# Database Oriented Grid Middlewares

Efthymia Tsamoura
Aristotle University of Thessaloniki
Thessaloniki, 54124, Greece
etsamour@csd.auth.gr

Anastasios Gounaris
Aristotle University of Thessaloniki
Thessaloniki, 54124, Greece
gounaria@csd.auth.gr

***Abstract*—Efficient management of massive data sets is a key aspect in typical grid and e-science applications. To this end, the benefits of employing database technologies in such applications has been identified since the early days of grid computing, which aims at enabling coordinated resource sharing, knowledge generation and problem solving in dynamic, multi-institutional virtual organizations through a distributed, scalable, adaptive and autonomous infrastructure. Nowadays, databases are playing an increasingly important role, both when the data is (semi-) structured and when it is stored in flat files. This survey paper discusses in detail existing database-oriented grid middleware. It focuses on several complementary aspects, such as dynamicity, autonomy, resilience to failures, and performance, and presents the characteristics, capabilities and limitations of existing solutions.**

## I. Introduction

Grid technologies promote and enable coordinated resource sharing, knowledge generation and problem solving in dynamic, multi-institutional virtual organizations through a distributed, scalable, adaptive and autonomous infrastructure [1]. One of their most important motivations has been the need to support novel e-science applications that typically involve complicated processing of massive datasets. Such datasets are produced at increasingly growing rates by modern scientific experiments belonging to several domains, such as astronomy, economy, social sciences and high energy physics. Although initially the focus of grid applications was more on performing computationally intensive tasks, the efficient management of high data volumes has become equally, if not more important. In centralized settings, database technologies provide the most appealing solution to the problem of management of large datasets, due to their well known ability to scale, process arbitrarily complex tasks expressed in a declarative manner and perform optimization without requiring human supervision. Non surprisingly, the benefits of employing database technologies in grid and e-science applications has been identified since the early days of grid computing [2].

The aim of this paper is to examine the level of incorporation of database technologies in grid solutions at the middleware level. We first identify and classify database-oriented grid middleware that is publicly available and application independent. Then, we focus on several complementary aspects, such as scalability, performance, dynamicity, resilience to failures, autonomy, transparency, expandability, and easiness to use. Scalability strongly relates to the capability of a system to perform well even when the data processed is very large, or the number of the collaborating resources becomes high, whereas, in this paper, we examine performance only from the perspective of how quickly tasks can be processed. The systems that utilize databases to store data are inherently scalable with respect to data volumes, but they are not necessarily scalable in terms of the number of interconnected resources. The grid is also subject to frequent and unpredictable changes in terms of the availability and the characteristics of the resources they are exposed on to it, and consequently, grid solutions should be as self-configuring as possible thus exhibiting dynamic behavior. In addition, since resources may fail or become unavailable while executing tasks, fault tolerance is important. Furthermore, the notion of pooling autonomous resources plays a key role in grid applications; any solutions should not compromise the resources' autonomy. Finally, in this work we are also interested in aspects relating to how easily new functionality can be added and tasks are expressed.

To the best of our knowledge, this is the first survey that focuses on database-oriented grid middleware. Two other surveys that are close to ours have appeared in [3] and [4], respectively. In the former, which mainly focuses on resource management and the potential benefits from employing agent-based solutions, a review is conducted which presents the fundamental problems of large scale data management in Grid systems and their characteristics, followed by an analysis of generic approaches based on Web services, P2P techniques and agents. In [4], the main requirements of a data grids are pointed out, advocating a solution based on P2P architectures.

The remainder of the paper is structured as follows. In Sec. II the database-oriented middleware systems that provide database access and query processing functionality are discussed. In Sec. III we briefly examine solutions that are inspired at least partially by database technologies but do not rely on databases for data storage. Finally, conclusions upon the current state of Grid data management middlewares are drawn in Sec. IV.

## II. Middlewares for managing structured and semi-structured DBs

In this section, we examine middleware solutions for managing structured and semi-structured data stored in traditional database management systems (DBMSs), such as MySQL, Oracle, Microsoft SQL Server and IBM DB2. These solutions are further classified into two categories. The first one deals with solutions that provide services for simple access to resources, while the second provides more advanced data integration and distributed query processing capabilities. Note that most of these middlewares are under continuous development and improvements, so, the discussion in this paper aims to reflect their main focus and orientation rather than a detailed technical description of the low level characteristics of their most recent releases.

### *A. Middlewares for access to databases*

OGSA-DAI [5] is a Web Service (WS) based middleware, both for accessing data resources and managing workflows. The main idea behind this initiative is to provide a WS wrapper interface to DBMSs, so that the latter can be exposed on to Grids in a uniform way. The basic elements of OGSA-DAI workflows are activities over such WSs. An activity dictates an action to be performed and can be classified into three main categories: (i) "statement activities" interact with a data resource, e.g. a common statement activity involves the submission an SQL query to a DBMS; (ii) "delivery activities" are responsible for delivering data, while (iii) "transform activities" perform data transformations.

A strong point of OGSA-DAI is that a wide range of relational, object-relational, and XML database products is supported, along with flat files. However, data resources cannot be automatically detected or configured. Instead, users must provide the appropriate configuration information, including user credentials, in order to expose them as OGSA-DAI services. However, the autonomy of DBMSs is not compromised; the databases are administered independently, and as such, the database administrator has full control on the account used to access the data through OGSA-DAI services. Users interact with OGSA-DAI through specific XML documents, called perform documents, which include the activities to be executed, along with their parameters, inputs, outputs, and data and control flow between them. The activities are defined in a semi-transparent way: data access activities require as input the location of the resource that wraps the relevant data, but do not require any knowledge about the type of the underlying DBMS and its data model. OGSA-DAI is also expandable. It provides Java libraries in order for users to add new activities or alter the implementation of existing ones.

The OGSA-DAI engine parses the perform documents, identifies the included activities' implementations and tries to optimize the workflow of activities. The OGSA-DAI engine may exploit parallelism in task execution, allowing the pipelined connection between tasks via streamed interconnections between activities. Also, it provides several options for data transfer and data compression, coupled with caching of intermediate results, with a view to facilitating optimized data transmission. Additionally, OGSA-DAI utilizes fault tolerance mechanisms at the workflow level. If a workflow shows potential problems, e.g. excess resource consumption, OGSA-DAI may reconfigure it or restart its execution from the last checkpoint. Of course, tasks already submitted to the underlying DBMS can benefit from the local fault tolerance mechanisms that are in place. OGSA-DAI architecture is inherently scalable, since it allows each DBMS to be exposed as a different resource; however scalability problems may arise in cases where a query aggregates data from different resources. Such problems are partially ameliorated by OGSA-DQP. OGSA-DAI is a free open source software and, at the time of writing, supports both Microsoft Windows and Linux operating systems.

GRelC Data Access Service (DAS) [6] is a Grid middleware based on WSs as well. It provides access to structured and semi-structured databases over the Grid and supports VOs management. GRelC-DAS is installed on a single server within a Grid, the DAS server. All user access and management requests are initially directed to this server, which is responsible for taking the appropriate action (the DAS server either redirects them to an integrated database or calls an internal component). This can cause potential bottlenecks, leading to problems with scalability. Users must specify all the relevant information to expose a data resource in GRelC, e.g. the type of data resource, the connection driver, user credentials, belonging virtual organization (VO), etc.

Interaction with GRelC-DAS is done through both traditional (graphical and command line interface) and more sophisticated tools, the so-called GRelC Portal. The GRelC Portal is a gateway providing access to GrelC-DAS through the web. In order to manage a DAS server utilizing the Portal, users must supply all the information associated with it, namely the server alias, the host location and the communication port. The query language depends on the specific data resource, e.g. SQL, XQuery, XPath, etc. Access is done in a semi-transparent way; users must specify the location of the resources that hold the data of interest but they need not be aware of their characteristics in terms of data model, specific product details, etc.

GRelC-DAS consists of several components. The Data Access component provides several database capabilities, e.g. connection, query and disconnection, utilizing SDAI. SDAI contains a C library which provides several DBs management interfaces. Using this library users can add drivers for new data resources. Moreover, the DAS SDK and the GRelCJProxy introduce another expandability point, since they provide a set of C++ and Java APIs for application developers. With the rest components users can manage

| Middleware / Property | OGSA-DAI | GRelC |
|---|---|---|
| **Performance** | Parallel activity execution. Several data transfer protocol and compression options. Caching of intermediate results. | Commands to transfer data within a data structure, or within a file using the HTTPG transfer protocol, or within a compressed file respectively. Prefetching and streaming options are also supported. |
| **Scalability** | Each DBMS can be exposed as a different resource. Scalability problems may arise when a single service aggregates data from multiple resources. | The installation of DAS on a single server may create bottlenecks leading to scalability problems. |
| **Autonomy** | Local DBMSs are autonomous. | Local DBMSs are autonomous. |
| **Transparency** | Users must supply the location of the data resources, but not any DBMS specific information. | Users must supply the location of the data resources. |
| **Dynamicity** | Static configuration of resources performed manually. | Static configuration of resources performed manually. |
| **Fault Tolerance** | Workflows can be restarted. | GRelC does not deal with fault tolerance issues. |
| **Expandability** | New drivers can be plugged. Java libraries to add or alter new activities. | C library to add drivers for new data resources. C++ and Java SDKs are provided to application developers. |
| **Easiness to use** | Perform documents containing user activities. Query submission language depends on the specific data resource, e.g. SQL for RDBMSs and XPath and XUpdate for XML DBs. GUI and Command-Line Interface interfaces. | Query submission language depends on the specific data resource. GUI, Command-Line Interfaces and a web Portal. |

Table I
SUMMARY PRESENTATION OF OGSA-DAI AND GRELC DAS.

integrated DBMSs, servers and user profiles, e.g. add/delete users from Grid DBMSs, change privileges, monitor and check the availability of hosts and DBMSs, etc. A global metadata catalog exist, which contains information about users, VOs, credentials, Grid DBs (DBMS type, login), etc. GRelC-DAS provides several data transfer services, in order to optimize data transmission. Finally each DBMS is autonomously administrated. The current GRelC-DAS release (as of March 2009) supports only Linux operating systems and can be downloaded from the project's website.

A summary of the characteristics of OGSA-DAI and GRelC is presented in Table I.

### *B. Middlewares for Distributed Query Processing (DQP)*

OGSA-DQP [7] extends OGSA-DAI middleware with distributed query processing (DQP) capabilities over OGSA-DAI wrapped resources. Currently it performs Select-Project-Join queries. To this end, it provides two extra WSs: the coordinator service and the evaluator service. The coordinator is responsible for managing dynamic federations of resources on top of which SQL queries can be submitted. It is also responsible for compiling, optimizing and scheduling such queries for execution. Evaluators are capable of evaluating a query fragment provided by the coordinator. Since OGSA-DQP is built on top of OGSA-DAI, it inherits all its services and functionalities, e.g. OGSA-DAI data transfer and transformation services.

There are two phases involved in OGSA-DQP, a set-up phase where clients define the data resources and analysis services to be utilized during query execution (performed once per session) and the query phase during which queries are submitted and evaluated. To accomplish the set-up phase, users must supply an XML document containing the location of participating data resources and analysis services. The latter are ordinary WS that can be called from within the queries in the form of typical user defined functions. After this step, a DQP data resource is dynamically created and initialized. It represents a federation of data resources, analysis and evaluator services over which queries may be composed. During the initialization of the DQP data resource, the schemas of the databases that will be used, and other metadata are imported by contacting the OGSA-DAI wrapped data resources.

Clients submit SQL queries in perform documents as in OGSA-DAI. Each query is parsed, compiled and scheduled for execution and a query plan is produced. This query plan is partitioned, where each partition is described in an XML document specifying the role of an individual evaluator in the evaluation of the query. The coordinator service sends produced partitions to evaluator nodes. After that, the evaluators retrieve data from OGSA-DAI data resources and invoke any analysis services required to evaluate the query. Moreover evaluators communicate between themselves. OGSA-DQP is an open source software too.

G-ParGRES [8] performs OLAP query execution over Grids, extending a cluster level middleware called ParGRES. Currently both of them support relational databases and queries in SQL. In ParGRES query optimization is done using the AVP algorithm [9]. This algorithm splits input queries into fragments and allocates them to local cluster nodes. The later can exchange messages in order to dynamically perform load balancing. G-ParGRES operates similarly. The Distributed Query Service (DQS) receives user queries, splits them into partitions using the AVP algorithm and passes them on to local clusters. This service also

performs final result composition. G-ParGRES extends ParGRES with monitoring components to redistribute queries if an executor node is heavily loaded. One DQS is created to control the execution of a user query. A disadvantage of G-ParGRES is that it requires full database replication. G-ParGRES is under development, while it may be integrated with OGSA-DAI.

GRelC Data Gather Service (DGS) [10] is a middleware built on top of GRelC DAS performing P2P query execution. Each participating VO has some servers, which act as super peers (being responsible for routing and control actions), and have the DGS installed. In GRelC-DGS, data is organized into classes. Each class is data resource specific and each peer maintains a class specific routing table. Users must define for each class a specific data resource, the authorized users and VOs, a query language, the DGSs and sub-peers which control sub-peers or have data belonging in this class. Routing tables improve performance, since they prune the peers which may participate in query evaluation. When a query is submitted, the DGS consults the class specific routing table and passes the query on to neighboring peers, which have data of the same class.

Open SkyQuery middleware [11] is the Grid flavor of SkyQuery [12], a federation of geographically separate astronomy databases on the Internet. It is implemented to optimize an application-specific distributed spatial query called Cross Match [12]. Queries are submitted in the Astronomical Data Query Language (ADQL) and the SkyQL [12].

Open SkyQuery consists of the following components: the Clients, the Portal, the SkyNodes and the VO registry. SkyNodes host databases and contain APIs for accessing to data resources. They implement a web service interface and can be discovered through the VO registry. Two different types of SkyNodes exist; the basic nodes execute ADQL requests on their own data, while Full SkyNodes participate in distributed Cross Match Queries. The Clients accept user queries and pass them on to the Portal, which serves as a mediator and manages the federation of data resources. When a SkyNode (either Basic or Full) wishes to join the Open SkyQuery middleware it must provide information about the supported services and the database schemas to the Portal. Administrators must also provide the appropriate connection drivers. Cross Match query execution proceeds in two phases. During the first one "Performance Queries" are executed in order to estimate the number of potentially contributed tuples from each Full SkyNode. During the second phase a federated execution plan is constructed. This plan contains pairs of query fragments and the Full SkyNodes where they will be executed. The software to built a SkyNode can be downloaded from [11] and supports Microsoft Windows, Linux and Mac/OSX operating systems.

*Performance*. The performance of the above middlewares relies on the efficient execution of the query. OGSA-DQP exploits the two step optimization algorithm [13] to produce an execution plan, which may employ both inter- and intra- operator parallelism. G-ParGRES employees parallelism both in grid and cluster level. In GRelC-DQS submitted queries are concurrently executed in multiple nodes, while the usage of class routing tables prunes the search space. Open SkyQuery utilizes an efficient algorithm for Cross Match Query execution, although, it does not utilize parallelism, since a Full SkyNode cannot pass the execution plan to another one before it finishes the evaluation of its fragment.

*Transparency and Fault tolerance*. Query submission is done in a semi-transparent way in OGSA-DQP and GRelC-DGS. In the former, users must supply the location of the data resources and analysis services that will be utilized during query execution only during the configuration phase and not during query submission, while in GRelC-DGS, users must define the data classes and the corresponding routing tables. On the other hand, in Open SkyQuery users do not supply such kind of information. The system identifies the SkyNodes that host input data from the Portal's catalog. Fault tolerance is an important issue in DQP. G-ParGRES implements mechanisms to redirect failed subqueries to other nodes. The P2P organization in GRelC-DGS provides fault tolerance since the execution of failed queries continues in the rest peers. However super-peers are single points of failure for their sub-peers [14]. In Open SkyQuery, no specific mechanism is implemented; it utilizes the fault tolerance mechanisms of integrated database systems.

*Scalability*. Two parameters affect the scalability in these middlewares. First, the fact whether a submitted query is partitioned into fragments and whether these fragments are executed in parallel or not. The second parameter is the number of components which control the overall query execution process. OGSA-DQP and G-ParGRES partition input queries and execute them in parallel (supporting both inter- and intra- operator parallelism). Similarly, in GRelC-DGS input queries are concurrently executed at multiple nodes. On the other hand, Open SkyQuery executes the partitioned query plan serially. In OGSA-DQP one coordinator service controls the execution of the queries in a federation, leading in potential bottlenecks. In Open SkyQuery, the Portal undertakes the optimization and overall control for all input queries, while in GRelC-DGS bottlenecks may arise if many queries are submitted to the same DGS.

## III. Other Middleware artefacts

In several middleware solutions for data grids, data is stored in flat files and the usage of databases is limited to metadata management.

In EDG [16], a middleware is developed which provides a set of services and tools for job execution and file management in computational Grids. Databases are used for logging, book-keeping and replica management purposes,

| Property \ Middleware | EDG | POQSEC | GridDB-Lite |
|---|---|---|---|
| **Transparency** | Users must supply the names of data files for job submission and data management. | Users must supply the names of data files. | Users must supply the names of data files. |
| **Dynamicity** | Central repository of configuration specifications, from which individual machines can be autonomously installed and configured. Changes to the central specification automatically trigger corresponding changes in the actual configuration of individual nodes. | Not dealt with. Installation and configuration of host machines is undertaken by NorduGrid [15] | Not dealt with. |
| **Fault Tolerance** | Tasks execution recovery and rescheduling. Generic checkpointing, restart and rollback mechanisms. | Coordinator resubmits failed jobs. | Not dealt with. |
| **Expandability** | APIs for application developers. | NorduGrid APIs. Users can define new functions in a declarative language. | Users can define filters and partitioning functions. |
| **Easiness to use** | A command line interface is provided for job submission and access to relational databases. Jobs are submitted in a jobs description Language. A GUI is currently being developed. | A command line interface is provided. Jobs are submitted as declarative queries. | Jobs are submitted as SQL queries. |

Table II
PROPERTIES OF EDG, POQSEC AND GRIDDB-LITE.

which is typical to many data grids [17]. Access to middleware's relational databases is done through the Spitfire WS [18]. The latter provides a simple interface to retrieve and query data held in any local or remote RDBMS, sharing to an extent the same vision as OGSA-DAI.

POQSEC [19] and GridDB-Lite [20] are two examples that stand to benefit from the easiness-to-use that database declarative languages offer. POQSEC [19] performs DQP over data stored in flat files and stands on the top of NorduGrid (NG) middleware. It processes scientific analyses specified as declarative SQL-like queries over data in files distributed in the Grid. The user poses queries to POQSEC in terms of a database schema available in the client database. The schema contains both an application-oriented part and Grid metadata. The application schema describes data stored inside files in Grid storage resources, for example events produced by ATLAS. GridDB-Lite [20] is a middleware motivated by data-intensive scientific applications on the Grid. It is equipped with a data source service, which provides a view of a dataset in the form of virtual tables to other services. However, as in POQSEC, user queries are not evaluated using traditional database technologies. Compared to initiatives like OGSA-DQP, these approaches benefit from the declarative manner of expressing potentially complex tasks in query processing, but develop their own execution mechanisms, thus not exploiting the full potential of a DQP system. Table II summarizes some properties of these systems with regard to their dynamicity, transparency, fault tolerance, easiness to use and expandability.

So far, we have described middlewares for job submission, data accessing and querying over data grids. Workflow management middlewares are developed to facilitate workflow submission on computational grids. Existing middlewares of this category do not exploit databases for workflow management; instead they may present a workflow's inputs and outputs as database tables. GridDB [21] is a middleware which undertakes tasks submission to computational Grids, through a database interface. Its key feature is that the inputs and outputs of each workflow process are presented to the user as database tables, over which queries can be submitted. GridDB exploits memoization; minimization of resources wasted due to reuse of previously generated data products, while it offers interactive capabilities so that users can drive grid process execution. An analogous system is proposed in [22].

## IV. SUMMARY AND DISCUSSION

In this work we presented existing database-oriented middlewares for data management and distributed query execution over the Grid. We briefly described their architecture and the services they offer. We have also moved a step deeper showing the degree existing middlewares can satisfy some very important properties, such as dynamicity, autonomy, expandability, interoperability, scalability, high performance, easiness to use and transparency. From this presentation, it can be concluded that major incompleteness exist in existing middlewares. Although projects that offer simple access to grid-enabled databases have become mature (e.g., nowadays there are thousands of OGSA-DAI/DQP

users), more advanced DQP systems must become more robust, scalable and dynamic. The fact that the vast majority of work in the area of distributed databases deals with the concurrent handling of small transactions while guaranteeing data consistency, and thus is not tailored to data- and cpu-intensive applications, has raised some concerns as to whether database-oriented artefacts are viable solutions for data grids [17]. However, key characteristics of modern database technology, such as optimization of declarative languages, and scalable performance are definitely necessary for grid and e-science applications: so, we expect next generation data grid middleware, not only to be database-centric, but also to employ advanced optimization and highly scalable mechanisms.

## REFERENCES


[1] I. Foster and C. Kesselman, *The Grid: Blueprint for a New Computing Infrastructure*, 2nd ed. San Francisco, CA, USA: Morgan Kaufmann Publishers, 2003.

[2] P. Watson, "Databases and the grid," in *Grid Computing: Making The Global Infrastructure a Reality*, F. Berman, G. Fox, and T. Hey, Eds. Wiley, 2003.

[3] A. Hameurlain, F. Morvan, and M. E. Samad, "Large scale data management in grid systems: a survey," *3rd Int. Conf. on Information and Communication Technologies: From Theory to Applications (ICTTA)*, pp. 1–6, 2008.

[4] E. Pacitti, P. Valduriez, and M. Mattoso, "Grid data management: Open problems and news issues," *Intl. Journal of Grid Computing*, vol. 5, no. 3, pp. 273–281, 2007.

[5] "OGSA-DAI: Ogsa data access integration middleware," *www.ogsadai.org.uk/*.

[6] "Grelc: Grid relational catalog project," *http://grelc.unile.it/*.

[7] "OGSA-DQP: Ogsa distributed query processing middleware," *www.ogsadai.org.uk/dqp/*.

[8] N. Kotowski, A. B. A. Lima, E. Pacitti, P. Valduriez, and M. Mattoso, "Parallel query processing for olap in grids," *Concurrency and Computation: Practice and Experience*, pp. 2039–2048, 2009.

[9] A. Lima, M. Mattoso, and P. Valduriez, "Adaptive virtual partitioning for olap query processing in a database cluster," *Proc. of the XIX Brazilian Symposium on Database Systems (SBBD)*, pp. 92–105, 2004.

[10] G. Aloisio, M. Cafaro, S. Fiore, M. Mirto, and S. Vadacca, "Grelc data gather service: a step towards P2P production grids," *SAC*, pp. 561–565, 2007.

[11] "Open skyquery: Advanced query processing of astronomical data," *http://openskyquery.net/Sky/skysite/*.

[12] T. Malik, A. S. Szalay, T. Budavari, and A. R. Thakar, "Skyquery: A web service approach to federate databases," *Proc. of the 28th Conf. on Very Large Databases (VLDB)*, 2002.

[13] D. Kossmann, "The state of the art in distributed query processing," *ACM Computing Surveys*, vol. 32, pp. 422–469, 2000.

[14] A. Oram, *Peer-to-Peer: Harnessing the Power of Disruptive Technologies*. O'Reilly, 2007.

[15] P. Eerola, T. Ekelof, M. Ellert, J. R. Hansen, A. Konstantinov, B. Konya, J. L. Nielsen, F. Ould-Saada, O. Smirnova, and A. Waananen, "The nordugrid architecture and tools," *Computing in High Energy Physics and Nuclear Physics (CHEP03)*, March 2003.

[16] "EU data grid project," *https://eu-datagrid.web.cern.ch*.

[17] S. Venugopal, R. Buyya, and K. Ramamohanarao, "A taxonomy of data grids for distributed data sharing, management, and processing," *ACM Comput. Surv.*, vol. 38, no. 1, 2006.

[18] W. H. Bell, D. Bosio, W. Hoschek, P. Kunszt, G. Mccance, and M. Sil, "Project spitfire: Towards grid web service databases," *Global Grid Forum Informational Document (GGF5)*, 2002.

[19] R. Fomkin and T. Risch, "Framework for querying distributed objects managed by a grid infrastructure," *First VLDB Workshop on Data Management in Grids (DMG)*, pp. 58–70, 2005.

[20] S. Narayanan, T. Kurc, U. Catalyurek, and J. Saltz, "Database support for data-driven scientific applications in the grid," *Parallel Processing Letters*, vol. 13, no. 2, pp. 245–271, 2003.

[21] D. T. Liu and M. J. Franklin, "Griddb: A data-centric overlay for scientific grids," *Procceedings of the 30th Conf. on Very Large Databases (VLDB)*, pp. 600–611, 2004.

[22] S. Shankar, A. Kini, D. J. DeWitt, and J. Naughton, "Integrating databases and workflow systems," *SIGMOD*, vol. 34, no. 3, pp. 5–11, September 2005.