Journal of Computer Science 7 (6): 832-835, 2011
ISSN 1549-3636
© 2011 Science Publications

Extending Binary Large Object Support to Open Grid Services Architecture-Data Access and Integration Middleware Client Toolkit

Kiran Kumar Patnaik and Bollam Nagarjun
ABV-Indian Institute of Information Technology and Management,
Autonomous Institute of Government of India,
Gwalior, Madhya Pradesh, 474010, India

Abstract: Problem statement: OGSA-DAI middleware allows data resources to be federated and accessed via web services on the web or within grids or clouds. It provides a client API for writing programs that access the exposed databases. Migrating existing applications to the new technology and using a new API to access the data of DBMS with BLOB is difficult and discouraging. A JDBC Driver is a much convenient alternative to existing mechanism and provides an extension to OGSA-DAI middleware and allows applications to use databases exposed in a grid through the OGSA-DAI 3.0. However, the driver does not support Binary Large Objects (BLOB). **Approach:** The driver is enhanced to support BLOB using the OGSA-DAI Client API. It transforms the JDBC calls into an OGSA-DAI workflow request and sends it to the server using Web Services (WS). The client API of OGSA-DAI uses activities that are connected to form a workflow and executed using a pipeline. This workflow mechanism is embedded into the driver. The WS container dispatches the request to the OGSA-DAI middleware for processing and the result is then transformed back to an instance of ResultSet implementation using the OGSA-DAI Client API, before it is returned to the user. **Results:** Test on handling of BLOBs (images, flash files and videos) ranging from size 1 KB to size 2 GB were carried out on Oracle, MySQL and PostgreSQL databases using our enhanced JDBC driver and it performed well. **Conclusion:** The enhanced JDBC driver now can offer users, with no experience in Grid computing specifically on OGSA-DAI, the possibility to give their applications the ability to access databases exposed on the grid with minimal effort.

Key words: Grid computing, Binary Large Objects (BLOB), Web Services (WS), Application Programming Interface (API), client toolkit, exposed databases, Uniform Resource Locator (URL), binary data

INTRODUCTION

Grid is a system that is concerned with the integration, virtualization and management of services and resources in a distributed, heterogeneous environment that supports collection of users and resources i.e., virtual organizations across traditional administrative and organizational domains i.e., real organizations. A grid middleware must ensure to provide ways for interoperability to find specific resources, allocate jobs and transfer files among resources and users.

Grid Computing required the development of middleware to provide the functionalities necessary for the systems. Today, we have dozens of opportunities to implement and deploy a grid infrastructure, for example: Globus Toolkit (Richard *et al*., 2008), Unicore (Richard *et al*., 2008) gLite (Latip *et al*., 2011) and so on.

Web Services architecture that defines and uses open protocols for exchanging messages in a language and platform independent manner makes it appropriate for dealing with the issues in dynamic environment of the grid. OGSA-DAI middleware allows data resources, such as relational or XML, to be accessed, integrated and federated via web services to the users over Grid environment. The main purpose of OGSA-DAI is to provide the required data resource sharing not only to support the access to disparate resources but also to transform, integrate and deliver the data (Antonioletti *et al*., 2005). OGSA-DAI middleware can be deployed in the Globus container, or the Tomcat Container. Along with the middleware, the OGSA-DAI team offers a client Application Programming Interface (API) for

Corresponding Author: Kiran Kumar Patnaik, ABV-Indian Institute of Information Technology and Management, Autonomous Institute of Government of India, Gwalior, Madhya Pradesh, 474010, India
Tel: 919406580064

writing programs that access the exposed databases. A Binary Large OBject is a collection of binary data stored as a single entity in a database management system. BLOBs are typically images, audio or other multimedia objects, though sometimes binary executable code is stored as a BLOB.

Using a new API to access the data of a DBMS with BLOB can discourage the domain experts, programmers using Grid as a platform. Moreover it makes the users feel difficult migrating existing applications to the new technology. The intention of picking JDBC is for its acceptance in industry and provides a common interface to access databases managed by any DBMS using the Java language.

MATERIALS AND METHODS

OGSA-DAI is a framework that executes workflows. Workflows are submitted by clients to OGSA-DAI web services. The client toolkit of OGSA-DAI provides components which contact OGSA-DAI web services submit the workflows to OGSA-DAI and parse the request status and data after a workflow has been executed. Prior to our effort in this direction any client attempting to access data had to perform the following steps:

- Get a server proxy
- Create activities
- Configure and connect activities
- Create the workflow
- Execute the workflow
- Run the client
- Get status information and data from the request status

The above approach was tedious for the users and moreover difficult to migrate to a new technology to write the Client programs.

JDBC API has been revised several times since its creation and its latest specification is the 4.0 version. The intention of JDBC is to provide a common interface to access databases managed by any DBMS using the Java language.

A JDBC Driver was designed (Brito and Sato, 2008) to bring together JDBC and the OGSA-DAI middleware, offering java developers, with no experience in Grid computing specifically on OGSA-DAI, the possibility to give their applications the ability to access databases exposed on the grid with minimal effort. A standard API is available based on a middleware that intend to be in conformity with open standards.

JDBC works fairly easy compared to OGSA-DAI client API. As shown in Fig. 1, in order to get a Connection instance, a class offered by sun called DriverManager, contacts the driver implementation of the Driver interface. In this implementation DeviceManager must be told if the driver can handle the request based on the connection Uniform Resource Locator (URL). If yes, the driver returns a connection with the database. The Statement, PreparedStatement and CallableStatement interfaces are responsible for, given a connection, querying the database and returning the results in the form of an integer in the case of update statements, or a ResultSet in case of a select statement.

The driver is implemented using the OGSA-DAI Client API; it transforms the JDBC calls into an OGSA-DAI workflow and sends it to the server using Web Services. When the WS container receives the request, it dispatches the request to the OGSA-DAI middleware. It processes the request accessing the database and returning the result of its execution. When delivered, the result is then processed again using the OGSA-DAI Client API and is returned to the user as an instance of ResultSet implementation.

A BLOB is a collection of binary data stored as a single entity in a database management system. BLOBs are typically images, audio or other multimedia objects, though sometimes binary executable code is stored as a BLOB.

For inserting a BLOB, the following steps are to be performed with OGSA-DAI client API:

- Get a server proxy
- Create WebRowSet
- Create WebRowSetCharacterDataToTuple, SQLParameterisedUpdate activities
- Configure and connect activities
- Create the workflow
- Execute the workflow
- Run the client
- Get status information and data from the request status

The WebRowSet interface and (Wohrer *et al*., 2010) format was introduced in J2SE Version 1.4. A WebRowSet contains three parts: properties, metadata and data. The WebRowSet used in this context to insert a BLOB contains the data viz., column value and column type such that tuples are formed using the activity WebRowSetCharacterDataToTuple. These tuples serve as an input to SQLParameterisedUpdate where this activity takes SQL query and the tuples as input and then the workflow is executed.

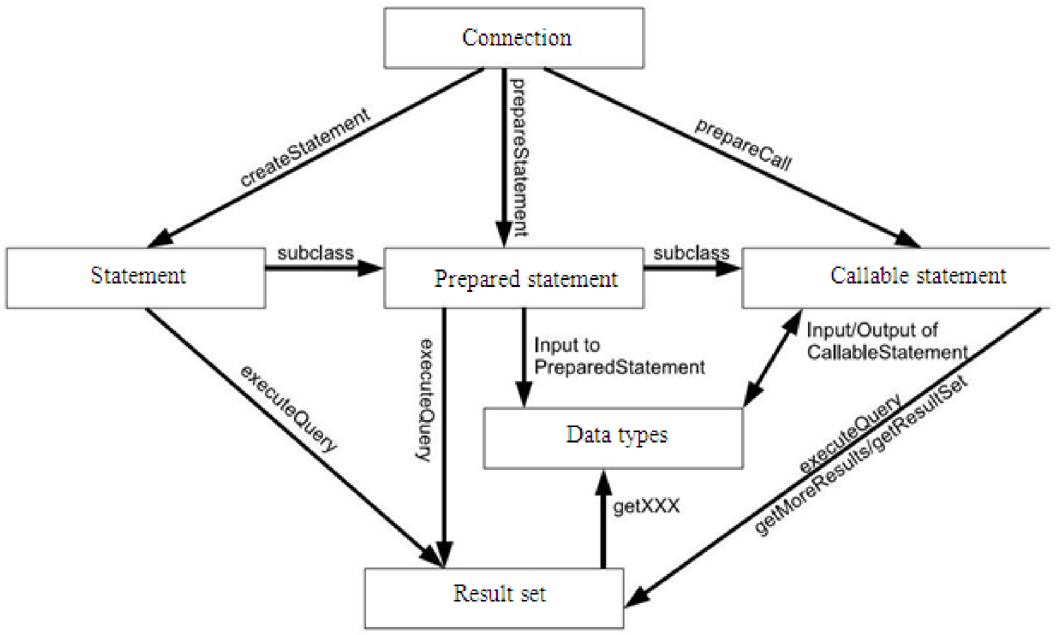

Fig. 1: Main interfaces of the JDBC API

The code below is an example of how to insert a BLOB using the JDBC Driver for OGSA-DAI:

```
/****Code above omitted…***/

        Class.forName("br.usp.pcs.lahpc.ogsadai.Driv
er");
         Connection con=
         DriverManager.getConnection("jdbc:ogsadai://
         localhost:8080/dai/services?resources=
         MyOracleDataResource","system","junnu");
         File imgfile = new File("arjun.jpg");
         FileInputStream fin = new
         FileInputStream(imgfile);
         PreparedStatement pre =
         con.prepareStatement("insert into
         extblackbook(id, picture) values(?,?)");
         pre.setInt(1,100);
         pre.setBinaryStream(2,fin,(int)imgfile.length(
         ));
         pre.executeUpdate( );

/****code below omitted…***/
```

Insertion of Large Objects cannot be done through Statement object; it has to be done through a PreparedStatement object. The above piece of code shows how a BLOB is inserted into the Database a PreparedStatement.

This code is very well known for Java programmers, where the differences are in specifying the driver class (line 2) and the resource URL (line 3). To do the same using the OGSA-DAI API we do a more difficult process, however the client will have more command over the process of retrieving information from the database.

In the previous version of OGSA-DAI 3.0 where BLOBs were not supported by the driver, CSVToTuple activity was used in the place of WebRowSetCharacterDataToTuple activity. However, CSVToTuple activity doesn't support tuples containing a BLOB column (OGSA-DAI 4.1 Axis - User guide). The CSVToTuple activity has been replaced by WebRowSetCharacterDataToTuple. The latter activity has a very sound support to all types of columns. The binary format support for tuples in the recent release of OGSA-DAI 4.1 also backs the use of WebRowSet instead of CSV format.

The BLOB which is to be inserted into the database is converted into a Binary Stream through setBinaryStream( ) method in PreparedStatement interface. The BinaryStream is encoded and converted into a byte array. A WebRowSet is constructed using the Byte array and other columns that are needed by the SQL query and this WebRowSet serves as an input for WebRowSetCharacterDataToTuple. Tuples are constructed from this activity and these tuples serve as an input for SQLParameterisedUpdate activity. This activity performs task analogous to PreparedStatement in JDBC. The SQLParameterized Update activity inserts the required BLOB into the database.

For the purpose of retrieval of a BLOB, the middleware's ResultSet contains getBlob( ) method which returns the BLOB object in accordance with the SQL select statement.

RESULTS

The enhanced driver code was tested using versions ranging from OGSA-DAI 3.1 to OGSA-DAI 4.1 installed under Windows and Linux operating systems on a single machine. The hardware was a Core 2 Duo machine with 1GB of RAM memory. The tests were carried out on Oracle, MySQL and PostgreSQL databases. BLOBs like images, flash files and videos ranging from size 1 KB to size 2 GB are tested and all of them inserted well into each database.

DISCUSSION

By the time the basic driver was written, OGSA-DAI 3.0 was the latest version available and it didn't support for BLOBs. From the version 3.1 to the latest 4.1, BLOBs and CLOBs are held in-memory with consequent implications on memory usage. As the BLOBs are held in-memory it is not feasible to store BLOBs above 2GB of size.

CONCLUSION

Providing access to resources exposed in a grid using existing technology and techniques can promote the OGSA-DAI and make it popular among programmers that don't have specific knowledge in grid computing. For an application developed in this direction, the contribution of supporting Binary Large Object feature to the basic version would definitely be a step forward in making OGSA-DAI exposed to a greater number of people.

ACKNOWLEDGEMENT

We would like to express our sincere thanks to Mathias Brito, the author of the Driver who was very helpful in carrying out the project. We would also like to thank the OGSA-DAI team for its valuable help through the discussion list specially Ally Hume and Mike Jackson without whom the project wouldn't have been this successful.

REFERENCES

Antonioletti, M., M. Atkinson, R. Baxter, A. Borley and N.P.C. Hong *et al*., 2005. The design and implementation of Grid database services in OGSA-DAI. Concurrency Comput. Pract. Exp., 17: 357-376. DOI: 10.1002/cpe.939

Brito, M. and L.M. Sato, 2008. Extending OGSA-DAI possibilities with a JDBC driver. Proceedings of the 11th IEEE International Conference on Computational Science and Engineering, July 16-18, IEEE Xplore, Sao Paulo, pp: 155-162, DOI: 10.1109/CSE.2008.55

Latip, R., H. Ibrahim and F.A. Al-Hanandeh, 2011. Scientific data sharing using clustered-based data sharing in grid environment. Am. J. Econ. Bus. Admin., 3: 146-149. DOI: 10.3844/ajebasp.2011.146.149

Richard, R.J.A., A.A. Joshi and C. Eswaran, 2008. Implementation of computational grid services in enterprise grid environments. Am. J. Applied Sci., 5: 1442-1447. DOI: 10.3844/ajassp.2008.1442.1447

Wohrer, A., T. Lustig and P. Brezany, 2010. Performance evaluation of webrowset implementations. Data Manage. Grid Peer-to-Peer Syst., 6265: 88-99. DOI: 10.1007/978-3-642-15108-8_8