tool, written in Java, uses JDBC to extract data and schema information from the database being used to archive simulation results. This default XUIS conforms to a DTD that we have created. The default XUIS can be customised prior to system initialisation.

The XUIS contains table names, column names, column types, sample data values for each column, and details of primary keys and foreign keys that participate in referential integrity constraints. The XUIS also allows aliases to be defined for table and column names.

In the next two sections we explain how this information is used to provide an interface with searching and browsing capabilities.

2.3 Searching and Browsing Data

Users of our interface can then begin to locate information in the scientific archive by searching or browsing data or by using a combination of both techniques.

2.3.1 Searching

To search for data a user selects a link to a query form for a particular table. This provides a QBE-like interface. The majority of users of our scientific data archives are frequent repetitive, simple queries. For this category of user, a form-based visual query system (VQS) represents a better alternative than an iconic VQSs or diagrammatic VQSs [12]. Form-based interfaces also facilitate non-expert users by capitalising on the natural tendency of people to organise data into tables [12].

On the query form, the user selects the fields to be returned. Also for each field present, restrictions may be put on the values of the data. Restrictions consist of an operator (=, <, >, >=, <=, <>, *like* and *not like*) and a

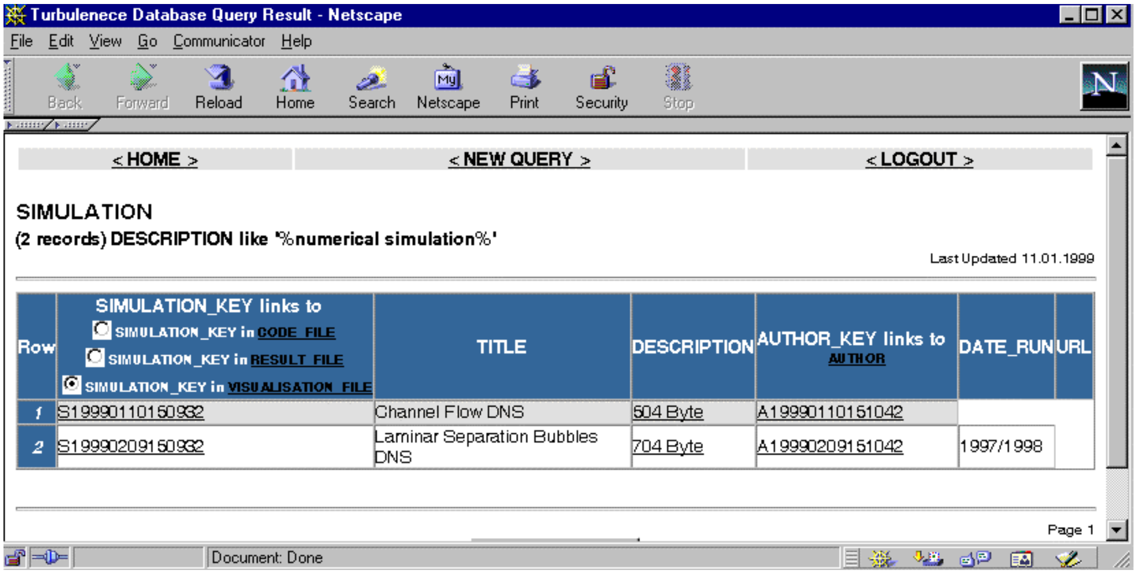

Figure 2: Result table from querying SIMULATION table.

text entry which can include wildcards. The operator is selected from a drop down list. Sample values for the text entry are also available in a drop down list. We provide these features to minimise two of the major problems inherent in text based query languages, such as SQL. These are a semantic one in choosing the correct attributes and relationships between them, and a syntactic one in building the correct textual expression that corresponds to the chosen query. By offering the user choices of column names, table names and operators, the instance of syntactic error can be reduced. Providing column names and sample values can aid substantially in narrowing the semantic meaning of a domain [13]. For example, a column called 'TITLE' in a personnel database could be used to store values such as 'Mr', 'Mrs', 'Dr', etc., or it could store a person's job title such as 'Chief Engineer'. A sample value will quickly clarify the intended meaning. Our interface randomly selects ten sample values to be displayed for each simple attribute type. The drop-down list of samples displays the SQL-type by default as this also provides useful information. The table names, column names, data types and sample values used in the query form are obtained from the XUIS used to initialise the system.

When the user submits a query to the database a result page is returned to the user in the form of a table whose columns are the marked fields and whose rows are the data items that satisfy any restrictions entered.

Users familiar with the structure of a database can get answers to particular queries using the query mechanism alone. However we also provide a convenient data browsing mechanism.

2.3.2 Browsing

After a result table is returned from the initial searching step the user is in browsing mode. Figure 2 shows a sample result table after performing a query on the SIMULATION table in the schema of Figure 3. In this mode the user may examine the table, print or save the data, scroll backward and forward through the pages returned from the database, or browse related information. Figure 2 shows that the SIMULATION result table contains data corresponding to simple SQL-types, such as the title and date for a simulation. It also contains a number of hypertext links from the SIMULATION_KEY, DESCRIPTION and AUTHOR_KEY columns. Our interface automatically inserts several different types of link that we describe below.

The SIMULATION result table (figure 2) displays a link on *each* AUTHOR_KEY attribute. This is because AUTHOR_KEY is a foreign key in the SIMULATION table that references the AUTHOR table (see Figure 3). Selecting a link on an AUTHOR_KEY value will retrieve full details of the author by displaying the appropriate row from the AUTHOR table. Since a primary key occurs exactly once in its table, following a *foreign key link* returns a single row.

The inverse relationship provides a *primary key link* to a table in which the primary key value appears as a foreign key. Since a foreign key may occur any number of times in a table, following a *primary key link* may change the number of rows in the table returned to the user. Also because primary keys may appear in several tables as a foreign key, there may be a choice of tables to browse to. Referring to Figure 3, SIMULATION_KEY is the primary key of the SIMULATION table. SIMULATION_KEY links to three tables where it appears as a foreign key; the RESULT_FILE table, CODE_FILE table and VISUALIATION_FILE table. Each individual SIMULATION_KEY value of Figure 2 therefore contains *a primary key link*. Selecting one of these values will return all the rows that the key appears in from one of the referenced tables. The particular table is indicated in the currently checked radio button in the column header.

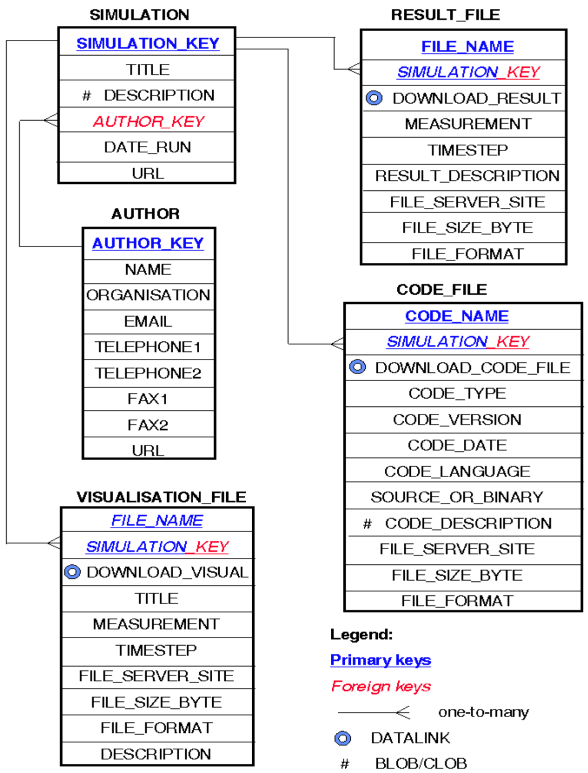

Figure 3: Sample database schema for UK Turbulence Consortium.

The table of data returned by our software interface may be quite large. Cells associated with simple values, such as numbers or short character strings, display the actual values. As well as simple types, we also use BLOB and CLOB types, to store small files that can be uploaded over the Internet. Cells associated with these types display a *LOB link* to the object. Clicking on such a link causes the data associated with the cell to be rematerialised and returned to the client. The link displays the size of the object in bytes, which may help users decide whether they want to retrieve the object. For example, Figure 3, indicates that the DESCRIPTION attribute in the SIMULATION table is a CLOB type. Selecting this link will retrieve the description and display it directly in the browser window since it contains character data. For binary data (e.g. a stored image or executable) selecting the link will allow the user to retrieve the data as a file.

The sample schema of Figure 3 also contains three attributes of the DATALINK SQL-type. These DATALINK attributes serve the main purpose of our architecture – to archive large scientific data via the web. The attributes in the Turbulence Consortium schema of Figure 3 are used for storing result files, code files, and visualisation files. These values are displayed as a filename in a result table, with a hypertext link that contains an encrypted key, required to access the file from the remote file server. This key is explained in more detail in section 2.4.

All of the link types that are described above are generated automatically in our system from information contained in the XUIS. This indicates which columns are of the BLOB, CLOB and DATALINK type and also contains details of primary key to foreign key relationships. A default XUIS will automatically contain these relationships if referential integrity constraints have been defined in a database. However, during customisation of the XUIS it is possible to add relationships between tables, for browsing purposes, even if the database does not specify these constraints.

2.4 SQL/MED: The New DATALINK Type

Both ANSI and ISO have accepted the proposal for SQL Part 9: Management of External Data [4], which includes the specification of the DATALINK type. ISO progressed SQL/MED from *working draft* to *committee draft* in December 1998. If things go according to schedule it should become a standard in late 2000.

DATALINKs provides the following features for database management of external files:

1. *Referential Integrity* – An external file referenced by the database cannot be renamed or deleted.
2. *Transaction Consistency* – Changes affecting both the database and external files are executed within a transaction. This ensures consistency between a file and its metadata.
3. *Security* – File access controls can be based on the database privileges.
4. *Coordinated Backup and Recovery* – The database management system can take responsibility for backup and recovery of external files in synchronisation with the internal data.

Our interface uses IBM's implementation of the DATALINK type [14, 15] that is available for DB2 version 5.2. We used the version of this software available for IBM's AIX operating system. This uses a DB2 database that stores the data associated with standard types internally, plus DATALINK specific software running on the remote file servers to manage external data. This software processes SQL *INSERT*, *UPDATE* and *DELETE* statements that affect DATALINK columns, to link and unlink files for the database. It manages information about linked files, and previous versions of linked files for recovery purposes. It also intercepts file system commands to ensure that registered files are not renamed, deleted and optionally, check the user's access authority.

An example of the SQL syntax for creating a table with a column containing a DATALINK SQL-type is given below (refer to the RESULT_FILE table in the schema of Figure 3):

```
CREATE TABLE RESULT_FILE (
download_result DATALINK
    LINKTYPE URL
    FILE LINK CONTROL
    READ PERMISSION DB
    …
    RECOVERY yes,
file_name VARCHAR(150) NOT NULL,
file_size_byte INTEGER, …
```

The *LINKTYPE URL* parameter indicates that values stored in the DATALINK column are specified using URL syntax. This has the advantage of identifying the remote file server host, directory structure and filename. The *FILE LINK CONTROL* parameter specifies that a check should be made to ensure the existence of the file during a database insert or update. *Read permission* can be set to *DB* (database) of *FS* (filesystem). If the database manages read permission then files can only be accessed using an encrypted file access token, obtained from the database by users with the correct database privileges. Without a valid token, the DATALINK software denies the read request. The *RECOVERY yes* parameter indicates that the DATALINK software coordinates backup and recovery between the referenced files and the database. Further details of these parameters, and additional parameters that can be specified in the DATALINK column definition, are available in [4].

A DATALINK value can be entered via a standard SQL *INSERT* or *UPDATE* statement. The value takes the form:

http://host/filesystem/directory/filename

The *filesystem* part of this URL is a specially mounted DATALINK File System (DLFS) in our AIX version of the software. The *directory* and *filename* are standard UNIX file systems. If read permission is managed by the database, an SQL *SELECT* statement retrieves the value in the form:

http://host/filesystem/directory/access_token;filename

The file can then be accessed from the filesystem in the normal way using the name:

access_token;filename

or, by using the full URL if the file is placed on a web server (as in our system). The access tokens have a finite life determined by a database configuration parameter. This can be set to expire after an interval set in seconds.

3. Future Work

3.1 Processing of Data Files Prior to Retrieval

An advanced customisation feature we are implementing is the ability to associate operations with SQL data types, or database attributes, through modification of a new 'operation' element in the XUIS. An operation can, for example, allow a post-processing code (stored internal or external to the database, as a BLOB or DATALINK value) to be associated with a database column that stores arrays of simulation data (as a DATALINK). The post-processing code then provides additional interface functionality, such as array slicing and visualisation, for the data contained in this attribute.

3.2 Federated Database Access

Our system can be modified to provide a 'multidatabase language system' (in the taxonomy by Bright et al. [16]) in which heterogeneous databases can be accessed at the user interface level through a query language and tools that can integrate data from different sources without a global schema. We are implementing this functionality by extending the DTD for the XUIS to contain multiple database user interface definitions. The relationships, which currently allow browsing of related data in associated tables, will be extended to allow browsing to related data in tables in disjoint databases. Queries can be shipped to remote database using JDBC drivers that operate over networks. As well as FDB browsing a new element will allow several tables to be grouped into a 'supertable'. A supertable will present a single view of the grouped tables for query purposes. Columns that participate in the supertable will be limited to columns assigned the same alias name in the XUIS. This FDB functionality, although limited, will serve a useful purpose of allowing not only distribution of result files, but also distribution and autonomous administration of the database containing the simulation metadata stored internally in the database.

4. Conclusions

We have constructed a prototype system[3] to meet a requirement of the UK Turbulence Consortium to make available to authorised users, large result files from numerical simulations, with a total storage requirement in the hundreds of gigabyte range.

Our interface is specifically aimed at users with a scientific background who are not familiar with SQL. As such, we aim to help users locate scientific data files of interest, using an intuitive searching and browsing mechanism in keeping with a web-based look and feel. One of the implications of this is that we automate the interface construction so that it requires little database or web development experience to install and access. This is a generic, schema-driven system that can be used to manage many different types of large, distributed data archives. We achieve the automated construction by allowing the user interface specification to be defined in an XML file used to initialise the system and by

---

[3] A demonstration system is available at URL http://www.hpcc.ecs.soton.ac.uk/~turbulence/database. Login with username and password both equal to guest. The guest account has the limitation that data files cannot be downloaded.

providing a tool that can generate a default XML specification.

Separating the user interface specification from the user interface processing can provide a number of further advantages:

- The user interface, although schema driven, can be customised to use aliases for table and column names and to present different sample values.
- Hypertext links to related data can be specified in the XML even if there are no referential integrity constraints defined for the database.
- Different Users (or classes of user) can have different XML files thereby providing them with different user interfaces to the same data.

Our architecture uses distributed commodity computing, that combines the new SQL DATALINK type, defined in SQL/MED, with a web-based interface to the data. This architecture provides web access to large, distributed files but maintains database security, integrity and recovery. As far as we are aware, this is the first use of this technology for web-based scientific data archiving.

Bandwidth is a limiting factor in our environment. We greatly reduce bandwidth requirements by avoiding costly network transfers associated with uploading data files to a centralised site. Data distribution also reduces retrieval bottlenecks at individual sites. Post-processing of data files can further reduce the bandwidth requirements of file retrieval.

Our on-going work is exploring ways to extend the XML user interface specification to allow operations to be specified for database columns, so that a user can extend the interface by including post-processing codes. These codes can then be applied dynamically to the data on the file servers to reduce the volume of data being returned. Finally, extensions are being implemented to define access to autonomous databases in a loosely coupled multidatabase environment.

5. Acknowledgements

The UK Turbulence Consortium (EPSRC Grant GR/M08424) provided data for this project. IBM's DB2 Scholars programme provided us with DB2 licenses. The DB2 Data Protection and Recovery Team at IBM were extremely helpful when responding to technical questions regarding DB2's DATALINK implementation.

References

[1] Sandham, N.D. and Howard, R.J.A. Direct Simulation of Turbulence Using Massively Parallel Computers. *In:* A. Ecer *et al.*, *eds. Parallel Computational Fluid Dynamics '97,* Elsevier, 1997.

[2] Williams, R., Bunn, J., Reagan, M., and Pool, C., T. *Workshop on Interfaces to Scientific Data Achives*, California, USA, 25-27 March, 1998, Technical Report CACR-160, CALTECH, 42pp. http://www.cacr.caltech.edu/isda

[3] Eisenberg, A. and Melton, J., SQL:1999, formerly known as SQL3, *SIGMOD Record*, 28(1), March, 1999.

[4] Mattos, N., Melton, J. and Richey, J. Database Language SQL-Part 9:Management of External Data (SQL/MED), ISO/IEC Committee Draft, CD 9075-9 (ISO/IEC JTC 1/SC 32 N00197), December, 1988. ftp://jerry.ece.umassd.edu/isowg3/dbl/YGJdocs/ygj023.pdf

[5] Extensible Markup Language (XML) 1.0, W3C Recommendation, 10 February, 1998. http://www.w3.org/TR/REC-xml

[6] Zloof, M., M. Query By Example. *American Federation of Information Processing (AFIPS) Conf. Proc.*, Vol. 44, National Computer Conference, 1975, 431-8.

[7] Manber, U. Future Directions and Research Problems in the World Wide Web. *Proc ACM SIGMOD Conf.*, Montreal, Canada, June 3-5, 1996, 213-15.

[8] Warren, M., S., *et al.* Avalon: An Alpha/Linux Cluster Achieves 10 Gflops for $150k. Gordon Bell Price/Performance Prize, Supercomputing 1998. http://cnls.lanl.gov/avalon/

[9] Sheth, A., and Larson, J. Federated Database Systems for Managing Distributed, Heterogeneous, and Autonomous Databases. *ACM Computing Surveys*, Vol. 22(3), 1990, 183-236.

[10] Davidson, J., D., and Ahmed, S. Java Servlet API Specification, Version 2.1a, November, 1988. http://java.sun.com/products/servlet/index.html

[11] White, S., Hapner, M. JDBC 2.0 API, Sun Microsystems Inc., Version 1.0, May, 1998. http://java.sun.com/products/jdbc/

[12] Catarci, T., Costabile, M., F., Levialdi, S., and Batini, C. Visual Query Systems for Databases: A Survey. *Journal of Visual Languages and Computing*, 8, 1997, 215-60.

[13] Haw D., Goble, C., A., and Rector, A., L. GUIDANCE: Making it easy for the user to be an expert. *Proc. of 2nd International workshop on User Interfaces to Databases*, Ambleside, UK, 13-15th July, 1994, 19-44.

[14] Quick Beginnings, IBM DB2 File Manager for AIX, Version 5.2. Part Number 04L6231, IBM Corporation, 1998, 85pp.

[15] Davis, J., R. DATALINKS: Managing External Data with DB2 Universal Database, White paper, IBM Corporation, August, 1997. http://www.software.ibm.com/data/pubs/papers/

[16] Bright, M., W., Hurson, A., R., and Pakzad, S., H. A taxonomy and Current Issues in Multidatabase Systems. *IEEE Computer*, 25(3), 1992, 50-60.