Integrating databases and workflow systems

Srinath Shankar Ameet Kini David J DeWitt Jeffrey Naughton

Department of Computer Sciences
University of Wisconsin
Madison, WI 53706-1685
{srinath,akini,dewitt,naughton}@cs.wisc.edu

Abstract

There has been an information explosion in fields of science such as high energy physics, astronomy, environmental sciences and biology. There is a critical need for automated systems to manage scientific applications and data. Database technology is well-suited to handle several aspects of workflow management. Contemporary workflow systems are built from multiple, separately developed components and do not exploit the full power of DBMSs in handling data of large magnitudes. We advocate a holistic view of a WFMS that includes not only workflow modeling but planning, scheduling, data management and cluster management. Thus, it is worthwhile to explore the ways in which databases can be augmented to manage workflows in addition to data. We present a language for modeling workflows that is tightly integrated with SQL. Each scientific program in a workflow is associated with an **active table** or view. The definition of data products is in relational format, and invocation of programs and querying is done in SQL. The tight coupling between workflow management and data-manipulation is an advantage for data-intensive scientific programs.

1 Introduction

Cutting edge science has been witness to an information explosion. In recent years, scientists from fields such as high energy physics, astronomy, environmental sciences and biology have been overwhelmed by the task of managing the vast quantities of data generated by their experiments. Some examples are ATLAS [7], SDSS [10], GEON [5] and BIRN [1]. In addition to the data, scientists also have to deal with a large number of specialized programs. There is a critical need for automated systems to manage scientific applications and data. Thus, the study of scientific workflows has gained importance in its own right. Several WFMSs (GriPhyn [6], Griddb [29], Zoo [26], Kepler [12]) have been proposed to provide functionality such as workflow modeling, execution, provenance, auditing and visualization. The importance of the Grid and systems like Condor [4] to workflow management has also been recognized. While database technology has been utilized to some extent in each of these systems, none of them really explore the full power of DBMSs in handling large magnitudes of data. Furthermore, contemporary WFMSs are built from multiple components, each of which performs a separate function. We believe it is a mistake to study the different aspects of workflow management in isolation. Databases should play a crucial role in the big picture, and this must motivate architectural decisions with respect to workflow modeling, planning, execution and data management.

2 The Grid - An added dimension

Recent research in scientific workflow management has focused on the Grid. Several definitions of the grid exist [16]. Theoretically, a computing grid is like a power grid. Users can 'plug into' it and avail themselves of the vast computing resources available around the world.

The most popular distributed computing system that approximates this behavior is Condor, which is used to manage clusters (or pools) of machines. Condor is an excellent way to harvest the CPU cycles of idle machines. It provides useful virtualizing services, such as migrating jobs and transferring the input and output files for a job to the right machine. Furthermore, it is easy to customize on a per-job basis. Users can specify the type of environments they need to run their jobs, and individual machines in the pool can implement policies regarding the kinds of jobs they are willing to run. Thus, it is an invaluable tool for scientists with long-running, resource-intensive jobs.

Condor has established itself as the work-horse of scientific computing. At last count, (Jul 2005) it was installed on more than 58000 machines organized across 1500 pools in universities and organizations world-wide. Condor was developed in the early '80s primarily as a cycle harvesting system. Since then a lot of the assumptions that motivated the design of Condor have changed. In the '80s computing resources were scarce and expensive. Now, large clusters of commodity machines have become affordable for almost everyone. Although the volumes of data available to and processed by typical scientific applications has increased, the cost of storage has also plummeted. In light of the advances in processor and disk technology over the last decade, we believe it is worthwhile to re-evaluate some of the design decisions made in the architecture of Condor. As the need for cycle harvesting on idle desktop machines diminishes and the need for better cluster management comes to the

fore, we feel that grid computing systems such as Condor can exploit database technology to provide improved functionality, scalability and reliability.

3 Related work

A lot of research has already been done in the field of managing *business* workflows with databases. Examples include active databases [31], enhanced datalog [15] and relational transducers [11]. Oracle 9i introduced job queues for the periodic execution of administrative and house-keeping tasks defined in the procedural PL/SQL language. Oracle 10g offers database views for monitoring the status of jobs, programs, program arguments and schedules. Solutions for business workflows have mainly concentrated on the *control* flow between processes.

Scientific workflows, on the other hand, pose an entirely new set of challenges. Scientific jobs are usually long-running and highly resource-intensive. Thus, efficient *data*-flow management is essential. In addition, scientists require sophisticated tools to query and visualize their data products. One of the first systems built to handle workflows in this domain was Zoo [26], a desktop experiment management environment. It used the object-oriented language Moose to model control flow between jobs as relationships between entities that represented jobs, input data and output data. Job invocation was handled by assigning rules on these relationships. Zoo also provided a variety of workflow auditing capabilities that allowed users to query the state of the database using the Fox query language. The Zoo system was architected with a data-centric view of workflows and was built on the Horse OODBMS.

The GriPhyn project [6] is a collaborative effort toward a standard solution for handling scientific workflows. Its components include Chimera [23], Pegasus [21] and a Replica Location Service [19]. Chimera allows users to declare programs, their logical data products and the composite workflow graph in a Virtual Data Language. Given a request for a particular virtual data product, Chimera analyzes the Virtual Data Catalog and comes up with an abstract DAG representing the sequence of operations that produce that data. Pegasus, the planner, locates physical file replicas (using RLS) and uses resource information (from the Globus Monitoring and Discovery Service [25]) to come up with a concrete plan of execution. The concrete plans produced by Pegasus are in fact Condor DAGMan [2] submit files. The DAGMan scheduler in Condor is used to execute the programs on a Condor pool. Pegasus uses artificial intelligence techniques to choose amongst multiple possible physical replicas and minimize resource usage in its planning algorithms.

The Ptolemy II [9] is based on the concept of actors, which are independent components that perform tasks such as data transformations, specific steps in an algorithm or simply opening a browser or a shell program. Actors have well-defined interfaces called ports and can be composed into scientific workflows. The idea is to promote the reuse of entities and thus make scientific workflows easy to design and more modular. Kepler [12] provides extensions to Ptolemy such as a *Webservice* actor to enable access to remote resources and *FileStager*, *FileFetcher* and *GlobusJob* actors to enable workflows to make use of the Grid. It also provides a *Director* actor, similar to Condor's DAGMan, that can be used to schedule the execution order of individual actors. The Kepler system has been used in scientific projects such as GEON, SEEK and SciDAC/SDM. While Kepler provides a useful way to model scientific workflows, it does not address the larger issues of planning workflows or providing fault-tolerance measures.

In GridDB [29], the inputs and outputs of programs are modeled as relational tables. It allows users to define programs and the relationship between their inputs and ouptuts in a functional data modeling language (FDM). Insertion of tuples in input tables triggers the execution of programs in the workflow. Programs are executed by submitting them to Condor.

Turning to workflow execution substrates, the Condor team at UW-Madison has worked on a number of projects aimed at increasing the functionality and efficiency of Condor. These include Stork [28] and DiskRouter [3] for efficient data placement, Condor-G (which integrates Condor with the Globus toolkit [25]) and Hawkeye [8] (which lets users monitor their jobs). These have been integrated with Condor in varying degrees.

Perhaps the most relevant work with respect to the execution of scientific programs was done by Gray et al. [24]. Using the cluster-finding example from the Sloan Digital Sky Survey, they demonstrated the benefits of modern DBMSs, such as using indices, parallelizing query execution and using efficient join algorithms. The performance obtained by using a database (Microsoft's SQL Server) to store and query astronomical data was orders of magnitude better than previous implementations of the algorithm. They advocate a tighter integration of computation and data, i.e. involving the DBMS in the analysis and computation of scientific applications and not relegating it to a passive store.

4 Limitations of current workflow systems

Most of the workflow management systems described above are composed of multiple components that were developed independently. Thus, inter-component communication is set up along very rigid channels. For example, consider the role of planning in a WFMS. In their

analysis of batch workloads across remote compute clusters, Bent et al. [14] showed that there are qualitative differences in the types of I/O performed by an individual job during its lifetime. *Batch* I/O refers to the input that is common to all jobs in a batch, while *pipeline* I/O is the data flow that occurs between jobs in a particular DAG. The authors recognized the importance of exploiting the sharing characteristics of batch input and providing locality of execution for programs that share pipeline I/O to minimize network traffic. Current WFMSs are not closely integrated with the user data, and are thus unable to do such detailed planning. For example Pegasus, the planning component of GriPhyN, which uses AI techniques to arrive at globally optimal solutions, doesn't have access to detailed job characteristics. GridDB currently has no way of planning workflows at all, since it is constrained by the Condor job-submission interface. Incorporating planning into such systems will require a substantial reworking of their design. While the approach pursued by Gray et al. seems the most holistic solution, it involves invoking program modules from within SQL statements (in the form of user-defined functions or stored procedures). Most databases do not directly execute binaries or interpret arbitrary code. Programmers who wish to invoke modules as part of SQL statements have to conform to strict specifications while writing and compiling them, and are usually constrained in the languages they can use (C,C++ and Java). Scientists may balk at having to rewrite their applications to allow them to be run by a database. We feel the continued usage of legacy applications in languages such as Fortran, and the importance of file-based (as opposed to database) I/O in the scientific domain may hinder the adoption of the technique espoused by Gray et al. Furthermore, rewriting workflows in SQL is not sufficient. Scientists also want ways to set up and execute workflows.

The Zoo system used objects and relationships to model workflows. Like GridDB, triggers were used to activate workflows. However, Zoo was designed as a desktop management system and not for cluster management, which is an important component of a WFMS. Zoo had no mechanisms for gathering workload characteristics, maintaining data replicas across nodes and scheduling data transfers between machines.

5 Why a database is essential

At this juncture, it is important to concretize the notion of a WFMS. How is it used, and what demands do we make of it in terms of functionality? From the user's perspective, there are three important aspects to a WFMS.

- The declaration and specification of workflows, processes and data
- The invocation of scientific workflows
- The ability to monitor workflows.

A WFMS must make it easy for a user to do the above while successfully abstracting away the following behind-the-scenes activities:

Workflow Planning – The various processes that comprise a workflow and the attendant data transfer must be planned to maximize system efficiency and throughput.

Scheduling – It is the responsibility of the WFMS to invoke individual programs and schedule data transfers according to a plan.

Data management – A WFMS must keep track of the data produced by user workflows, manage data replicas and consistency, provide data recovery in the face of failure and maintain versions of user programs and data as a workflow evolves.

Cluster management – A WFMS must monitor the state of its cluster, including network connectivity and machine parameters such as disk space and load averages. It must handle temporary and permanent machine failure in a transparent manner while distributing load evenly.

As Sections 3 and 4 show, database technology has been used only to a limited extent, mostly to store metadata and task descriptions. For instance, Griphyn's RLS simply uses a database to store its replica catalog. GridDB uses a database to store *memo tables*, which are correspondences between the inputs and outputs of a program that has completed, and *process tables* that contain state information of currently executing programs. In commercial databases, job queues have been used to schedule administrative tasks and for interprocess communication. We believe database technology has a lot more to offer the realm of scientific workflow management.

Planning – The research in [14] demonstrates the need for workflow planning if clusters are to scale to a large number of machines (about 100,000) executing huge batch workloads. No satisfying solution currently exists to this problem, and we believe it is an important one to solve in the near future. Database technology has long concerned itself with the issue of optimizing queries over distributed data sources to minimize CPU time and network and disk I/O ([22],[30]). Moreover a great deal of attention has been paid to dynamic query optimization [20], which uses feedback from currently executing queries to make plan adjustments. Thus, databases are ideally suited to planning data-intensive scientific workflows in dynamic environments.

Provenance – A good workflow management system should provide administrative assistance in addition to job scheduling services. The importance of data provenance has already been recognized in the GriPhyn and GridDB projects. Data provenance has been studied from both theoretical [17] and practical perspectives

by the database community. Data management systems such as [32] have demonstrated the ability of databases in providing real provenance services to users.

Concurrency control – As cluster and workload sizes increase, it becomes necessary to recognize and prevent interference between simultaneously executing jobs. Current WFMSs do not address this problem adequately. Databases are good at handling concurrent access at various levels of granularity, maintaining replica consistency for data objects and resolving conflicts. In addition, databases can provide different degrees of consistency for different transactions.

Recovery – Recognizing and handling multiple failure modes in a grid computing environment is very important. For example, a job must not be 'lost' or 'forgotten' when machines involved in its execution crash. Databases are good at recovering data and system state in a transparent way while minimizing the impact on currently executing processes and efficiently balancing load on the remaining machines.

Transactional semantics and persistence – The notion of a transaction is essential to workflows. The atomicity and durability of certain sequences of actions, such as moving data and jobs from one machine to another, is important. Condor's DAGMan is a custom-designed tool that allows users to specify workflow graphs that are guaranteed to be executed. Even this popular tool doesn't allow users to 'rollback' portions of a graph when their programs fail or abort jobs that were mistakenly submitted or violated certain constraints. Transactional semantics would provide greater functionality and simplify the design of WFMSs.

Querying capabilities –In addition to user data, grid compute clusters generate large amounts of operational data on a daily basis. This data includes the status of machines and jobs in the cluster, user information and file access information. For example, in Condor, such data is spread over log files across multiple machines, making it very hard to administer. In fact, a lot of it is thrown away due to archaic space management policies. The tasks of managing a pool of machines, monitoring its health and diagnosing problems could be simplified and even automated if its operational data were accessible using SQL in a uniform, declarative fashion. The support for SQL provided by a DBMS would also allow users to query the status of their personal jobs and data without impinging the privacy of other users.

To summarize, many components required of a scientific WFMS have been a part of DBMS technology for a long time. A comprehensive system that encompasses workflow planning, data and cluster management must have a database system as its core. In fact, since databases provide much of the functionality required of a WFMS, we feel it makes sense to ask – Can we augment databases to handle workflows in addition to data? A tightly integrated architecture is needed and the design of all components of a WFMS must be motivated by the use of database technology. No aspect of workflow management should be viewed in isolation from the others. For instance, a workflow modeling language that is similar to SQL could leverage database query optimization techniques for the efficient execution of data-intensive scientific jobs. In addition, in order to plan and schedule workflows, query optimizers must have access to information such as replica locations, machine availability and load in relational format. This would allow most WFMS management operations to be performed in a way analogous to traditional data management operations.

6 A modeling framework

In this section we present a workflow modeling language that tightly integrates WFMSs and DBMSs. First, we identify some of the features a workflow modeling language should possess:

Program specification – The language should let users declare programs and specify an 'invocation format' for the programs. By invocation format we mean the format in which the program and its parameters are presented to a cluster for execution. For example, the invocation format could be a shell command, or a Condor submit file.

Data specification – Users may wish to explicitly model the input and output of their programs, and a modeling language must allow users to assign a schema to this data.

Data 'transducers' – Almost all scientific programs are written to deal with data in files. Thus, a modeling language must provide for transducers that interpret data resident in flat files and provide a relational 'view' of it.

Specification of control and data flow – Users must be able to compose their programs and data into a unified workflow.

Workflow invocation – The modeling language must allow users to activate workflows in a transparent and well-defined manner.

To demonstrate our modeling language we use the ATLAS high energy physics workflow ([29]). An event generator program (*gen*) is used to feed two different simulator (*atlsim* and *atlfast*) and their output is compared (Figure 1).

The modeling language we present is based on the concept of an **active table**. Every program in a workflow is associated with an active relational table. An active table has two parts to its schema – one for the input of the program and one for its output. For example, the event generator *gen* is declared in the following way (keywords are in uppercase)

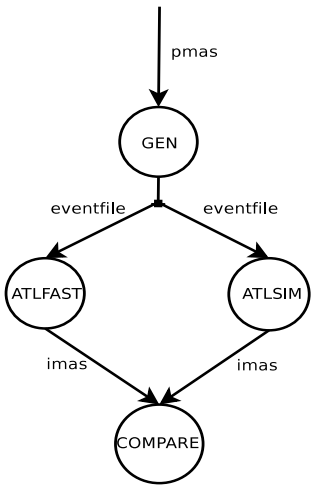

Figure 1: The simplified Atlas Workflow

```
Example I (GEN):
CREATE ACTIVE TABLE Gen
WITH PROGRAM '/path/to/gen'
INPUT (pmas INTEGER) INVOCATION (" $pmas")
OUTPUT (eventfile VARCHAR) FORMAT 'genschema.xml'
```

Program *gen* takes as input an integer *pmas* and produces a file of events (to be used by the simulators). The `INPUT` and `OUTPUT` clauses capture this information – `pmas` contains the integer parameter for *gen*, and `eventfile` contains the name of the file it produces. The schema of an active table is the concatenation of its input and output fields. Thus, the table `Gen` has the schema `Gen(pmas INTEGER,eventfile VARCHAR)`. The `INVOCATION` clause specifies that the program is invoked by issuing the command '`gen <pmas>`'. The purpose of the `FORMAT` clause is explained in the next example.

Workflows are composed by declaring active views, which are similar to active tables. Consider the program *atlfast*, which takes as input the event file produced by *gen*. The declaration of *atlfast* is as follows

```
Example II (ATLFAST):
CREATE ACTIVE VIEW Atlfast
WITH PROGRAM '/path/to/atlfast'
INPUT (eventfile varchar)
INVOCATION ("-events $eventfile")
AS SELECT Gen.eventfile FROM Gen
OUTPUT (imas integer) FORMAT 'atlschema.xml'
```

As was the case in the Example I, the relational view `Atlfast` is associated with the program *atlfast*. The active view has been defined to take as input the filename `Gen.eventfile`, which is output by the program *gen*. (It is not required that the `INPUT` field in `Atlfast` also be called `eventfile`). The table `Atlfast` has the schema `Atlfast(eventfile VARCHAR, imas INTEGER)`.

Most scientific programs process data in a filesystem. For instance, the output of the *atlfast* program is an integer which it stores in a file. To interpret data in files, the WFMS needs to know the format in which it is stored, and what the desired relational format is. The `FORMAT` clause is used to specify an XML file containing this information.

Workflow automation is accomplished in the following way – In examples I and II, `Gen` is the base table, and `Atlfast` is the derived table. The workflow is set up by populating the base table with input data. Each record of input corresponds to a separate execution of the workflow. A section of the workflow is invoked by issuing a query on the corresponding derived table. For instance, `Gen` could be populated as follows:

```
INSERT INTO Gen(pmas) VALUES (100)
INSERT INTO Gen(pmas) VALUES (200)
etc.
```

The *gen-atlfast* section of the ATLAS workflow can be explicitly started using the SQL query:

```
SELECT Atlfast.imas FROM Atlfast
```

The general idea is that a program is invoked whenever a query is issued on the `OUTPUT` fields of its active table (in the above example, the output field is `imas`). Thus, *gen* is invoked for each value of `pmas` (100,200,...), and *atlfast* is called for each event file *gen* produces. If a program fails due to an error or abnormality, the corresponding `OUTPUT` field is filled with `null` or an appropriate error value. For example, suppose the program *gen* with input 200 fails. Then the corresponding `Gen.eventfile` field is set to `null`. The exact input that caused the error can be detected by the simple query:

```
SELECT Gen.pmas FROM Gen
WHERE Gen.eventfile IS NULL
```

To complete the ATLAS example, here is the declaration of the *atlsim* program. Its input and output are similar to *atlfast*

```
Example III (ATLSIM):
CREATE ACTIVE VIEW Atlsim
WITH PROGRAM '/path/to/atlsim'
INPUT (eventfile varchar)
INVOCATION ("-events $eventfile")
AS SELECT Gen.eventfile FROM Gen
OUTPUT (imas integer) FORMAT 'atlschema.xml'
```

The comparison between the output of the two simulator programs can now be done by means of a SQL query

```
Example IV (COMPARE):
SELECT F.imas, S.imas FROM Atlfast F, Atlsim S
WHERE F.eventfile = S.eventfile
```

Thus, in this workflow modeling language, initial input data and programs are represented as active tables, and derivative programs and data as active views. For each program in the workflow, exactly one table/view has to be defined. Any section of the workflow can be explicitly invoked by issuing a SQL query on the corresponding views or tables (Such a query is the comparison query in Example IV). Workflow automation is simply an extension of active view maintenance. Since the

views are generated by the output of possibly long running programs, it might be best to materialize them instead of regenerating them on each query. This prevents unnecessary re-execution of programs. Like traditional materialized views, active views can be updated in two ways

- When base data changes (that is, input is inserted into the initial tables in the workflow), successive stages of the workflow can be invoked – the 'push' method
- When a query is issued (explicit invocation) views can be updated if necessary – the 'pull' method.

If the 'pull' mechanism is adopted, users might have to wait long periods of time for the result of a query. We feel a 'push' mechanism would increase the degree of automation the WFMS provides.

The modeling language presented above is meant for tight integration with a database. The declaration and definition of workflows is in relational format, and the invocation and querying is done in SQL. Thus, as example IV shows, data manipulation can be closely integrated with workflow execution. Moreover, the query graph that represents the output (in this case, example IV) is the same as the workflow graph. Coupled with a knowledge of the data products, this opens the door to workflow optimization using database techniques. For instance, the WFMS need not materialize the outputs of *atlfast* and *atlsim* before executing the join – a pipelined plan can be chosen in which the join is executed 'on the fly', thus improving efficiency. Architectures that simply layer a modeling framework over an execution substrate do not lend themselves to such optimizations.

7 An execution framework

We now turn to the execution substrate, with Condor as the standard. Condor consists of several daemons such as the *schedd* to which users present jobs for submission, the *startd* which is responsible for advertising machines in the pool, and the *negotiator* which matches jobs to machines. The task of managing, debugging and querying the system is a complex task involving multiple daemon log files across hundreds of machines in the pool. Recent research at UW-Madison has addressed many of the problems facing system administrators and users by providing a powerful window into the system while simultaneously laying the foundation for a more comprehensive DB-managed cluster architecture. The original Condor daemons on different machines in the pool were modified to report operational data to a central database. This data lends itself to diagnostic queries (Which machines are down and for how long? Why hasn't a job run?), as well as user queries (What is the status of my jobs? How much system run-time have I got?). Additionally, certain information that was hard to obtain previously is now easily accessible. For instance, any aggregate query on the I/O performed by a job would previously involve scanning multiple log files over all the machines the job ever executed on – now it's a simple SQL query.

The instrumentation of the Condor daemons is also an important first-step toward building a WFMS around a DBMS – the operational data collected could be used to compile statistics about jobs (such as run-time, I/O activity, disk usage), data (such as file usage, size and sharing characteristics) and machines (load average, job preference etc). This data would be invaluable in planning, scheduling and executing scientific workloads. Indeed, fundamental changes can be made to the Condor architecture itself. For example, a regular database join can be adapted to perform the task of *matchmaking* – pairing jobs and machines based on their requirements. Initial work in this area has produced encouraging results [27]. The daemons themselves can be simplified if the database is used smartly. For example, job submission can be reduced to inserting a few records in a 'Jobs' table in the database. Log files can eventually be eliminated. This architecture paves the way for a more active role for the database in tasks such as planning, job scheduling, data transfer and cluster management.

8 Challenges and conclusion

Several challenges lie in the way of a DB-integrated workflow management system. Firstly, there is an obvious 'impedance mismatch' between the execution environment offered by a database and that offered by a traditional OS. Most scientists are used to programs that handle data in a traditional file system. Storing data in relations is quite different. Databases usually require a fixed schema and do not allow arbitrarily complex objects. While traditional user programs can access files in any pattern, access to relations is usually done via cursors. A practical approach would involve allowing the database to execute external user programs and providing a real-time relational 'view' of user data, queryable via SQL. A preliminary implementation of this functionality has been made in the PostgreSQL open source ORDBMS. It allows users to specify a schema for data resident in the file system hierarchy and query this data using SQL. It supports the creation of indices on these relational views of file data. However, updates to these views via the database are not allowed. Handling complex data types that scientists are used to and synchronizing updates between the operating system and the database is still an open problem. Ideally, the DBMS must allow the integration of 'external' services such as filesystem access with traditional SQL access. This would make it possible to rewrite the more data-intensive portions of a workflow in SQL for better performance.

Becla and Wang [13] present an interesting perspec-

tive on using database techniques for scientific workloads. The authors recognized the importance of concurrency control, scalability, durability and administrative ease in dealing with two terabytes of data produced by 2000 nodes per day in the BaBar high energy physics experiment. An initial OODBMS (Objectivity/DB) based approach was eventually abandoned in favor of a hybrid RDBMS/file-based solution. The authors felt that general purpose solutions are not applicable in dealing with systems of this magnitude, and that some customization is required to tune off-the-shelf data and workflow management products to handle specific scientific computing environments.

Scientific workflows have traditionally dealt with compute-intensive tasks, while most databases are designed to minimize disk I/O. In recent years, research has been done in the optimization of queries with User-Defined Functions [18]. Like scientific tasks, UDFs are usually treated as black-boxes and are often compute-intensive. However, unlike scientific programs, UDFs are used in conjunction with SQL queries and run for seconds instead of hours. Gathering statistics such as job execution time and data access patterns is essential to planning scientific computation. It remains to be seen how such information can be obtained by the database from the execution of scientific programs. Perhaps some combination of user-provided hints and historical information will be necessary.

In conclusion, though the area of scientific data management is still young, we feel that any direction it takes will certainly incorporate database technology in a crucial way. Research in the field must reflect this. Since the contributions database research can make to the scientific realm are quite clear, it is hoped that exploring the relation between the two will open up new and exciting avenues of research for the database community.

References

[1] Biomedical informatics research network. http://www.nbirn.net.

[2] Condor dagman. http://www.cs.wisc.edu/condor/dagman/.

[3] Condor diskrouter. http://www.cs.wisc.edu/condor/diskrouter/.

[4] Condor high throughput computing. http://www.cs.wisc.edu/condor.

[5] Cyberstructure for the geosciences. http://www.geongrid.org.

[6] Grid physics network. http://www.griphyn.org.

[7] Grid physics network in atlas. http://www.usatlas.bnl.gov/computing/grid/griphyn/.

[8] Hawkeye. http://www.cs.wisc.edu/condor/hawkeye/.

[9] Ptolemy ii: Heterogenous modeling and design. http://ptolemy.eecs.berkeley.edu/ptolemyII/.

[10] Sloan digital sky survey. http://www.sdss.org.

[11] S. Abiteboul, V. Vianu, et al. Relational transducers for electronic commerce. In *PODS*, pages 179–187, 1998.

[12] I. Altintas, C. Berkley, E. Jaeger, M. Jones, B. Ludäscher, and S. Mock. Kepler: An extensible system for design and execution of scientific workflows. In *SSDBM*, pages 423–424, 2004.

[13] J. Becla and D. L. Wang. Lessons learned from managing a petabyte. In *CIDR*, pages 70–83, 2005.

[14] J. Bent, D. Thain, A. C. Arpaci-Dusseau, R. H. Arpaci-Dusseau, and M. Livny. Explicit control in the batch-aware distributed file system. In *NSDI*, pages 365–378, 2004.

[15] A. J. Bonner. Workflow, transactions, and datalog. In *Proceedings of the Eighteenth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, 1999*, pages 294–305. ACM Press, 1999.

[16] M. Bote-Lorenzo and E. Dimitriadis, Y.and Gomez-Sanchez. Grid characteristics and uses: a grid definition. In *Proceedings of the First European Across Grids Conference*, pages 291–298, February 2003.

[17] P. Buneman, S. Khanna, and W. C. Tan. Why and where: A characterization of data provenance. In *ICDT*, pages 316–330, 2001.

[18] S. Chaudhuri and K. Shim. Optimization of queries with user-defined predicates. *ACM Trans. Database Syst.*, 24(2):177–228, 1999.

[19] A. L. Chervenak et al. Giggle: a framework for constructing scalable replica location services. In *SC*, pages 1–17, 2002.

[20] R. L. Cole and G. Graefe. Optimization of dynamic query evaluation plans. In *SIGMOD Conference*, pages 150–160, 1994.

[21] E. Deelman, J. Blythe, et al. Pegasus: Mapping scientific workflows onto the grid. In *European Across Grids Conference*, pages 11–20, 2004.

[22] D. J. DeWitt and other. The gamma database machine project. *IEEE Trans. Knowl. Data Eng.*, 2(1):44–62, 1990.

[23] I. T. Foster, J.-S. Vöckler, M. Wilde, and Y. Zhao. Chimera: Avirtual data system for representing, querying, and automating data derivation. In *SSDBM*, pages 37–46, 2002.

[24] J. Gray et al. When database systems meet the grid. In *CIDR*, pages 154–161, 2005.

[25] K. He, S. Dong, L. Zhang, and B. Song. Building grid monitoring system based on globus toolkit: Architecture and implementation. In *CIS*, pages 353–358, 2004.

[26] Y. E. Ioannidis, M. Livny, A. Ailamaki, A. Narayanan, and A. Therber. Zoo: A desktop experiment management environment. In *SIGMOD Conference*, pages 580–583, 1997.

[27] A. Kini, S. Shankar, D. DeWitt, and J. Naughton. Matchmaking in database systems, submitted for publication. Submitted for publication.

[28] T. Kosar and M. Livny. Stork: Making data placement a first class citizen in the grid. In *ICDCS*, pages 342–349, 2004.

[29] D. T. Liu and M. J. Franklin. The design of griddb: A data-centric overlay for the scientific grid. In *VLDB*, pages 600–611, 2004.

[30] G. M. Lohman, C. Mohan, et al. Query processing in R*. In *Query Processing in Database Systems*, pages 31–47. 1985.

[31] N. W. Paton and O. Díaz. Active database systems. *ACM Comput. Surv.*, 31(1):63–103, 1999.

[32] J. Widom. Trio: A system for integrated management of data, accuracy, and lineage. In *CIDR*, pages 262–276, 2005.