Scientific Programming 14 (2006) 173–184
IOS Press

Federated database services for wind tunnel experiment workflows

A. Paventhan[a], Kenji Takeda[a,*], Simon J. Cox[a] and Denis A. Nicole[b]
[a]*Microsoft Institute for High Performance Computing, School of Engineering Sciences, University of Southampton, SO17 1BJ, UK*
E-mail: {povs, ktakeda, sjc}@soton.ac.uk
[b]*School of Electronics and Computer Science, University of Southampton, SO17 1BJ, UK*
E-mail: dan@ecs.soton.ac.uk

Abstract. Enabling the full life cycle of scientific and engineering workflows requires robust middleware and services that support effective data management, near-realtime data movement and custom data processing. Many existing solutions exploit the database as a passive metadata catalog. In this paper, we present an approach that makes use of federation of databases to host data-centric wind tunnel application workflows. The user is able to compose customized application workflows based on database services. We provide a reference implementation that leverages typical business tools and technologies: Microsoft SQL Server for database services and Windows Workflow Foundation for workflow services. The application data and user's code are both hosted in federated databases. With the growing interest in XML Web Services in scientific Grids, and with databases beginning to support native XML types and XML Web services, we can expect the role of databases in scientific computation to grow in importance.

Keywords: Application workflows, workflow activities, scientific data management, database federation, RDBMS

1. Introduction

Scientific and engineering experiments often involve people and facilities that are distributed within and across organizations. The large volumes of data acquired during many of these experiments are often transferred to different network locations for storage and processing. In the last few years, Grid computing has generated much interest among scientists and is increasingly being adopted in many scientific projects. The majority of scientific applications in the Grid rely on file systems for data management, with very limited use of Relational Database Management Systems (RDBMS). Where used, the RDBMS is often exploited as a query engine to retrieve metadata and/or results.

In any data-centric application the important functionality is to provide effective storage of and access to data. The file systems may provide better raw read/write performance than database systems, but there are additional benefits database systems can bring in transaction support to guarantee data integrity, query language capability, secured access to data and other features that include support for procedural language stored procedures and functions, native XML types and web services, transactional messaging, publish/subscribe replication, data mining extensions and so on. The development of such new capabilities is driven by the business market and it has the potential to enable new approaches to scientific data management in the Grid environment. An RDBMS with these rich new capabilities may be viewed as *database operating systems* [15] into which one can plug subsystems and applications.

Database federation can help heterogenous data produced at different geographical locations to be managed, and provide the user with a single logical view. The individual database instances in the federation are

*Corresponding author.

ISSN 1058-9244/06/$17.00

autonomous and any of them temporarily being unavailable does not affect their interactions. Although database federation as an approach to data integration [16] can support functions such as query optimization, the issue we address in this paper is geographical separation of data sources, be it within campus or across organizations.

Different scientific applications in fields such as High Energy Physics [30], Earth Sciences [19] and Geosciences [21] have already utilized database-centric approaches in a Grid environment. In the Grid context, there is also a valuable review of database integration in [35]. The work described in this paper differs in that we provide an end-to-end experiment workflow solution exclusively using the database capabilities. We present an architecture based on federation of database instances managing both data and the processing code. We also show how the user is able to compose their customized workflow by leveraging database-centric activities. The data movement operations to transfer data from experimental sites are enabled by database replication. The user is able to register their custom processing codes for a particular application and maintain different versions of them. The registered code runs under the user's security credentials, as we are able to leverage the database security features, such as certificate based, domain based or password based authentication schemes.

The rest of the paper is organized as follows. Section 2 covers some of the recent developments in databases and how they can be exploited in scientific applications. In Section 3, we present the federated database architecture for wind tunnel experimental workflow. Section 4 covers the implementation details of the database activities that enable workflow integration. In Section 5, we discuss wind tunnel experiment workflow based on database activities. Section 6 presents discussions on how some of the existing scientific projects exploit database technologies. Finally, conclusions and future work are presented in Section 7.

2. Recent database trends – Leveraging for scientific applications

The capabilities of database systems are increasing and their architectures are undergoing continuous change. Some of the features that provide new possibilities for scientific application development are discussed below.

2.1. Language runtime

Many popular database systems now host language runtimes supporting high-level language stored procedures, functions, triggers, and user-defined data types. For example, SQL Server 2005 hosts the Microsoft .NET Common Language Runtime (CLR) [11]; the Java Virtual Machine (JVM) and .NET CLR are supported in Oracle [36] and IBM DB2 [3]. This enables scientific applications to manage both data and the processing code in databases. The implementation approach discussed in Section 4 leverages SQL Server CLR integration feature enabling user to register compute-intensive code written in any of the CLR languages (C++, Java, C#, and so on).

2.2. Native XML support

With XML becoming a data type, storing XML documents, validating them against a schema, and querying based on XQuery expressions are all part of the core XML functionalities built into popular database systems [18,20,26]. This feature is useful in processing XML message exchanges between Grid services and to store semi-structured scientific data in XML format.

2.3. XML Web Services

With the increasing interest in XML Web Services, database systems [2,8,9] are beginning to support web services hosting inside the databases, eliminating the need for external hosting containers or web servers. This would enable Web Services Resource Framework (WSRF) [10] based, or similar, Grid services to be exposed directly from the databases.

2.4. Transactional messaging

Asynchronous and reliable messaging between database instances are possible in present day database systems (SQL Service Broker [34] or Oracle streams [7]). We have utilized Microsoft SQL Server 2005 Service Broker for service level interactions which is discussed in Section 4. Service Broker objects include queues, dialogs, message types, contracts and services. These objects can be created using regular CREATE, ALTER and DROP Data Definition Language (DDL) commands. The messages from the transmit queue of the local database instance to the receive queue of the remote database instance can be transferred inside a transaction making the message transfer reliable. This database feature can be exploited for developing reliable Grid services.

2.5. Replication

The publish/subscribe model in replication allows tables, stored procedures or any other database objects to be published. Different replication styles determine when and how the data reaches the subscriber. For example, transactional push-style replication moves data to the subscriber in near-realtime. Database replication can be utilized when scientific applications have to deal with distributed data and the availability of data is to be ensured in more than one location.

3. Architecture

Figure 1 shows a federated database architecture for a typical multi-site wind tunnel facility which is similar to other applications. Wind tunnels are widely used to design, test and verify aerodynamics of aircraft, cars, yachts, and buildings, amongst others. The University of Southampton has three main wind tunnel facilities (11' $\times$ 8', 7' $\times$ 5' and 3' $\times$ 2') spread over the campus, housing heterogenous, specialized experimental hardware and software for academic and industrial research. The high volume of data generated from multiple experiments are transferred from the data acquisition system to a suitable network location where user can carry out further processing and analysis.

There are three logical database instances participating in the federation – (1) Site databases (SiteDB): Considering the importance of timely data movement and near-realtime requirements, the SiteDB publishes the experiment data tables to the MasterDB using transactional and push-style replication. This ensures immediate transfer of experimental data to the master as soon as the data imported into SiteDB from the data acquisition system. (2) Master database (MasterDB): This maintains user and application tables, and publishes them to sites and worker databases. It subscribes to experimental data from all the sites. The master node also runs workflow services for users to register, run and monitor their application workflow. (3) Worker databases (WorkerDB): This set of database instances is managed as a cluster of nodes. It carries out the processing work assigned by the master. It also manages different versions of custom user code for processing.

The database instances in the federation enable a complete end-to-end wind tunnel experimental workflow to be created and executed by hosting a set of database services (*activities*) with master node providing additional workflow services. The master schedules the processing activities from multiple user workflows onto worker nodes for load balancing. Access to other Grid resources, such as compute clusters, enabled using Grid and/or Web Services, is also supported based on our earlier work [22].

Figure 2 shows the sequence of messages and data exchanges between different database instances and the user's wind tunnel grid client. The actions labeled with letters A, B, C and D are independent of a particular workflow instance and they can happen at any stage. The users can compose workflows based on database activities, compile into a workflow assembly and submit to MasterDB using workflow services for scheduling (step A). They can also monitor the status of their currently running workflows (step B). They can compile a customized assembly and register it through the assembly management services running in MasterDB (step C1). The master in turn makes the assembly available to WorkerDB for registration and subsequent load balancing of users jobs (step C2). Each assembly is registered with a unique name derived from the username, application type and user specified version number. This unique name registration enables a user to maintain different versions of algorithms to process the experimental data.

The actual workflow execution starts when the user initiates data acquisition during an experimental run (step 1). When the data acquisition is over, the service waiting for acquisition to complete (step 2) changes the state of the current experimental run from "*Waiting for DAQ*" to "*DAQ over*". As the workflow is based on a state machine model, this state change transitions the workflow to the next stage, triggering an import data activity (step 3). Since the application data tables are subscribed at MasterDB and published by means of transactional push publication in sites, the newly imported data is transferred to MasterDB in near-realtime (step 4). Now, with data available at MasterDB and the user's application code registered with master and workers, the data can be distributed for processing (step 5). The processing requests to workers comprise username, application code and version to uniquely identify the assembly for processing (step 6). On receiving the processing request, the worker either invokes the default processing or a customized method registered by the user (step 7). The worker node sends the computed results and the status of the processing to the master (step 8). The master receives, consolidates and records the results (step 9). The final step involves a call to the Matlab interface to generate a plot and save it into the results table (step 10).

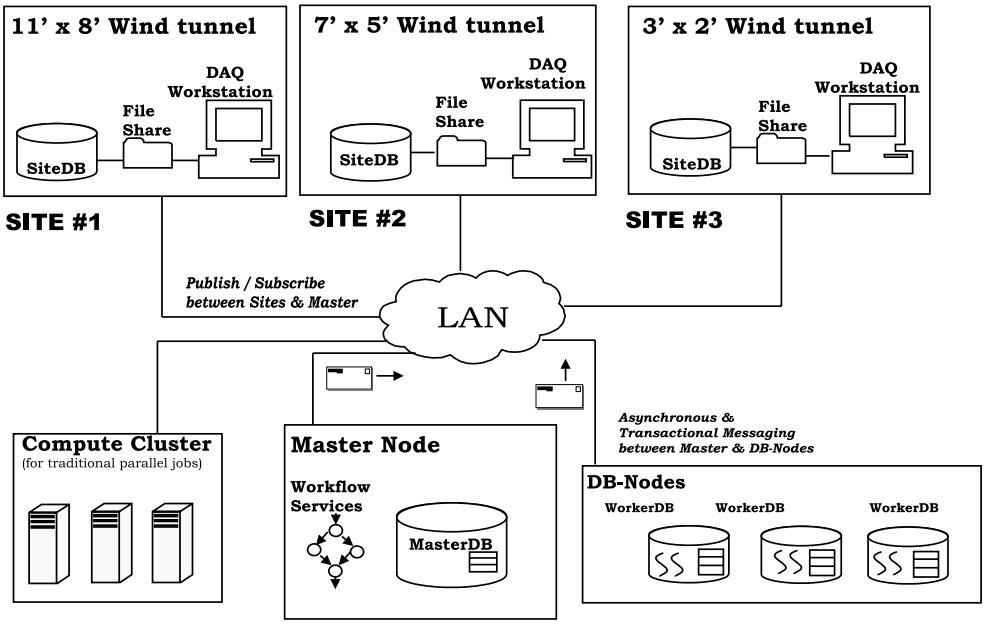


Fig. 1. Federated database architecture for wind tunnel application.

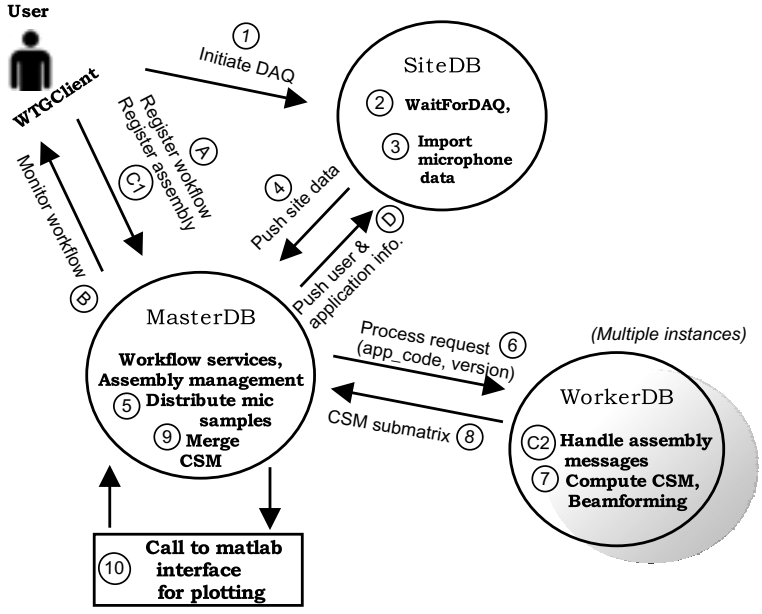


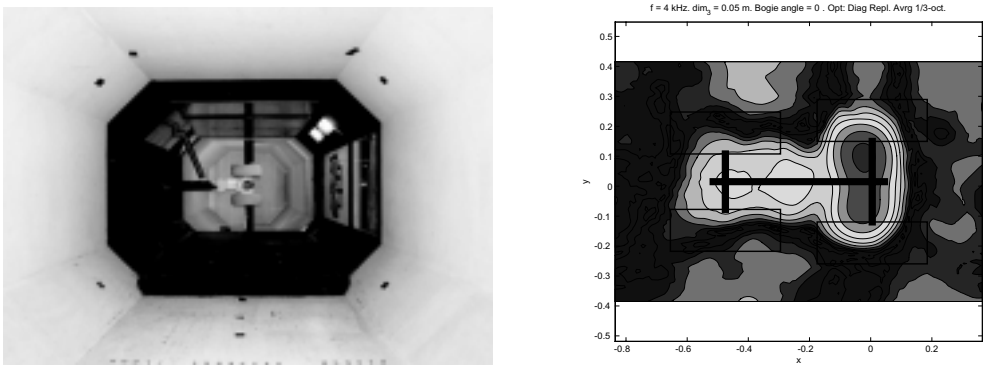
Fig. 2. Data and message flow for microphone array application.

4. Implementation

The implementation details we discuss in this section are based on Microsoft SQL Server 2005, leveraging SQL Service Broker [34] and .NET integration [11] features. These two features are typical of a modern RDBMS as discussed in Section 2, and we believe the generic approach is applicable to other database systems.

4.1. Motivating example – Microphone arrays

The microphone array technique (Fig. 3) is used to measure noise of aircraft components (slats, landing-



(a) Landing gear test (b) Typical noise contour plot

Fig. 3. Microphone phased array experiment [29].

gears, flaps etc) to help aerospace engineers improve aircraft design and to reduce the overall airframe noise [32]. Microphone arrays consist of multiple, around 100, microphones that must be simultaneously sampled. The phase shift between channels is then used to derive acoustic source information [29,32]. The microphone array processing happens in two stages: cross spectral matrix computation and beamforming. The cross-spectral matrix (CSM) is an $M \times M$ matrix, where M is the number of microphones. The CSM computing steps involve data calibration of the raw samples, Frequency Fourier Transform (FFT) computation, block averaging of cross spectral components and background noise removal [29,33]. By dividing the microphone samples, the CSM steps can be run in parallel as can be seen in Section 4.3. In the beamforming step, the beamforming expression is formed using precomputed CSM and grid coordinates are generated for plotting. The generation of beamforming plots for multiple frequencies can be run in parallel. Even though we consider microphone arrays as an example in this paper, the approach and the discussions are valid for other wind tunnel experiments as well [23].

4.2. Wind tunnel database

Figure 4 shows the database schema used for the wind tunnel experimental data management. The user table holds the username, user's role, user's X.509 certificate subject and other user-specific details. When the new application is created, the applications table is added with dedicated data, results and run table names. Both user table and applications table are published by the master and subscribed by sites and worker nodes. The run and results tables are maintained in the master. The data tables store both the raw data and the configuration information from an experiment. For example, *LDAData* and *MicArrayData* hold data for Laser Doppler Anemometry (LDA) experiment and microphone arrays experiment respectively. The data tables are maintained at wind tunnel sites and published to master by transactional replication. The raw data import and update happen at sites independent of the master and at the same time the data gets propagated from sites to the master in near-realtime. Every experimental run in the run table has an associated dataset in a data table. The relationship between runs and data is many-to-one. This allows multiple runs with different processing parameters to be associated with a single dataset. Similarly, the results table holds multiple result records for each experimental run. The user assemblies table is maintained at the master and worker nodes. It has one entry for each assembly registered by the user as will be discussed further in Section 4.3.

4.2.1. Stored procedures

The service code for the database activities are written as CLR stored procedures in three different assemblies, namely, site assembly, master assembly and worker assembly and they are registered in *SiteDB, MasterDB* and *WorkerDB* respectively. The main stored procedure in *SiteDB* is *ImportMicData*. The stored procedures that run at *MasterDB* are *Register/Remove/ReinstallAssembly, AddMicArrayRun, MasterCSMScheduler, CSMCompute* and *Beamforming*. At the worker the main stored procedures are *WorkerCSMScheduler, CSMCompute* and *Beamforming*.

4.2.2. Service broker objects

Table 1 shows how different service broker objects (messages, contracts, queues and services) are cre-

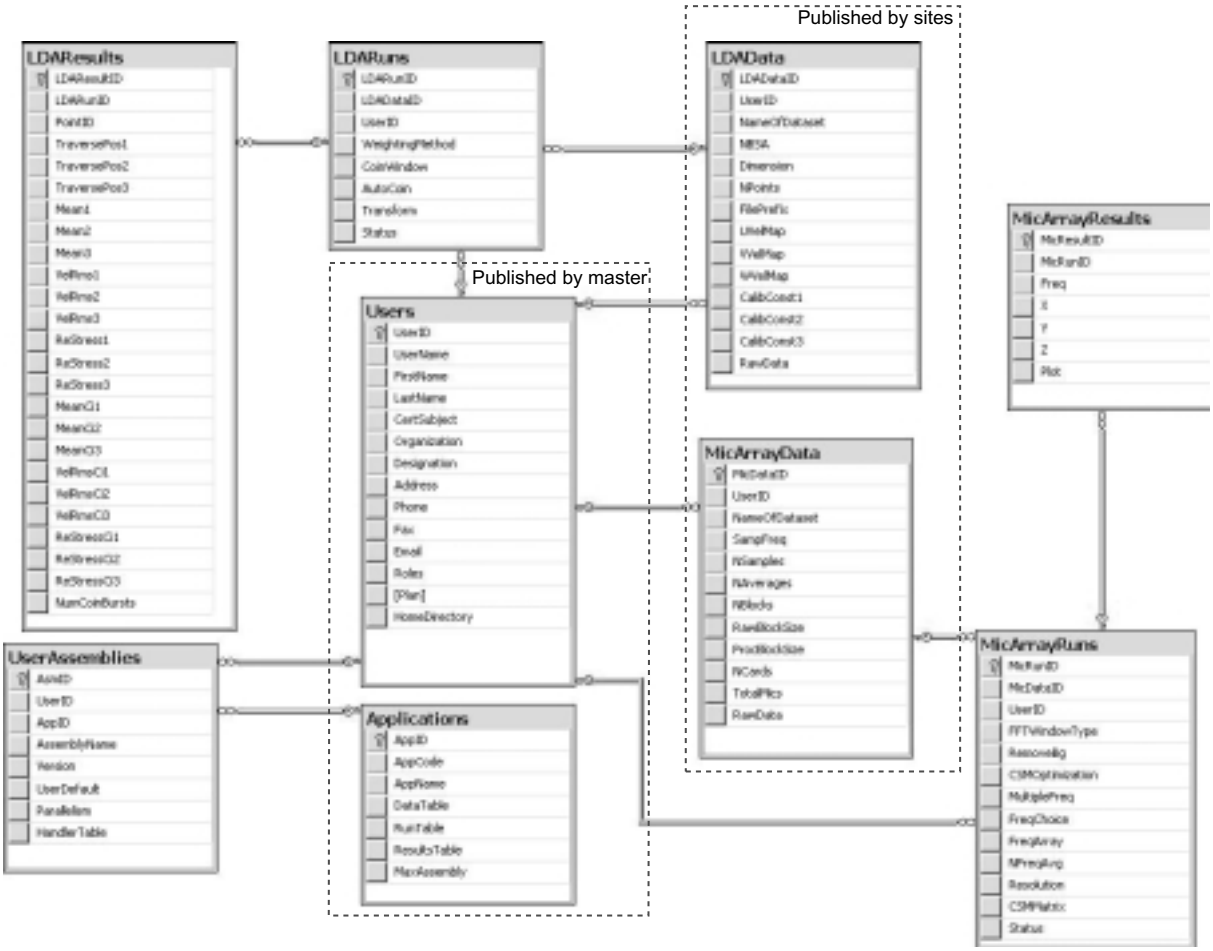


Fig. 4. Database schema for wind tunnel application data management.

ated using regular SQL DDL commands in the worker database. The message types can be binary or XML with a validation schema. A contract specifies the types of messages that can flow between a sender and receiver. Queues are placeholders for messages and central to transactional messaging; on new message arrival the activation stored procedure (for example, WorkerServiceProc in the *Queue* declaration) is invoked. A service is an endpoint that participates in a conversation. The master and worker communicate with the help of broker objects. Table 2 shows how the master would send a CSMComputeRequest message to worker.

4.3. Database activities

The following are the database activities that form the basis for the development of customized wind tunnel experiment workflows.

4.3.1. Assembly activities

Assemblies are libraries containing a user's application-specific processing code, which can be registered with the master and worker databases. The user can choose the default processing functions available or write a custom one, register with the master, and in turn, with the worker. Once the assembly is registered, the public interfaces (public class, static functions, data) are available for access from service stored procedures. The assembly activities RegisterAssembly, ReinstallAssembly and RemoveAssembly will internally translate into SQL DDL statements CREATE ASSEMBLY, ALTER ASSEMBLY and DROP ASSEMBLY respectively. On receiving register assembly message, the service procedure catalogs the assembly with a unique name derived from username, application code and version. The user's custom processing functions are identified and a lookup table (*HandlerTable* column)

Table 1
Service Broker objects creation using SQL DDL commands

Message types:

```
CREATE MESSAGE TYPE RegisterAssemblyRequest VALIDATION = WELL_FORMED_XML;
CREATE MESSAGE TYPE ReinstallAssemblyRequest VALIDATION = WELL_FORMED_XML;
CREATE MESSAGE TYPE RemoveAssemblyRequest VALIDATION = WELL_FORMED_XML;
CREATE MESSAGE TYPE AssemblyReply VALIDATION = WELL_FORMED_XML;
CREATE MESSAGE TYPE CSMComputeRequest VALIDATION = NONE;
CREATE MESSAGE TYPE CSMComputeReply VALIDATION = NONE;
CREATE MESSAGE TYPE BeamformingRequest VALIDATION = NONE;
CREATE MESSAGE TYPE BeamformingReply VALIDATION = NONE;
```

Contract:

```
CREATE CONTRACT WorkerContract(
       RegisterAssemblyRequest SENT BY INITIATOR,
       ReinstallAssemblyRequest SENT BY INITIATOR,
       RemoveAssemblyRequest SENT BY INITIATOR,
       AssemblyReply SENT BY TARGET,
       CSMComputeRequest SENT BY INITIATOR,
       CSMComputeReply SENT BY TARGET,
       BeamformingRequest SENT BY INITIATOR,
       BeamformingReply SENT BY TARGET );
```

Service Procedure:

```
CREATE ASSEMBLY WorkerAssembly FROM 'path/to/assembly' WITH PERMISSION_SET = SAFE;
CREATE PROC WorkerServiceProc
       EXTERNAL NAME WorkerAssembly.[WTG.DBLibrary.WorkerDBService].ServiceProc
```

Queue:

```
CREATE QUEUE WorkerQueue
       WITH STATUS = ON, RETENTION = OFF,
       ACTIVATION( STATUS = ON, PROCEDURE_NAME = WorkerServiceProc,
             MAX_QUEUE_READERS = 4, EXECUTE AS SELF );
```

Service:

```
CREATE SERVICE WorkerService ON QUEUE WorkerQueue (WorkerContract);
```

Table 2
An illustration of a transactional message exchange

Master database	Worker database
`BEGIN TRANSACTION`	`BEGIN TRANSACTION`
`SET @Message = <CSMComputeRequest>...`	`WAIT FOR(`
`BEGIN DIALOG @conversationHandle`	`RECEIVE TOP(1)`
`FROM SERVICE [MasterService]`	`@mesg_type = message_type_name,`
`TO SERVICE [WorkerService]`	`@Message = message_body`
`ON CONTRACT [WorkerContract]`	`FROM [WorkerQueue]`
`SEND ON CONVERSATION @conversationHandle`	`WHERE conversation_handle =`
`MESSAGE TYPE [CSMComputeRequest]`	`@ConversationHandle`
`(@Message)`	`);`
`COMMIT`	`COMMIT`

with metadata for function invocation is stored into the *UserAssemblies* table.

4.3.2. Process activities

The process activities execute application-specific code either from the default assembly or from a user registered one. When a processing message is sent from the master, corresponding lookup table is retrieved to invoke the user's custom processing function. For example, the microphone array processing involves a cross spectral matrix (CSM) computation and beamforming step. The CSM computation can be executed in parallel by splitting the raw microphone array samples equally among processing threads to improve the performance. The threads running on worker nodes compute the cross spectral matrix in parallel on portion of the data. The master receives the part results from workers, performs an averaging operation to form the cross spectral matrix and stores this in the *MicResults* table. The beamforming step is executed once for each frequency to generate the beamforming plot. In the case of multiple frequencies, different frequency values can

be sent to worker nodes to generate the beamforming plot in parallel. The output of individual beamforming steps are three square matrices X, Y and Z with grid point values for plotting.

4.3.3. Plot activity

Matlab has been chosen in order to generate publication-quality scientific plots. The plot function is written in Matlab which takes various arguments for plotting. The invoking of Matlab code from .NET was achieved using the Matlab .NET builder tool [5] which wraps the Matlab function into a .NET class. The plot activity uses the wrapper class to generate the plot and stores the image into the results table for download/visualization.

4.4. Microphone processing: Performance

Table 3 shows the cross spectral matrix processing timings for a C# command line application and for the same code hosted inside SQL Server as a Common Language Runtime (CLR) stored procedure. The two nodes utilized for this test are Dual Pentium III 1 GHz with 1 GB RAM running Windows Server 2003 and SQL Server 2005, and connected over a 100 Mbps LAN. The SQL Server hosts the .NET 2.0 runtime. The raw data samples used for the test were acquired from 56 microphones consisting of 100 blocks with a *block size* of 2048 (total samples = 204800) [29]. The load time is the time taken to read the microphone samples into memory for processing. In the command line case, the samples are read from a delimited text file and in the SQL CLR case, they are read from the *RawData* BLOB (Binary Large Object) column in the *MicArrayData* table. As can be seen from the table, parsing of samples from text file takes more than double the time of deserializing the raw data BLOB into memory. The SQL CLR cross spectral matrix processing timing is comparable and the overhead due to processing inside the database is marginal, as can be seen from the Fig. 5. The split time is the time taken to partition the samples among the threads and the merge time is the time taken to combine the cross spectral matrix received from threads by an averaging operation. The split and merge time for 2 threads of SQL CLR case on a single node is again comparable with the command line timings.

The computing of the beamforming expression involves multiplying the cross spectral matrix with a weight vector for each grid point of the plot. This matrix-vector multiplication can be optimized with specialized Intel's Streaming SIMD (Single Instruction Multiple Data) Extension [17] instructions. The optimized command line beamforming timing is obtained using the NMath Core [6] C# library (which in turn uses Intel's Math Kernel Library). This optimization library could not be registered into the database due to the SQL CLR strict versioning policies (we expect this to be resolved in the future). In order to provide a fair comparison, we measured the beamforming timings without optimizations (marked with $\star$ in Table 3); the SQL CLR timings is comparable to the same command line version.

The overhead due to queueing of the raw data is noticeable in the four thread case. But, this particular test is more to illustrate the advantage of reliably partitioning and load balancing a service using database messaging, than to show any speedup. With multiple users running different experiments producing a high volume of data, a reliable service to the wind tunnel experimental environment is more important. This is discussed further in Section 6. Also, for cases, where the experimental processing is long running due to data volume or the nature of the processing, the queueing overhead can be amortized. Further, the database messages can be routed through a low latency and high bandwidth network technologies, such as, InfiniBand and other high speed interconnects, to improve the performance.

5. Workflow integration

The workflow integration is achieved using Microsoft Windows Workflow Foundation that is part of the upcoming Microsoft .NET development framework 3.0 [4].

5.1. Windows Workflow Foundation

The workflow in Windows Workflow Foundation can be composed using a visual workflow designer or declaratively written in XML Applications Markup Language (XAML) or coded completely in CLR languages. The workflow must be compiled with a workflow compiler before it can be run. There are two workflow models supported [12]: (1) *Sequential* workflow model – comprising activities that execute in a predictable sequential path, and (2) *State machine* model – a flow driven by events triggering state transitions. In both these models the basic element of the workflow is called an *activity*. Some of the Windows Workflow Foundation's activ-

Table 3
Microphone array processing (CSM timings)

Dual P-III 1 GHz CPU; 1GB RAM	Load data	Split	Merge	CSM step	CSM Total	Beamforming step (single frequency)
Sequential (Command line)	30.231	–	–	89.400	89.400	99.043 (288.149⋆)
2 threads (Command line)	28.587	0.996	1.499	51.287	53.782	99.043 (288.149⋆)
Sequential (SQL CLR)	12.253	–	–	89.635	89.635	292.553⋆
2 threads (SQL CLR)	12.529	1.321	1.604	54.423	57.348	292.553⋆
4 threads (SQL CLR) on two nodes	11.850	13.971†	84.496‡	30.899	129.366‡	292.553⋆

⋆Without using matrix-vector multiplication optimizations. *All timings in seconds.*
†Time to split, serialize & send.
‡Time due to queuing delay & merge.

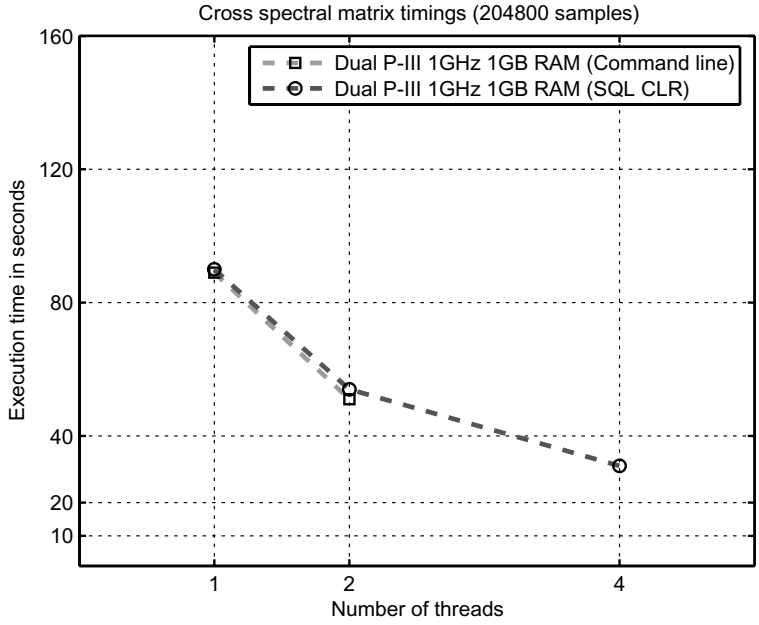


Fig. 5. Microphone cross spectral matrix performance.

ity types include control-flow (While, IfElse, Delay), exception (throw, exception-handler and BPEL compensations), data handling (Update, Select), transactions (and compensations for long-lived "transactions" that cannot be directly unwound) and communication (InvokeWebService, InvokeMethod). The *System.Workflow.ComponentModel.Acitivity* is the base class for all the activities. This extensible development model enables creation of domain-specific *activities* which can then be used to compose workflows that are useful and understandable by domain scientists. The workflow hosting layer part of the Windows Workflow Foundation is responsible for communication, persistence, tracking, transaction, timing, dynamic updates and threading. A long running workflow instance can be persisted, when it is faced with resource constraints, with all its state in a database so that it can be restarted again. With this flexible approach to workflow hosting and an extensible framework for workflow activities, most of the functionality of typical state-of-the-art scientific workflow systems [37] can be hosted on top of Windows Workflow Foundation.

5.2. Wind tunnel experiment workflow

The database activities discussed in Section 4 are wrapped into an experiment-specific workflow activity library for users to compose workflow and submit to the master node for hosting. Figure 6 shows the composition of custom microphone workflow based on a state machine workflow model.

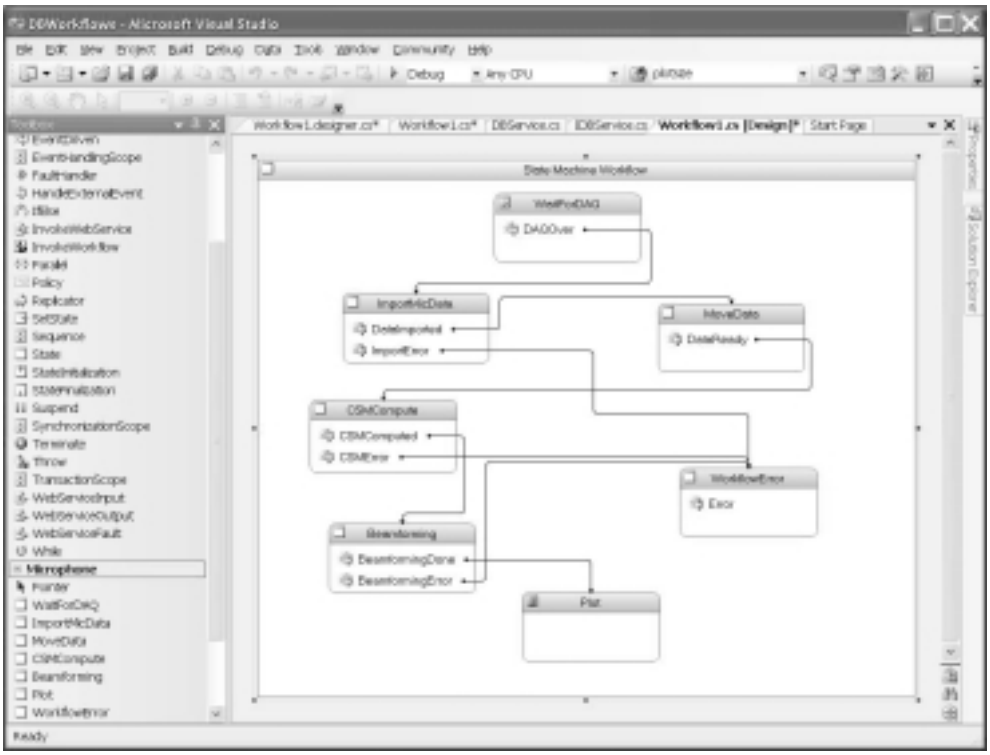


Fig. 6. Microphone experiment workflow based on database activities.

The initial state of the microphone workflow is *WaitForDAQ* and the final state on success is *Plot* or *WorkflowError* in case of any error during workflow execution. The experiment-specific activities are derived from the *State* activity. The *State* activity consists of one more event driven activities. For example, the *ImportMicData* state has *DataImported* event transitioning to *MoveData* state and *ImportError* event transitioning to *WorkflowError* state. On completion of an event, the *SetState* as part of the event-driven activity sequence transitions the workflow to the next state. The *CSMCompute* and *Beamforming* are workflow states representing processing.

The user composes the workflow, compiles it into an assembly and submits it to the master node for hosting. The workflow is scheduled and run at the master node. The workflow activities connect to the master database to execute the corresponding database activities described in Section 4. The state transitions of the workflow are recorded into the run tables of the master database. Users can monitor the status of the submitted workflow instance, to find out whether it is still running, completed successfully, or terminated with an error.

A similar, customized workflow approach to Laser Doppler Anemometry (LDA) experiment based on sequential workflow model is presented in [23].

6. Discussions

The nature and degree of use of Relational Database Management Systems (RDBMS) in scientific data management has been variable. Some of the usage has been to stream data near-realtime, to host scientific services by means of static stored procedures, to partition data to improve query performance, to store results, to store metadata and so on. In this section, we discuss related scientific projects, highlighting the degree of database systems usage and offer our arguments in favor of keeping databases central to the entire experimental workflow.

The NEESgrid framework [24], part of the Network for Earthquake Engineering Simulation project, supports instrument integration and exposes domain-specific Grid services for conducting and monitoring distributed earthquake engineering experiments. In terms of the experimental facilities, NEESgrid has some close similarities with wind tunnel experiments, but the emphasis is more on remote access to instruments in a multi-site environment. NEESgrid uses databases for metadata management only. The myLEAD [25] tool, part of the Linked Environment for Atmospheric Discovery (LEAD) project, provides spe-

cialized services for atmospheric scientists to search, store and catalog data objects generated during their investigations. The metadata catalog is managed in an RDBMS along with a set of database-stored procedures to expose persistent Grid services. It uses OGSA-DAI as a middleware for client interactions. OGSA-DAI [13] provides Grid service interfaces to different data sources (relational, XML, flat files). The advantages of database management systems in real-world scientific application have been demonstrated in Sloan Digital Sky Survey (SDSS) [31] project. With efficient indexing, join and parallel query operations, a twenty times speedup was achieved as compared to a file-based implementation. MySQL's streaming support has been utilized while hosting archival and real-time Geographical Information Systems (GIS) Grid services in [14]. The BioSimGrid project [1] manages large-scale biomolecular simulation data in flat files and associated metadata in RDBMS (Oracle). It supports simulation data to be deposited into a repository which is then replicated to different sites for retrieval and analysis.

As can be seen from the above applications, the major factors that influence how database systems are utilized in a scientific environment include data characteristics, nature of acquisition, processing requirements and performance. In the case of multi-user facilities such as wind tunnels where different experiments, multiple locations, multiple runs, changing parameters, high volume data and customized processing are the order of the day, it requires an approach that meets this set of demanding requirements.

The emphasis in the federated database approach [28] is on the ability of the local database instances to continue to support local operations autonomously, while they are part of the federation, to provide a set of global operations. Our architecture takes this approach while integrating the geographically distributed sites. The site databases operate independently in the federation, sharing information through publish/subscribe replication with the master. Also, the communication between the master and the worker nodes are by means of reliable transactional messaging. Any site node or worker node or even the master not being available temporarily will not affect the global operations. This is of particular importance in wind tunnel operations, which are deemed mission critical. A typical industrial scenario would be for a Formula One racing team. Sites would include the factory, multiple wind tunnel sites, testing and race tracks in different countries, where the network bandwidth and quality of service cannot always be guaranteed. Any of these sites could be offline for a number of reasons, and many are required to operate around the clock. Hence, local autonomy and reliability are important for such an application.

In general, scientific projects keep their raw data in flat files. In our approach, the wind tunnel raw data is imported as a Binary Large Object (BLOB) into the database at experiment sites and these are replicated to the master node for processing. As can be seen from [27], the load performance of read-only BLOB objects are comparable to the file systems, whereas write/update operations that result in fragmentation of the BLOB affect the load time. In our microphone experiment example, the raw data is a read-only object, the BLOB structure is accessed before *CSMCompute* and never gets updated. Similarly the CSM matrix BLOB, once stored, is always read before the *Beamforming* step. In this usage scenario, deserializing the BLOB is more advantageous as can be verified by the load time in Table 3 when compared to loading the raw data from text file stored in the file system. There is an additional overhead in parsing the floating-point samples from a text file.

In typical wind tunnel processing, an aerodynamicist would be interested in changing and customizing the processing algorithms. User customization of the processing algorithm, together with an ability to use the default processing steps, is an essential requirement. By taking advantage of the language runtime support inside databases, managing the user customized algorithm is possible in our approach. The Microsoft .NET development environment has an extensive support for languages such as Java, C++ and C#, with other language compilers such as Python and FORTRAN also being available, users can program in their language of choice. The user code is registered to run under the user's security credentials to gain authorized access to the dataset and results.

7. Conclusions

In this paper, we presented an approach to supporting an end-to-end engineering workflow based on federation of databases. The database instances host database services which are invoked from a state machine model workflow. We have demonstrated this approach with a reference implementation leveraging the features of SQL Server and Windows Workflow Foundation. The architecture is generic and can be implemented using

database systems other than SQL Server, such as ORACLE or IBM DB2.

The advantages in this approach include reduction in the overall turnaround time by providing an easy-to-use, extensible workflow framework, relieving the user of data management issues, and providing a robust and reliable system by using the features typical of commercial database systems, such as replication and transactional messaging. Even though the approach and the implementation discussed in this paper is with reference to wind tunnel experiments, it can be easily extended to other scientific and engineering application with similar characteristics.

Acknowledgments

The authors would like to thank Microsoft for their ongoing support. The microphone array work is funded by the UK EPSRC under grant number GR/S68446/01.

References

[1] BioSimGrid project. http://www.biosimgrid.org, [September 2006].
[2] DB2 Web Services: The Big Picture. http://www.ibm.com, [August 2002].
[3] A detailed look at DB2 Stinger .NET CLR Routines. http://www.ibm.com, [June 2004].
[4] Introducing the .NET Framework 3.0. http://msdn.microsoft.com/, [July 2006].
[5] MATLAB Builder for .NET User's Guide Version 2. http://www.mathworks.com, [March 2006].
[6] NET Numerical Applications with NMath Core, white paper, http://www.centerspace.net, [July 2006].
[7] Sharing Information with Oracle Streams. www.oracle.com, [May 2005].
[8] Using Native XML Web Services in SQL Server 2005. http://msdn2.microsoft.com, [July 2006].
[9] Virtualize Your Oracle Database with Web Services. http://www.oracle.com, [November 2005].
[10] Web Services Resources Framework v1.2 OASIS Standard. http://www.oasis-open.org/, [April 2006].
[11] A. Acheson et al., *Hosting the .NET Runtime in Microsoft SQL Server*, in ACM SIGMOD Conference, 2004, 860–865.
[12] P. Andrew et al., *Presenting Windows Workflow Foundation*, Beta Edition, Sams, September 2005.
[13] M. Antonioletti et al., The design and implementation of Grid database services in OGSA-DAI, *Concurrency and Computation – Practice and Experience* **17** (2005), 357–376.
[14] G. Aydin, G.C. Fox et al., *Streaming Data Services to Support Archival and Real-Time Geographical Information System Grids*, in Sixth Annual NASA Earth Science Technology Conference, June 2006.
[15] J. Gray, The Revolution in Database Architecture, Tech. Report MSR-TR-2004-31, Microsoft Research, March 2004.
[16] L.M. Haas, E.T. Lin and M.A. Roth, Data integration through database federation, *IBM Systems Journal* **41** (2002), 578–596.
[17] Intel Corporation, IA-32 Intel Architecture Software Developer's Manual, Programming with Streaming SIMD Extensions 3 (SSE3), June 2006.
[18] Z.H. Liu, M. Krishnaprasad and V. Arora, *Native XQuery processing in Oracle XMLDB*, in ACM SIGMOD Conference, 2005, 828–833.
[19] S. Narayanan, T. Kurc, U. Catalyurek and J. Saltz, Database Support for Data-driven Scientific Applications in the Grid, *Parallel Processing Letters* **13** (2002), 245–271.
[20] M. Nicola and B.V. der Linden, *Native XML Support in DB2 Universal Database*, in Proceedings of the 31st VLDB Conference, 2005, 1164–1174.
[21] S. Pallickara, B. Plale, S. Jensen and Y. Sun, *Structure, Sharing and Preservation of Scientific Experiment Data*, in IEEE Workshop on Challenges of Large Application in Distributed Environments, July 2005.
[22] A. Paventhan, K. Takeda, S.J. Cox and D.A. Nicole, MyCoG.NET: a Multi-language CoG toolkit, Concurrency and Computation – Practice and Experience, DOI: 10.1002/cpe.1133.
[23] A. Paventhan, K. Takeda, S.J. Cox and D.A. Nicole, *Workflows for Wind Tunnel Grid Applications*, in International Conference on Computational Science, 2006, 928–935.
[24] L. Pearlman et al., *Distributed Hybrid Earthquake Engineering Experiments:Experiences with a Ground-Shaking Grid Application*, in Proceedings of the 13th IEEE Symposium on High Performance Distributed Computing, June 2004.
[25] B. Plale et al., Active Management of Scientific Data, *IEEE Internet Computing* **9** (2005), 27–34.
[26] M. Rys, *XML and Relational Database Management Systems: Inside Microsoft SQL Server 2005*, in ACM SIGMOD Conference, 2005, 958–962.
[27] R. Sears, C. van Ingen and J. Gray, To BLOB or Not To BLOB: Large Object Storage in a Database or a Filesystem?, Tech. Report MSR-TR-2006-45, Microsoft Research, June 2006.
[28] A.P. Sheth and J.A. Larson, Federated database systems for managing distributed, heterogeneous, and autonomous databases, *ACM Computing Surveys* **22** (1990), 183–236.
[29] M.G. Smith, B. Fenech et al., Control of noise sources on aircraft landing gear bogies, in 12th AIAA/CEAS Aeroacoustics Conference, AIAA Paper 2006-2626, May 2006.
[30] H. Stockinger, *Distributed Database Management and the Data Grid*, in Proceedings of the Eighteenth IEEE Symposium on Mass Storage Systems and Technologies, April 2001.
[31] A.S. Szalay, J. Gray et al., *The SDSS SkyServer – Public Access to the Sloan Digital Sky Server Data*, in ACM SIGMOD Conference, 2002, 570–581.
[32] K. Takeda, X. Zhang and P.A. Nelson, *Unsteady aerodynamics and aeroacoustics of a high-lift device configuration*, in 40th AIAA Aerospace Sciences Meeting and Exhibit, AIAA Paper 2002-0570, 2002.
[33] R.J. Underbrink, *Aeroacoustic Measurements*, Springer, 2002.
[34] R. Walter, *The Rational Guide To SQL Server 2005 Service Broker*, Rational Press, 2005.
[35] P. Watson, Databases and the Grid, in: *Grid Computing: Making the Global Infrastructure a Reality*, F. Berman, G.C. Fox and T. Hey, eds, Wiley, 2003.
[36] M.A. Williams, Pro .NET Oracle Programming, Apress, 2004.
[37] J. Yu and R. Buyya, A taxonomy of scientific workflow systems for grid computing, *ACM SIGMOD Record* **34** (2005), 44–49.