2015

Desiderata for a Big Data Language

David Maier
Portland State University

Desiderata for a Big Data Language

David Maier
Portland State University
PO Box 751
Portland, OR 97207-0751
+1 (503) 725-2406
maier@cs.pdx.edu

ABSTRACT

Data management and analytics systems for big data have proliferated, including column stores, array databases, graph-analysis environments and linear-algebra packages. This burgeoning of systems has lead to a surfeit of language and APIs. It is time to consider a new framework that can span these systems and simplify the programming and maintenance of Big Data applications. There are two key goals for such a framework:

Portability: It should be relatively easy to move an application or tool developed on one platform to operate against another. As a corollary, back-end data and analytics services should be swappable in a particular platform.

Multi-Server Applications: It will be more common than not that a given application will need the services of multiple systems. The framework should make is easy to construct and deploy such applications.

Such an organizing framework needs a central abstraction to facilitate communication between front-end clients and back end services. LINQ (Language Integrated Query) provides an example of such a framework, albeit for a narrower class of structures and operations. In LINQ, the central abstraction is the Standard Query Operator (SQO) API, which defines a collection of functions on ordered collections such as `Select()`, `Join()` and `Reverse()`. LINQ clients deliver queries as expressions over these operators, and servers (called *LINQ Providers*) accept SQO as expressions as input. There are a wide range of Providers, spanning diverse data types, such as SQLServer, LDAP, XML and RDF. LINQ has many beneficial properties, including:

- It is algebra at the core. The semantics of the SQO API is much easier to understand than a surface-language specification such as SQL. (However, client languages are free to provide syntactic sugar to provide a more declarative specification of queries.)
- It can pass queries to Providers in the form of an expression tree, rather than as a series of remote function calls. This capability obviously cuts down on communication between client and Provider, but also permits optimization and query planning at the Provider.
- The result of a query is a collection in the client environment. There is not the awkwardness of cursors.

We believe a LINQ-like approach is viable for a Big Data organizing framework. However, there need to be extensions and adjustments.

- The data model and operations must be more expressive. In particular, they should include multi-dimensional arrays and operations upon them.
- There should be support for "control iteration". Data algebras rightly encapsulate "data iteration," but many areas, such as graph analytics and data mining, require repeated execution of an expression until some convergence criterion is met.
- Multi-server queries. It should be easy to evaluate a query over a combination of data and analytics servers (such as SciDB and ScaLAPACK), without routing intermediate results through the application level.

Thus, I advocate an algebraic intermediate form as the nexus for a multi-server Big Data framework. A possible start would be a fusion of tabular and array models, with 0 or more attributes in a table structure being tagged as dimensions, and operators being dimension-aware.

How will we judge various options for such an algebra? There are at least four desiderata that should be met.

1. *Coverage*: Big Data algebra should express the operations commonly requested of data and analysis servers. It should at least span standard relational and array operations.
2. *Translatability*: Every algebra operator should be translatable to a back-end system (or a combination of such systems).
3. *Intent Preservation*: The mapping from client APIs and languages into Big Data algebra should not obscure the original intent of an expression. For example, if the original function is matrix multiply, it should be recognizable as such at a server that has a direct implementation of matrix multiply.
4. *Server Interoperation*: An algebra query that spans servers should be realizable as a plan where intermediate results pass directly between servers, rather than being routed through the application or a middle tier.

7th Biennial Conference on Innovative Data Systems Research (CIDR '15) January 4-7, 2015, Asilomar, California, USA.