Towards a Grid-based DBMS

Craig Thompson
University of Arkansas

In certain high-end data-centric applications, practitioners are discovering that traditional relational database technology is not meeting their requirements for huge data sets, high transaction throughput, cheap storage, and flexible workflow support – so they are switching to grid technology.

But, can we have our cake and eat it too? Do we have to give up relational capabilities to get grid benefits?
What is a Grid?

• A grid is a collection of components with scalable membership that federate to share some resource like computation or data processing.

• Grid computing is way of organizing computing resources so that they can be flexibly and dynamically allocated and accessed
  – Processors, storage, network bandwidth, databases, applications, sensors and so on

• The objective of grid computing is to **SHARE** information and processing capacity so that it can be more efficiently exploited
  – Offer end-to-end QoS guarantees
  – Offer end-to-end security
  – Workflow and resource management
  – Robust operation

• Types of grids
  – Computational grids – reducing execution time
  – Data grids – large scale data management problems
Why Use a Grid?

- Commodity processors
- Cheap storage
- Custom supercomputer are expensive
- Reduce application run-time
- Increased availability
- Dynamic allocation of resources
- Large datasets
- Scalable
- Resource sharing
  - Computers, storage, sensors, networks, …
  - Sharing always conditional: issues of trust, policy, negotiation, payment, …
- Coordinated problem solving
- Dynamic, multi-institutional virtual organizations
Query Layer

Query Tree consists of relational algebra operators like join, project, select but also stored procedure calls

Indexing Layer

Query Execution results in tables

Disk-based Storage Layer

Relational DBMS Architecture

Workflow Layer

Workflow Execution results in data sources

Workflow $w_i$

Data Grid Architecture
Example workflow

Many data sources
Embarrassingly parallel

Very like a query
(input is relations, output is relations)
Storage Level Interface

Requirements
High Throughput and Performance
Initial Partitioning
Dynamic Re-partitioning
Scalability
Load Balancing
Failover

Can be implemented via disk or via an in-memory distributed database or a hybrid
Comparing DBMS and Grid-based Workflow Architectures

Similarities between the two architectures

• A query operates on relations, that is, tables – similar to flat files except that they are stored within the DBMS and there might already exist various primary and secondary indices that organized them. A workflow operates on data sources that might come in many varieties, from flat file to relation to excel spreadsheet to XML. But some of the files may be pre-processed to provide indexing. So the grid architecture could be viewed as a straightforward generalization of the relational architecture … especially if it were possible to attach new relations to a relational DBMS via a plug-in capability.

• Both a query and a workflow consist of operators that transform input sets of records into output sets. SQL queries are translated into query plans and then optimized into query execution plans. Workflows are often manually developed but this process could be automated. Both map input data source formats into output data source formats, so they play a similar role.

• In both cases, there is provision for persistent storage, at least for the life of an application living in the database or on the grid.

• Indexing on a grid and in a database also play similar roles – the indexes may or may not preserve order of records depending on the needs of applications.

• At execution time, each workflow including at the data node level could be scheduled to operate in a data flow/demand driven way. That is, there is a similar need to manage concurrent access if two workflows or SQL commands needed to update a common data resource at the same time.

• Data mining and visualization tiers could be built on top of the workflow or the query layer.

Apparent differences between the two architectures

• It could be argued that the biggest obvious difference is that the backend storage system of the grid is PC and main memory based whereas the similar function for a relational DBMS is supplied by disk technology. But - we could view these as different storage level implementations. After all, the three level architecture of DBMS systems is intended to accommodate changes in one layer without affecting other layers, a use of encapsulation.

• There may be many directors supplying various data nodes and all nodes are operating in parallel. But this might be a nice addition to relational DBMS indices. Also, in the grid, data can be scheduled using data flow technology so that some operations on a later operator are started before all operations on an earlier operator are complete. But this could happen in a relational DBMS as well.

• A relational DBMS does not expose interfaces to the query tree or query execution plan. Experienced users can tune a relational DBMS by creating indices but they cannot usually define new kinds of indices. In contrast, Grid-Workflow systems may expose the workflow layer explicitly (since there is no standardized correlate to a Workflow Calculus language similar to SQL for queries). It is not clear how much Grid-Workflow applications need to actually develop new indices if a reasonable base library were developed.
Implications

• Are there benefits from noticing that the two architectures are variants of a more general architecture?
  – Perhaps high level operators from relational DBMS will be useful in the workflow grid. For instance, the Grant command is used in relational systems to grant rights to users on entities like relations and views. A similar verb would be useful in workflows to provide access control on workflow data sources. Similarly, integrity constraints from relational DBMS might be used to state that No records for minors will be processed. Using automated query modification techniques, such constraints could automatically be laced into workflows to assure they meet security, privacy, or other constraints.
  – It is a useful generalization for database systems to be able to operate on more general data sources and to be able to extensibly add new operators and index types. It is potentially valuable even for relational DBMS architectures to expose the storage level API, so new data structures that meet the API can be viewed as new implementations of storage engine constructs, leading to the idea of plug-in index types or even plug-in storage engines.

• Having well-defined interfaces between layers might make it much easier to extend functionality at one layer without disturbing layers above or below. One important case to consider – could the indexing layer operate at times over disk-based storage and at other times over grid-based storage. That is, can we generalize these two architectures into one which supports both kinds of storage.

• If the analogies between Relational DBMS architectures and Grid-based Workflow architectures are strong, we might well find that a generalized architecture that covers both might provide the best of both worlds. Another way to reason is, if architecture A has capability C and A is similar to architecture B, then might it be likely that, as B evolves, B will need C as well? We can perhaps reuse the design for C in B. But, a corollary is – if B does not need C then maybe it should be optional in the design of A. [The idea of optional composition would benefit many software projects. Very often, we wish we had a system like S but with or without some capabilities. Our choices are to adopt S and live with more capabilities than we need and either have to license S if it is proprietary (in which case it will be harder to extend since we do not own it) or we adopt open source and have to take on maintenance of the adapted system. Of course, since we do not quite know how to build composable software, we live with this dilemma.]
Conclusions

• A collection of architectural views helps direct development towards an end result more quickly and also provides a platform for reasoning about possible extensions or evolutions of a system.
  – For instance, how can you assure that a system is end-to-end secure or provides end-to-end performance guarantees.
  – An architecture can also help an entire organization or community communicate or divide up the work of developing and maintaining subsystems.

• We considered Relational DBMS and Grid-based Workflow architectures. They are similar – so we can learn from one to benefit the other.
  – Implementing a workflow grid is similar in complexity to that of a relational DBMS.
  – Some organizations will focus on the technical architecture and sell that capability – others will focus on developing such an architecture because they need one to process data to insure commercial success.