IMPLEMENTING COMPUTATIONAL RESOURCES IN DISTRIBUTED DATABASES

Ǧorgi Kakaševski
Faculty of Informatics
FON University
Skopje, Macedonia

Anastas Mišev
Faculty of Computer Science and Engineering
Ss. Cyril and Methodius University
Skopje, Macedonia

ABSTRACT

Distributed databases serve for storing and organizing heterogeneous data in distributed manner. Basic operations on data are included in these systems, like executing SQL queries or distributed query processing. But basic operations are not enough and users have needs for more complex operations, mainly based on data mining algorithms, like association rule mining. Other operations (algorithms) from third party vendors is difficult to integrate, these algorithms can be CPU intensive and can make difficult the normal functioning of data servers. OGSA-DAI is an example of open source extensible service-oriented distributed database system. On the other side, many algorithms are not built in Java, they aren't open source projects and can't be easily incorporated in OGSA-DAI workflows or their CPU needs are over the capacity of data server. For that reason, we propose architecture (OGSA-DAI extension) with new computational resources and activities that allow execution of external jobs, and returning data into OGSA-DAI workflow.

I. INTRODUCTION

E-Science is computationally intensive science that is carried out in highly distributed network environments, or science that uses immense data sets [1]. Digital data play major role in e-Science applications, from small projects [2] to The Large Hadron Collider [3]. The Grid provides both computational and storage resources to the scientific applications [4]. Via dynamic and distributed Virtual Organizations, Grid computing technologies provides approach to use geographically distributed heterogeneous large-scale data resources. The key characteristics distinguishing Grid computing over the conventional technologies are: it is flexible, autonomous, dynamic and service-oriented. In this are included systematic and effective access and interaction of distributed data resources.

Because of mainly file-based organization of Grid data, there is a need how to use data which are originally organized as shared and structured collections, stored in relational databases, or in some cases in structured documents or in assemblies of binary files. These requirements of Grid applications require developers of Grid systems to provide practical implementations of "relational Grid databases" (or data access and integration [5]). One of the widely used middleware for heterogeneous and distributed database is OGSA-DAI [6] build on WS-DAI standards for data access and integration.

While other Grid middleware's use data in file based manner, OGSA-DAI workflows consists of activities that use relational databases. One of the open issues in OGSA-DAI is how to execute external jobs. By external jobs we assume programs that are already compiled and can be executed without user interaction. These programs can be useful for lightweight data processing, like data pre-processing in data mining algorithms, creating data structures for fast data search or different algorithms which have need from data, stored in heterogeneous distributed relational databases. External jobs take input, usually via input file, and return output in output file. Input and output files usually are text files in CSV, XML or JSON format. Some applications defined their type of text files. Good to mention example of such external job is widely used Weka [7, 8] application for data mining, which have algorithms for preprocessing, clustering, classification etc. Weka takes input in textual ARFF file format and produce output in text file (also there is support for XML and other file types).

The goal of this paper is to present architecture for implementing computational resources which are capable for execution of external jobs in OGSA-DAI workflows and to present the current status of our implementation. With this upgrade, in OGSA-DAI can be added more complex algorithms for distributed data analysis, which are not supported in databases by default.

The rest of the paper is organized as follow. In Section 2 we describe standards for data access and integration which are key concepts for practical implementations of relational database middleware for Grid. We described OGSA-DAI, a practical release of WS-DAI specifications in Section 3. In Section 4 we present extension of OGSA-DAI that we propose, and in Section 5, the current status of practical implementation. We conclude paper and give recommendations for future work in Section 6.

II. STANDARDS FOR DATA ACCESS AND INTEGRATION

One of the main characteristics of the Grid is it heterogeneity, which means that Grid is made of different type of resources with different characteristics, and as whole must act as one system. This property is true for databases. Another key moment in organizing databases on Grid is the service-oriented architecture of Grid and Grid services, defined by Open Grid Service Architecture (OGSA) [9]. OGSA has a one of the central roles in Grid computing, because it simplifies the development of secure, robust systems and enable the creation of interoperable, portable, and reusable components. The main idea of transient Grid services is to extend well-defined Web service standards (WSDL, XML schemas, SOAP, etc.), to support basic Grid behaviors. These standards are proposed by Open Grid Forum and this organization have key role in development of new Grid technologies.

OGF issued the Web Service Data Access and Integration (WS-DAI) family of specifications [10]. This specifications defines web service interfaces to data resources, such as relational or XML databases. WS-DAI specifications have five main components: interfaces, data services, properties, messages and data resources. Interfaces are used to interact with data services. Data services are web services that implements interfaces to access distributed data resources. Properties describe data services, messages are sent to data services and data resources are system that can act as a source of data. Data resources are already existent database management systems, for example, MySQL, and this system has no real connection with WS-DAI specifications.

In order to implement these specifications on real system, on Fig. 1 is given proposed architecture of Grid database system. During the fact that these traditional DBMSs exist for many years and are well developed and tested, in our case we can use some of the existing. For example, DBMS can be MySQL, Oracle or XML. On top of this system Grid Data Services must be implemented, as part of the middleware that is responsible for communications among heterogeneous distributed database.

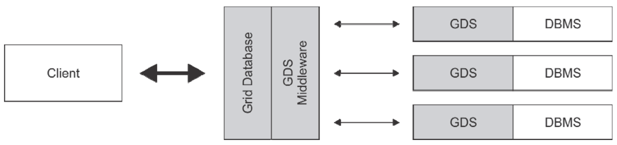


Figure 1. Architecture of Grid Database System

Grid Data Service Middleware is one kind of transparent wrapper for complex Grid database. Transparency refers to: location, name of database, distribution, replication, ownership and cost, heterogeneity and schema. Clients exchange data with Grid database, even if they don't know where that data is stored or in which format. Important for client is that in Grid database must be implemented all (or common) functionalities of regular database management systems and in natural way. That ensures that clients (users, software developers) can easily use Grid database and easily adapt their applications with new kind of database.

As a conclusion, a heterogeneous distributed database system (or simply Grid database) should provide service-based access to existing database management systems that are located on Grid nodes. This database will be some form of virtual database and clients will access to data on transparent way no mater of physical organization of data [11].

III. Practical Implementation of WS-DAI

To be accepted one OGF standard it must have at least two independent implementations. In WS-DAI and WS-DAIR Implementations - Experimental Document [12] is given reports on the interoperability testing results arising from two implementations of WS-DAIR (which also implement WS-DAI). By interoperability is mean interoperability of the clients accessing the DAIS defined services, i.e. not inter-service interoperability. To be more specific, the actual SOAP messages sent by a client, in this case a third party application, must produce the same behavior and results for each of the DAIS compliant services in order for those services to be defined as interoperable.

The implementations that participated in this exercise were:

- AMGA WS-DAIR implementation [13].
- OGSA-DAI WS-DAIR implementation [6].

AMGA WS-DAIR is developed as a part of gLite middleware. Its implementation uses gSoap & C++, supports the following underlying database management systems: PostgreSQL, MySQL, SQLite and Oracle. Supported languages for querying data are SQL-92 entry level "direct data statement" and the AMGA metadata language, but it doesn't support stored procedures. It supported WebRowSet dataset and this security features: SSL, GSI, VOMS, permission, and ACL.

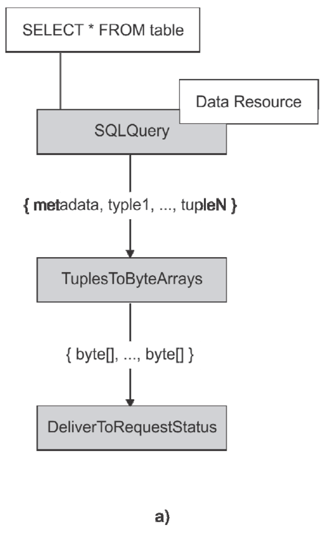
OGSA-DAI stands for Open Grid Services Architecture - Data Access and Integration. It is a Java open source project and it is a one of the most comprehensive heterogeneous database system, which can be used in Grid.

One of the primary goals of data access and integration is to treat these data, whatever their origin, within a uniform framework. OGSA-DAI make the DBMSs, which can be heterogeneous and distributed on Grid, transparent to the users [14]. It hides the underlying drivers that are needed to communicate with the database management and as name tells, OGSA-DAI provide service-oriented interface to the clients. This process naturally will add some overhead on top of the communication itself but it is not that significant [15].

It supports several kinds of databases: relational (Oracle, DB2, SQL server, MySQL, Postgress), XML databases like xindice and eXist and files. There is an opportunity for programmers to develop new drivers for DBMS which are not supported in original version. Acting in the top of local DMBS that are located on different sites, OGSA-DAI provides uniform Grid database.

Today, OGSA-DAI has a several thousand users and a lot of scientific projects that are using it as a Grid database (i.e. ADMIRE1, BIRN2, GEO Grid [16], MESSAGE [17], BEinGRID5, LaQuAT6, Database Grid7, mantisGRID [18] etc.). List of most important projects can be found in [19].

Architecture of OGSA-DAI is in compliance with needs described in Section 2. In OGSA-DAI, data resource represents externally managed database system and data service provide interface to access to this data resources. A data request execution resource (DRER) executes OGSA-DAI workflows (and termed requests) provided by clients. A DRER can execute a number of such requests concurrently and also queue a number more. An OGSA-DAI server must have at least one DRER. Clients execute workflows using OGSA-DAI as follows: a client submits their workflow (or request) to a data request execution service (DRES). This is a web service which provides access to a DRER. A workflow consists of a number of units called activities. An activity is an individual unit of work in the workflow and performs a well-defined data-related task such as running an SQL query, performing a data transformation or delivering data. Activities are connected. The outputs of activities connect to the inputs of other activities. Data flows from activities to other activities and this is in one direction only. Different activities may output data in different formats and may expect their input data in different formats [20]. Main activities are organized in three types: Statement Activity, Translation Activity and Delivery Activity. On Fig. 2 a) is shown example of OGSA-DAI workflow with three activities. The interaction between client and data through DRES, is shown on Fig. 2 b).



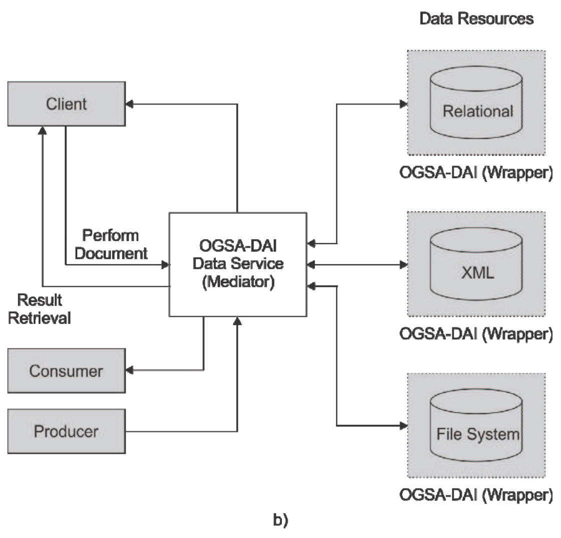


Figure 2. a) Example of OGSA-DAI query-transform-deliver workflow. b) Interaction between client and data.

IV. Model for Computational Resources in OGSA-DAI Workflows

At the beginning we analyze several systems for scheduling of jobs in distributed environments [21, 22]. As we explain in previous section, key components, or extensibility points, of OGSA-DAI are resources and workflow activities. The idea of our proposed model is to define computational resource (CR) and execution activity (EA), as shown on Fig. 3.

A. Defining the Model

Like data resources which are connected to existing database and allow access to data, CR are places (computers) on which can be executed external jobs. CR have necessary configuration for executing jobs in the operating system. It has parameters which are read from configuration file (maxNoJobs, OperatingSystem, etc.) or created dynamically in addiction of current system status (if processor usage is high or number of jobs is equal to number of maximum allowed jobs, CR doesn't execute job and put job in activity queue). CR has list of predefined jobs with parameters and statistics of previous executions. If job is not previously defined, parameters and statistics are not present.

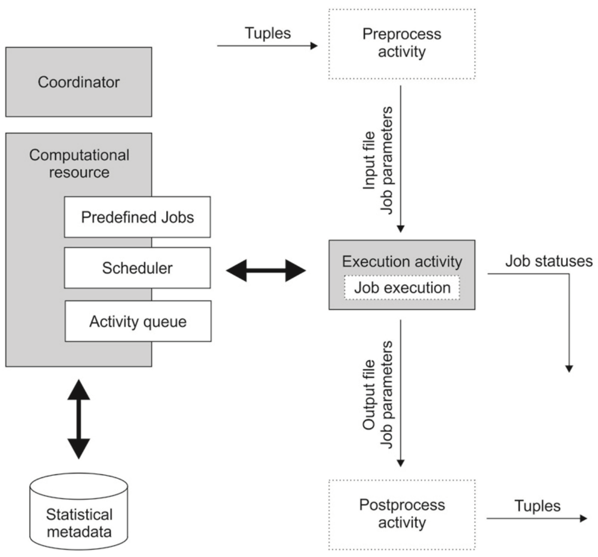


Figure 3. Architecture of proposed model for executing external jobs

CR has scheduler which is responsible for starting the external jobs. This component read statistics, CR static and dynamic properties, and gives system resources to EA. Scheduler have activity queue and keep all activities that should be executed. For each DBMS (MySQL, Oracle, XML), is defined specific wrapper. In this case, each type of CR must have different implementation (for example, CR for executing batch jobs in Windows has different implementation from CR in Linux).

EA is activity which is connected to the CR and is wrapper to job/application that should be executed in workflow. Activity can be part of regular OGSA-DAI workflow but must be specified with one computational resource. OGSA-DAI in general don't have workflow coordinator which can choose appropriate resource for activity and user must specify computational resource. In our model we propose coordinator which can choose on which resource to execute EA. For example, if activity is executed in several seconds (data about execution are read from statistics), it is executed on same computer, but if activity last for several hours it is executed on other CR.

On Fig. 3 EA is surrounded by two activities, one for preprocess and one for postprocess data. These activities are specific for different type of applications. First activity convert and export data in file format, while last activity read data from output file of external job and convert it into stream of data. EA have other pipe that report job statuses and rest of the workflow know the current system status and make decisions from this information - workflow can have maximum waiting time and if job is not finished, workflow can continue without that data (or job can be started again).

B. Scenarios of Practical Usage of the Model

In this section we present two scenarios of executing external jobs, which can commonly occurred in real situations. These scenarios are defined on cost bases of data transfer and computational capabilities of OGSA-DAI servers.

Communication cost can be very expensive when data are transferred between servers. If application is data-intensive; have need of distributed queries; or algorithms are iterative and exchange data between them very often; then a lot of messages must be exchanged. While this is a general rule to be followed in the design of distributed systems in which there is the potential for large amounts of information to be moved over networks, the problem is accentuated in service oriented architectures, like Grid and OGSA-DAI concretely, due to the fact that exchanging messages between web services is more expensive. Currently, the main expense is due to the fact that data held within a service (e.g. in a database) is converted to and from a character-based representation (of XML or JSON) so that it can be packaged in a SOAP message and sent between services. This imposes a CPU cost for converting to and from the specific format and also results in a large increase in size between an efficient representation within a service (e.g. within a database) and the tagged, character-based representation, incurring increased transfer costs as more bytes are shipped [23].

The idea of first scenario is to try to avoid moving data as much as possible through moving the computation to the data, rather than vice versa. This is the main way to minimize the number of messages; the amount of data exchanged and this approach avoid communication costs. Here, computations are done on the database servers and data transfer is avoided as much as possible. This scenario can be apply when algorithms use data from one server; when algorithms don't have need from high computational power - like preprocessing of data; etc.

Our second scenario of using OGSA-DAI with external jobs is to separate computation tasks from database computers. This can be done from several reasons: database servers don't have enough computational power, algorithms work with small amount of data from overall database but impose high computational power - therefore database servers can be bottleneck; data are highly distributed and must be transferred to one location for computation to be done; etc.

For example, data can be distributed on several locations in different organizations, but their computers don't have enough computational resources. Although data transfer can be expensive in such a situation, the computation can be done on other computers, or clusters, adopted for that purpose.

Another possible situation is when computation is near the data (but not on same server), the data transfer is realized with great speed. In this scenario computation can be done on dedicate cluster nodes in the same cluster.

V. Current Status and Practical Implementation

With cooperation with EPCC institute at University of Edinburgh, we start to implement proposed architecture and now is work in progress. OGSA-DAI is open source project build in Java and have good documentation for developers. We extend latest OGSA-DAI 4.2 release with:

- Computational resources
- Activities connected with CR

Currently CR support only static properties for execution of activities. These properties are read from configuration file (in OGSA-DAI manner).

Until now we concentrate on implementation of EA that runs batch jobs or standalone applications on CR computer. These jobs have one input and one output file. EA is extending MatchedIterativeActivity and implements ResourceActivity interface. Matched iterative activities take input into blocks, in our case jobs, that should be executed, and process each block individually. Inputs of activity are consisting of job parameters and input file. Scheduler is in early phase of development and check maximum number of jobs that can run in parallel. If number of active jobs is lower than maximum number, the job is started in separate thread. For defining job we use Callable interface, instead of Runnable, because Callable allow more control on threads - canceling thread if error occur and can return result. Threads are submitted to ExecutorService which automatically take care of maximum number of active threads (jobs). Output of activity is jobs states, which have job description and job current status (submitted, started, executing, and finished). Job state for each job is produced periodically on several seconds. With this output, rest of the workflow (postprocess activity) knows the current state of the jobs.

Our test case is consisting of crawling (downloading) articles from news web sites. External job is made in Python, have one textual input file and one XML output file. With our workflow we want to collect news articles from several web sites. Preprocess activity take tuples as input, export them into text file and create jobs for executing. These jobs are followed to the EA. With several executions of workflow we conclude that the maximum number of jobs parameter depends on CPU usage of external program, and in our case we set it to 20. This is the main reason why we propose list of predefined jobs in CR, and keeping different parameters for each job. Another moment is overall CPU usage, because several workflows can be run in parallel in OGSA-DAI. For that reason we propose dynamic properties of CR, and if CPU usage is above 90%, EA will be put into waiting queue.

VI. Conclusion and Future Work

We can highlight three conclusions. First, Grid applications have an increasing need of database systems. Combining Grid and database technologies is an essential approach to meet the requirements of large-scale Gird applications. Our second conclusion is that today there are many efforts to enable organizing and querying data in distributed Grid environment. OGSA-DAI is a good example. And, finally, we show that there is possibility to implement computational resources in OGSA-DAI as extension, which can allow execution of external jobs.

Our proposed model allows different types of algorithms for data analyzing, built in different languages or like separate applications, to be incorporated in heterogeneous distributed databases. More complex cases cover execution of external jobs on other platforms, like desktop grid, Condor clusters, cloud services etc.

VII. Acknowledgements

This work is done in collaboration with Armey Krause, Ally Hume, Michael Jackson and Mario Antonioletti from EPCC at the University of Edinburgh.

References

[1] J. Taylor, Defining e-Science, http://www.nesc.ac.uk/nesc/define.html.

[2] G. Kakasevski, et al., Grid enabled system for gathering and retrieving images from WEB, ETAI 2007.

[3] The Large Hadron Collider Homepage. http://lhc.web.cern.ch/lhc/.

[4] I. Foster, C. Kesselman. The Grid: Blueprint for a New Computing Infrastructure. Elsevier, 2004.

[5] N.P.C. Hong, et al., Grid Database Service Specification, GGF Informational Document, Feb. 2003.

[6] M. Antonioletti, M. Atkinson, R. Baxter, A. Borley, N. Hong, B. Collins, N. Hardman, A. Hume, A. Knox, M. Jackson, et al., "The design and implementation of Grid database services in OGSA-DAI," Concurrency and Computation: Practice and Experience, vol. 17, no. 2-4, pp. 357–376, 2005.

[7] G. Holmes, A. Donkin, and I.H. Witten. Weka: A machine learning workbench. In Proc Second Australia and New Zealand Conference on Intelligent Information Systems, Brisbane, Australia, 1994.

[8] The University of Waikato; WEKA - Open source datamining software, http://www.cs.waikato.ac.nz/ml/weka/, 2013.

[9] I. Foster, C. Kesselman, J. M. Nick, and S. Tuecke, "The physiology of the grid: An open grid services architecture for distributed systems integration," in Open Grid Service Infrastructure WG, Global Grid Forum, vol. 22, pp. 1–5, Edinburgh, 2002.

[10] Atkinson, M.P., Dialani, V., Guy, L., Narang, I., Paton, N.W., Pearson, D., Storey, T. and Watson, P. Grid Database Access and Integration: Requirements and Functionalities, DAIS-WG, Global Grid Forum Informational Document (GFD.13), March 2003.

[11] A. Lee, J. Magowan, P. Dantressangle and F. Bannwart. Bridging the Integration Gap, Part 1: Federating Grid Data. IBM Developer Works. August 2005.

[12] Steven Lynden, Mario Antonioletti, Mike Jackson, Sunil Ahn; WS-DAI and WS-DAIR Implementations - Experimental Document; DAIS Working Group 2009.

[13] Nuno Santos, Birger Koblitz; Distributed Metadata with the AMGA Metadata Catalog; CoRR abs/cs/0604071 2006.

[14] Jackson, M. et al. OGSA-DAI 3.0–the whats and the whys. In Proc. UK e-Science All Hands Q6 Meeting pp. 158–165, 2007.

[15] Kumar Abhinav. Evaluation of Grid Middleware OGSA-DAI. School of Computing Science, Vellore Institute of Technology, Deemed University. May 2006.

[16] Tanimura Y, Yamamoto N, Tanaka Y, Iwao K, Kojima I, Nakamura R, Tsuchida S and Sekiguchi S. Evaluation of Large-Scale Storage Systems for Satellite Data in GEO GRID. The International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences. Vol. XXXVII. Part B4. pp 1567-1574. Beijing 2008.

[17] MESSAGE (Mobile Environmental Sensing System Across Grid Environments) Project. http://bioinf.ncl.ac.uk/message/. 2010.

[18] Garcia Ruiz, M., Garcia Chaves, A., Ruiz Ibañez, C., Gutierrez Mazo, J.M., Ramirez Giraldo, J.C., Pelaez Echavarria, A., Valencia Diaz, E., Pelaez Restrepo, G., Montoya Munera, E.N., Garcia Loaiza, B. and Gomez Gonzalez, S. mantisGRID: A Grid Platform for DICOM Medical Images Management in Colombia and Latin America. J Digit Imaging. Feb 2010.

[19] OGSA-DAI open source project publications. http://sourceforge.net/apps/trac/ogsa-dai/wiki/Publications. 2013.

[20] OGSA-DAI user documentation. http://sourceforge.net/apps/trac/ogsa-dai/wiki/UserDocumentation. 2013.

[21] Gurmeet Singh, Carl Kesselman, Ewa Deelman: Optimizing Grid-Based Workflow Execution. J. Grid Comput. 3(3-4): 201-219 (2005).

[22] Andre Luckow, Lukasz Lacinski, Shantenu Jha: SAGA BigJob: An Extensible and Interoperable Pilot-Job Abstraction for Distributed Applications and Systems. CCGRID 2010: 135-144.

[23] Paul Watson: Databases in Grid Applications: Locality and Distribution. BNCOD 2005: 1-16.