A Scalable Architecture for e-Science Data Management

Salman Toor
Dept. Information Technology
Uppsala University
Uppsala, Sweden
salman.toor@it.uu.se

Manivasakan Sabesan
Dept. Information Technology
Uppsala University
Uppsala, Sweden
manivasakan.sabesan@it.uu.se

Sverker Holmgren
Dept. Information Technology
Uppsala University
Uppsala, Sweden
sverker.holmgren@it.uu.se

Tore Risch
Dept. Information Technology
Uppsala University
Uppsala, Sweden
tore.risch@it.uu.se

***Abstract*—The massive increase in the size of the data provided by e-Science applications requires not only to increase the capabilities of resources, but also to design new strategies for efficient utilization of already available resources. In this paper we present a scalable approach to extend a file-oriented storage system, Chelonia, with geographically distributed databases defined by a generic database schema. The database schema is able to model the data from typical e-Science applications. The system includes web service query service allowing e-Science applications to query the required data.**

I. Introduction

The evolution of distributed computing infrastructures has opened a whole new world for the large scale computationally expensive applications. Initially the focus of such applications was to exploit the computational power. Due to the availability of more resources, the amount of experiments from the e-Science community representing natural sciences has increased dramatically. It was realized that due to the increasing amount of computational experiments there is a vital requirement of reliable and consistent storage systems.

To manage the scalability of data a concept has evolved, known as *data grids* [15] or *storage clouds* [24]. Often such systems are designed to handle file oriented storage capabilities, i.e. the application/user can put, get, delete or replicate entire files within the system. This is mainly because most of the existing applications are designed to deal with flat files and to keep focus on the computational aspect. The *Chelonia storage system* [2] is one of the available solutions to provide a scalable and efficient distributed file store.

As the magnitude of data generated by the applications increased significantly the data management becomes non-trivial. Therefore a number of middlewares have started to investigate different strategies to provide efficient and reliable storage systems. These systems often uses the database only for managing the meta-data of the system. However the requirements of the applications are becoming more and more sophisticated. There is a number of applications from system biology, astronomy and different branches of physics which generate huge amounts of data that require scalable storage solutions to provide functionalities more than just put, get or delete files.

The use of multidimensional matrices is one of the essential components in computational sciences. However, it has been observed that in many disciplines there are applications that generate thousands of matrices representing trajectories. Such applications use lots of computational power and in case of reanalyzing the data from old experiments require all the trajectories used in those experiments. The proposed architecture allows not only to store such data but also to query them according to the requirements. The emerging storage requirements motivate us to extend the capability of the Chelonia storage system to handle matrices.

The input data files can be as small as a few hundred kilobytes or can be in size of gigabytes. Often in a file only a part of the data is required. Since everything is in the file, the whole file first needs to be transferred to the computational resource where the application reads and extracts the desired data from the file. This entire activity is wasting computational time and network bandwidth. This waste of resources will increase dramatically with the increasing number of such applications. Retrieving only the required part from the stored file enables the applications to scale well in terms of efficient computational and network resources. This leads to an approach to store the data and use queries to extract relevant parts of stored data in a structured way. For this we have extended the architecture of Chelonia with a regular *Relational DataBase Management System (RDBMS)*.

The importance of the RDBMSs in terms of data management has already been well proven, but the efficient utilization of databases depends on carefully designed database schema. In the scientific environment, designing a schema is rather difficult because of the complex nature of data. The strength of the extended Chelonia architecture is the underlying *Chelonia database schema* which is generic enough to handle the needs of scientific applications from a number of different disciplines.

The Chelonia database schema frees the user from designing a schema for each new scientific application. It is designed to handle the various different datatypes used in scientific experiments ranging from simple datatypes such as integers, reals and strings to complex datatypes such as multidimensional matrices, and even trajectories. In addition, a user can make queries to select desired data elements from the stored scientific results in a uniform way because the database schema is the same for all kinds of scientific applications.

A user makes declarative queries specified in *Structured Query Language (SQL)* to the database. The declarative SQL queries allow the user to specify the data to be retrieved from the database on a very high level. Such SQL queries free the user from writing the detailed application programs for how to retrieve the required data. In general SQL is a widely accepted and mostly used standard for querying data. The declarative queries provide an efficient way for the RDBMS to optimize the queries to improve the query execution time. The optimization functionalities are already well established with existing RDBMSs. In addition, the RDBMS provides the facility to extend its functionalities by defining *User Defined Functions (UDF)* in some programming language. By defining new UDFs, arbitrary functionalities that are not standard RDBMS functions can be incorporated within the RDBMS. In the extended Chelonia architecture, UDFs are used to enhance the SQL functionality to meet the requirements from e-Science applications so that processing can be done inside the database server without having to return the entire dataset(E.g. matrix) back to the client.

The Chelonia storage system is managing several geographically distributed storage nodes. With the extended architecture of Chelonia, a storage node incorporates a RDBMS. Utilizing the geographically distributed storage nodes, Chelonia scales the data in multiple autonomous databases and provides a seamless view over these databases.

Most of the existing operations of Chelonia are accessed via web service operations. To execute queries the *Web Service MEDiator (WSMED)* [18] system is used that provides a web service to query the scientific data without any further programming. The search is completely specified by SQL queries. The WSMED adheres to the *Everything as a Service (XaaS)* [11] paradigm by providing a general web service to process queries over the other web services, known as the *WSMED web service*.

This summarizes the contributions of our work:

- Chelonia is extended with a RDBMS to enrich the capability of its file oriented storage by storing the data generated from different e-Science applications in a structured way.
- A generic scientific database schema, the Chelonia database schema, is used to model the data from different e-Science applications.
- The stored data can be queried in a uniform and structured way.
- A web service querying service, the WSMED web service, is provided to search the scientific data in Chelonia.

The rest of the paper is organized as follows. Section II analyses related work of state-of-the-art projects. In Section III and IV we present the overview of the Chelonia storage system and WSMED. The proposed extended architecture is discussed in Section V. Preliminary results have been analyzed in Section VI and finally conclusion and some future directions are presented in Section VII.

II. Related Work

Google App Engine [5] is a web based execution environment that allows very many users to store tables. The resource utilization of Google App Engine is very limited. The applications can make queries with limited SQL functionalities. In contrast, Chelonia manages the scientific data and enables queries without any limitations.

A highly available and scalable cloud database service *Microsoft SQL Azure* [12] is built on SQL Server technologies. The users do not need to do any additional software installations to store and query their data.

Relational Cloud [16] is providing database as a service by supporting high availability via transparent replication, automatic workload partitioning, and live data migration on a pool of commodity servers within a single data center. In contrast, Chelonia allows to utilize geographically distributed storage. Automated workload partitioning as done in Relational Cloud in the context of scientific application's data and live data migration can be interesting future directions for Chelonia.

In comparison with the above mentioned systems, Chelonia provides a very generic schema suitable for e-Science applications to free the users from the complicated schema design and supports querying complex datatypes.

NetCDF (Network Common Data Form) [8] supports machine-independent data formats that support the creation, access, and sharing of array-oriented scientific data stored in regular files. Chelonia also provides a framework to represent array-oriented e-Science data. In addition, Chelonia stores the data in a database using a generic scientific database schema and enables queries to the stored data.

The *OGSA-DAI* (Open Grid Services Architecture Data Access and Integration) [10] is a framework that allows workflows to be executed for various applications. Such workflows involve accessing, updating, combining, transforming or delivering data that could be distributed across a number of databases and held in various formats. Chelonia is a grid-aware storage solution and thus grid jobs or workflows can utilize all the functionalities provided by Chelonia. Further Chelonia provides a very generic scientific database schema to relieve the user from the burden of designing a specific schema for each application. Then a user makes queries to the stored data in a uniform way.

III. The Chelonia Storage System

The Chelonia storage system [23] was designed to address medium scale storage requirements. It was developed within the *knowARC* [4] project under the *NorduGrid* [9] collaboration. Chelonia was developed in conjunction with the next generation *ARC* middleware [1] [17]. Initially, Chelonia was developed as a distributed storage for flat files. It has four fundamental web services which have well defined tasks and all together it provides a scalable, flexible and fault tolerant storage system for flat files.

The web services provided by Chelonia are *Bartender, A-Hash, Librarian* and *Shepherd*. Following are the brief functionality of each of the services.

- **Bartender** is the front-end service of the Chelonia storage system. All the incoming requests invoke this service first. The operations provided by Bartender allow to get, put, delete or move the files and collections (directories).
- **A-Hash** is a meta-data store of the Chelonia system. It contains the information about the available objects in a key-value pair format. A-Hash can be deployed either in a centralized or a replicated manner.
- **Librarian** is a state-less service which gets requests from Bartender and helps to take decisions. The responsibility of the Librarian service is to handle logical names. Since the service is state-less itself, it uses the A-Hash service to store all the information regarding the available files, collections and the mount-points.
- **Shepherd** runs at a storage node known as the *Shepherd node*. This service is responsible for checking the available files periodically and update the information in the A-Hash through the Librarian service. The Shepherd service also ensures that the files have correct number of replicas.

The Chelonia Storage system provides several interesting features which makes it flexible and easy to use and deploy. Following are the key features of Chelonia:

- The non-intrusive architecture of Chelonia avoids the possibility of a single point of failure. The system can have multiple Bartenders connecting with a single or multiple Librarians. Similarly, if there is a replicated A-Hash, a single or multiple Librarians can connect to any of the available A-Hashes, while the integrity of the results will be intact.
- A global hierarchical namespace allows to access files and collections without using their URLs.
- The accessibility of Chelonia is through *Chelonia command-line tool*.
- In Chelonia, a third party transfer mechanism is used to transfer files, i.e. once the users have been authorized to access a file, a transfer URL is generated by the Shepherd service which is returned back to the user and later the user can fetch the file directly from the Shepherd node.

All the web services of Chelonia are used in the extended architecture to manage the stored data. [20], [21], [22] explain the functionality and features of the Chelonia in detail.

IV. The Web Service MEDiator (WSMED) System

The eminent application of web services is to search different kinds of data from servers providing information of different kinds. For a given set of parameters, such *data providing web services* return collections of objects without any side effects. The Web Service MEDiator (WSMED) [18] [19] [25] is built to query any data providing web service operations without any further programming. The search is completely specified by declarative SQL queries that retrieves data from the data providing web services. To comply with the Everything as a Service (XaaS) paradigm WSMED provides a general WSMED web service to process SQL queries over other web services.

To query any data providing web service, WSMED can import any WSDL file and automatically generate relational views for the web service operations defined in the WSDL file. These views can be queried and joined with SQL. The execution time of a web service operation call depends on the operation properties. Such a property is known as *web service operation cost* to execute a web service operation. The cost is relying on when and where the operation is invoked. In general the cost for an operation call is not explicitly available and very hard to calculate. To speed up the operation calls, WSMED deploys adaptive parallelization operator. For a given SQL query, WSMED dynamically composes the web services. Further it optimizes the web service operation calls by adaptively parallelizing such calls without knowing the cost of calling each web service operation.

Figure 1 illustrates the service oriented architecture of WSMED with web service operations *INIT, IMPORTWSDL, AUTHENTICATION, VIEWINFO, QUERY, and EXIT_S*.

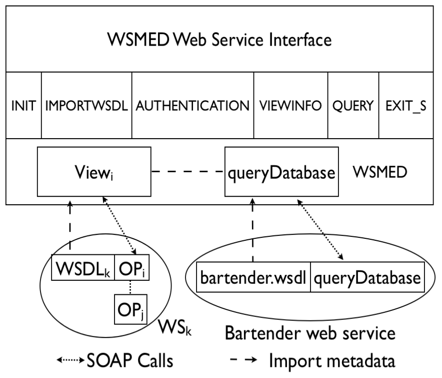

Fig. 1. Service Oriented Architecture

- The INIT operation registers a WSMED user session.
- For a given URL of a WSDL document, the IMPORTWSDL operation imports WSDL meta-data information and automatically creates an SQL view View$_i$ for each operation OP$_j$ provided by a web service WS$_k$ described by an imported WSDL document WSDL$_k$.
- The AUTHENTICATION operation provides authentication information for web service operations.
- The VIEWINFO operation provides information about the SQL view over a given web service operation. For example, it lists view attributes that must always be specified in the queries and attributes to be returned once the given web service operation is called.
- The QUERY operation accepts SQL queries to the generated views. The results from the operation is automatically flattened, optimized, and post processed by WSMED in order to deliver a proper SQL result as a collection of tuples.
- Finally, the operation EXIT_S terminates a user session.

The WSMED web service is demonstrated through a publicly accessible *WSMED Demo* [13] from any browser. It enables the user to access any data providing web services.

The schema of the generated views can be inspected and the query can execute general SQL queries over the views. The demonstration is fully implemented as a JavaScript calling WSMED web service using SOAP. The WSMED Demo fulfills the major characteristics of XaaS. The WSMED web service is used to query the stored scientific data in Chelonia.

V. The Extended Architecture of Chelonia

The proposed architecture provides a loosely coupled model for the integration of Chelonia, WSMED and the RDBMS. Figure 2 illustrates the complete overview of the architecture. Initially Chelonia could only handle flat files, but with the extension Chelonia can be used to generate and query databases managed by a RDBMS. The proposed approach allows both Chelonia and WSMED to work as stand-alone systems. The responsibilities of each of the component is according to its strength, i.e. due to its none-intrusive architecture and storage capability, Chelonia is responsible for populating databases in the RDBMS, whereas WSMED is providing a web service for querying scientific data. The interaction between applications/users and the new architecture is exactly the same as in the original version of Chelonia. All the requests goes through the Bartender, A-Hash is used to store the meta-data and Shepherds are responsible for underlying data integrity and third party transfer.

The rest of the section is divided into three sub sections. Section V-A depicts the generalized database schema. The population of data in the Chelonia is discussed in Section V-B and Section V-C describes how to specify the SQL queries using WSMED.

A. Chelonia Database Schema

To store the data from an application, we have devised a generic database schema. The schema is implemented in MySQL server [7]. The schema contains six different relations TASKS, VARIABLEDIRECTORY, ITRIPLES, FTRIPLES, STRIPLES, and ATRIPLES. Following are the relations where the names appearing in capital letters depict the relation (table) names, and underlined attributes represents the respective primary keys.

```
TASKS( taskId : int, taskName : char,
sourceCode : text, whoDid : char,
executionDate : date)
```

The relation TASKS store data about the executed tasks. It has five attributes taskId, taskName, sourceCode, whoDid, and executionDate with the respective datatypes int, char, text, char, and date. taskId represents the identifier for each executed task, taskName indicates the name of the task, sourceCode contains the programming language code or pseudo code for the task, whoDid depicts the name of the person who has designed and executed the task and executionDate represents when the task is executed.

```
VARIABLEDIRECTORY (vName : char,
vType : char, dimension : char)
```

The VARIABLEDIRECTORY stores the data about the different variables measured during each task execution. The attributes are vName, vType, and dimension with datatypes char, char, and char. vName denotes the variable name and vType notifies the type of a variable (e.g. int or float or string or array_of_int or array_of_float or array_of_string). In general the value for each stored attribute vType array_of_(int/float/string) may have different dimensions. For example values for a variable with vName matrixVal and vType array_of_int could have the dimension three. The dimension value three is stored as the dimension attribute. The default value of the dimension of vTypes int, float and string is one.

The ITRIPLES, FTRIPLES and STRIPLES are used to store the values for each variable, representing the vTypes int, float and string respectively. Therefore the value attribute is defined with the datatypes int, float and longtext for the relations ITRIPLES, FTRIPLES, and STRIPLES. The other attributes taskId and vName has the datatype int and char respectively for all three relations. The datatype longtext supports storing long strings and enables sophisticated text searches on the stored strings.

```
ITRIPLES(taskId : int, vName : char,
value : int)
FTRIPLES(taskId : int, vName : char,
value : float)
STRIPLES(taskId : int, vName : char,
value : longtext)
```

The ATRIPLES relation is used to store array values of variables. It has the attributes taskId, vName and value, with the datatypes int, char and longblob. The longblob datatype enables to store huge arrays as binary objects.

```
ATRIPLES(taskId : int, vName : char,
value : longblob}
```

The Shepherd service receives the data file(s) to be stored. Then it initiates the request to populate the data into the MySQL database. Once the database is created with the above schema and populated, any SQL queries can be made by a user. In general e-Science applications require to store multidimensional matrices and trajectories known as complex datatypes. Such multidimensional matrices are stored as binary objects (longblob datatypes) in the MySQL database. To search the contents of such binary objects, we defined UDFs written in C++. In our approach, when a Shepherd receives a data file with matrices or trajectories, it forwards that file to the MySQL database to store the data.

This approach gives the following benefits:

- The functionality of normalizing and storing binary objects in databases is comparably a better option in terms of execution time for populating and querying data compared to generating a data structure at the application level.
- The usage of UDFs along with several other standard SQL functions enrich the declarative functionality of complex queries for e-Science applications.
- The approach of using UDFs improves the loosely coupled model. That is the Shepherd is still completely

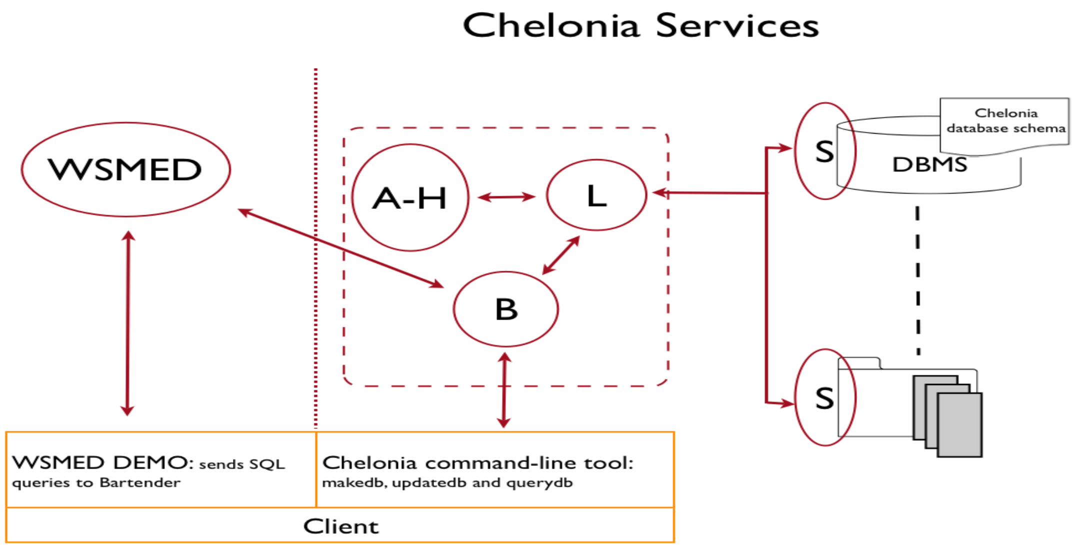
Chelonia Services
WSMED
A-H
L
B
S
S
DBMS
Chelonia database schema
WSMED DEMO: sends SQL queries to Bartender
Chelonia command-line tool: makedb, updatedb and querydb
Client

Fig. 2. Extended architecture for enabling databases using Chelonia services (Bartender (B), Librarian (L) AHash (A-H) and Shepherd (S)) and querying with Web Service MEDdiator (WSMED).

independent of how the data is to be stored and queried in the underlying database.
- In contrast to the conventional downloading of the whole file to applications for computations, UDFs return only the required data from the file.

B. Populating data with Chelonia

In order to populate the data into the underlying database, the user needs to provide the data in a file that comply with an XML schema required by MySQL. The user needs to send his data in an XML file, the *input file*. If the input data consists of only simple datatypes it is sufficient to represent all the data to be stored. In case of complex datatypes, an additional file, the *complex data file*, also has to be supplied. *LOAD_XML* [6] is used to populate the data from the XML file to the MySQL database.

The complex data file should follow a proposed structure to be readable by Chelonia. The format is flexible enough to handle multidimensional matrices as well as trajectories. For the complex datatypes, we are using a semicolon separated format in which the first section denotes the datatypes, the second section contains the information about the dimension and the third section contains the actual data.

Listing 1. Generalized format for representing multidimensional matrix

```
dimType_1 , dimType_2 , ... , dimType_i , valType ;
dimVal_1 ; dimVal_2 ; ...; dimVal_i ;
   {  {
         {val_{1,...,1} , ... , val_{1,...,dimVal_i} }
         {val_{2,...,1} , ... , val_{2,...,dimVal_i} }
         ... dimVal_{(i-1)}
      }_{dimVal_2}
   }_{dimVal_1} ;
```

Listing 2. Example of three dimensional matrix

```
int , int , int , float ; 3; 3; 2;
  {
    {{1.1 ,1.2}{2.1 ,2.2}{3.1 ,3.2}}
    {{4.1 ,4.2}{5.1 ,5.2}{6.1 ,6.2}}
    {{7.1 ,7.2}{8.1 ,8.2}{9.1 ,9.2}}
  };
```

Listing 3. Generalized representation of n dimensional trajectory

```
dimType_1 , dimType_2 , .... , dimType_n , valType ;
   xCord_1 , xCord_2 , ... , xCord_i ;
   yCord_1 , yCord_2 , ... , yCord_i ;
   ...
   ...
    nCord_1 , nCord_2 , ... , nCord_i ;
   {
      {val_1}{val_2}...{val_i}
   };
```

Listings 1 and 3 illustrate the generalized format for representing the complex datatypes: multidimensional matrices and trajectories. Listing 2 provides an example of a matrix with the dimension $[3 * 3 * 2]$. The *dimType* is the datatype for each dimension. In case of matrices it is an integer but in case of trajectories it can be either integer or real. *valType* is the datatype of the actual data to be stored. The number of elements in each dimension is represented in *dimVal* for matrices. For trajectories the comma separated list of elements represent the number of elements. The last section contains the actual data organized within a set of curly brackets where each curly bracket set represents one dimension.

All the data management functionalities of the proposed architecture is provided by the four basic commands using the Chelonia command-line tool. **makedb** and **unmakedb**: used to create and remove the database objects in the underlying RDBMS. **updatedb**: used to populate and update the data. **querydb**: is used to send the SQL query to the database.

The operations of populating and querying the database are all asynchronous i.e when a user sends a request to create a database, Bartender checks the paths validity and passes on the request to the appropriate Shepherd node. In the Chelonia, an object (file or database) can be in five different stages:

- *CREATING*: When an object is in the process of uploading.
- *ALIVE*: Object is available without any problems.
- *STALL*: Object is in the store, but having some problem.
- *THIRDWHEEL*: When an object needed to be removed, the Shepherd service marks that object with this state and later it will be removed from the Shepherd node.
- *UPDATING*: This state is only valid for the databases.

While processing the request for the first time, the state of the database is "CREATING" which basically prevents any user requests to access the database. Users can only populate data or send queries once the state of the database becomes "ALIVE". For each query, the Shepherd service generates a transfer URL which points towards the file containing the results. Each **updatedb** request changes the state of the database to *UPDATING*, which stops the users to send queries until the database will be available again.

Since Chelonia is a grid-aware storage system, all these commands can be issued form the grid jobs as well. One of the major gains of the proposed system is to allow query execution that enables to access only the desired data rather then transferring complete files. Chelonia can run in fully secure and insecure manner. Currently, the communication of WSMED with Chelonia is through insecure HTTP channel. If there is a requirement for a secured system, one can only use the Chelonia command-line tool. The work related to the secure communication between Chelonia and WSMED is an ongoing work.

C. Querying with WSMED

The web service operation named *queryDatabase* provided by the Bartender web service enables to search any stored data in Chelonia by specifying an SQL query. Because the Bartender web service is invoked from the WSMED Demo, the wsdl document *bartender.wsdl* has to be imported by providing its URL [3]. Then WSMED automatically creates an SQL view queryDatabase as described in Listing 4.

After this initialization, one can make any SQL queries using the SQL view queryDatabase to dynamically call the web service operation also called queryDatabase. The schema of the view queryDatabase has view attributes REQUESTID, DBNAME, QUERY, PROTOCOL, SUCCESS, TURL, FINISHED_REQUESTID, and POSSIBLE_PROTOCOL:

Listing 4. SQL view queryDatabase

```
queryDatabase(REQUESTID : char,
DBNAME : char, QUERY : char,
PROTOCOL : char, SUCCESS : char,
TURL : char, FINISHED_REQUESTID : char,
POSSIBLE_PROTOCOL : char)
```

The values should be specified for the following view attributes to call the web service operation queryDatabase .

- REQUESTID: An identifier given by a user. E.g. 0R
- DBNAME: the name of the database the user would like to query. E.g. /db1
- PROTOCOL: list of comma separated protocols the user would like to use to fetch the result file that contains the result of the given user SQL query. E.g. http.
- QUERY: the SQL query the user would like to make to the database specified by DBNAME. E.g:

Listing 5. SQL query1

```
select read_array("100:6000,100:8000",value)
       from ATRIPLES
       where taskId=2 and vName="matrixVal"
```

Listing 5 illustrates an example *query*1 that returns a sub section of a matrix specified by the attribute value in the table *ATRIPLES* for a task specified by the attribute *taskId* = 2 and a measurement specified by the attribute *vName=matrixVal*. *read_array* as a UDF. It requires two arguments:

- dimension: a string that specifies the dimension of the required subsection of a matrix. The dimension string should start from lower and upper limits of the first dimension up to the required dimension i of the given matrix value (i $\leq$ dimension of the matrix). That is the dimension string is specified in a format *lower limit of dimension1:upper limit of dimension1, ... , lower limit of dimension*$_i$*:upper limit of dimension*$_i$. *query*1 requests a subsection of a matrix with dimensions (100:6000,100:8000) from the stored two dimension matrix [10000*10000] specified by the attribute value.
- matrix: denotes the stored matrix. In *query*1 the matrix is specified by the attribute value.

The attributes SUCCESS, TURL, FINISHED_REQUESTID and POSSIBLE_PROTOCOL are filled with results returned by the web service operation call queryDatabase.

- SUCCESS: specifies the status about the call queryDatabase to execute *query*1.
- TURL: depicts the URL of the result file for the given *query*1. Such a result file is to be fetched later by the user by one of the protocols suggested by the returned comma separated list of protocols in POSSIBLE_PROTOCOL.
- FINISHED_REQUESTID denotes the identifier of the finished request.

In Listing 6 *query*2 represent the SQL query to call the web service operation queryDatabase using WSMED web service.

Listing 6. SQL query2

```
select TURL,POSSIBLE\_PROTOCOL
from queryDatabase
where REQUESTID='0' and DBNAME='/db1'
    and PROTOCOL='http' and
    QUERY= 'select
    read_array ("100:6000,100:8000", value)
```

```
        from  ATRIPLES
        where  taskid=2  and  vName="matrixVal"  '
```

Then *query*2 returns :

http://sal6.uppmax.uu.se:60005/hopi/304876e0-49fdf3f-fd354, http

Once the result file for the *query*1 is ready, user can use any http client tools such as *wget* or *curl* by using TURL to fetch the file.

VI. Experiments and Results

For the experiments, all the Chelonia services have been deployed on the UPPMAX (Uppsala Multidisciplinary Center for Advanced Computational Science) resources whereas WSMED is running from the UDBL (Uppsala DataBase Laboratory) resources.

The Chelonia services are running on three different machines each having 2 GB of RAM and dual core Intel Pentium-4 3.0 GHz processor with the operating system Scientific Linux-5.3. One of the machines is dedicated for the Shepherd node and the other two are running the Bartender, Librarian and A-Hash services. WSMED is running on a 3 GHz single processor Intel Pentium-4, 2.5 GB RAM and with Windows 7 operating system.

While the system is still under active development, we are able to present promising initial results that showcase the key features of the extended architecture of Chelonia. To get a clear understanding, we have divided our results into three sections. The first Section VI-A discusses the requirements from an existing e-Science application, *QTL Analysis*[14] which can potentially use the extended architecture of Chelonia. In the last two subsections, we have discussed the execution time while populating and querying large datasets. These datasets depict the same datatypes as required by the QTL application.

A. QTL Analysis

In the field of population genetics and Quantitative Trait Locus (QTL) analysis, one is interested in finding sets of QTL influencing a specific property, or trait, in a population. A QTL is then a genetic position where there exists genetic variation, in the form of different alleles, that control the trait. In order to find QTL, data on trait values (phenotypes), genetic markers (genotypes) and pedigrees need to be available. In recent years, treating expression of individual proteins determined by microarrays as phenotypes has become commonplace. Finding such expression QTL, or eQTL, means sifting through tens of thousands of expression profiles for all individuals, finding correlations against those genetic variation. In this setting, it is beneficial to only extract the expression data for a single protein in each search, i.e. extracting a single column from an individual-phenotype matrix. When interactions between multiple genetic positions are studied, the search can be distributed to several nodes. In that case, only a subset of the genetic marker data is needed on each node, meaning that only specific contiguous blocks of the individual-genotype matrix is needed. By using the extended architecture of Chelonia, engineering complexity is reduced while the bandwidth and local storage needs are kept at a minimum.

B. Data Population

In the first experiment we have populated data with a number of different variables with both simple and complex datatypes. We have populated matrices with varying number of elements ranging from 1 million to 100 million. The experiments have been conducted both for the integer and real numbers. Figure 3 shows the execution time taken by the Shepherd to populate the underlying MySQL database. Note that this execution time excludes the time to transfer the files from the client side to the Shepherd node. The time taken by real number matrices is higher than for integers as the parsing of integer numbers take less time compare to real numbers. Notice that it took only 13 minutes to populate a matrix with 100 million elements. Since we are storing the data files as binary objects inside the database, the system can only handle the data according to the maximum size ($4GB$) of the longblob datatype supported in the MySQL server, which is more than sufficient for our applications to run.

The system is capable of handling character and strings the same way as integers and reals. This feature can be particularly interesting for applications like genome analysis.

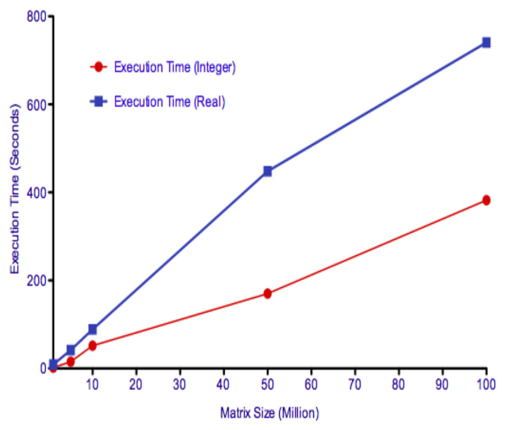

Fig. 3. Execution time while populating matrices of size 1, 5 10 and 100 million both with integer and real numbers

C. Query Response

We have also measured the query execution time. The syntax of the queries is illustrated in listing 5. The first query accesses $10,000$ elements with the index range $101 : 200$ to $501 : 600$. The second query accesses $100,000$ elements with the index range $601 : 700$ to $1 : 1000$. Both queries have been executed on the stored matrices with integer and real datatype elements. Figure 4 shows the query execution time. The execution time increases with the size of the matrix as we have to load the matrix before every query and then store the results to a output file which user retrieves from Shepherd later. With the maximum matrix size of 100 million elements both for integer and real numbers the system manages to retrieve queried section in less the 15 seconds.

It is also important to note that the size of the data files with 100 million for integer and real numbers are 510 MB and 897 MB and the result file for 10,000 and 100,000 are 28 KB and 380 KB, which is significantly smaller than the original data files. Consider a scenario where a user would like to submit 1,000 grid jobs for a certain scientific experiment. For each job, the complete input data file needs to be uploaded. But with the proposed system, only the required section of the data is called, which is significantly smaller and faster to transfer over the network. That is the proposed architecture substantially improves the execution time of the jobs in terms of computational and network resources.

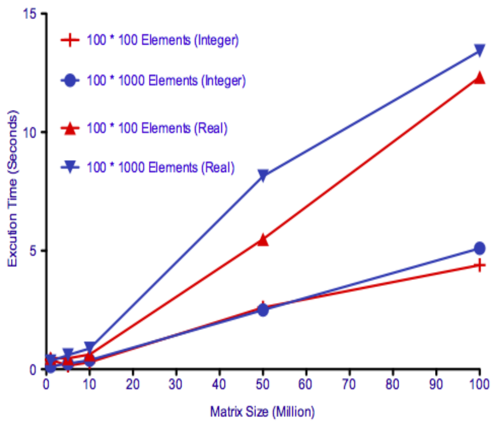

Fig. 4. Query execution time to fetch 10,000 and 100,000 elements from the available matrices.

VII. Conclusion and Future Directions

The extended Chelonia architecture stores the data in geographically distributed autonomous databases and supports SQL queries to retrieve sections of the required data. The Chelonia database schema is generic enough to address the needs of applications from multiple disciplines and frees the users from designing specific schema for different applications. The users are able to query the underlying databases from a browser without installing any additional software. This project has recently been launched. The architecture of the system and the results show that the proposed architecture has the strength to provide a better model for data intensive applications than a file based approach.

There are multiple directions for future work. One of the prominent characteristics supported by the overall framework is the scalable distribution of the data. It can be further enriched by horizontally fragmenting the database for different *taskIds* and distribute such fragments among different Shepherd nodes.

WSMED enables scalable SQL queries calling web service operations via its adaptive parallelization operators. To utilize such scalable querying functionality the Chelonia needs to be extended to handle parallel web service operation calls.

Currently the binary object size is limited by the size provided by the underlying RDBMS. One possible enhancement is to divide the objects into medium sized chunks (binary blocks). This technique will help to enhance the efficiency and can also be helpful to optimize the query response time and will support unlimited matrix size.

Acknowledgements

We would like to thank Alex Read, Jon. K. Nilsen, and Zsombor Nagy for their contribution in the development of Chelonia, Carl Nettelblad for his useful contribution regarding the QTL application and Mattias Ellert for the comments.

References

[1] Advanced Resources Connector. http://www.nordugrid.org/arc/.
[2] Chelonia Web page. http://www.nordugrid.org/chelonia/.
[3] Chelonia web service wsdl. http://udbl2.it.uu.se/WSMED/bartender.wsdl.
[4] EU KnowARC project. http://www.knowarc.eu/.
[5] Google App Engine. http://code.google.com/appengine/docs/whatisgoogleappengine.html.
[6] LOAD XML. http://dev.mysql.com/doc/refman/5.5/en/load-xml.html.
[7] My SQL 5.5 Reference Manual. http://dev.mysql.com/doc/refman/5.5/en/.
[8] NetCDF. http://www.unidata.ucar.edu/software/netcdf/.
[9] NorduGrid Collaboration. http://www.nordugrid.org/.
[10] OGSA-DAI frame work. http://www.ogsadai.org.uk/.
[11] The Next Wave: Everything as a Service. http://www.hp.com/hpinfo/execteam/articles/robison/08eaas.html.
[12] WINDOWS AZURE. http://www.microsoft.com/windowsazure/sqlazure/database/.
[13] WSMED Demo. udbl2.it.uu.se/WSMED/wsmed.html.
[14] L. Bao, L. Wei, J.L. Peirce, R. Homayouni, H. Li, M. Zhou, H. Chen, L. Lu, R.W. Williams, L.M. Pfeffer, D. Goldowitz, and Y. Cui. Combining gene expression qtl mapping and phenotypic spectrum analysis to uncover gene regulatory relationships. *Mamm Genome*, 17(6):575–83, 2006.
[15] Ann Chervenak, Ian Foster, Carl Kesselman, Charles Salisbury, and Steven Tuecke. The data grid: Towards an architecture for the distributed management and analysis of large scientific datasets. *Journal of Network and Computer Applications*, 23(3):187 – 200, 2000.
[16] Carlo Curino, Evan Jones, Raluca Ada Popa, Nirmesh Malviya, Eugene Wu, Samuel Madden, Hari Balakrishnan, and Nickolai Zeldovich. Relational Cloud: A Database Service for the Cloud. In *5th Biennial Conference on Innovative Data Systems Research*, Asilomar, CA, January 2011.
[17] M. Ellert et al. Advanced Resource Connector middleware for lightweight computational Grids. *Future Gener. Comput. Syst.*, 23(1):219–240, 2007.
[18] Sabesan Manivasakan. *Querying Data Providing Web Services*. PhD thesis, Uppsala UniversityUppsala University, Division of Computing Science, Computing Science, 2010.
[19] Sabesan Manivasakan and Tore Risch. *Adaptive Parallelization of Queries Calling Dependent Data Providing Web Services*. Lecture Notes in Business Information Processing. Springer, 2011.
[20] Zs. Nagy, J. K. Nilsen, and S. Toor. *Chelonia - Self-healing distributed storage system*. NorduGrid. NORDUGRID-TECH-17.
[21] Zs. Nagy, J. K. Nilsen, and S. Toor. *Chelonia Administrator's Manual*. NorduGrid. NORDUGRID-MANUAL-10.
[22] Zs. Nagy, J. K. Nilsen, and S. Toor. *Chelonia User's Manual*. NorduGrid. NORDUGRID-MANUAL-14.
[23] J. K. Nilsen, S. Toor, Zs. Nagy, and B. Mohn. Chelonia – A Self-healing Storage Cloud. In M. Bubak, M. Turala, and K. Wiatr, editors, *CGW'09 Proceedings*, Krakow, 2 2010. ACC CYFRONET AGH. ISBN 978-83-61433-01-9.
[24] Mayur R. Palankar, Adriana Iamnitchi, Matei Ripeanu, and Simson Garfinkel. Amazon S3 for science grids: a viable solution? In *DADC '08: Proceedings of the 2008 international workshop on Data-aware distributed computing*, pages 55–64, New York, NY, USA, 2008. ACM.
[25] Manivasakan Sabesan, Tore Risch, and Feng Luan. Automated web service query service. *International Journal of Web and Grid Services*, 6(4):400–423, 2010.