XG: A Grid-Enabled Query Processing Engine

Radu Sion[1], Ramesh Natarajan[2], Inderpal Narang[3], and Thomas Phan[3]

[1] Stony Brook University
sion@cs.stonybrook.edu
[2] IBM TJ Watson Research Lab
phantom@us.ibm.com
[3] IBM Almaden Research Lab
narang@us.ibm.com
[4] IBM Almaden Research Lab
phantom@us.ibm.com

Abstract. In [12] we introduce a novel architecture for data processing, based on a functional fusion between a data and a computation layer. In this demo we show how this architecture is leveraged to offer significant speedups for data processing jobs such as data analysis and mining over large data sets.

One novel contribution of our solution is its data-driven approach. The computation infrastructure is *controlled from within the data layer*. Grid compute job submission events are based within the query processor on the DBMS side and in effect controlled by the data processing job to be performed. This allows the early deployment of on-the-fly data aggregation techniques, minimizing the amount of data to be transfered to/from compute nodes and is in stark contrast to existing Grid solutions that interact with data layers as external (mainly) "storage" components. By integrating scheduling intelligence in the data layer itself we show that it is possible to provide a close to optimal solution to the more general grid trade-off between required data replication costs and computation speed-up benefits. We validate this in a scenario derived from a real business deployment, involving financial customer profiling using common types of data analytics.

1 Demonstration Outline

In this demo we will show how integrating a computation grid with a data layer results in significant execution speedups. For example, with only 12 non-dedicated nodes, a speedup of approximately 1000% can be attained, in a scenario involving complex linear regression analysis data mining computations for commercial customer profiling. In this demo we deploy XG with live connections (on-site demo only also possible) to our 70+ nodes CPU Grid located in IBM Almaden. We then demonstrate the data processing scenarios discussed below, including the mining query execution speedup shown in Figure 2. Additionally, we propose to demonstrate some of the more subtle features of our system, opening insights into important underlying design decisions. These features include: (i) the runtime mechanism for on-demand QoS provisioning of data replication and computation (provisioning the right amount of resources per data processing task to meet QoS demands, e.g., deadlines), (ii) the on-the-fly recovery on failure in both computation and data layers, (iii) the automatic discovery of computation resources, (iv) the runtime monitoring of computation and data flows between the data layer and the grid. This demonstration is facilitated by the fact that these features are exposed through a user-friendly GUI.

2 Introduction

As increasingly fast networks connect vast numbers of cheaper computation and storage resources, the promise of “grids” as paradigms of optimized, heterogeneous resource sharing across boundaries [2], becomes closer to full realization. It already delivered significant successes in projects such as the Grid Physics Network (GriPhyN) [4] and the Particle Physics Data Grid (PPDG) [10]. While these examples are mostly specialized scientific applications, involving lengthy processing of massive data sets (usually files), projects such as Condor [1] and Globus [3] aim at exploring “computational grids” from a declared more main-stream perspective.

There are two aspects of processing in such frameworks. On the one hand, we find the computation resource allocation aspect (“computational grid”). On the other hand however data accessibility and associated placement issues are also naturally paramount (“data grid”). Responses to these important data grid challenges include high performance file sharing techniques, file-systems and protocols such as GridFTP, the Globus Replica Catalog and Management tools in Globus, NeST, Chirp, BAD-FS , STORK , Parrot , Kangaroo and DiskRouter in Condor. The ultimate goal of grids is (arguably) an increasingly optimized use of existing compute resources and an associated increase of end-to-end processing quality (e.g. lower execution times). Intuitively, a tighter integration of the two grid aspects (“computational” and “data”) could yield significant advantages e.g., due to the potential for optimized, faster access to data, decreasing overall execution times. There are significant challenges to such an integration, including the minimization of data transfer costs by performing initial data-reducing aggregation, placement scheduling for massive data and fast-changing access patterns, data consistency and freshness.

In this work we propose, analyze and experimentally validate a novel integrated data-driven grid-infrastructure in a data mining framework. Computation jobs can now be formulated, provisioned and transparently scheduled from within the database query layer to the background compute Grid. Such jobs include e.g., the computation of analytical functions over a data subset at the end of which the result is returned back in a data layer (either by reference to a specific location or inline, as a result of the job execution). We designed and built a specialized experimental grid infrastructure. One of the main design insights behind it is that (arguably) any “global” grid is ultimately composed of clustered resources at the “edge”. It is then only natural to represent it as a hierarchy of computation clusters and associated close-proximity data sources. Using our end-to-end solution (data-layer aggregation and compute grid invocation), in our considered application domain (data analysis for predictive modeling) significant speed-ups have been achieved versus the traditional case of data-layer processing.

Scenario: Customer Analytics. Let us now explore an important commonly encountered operation scenario for data mining in a commercial framework that yielded significant cost and speed-up benefits from our solution: a large company (i.e., with a customer base of millions of customers), maintains an active customer transaction database and deploys data mining to better customize and/or optimize its customer-interaction response and associated costs. There are two types of customer interactions, each subject to different types of requirements and response mechanisms, namely (i) incoming ("pull" model) inquiries and (ii) outgoing advertisements ("push" model). For space reasons, here we are discussing (i).

Incoming inquiries (e.g., over the phone, online) are handled in a real-time or short-notice manner. There are bounds on response-time (e.g., Human-Computer interaction experiences should feature response times of under 7-8 seconds to be acceptable) [11] to be satisfied. An imprecise but fast initial "pre"-response might be often preferable to an exact but slow one, as it is likely that higher waiting-times would result in a drop in overall customer satisfaction.

The company's response data is based on previously recorded customer "profiles", composed of a history of transactions and a set of related predictive models. Such profiles need to be maintained with sufficient (preferably maximal) accuracy and the associated (predictive) models re- computed periodically or as part of an event- driven paradigm in which associate customer events trigger individual model re- computations.

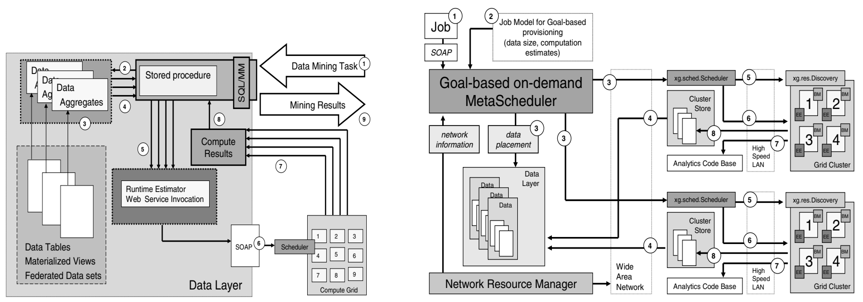
In such an interaction, often the center-point (and likely the most expensive) is processing a function of the immediate input data and the customer predictive models in the stored profile ("model scoring"). Often, also, new models need to be computed on the fly. Because of its real-time nature, and the potential for thousands of simultaneous incoming customer requests, this scenario is extremely challenging.

To understand the size of this problem, let us quantify some of the previous statements with real data. Let us assume a customer base of over 10 million customers. Roughly 0.1% (10k) of them are active at any point in time (interactive and automated phone calls, web access, other automated systems). Preferably, the company response in each and every transaction should be optimally tailored (i.e., through on-demand data mining) to maximize profit and customer satisfaction. On average, only 75% (7.5k) of these active (meta)transactions are resulting in actual data mining tasks and, for each second, only 20% of these task- triggering customers require data mining. To function within the required response-behavior boundary, the company has to thus handle a continuous parallel throughput of 1500 (possibly complex) simultaneous data mining jobs. Achieving this throughput at the computation and data I/O level is very challenging from both a cost and scalability viewpoint.

3 System Architecture

The end-to-end solution comprises several major components: modeling, aggregation and computation outsourcing (in the data layer) and grid scheduling and management (grid layer).

Data Layer. Designing specifically for data mining over large data sets, requires a careful consideration of network data transfer overheads. As traditional data mining solutions are often based on code directly executed inside the database query processor, these overheads could often be reduced by an initial data aggregation step performed inside the data layer, before outsourcing the more computation heavy model generation tasks.



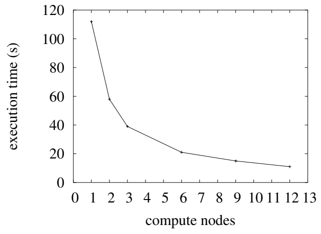
(a) (b)

Fig. 1. (a) Data Layer Overview: The grid is leveraged transparently from the data side. Mining tasks can execute normally within the query processor (e.g., as stored procedures). (b) Both data replication/placement (to cluster stores) and job scheduling in the hierarchical grid infra-structure is controlled by a meta scheduler.

The solution allows for dispatching of multiple simultaneous data processing tasks from within the query processor (i.e. SQL level) by performing simple calls through a user defined function (UDF) mechanism. At the completion of these tasks, their results become available within the actual data layer, ready for further processing. The interaction between the data layer and compute grid is composed of two elements: job submission control and data placement/replication. Job submission is initiated in the database and forwarded to the main computation grid through a webservice interface exposing the main grid scheduling control knobs. This interaction is enabled by user defined functions (UDF) within DB2, (constructs present also in a majority of big-vendor DBMS solutions including DB2 [6], Oracle [9] and SQL Server [8]).

The grid scheduler controls are exposed through a webservice interface. This is achieved through the XML Extender [7] and its SOAP messaging capabilities which allows the invocation of job submission methods exposed by the schedulers in the compute grid layer (see Figure 1 (a)). The invocation is asynchronous so as to not block the calling thread and to allow actual parallelism in data processing. While extensive details are out of the current scope, for illustration purposes, let us discuss here the query in Figure 2 (a).

After the initial aggregation step (performed by $agg()$) the resulting computations are outsourced to the grid (grouped by customer, c_id) through the $analysis\_grid()$



```
SELECT   c_id,   analysis_grid(agg(c_assets))
FROM   customer_tx
WHERE   c_age   <  21
GROUP   BY  c_id
```

Fig. 2. (a) With increasing number of computation nodes, query execution time decreases. (b) Sample Query.

UDF. This constructs the necessary SOAP envelopes, converts the arguments to a serializable format and invokes the grid scheduler with two parameters for each customer: (i) an URL reference to the external regression analysis code (located in the grid code-base, see Figure 1 (b)) and (ii) a reference to the aggregate input data.

There are two alternatives for data transfer to/from the compute grid: *inline* (as an actual parameter in the job submission – suitable for small amounts of input data and close local clusters) and *by-reference* where actual input data sources are identified as part of the job submission – suitable for massive data processing in a global grid. To support the input/output data by reference paradigm, in our design data replication is activated by the meta-scheduler at the grid cluster level, leveraging "close" data stores and linking in with data replication/placement mechanisms (Information Integration [5]) if the data is "far" (see Figure 1 (b)).

Computation Layer. XG is a grid management solution designed for tight data-layer integration. It enables a hierarchical grid structure of individual fast(er)-bus compute clusters (at the extreme just a single machine). This design derived from the insight that (arguably) a majority of grid infra-structures are to be composed of multiple high-speed cluster networks linked by lower speed inter-networks. Designing an awareness of this clustering structure in the actual grid allows for location-based scheduling and associated data integration and replication algorithms.

The grid hierarchy is supported by the concept of a meta-scheduler, (Figure 1 (b)) a software entity able to control a set of other individual (meta)schedulers in a hierarchical fashion, composing a multi-clustered architecture. Job submission at any entry-point in this hierarchy results in an execution in the corresponding connected subtree (or sub-graph). At the cluster level a scheduler is managing a set of *computation nodes*. A node is composed (among others) of an Execution Engine and an Execution Monitor.

The scheduler deploys a discovery protocol for automatic discovery of available compute resources, a polling mechanism for monitoring job progress and notifications for job rescheduling (e.g., in case of failures) and a scheduling algorithm for job scheduling. The scheduling algorithm is designed as a plugin within the scheduler, allowing for different scheduling policies to be hot- swapped. For inter-operability, at all levels, the schedulers are designed be invoked (e.g. for job scheduling) through a web-service interface. This interface allows for job submission (with both inline and by-reference data), job monitoring and result retrieval among others. More details are out of scope here.

Results. Our experimental test-bed consists of a grid cluster composed of 70 general purpose 1.2GHz Linux boxes with approximately 256MB of RAM each. The data layer deploys IBM DB2 ver. 8.2. with the XML Extender [7] enabled. We evaluated the actual speed-ups of the model generation process with increasing number of grid nodes. Figure 2 (b) illustrates a data mining scenario (for 100 customer profiling jobs) for which execution time went down from roughly 112 seconds for one compute node, to about 11 seconds when 12 nodes where deployed. The deployed code in both the data layer and the grid remained the same. What changed was just the availability of new computation power on the grid side.

4 Conclusions

In this work we introduced a novel architecture for data processing, a functional fusion between a data and a computation layer. We then experimentally showed how our solution can be leveraged for significant benefits in data processing jobs such as data analysis and mining over large data sets.

There are significant open avenues for future research including: the integration of a message passing interface solution in the compute grid, more complex failure recovery augmented by check-pointing, the implementation of privacy-preserving primitives for data processing, the exploration of both data placement and computation scheduling as first class citizens in resource scheduling. This becomes especially relevant in on-demand environments where a maximal throughput in data and compute intensive application is not possible using current manual data partitioning and staging methods. Last but not least, we believe *grid-aware query processing* to be an exciting avenue for future research, ultimately resulting in a computation aware grid query optimizer within a traditional DBMS query processor.

References

1. The Condor Project. Online at `http://www.cs.wisc.edu/condor`.
2. The Global Grid Forum. Online at `http://www.gridforum.org`.
3. The Globus Alliance. Online at `http://www.globus.org`.
4. The Grid Physics Network. Online at `http://www.griphyn.org`.
5. The IBM DB2 Information Integrator. Online at `http://www.ibm.com/software/data/integration`.
6. The IBM DB2 Universal Database. Online at `http://www.ibm.com/software/data/db2`.
7. The IBM DB2 XML Extender. Online at `http://www.ibm.com/software/data/db2/extenders/xmlext`.
8. The Microsoft SQL Server. Online at `http://www.microsoft.com/sql`.
9. The Oracle Database. Online at `http://www.oracle.com/database`.
10. The Particle Physics Data Grid. Online at `http://www.ppdg.net`.
11. Julie Ratner. *Human Factors and Web Development, Second Edition*. Lawrence Erlbaum Associates, 2002.
12. Radu Sion, Ramesh Natarajan, Inderpal Narang, Wen-Syan Li, and Thomas Phan. XG: A Data-driven Computation Grid for Enterprise-Scale Mining. In *Proceedings of the International Conference on Database and Expert Systems Applications DEXA*, 2005.