XG: A Data-driven Computation Grid for Enterprise-Scale Mining.

Radu Sion[1], Ramesh Natarajan[2], Inderpal Narang[3], Wen-Syan Li[3], and Thomas Phan[3]

[1] Computer Sciences, Stony Brook University, Stony Brook, NY 11794,
sion@cs.stonybrook.edu
[2] IBM TJ Watson Research Lab, Yorktown Heights, NY 10598,
nramesh@us.ibm.com
[3] IBM Almaden Research Lab, 650 Harry Rd, San Jose, CA 95120,
{narang,wsl }@us.ibm.com

Abstract. In this paper we introduce a novel architecture for data processing, based on a functional fusion between a data and a computation layer. We show how such an architecture can be 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 (e.g., linear regression analysis).
Experimental results show significant speedups. For example, using a grid of only 12 non-dedicated nodes, we observed a speedup of approximately 1000% in a scenario involving complex linear regression analysis data mining computations for commercial customer profiling.

1 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 [6], becomes closer to full realization. It already delivered significant successes in projects such as the Grid Physics Network (GriPhyN) [9] and the Particle Physics Data Grid (PPDG) [15]. While these examples are mostly specialized scientific applications, involving lengthy processing of massive data sets (usually files), projects such as Condor [5] and Globus [7] 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 [8] in Globus, NeST, Chirp, BAD-FS [16] , STORK [4] , Parrot [3] , Kangaroo [2] and DiskRouter [1] 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, increasing associated Quality of Service metrics. 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, the ability to directly handle 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).

To do so, we designed and implemented a light-weight, minimum overhead grid scheduling and management software, a proof of concept implementation of the data-driven computation scheduling paradigm. The tightly data-layer integrated design presented significant new foundational and implementation challenges in both the data and computational aspects. On the one hand, apparently, the manipulation of large data sets *from within the data layer* becomes easier. While this is true in principle, it does not come without additional problems to be tackled. Massive parallel data access patterns have the natural potential to result in data processing bottlenecks. This requires mechanisms for automatic smart data placement and replication as part of computation staging. On the other hand, equally challenging we find the (portable) integration of computation scheduling awareness in a traditional data-layer query engine, without the requirement of significant alterations to its core. We achieved this by operating within the boundaries of traditional SQL by providing a set of functional extensions allowing for computation formulation and external grid scheduling. This was important for two reasons: (i) seamless portability with other DBMS (e.g. MySQL), (ii) backwards compatibility with existing applications requiring no major code rewriting to benefit from the grid infrastructure.

Another main design insight behind our implementation is that (arguably) any global grid is ultimately composed of clustered resources at its 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 model-

ing) significant speed-ups have been achieved versus the traditional case of data-layer processing.

Using a grid of only 12 non-dedicated nodes, we observed a speedup of approximately 1000% in a scenario involving complex linear regression analysis data mining computations for commercial customer profiling.

Thus, the main contributions of this work include: (i) the proposal, design and implementation of a novel paradigm for grid scheduling from within a relational DBMS data-layer, allowing for speedups to be of impact at the actual relational algebra level, (ii) the use of data-aggregation techniques minimizing transfer overhead and optimizing the trade-off between required data migration costs and actual speed-up benefits, (iii) the design and implementation of a supporting grid management solution and (iv) the experimental evaluation thereof in a commercial customer profiling data analysis scenario.

The paper is structured as follows. Section 2 introduces our main commercial use-case scenario and explores some of the associated data analytics. Section 3 introduces the main solution. Section 4 analyzes system performance and discusses experimental results. Section 5 discusses avenues for further exploration and concludes.

2 Scenario: Real-time 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, see Figure 1). For space reasons, here we are discussing (i).

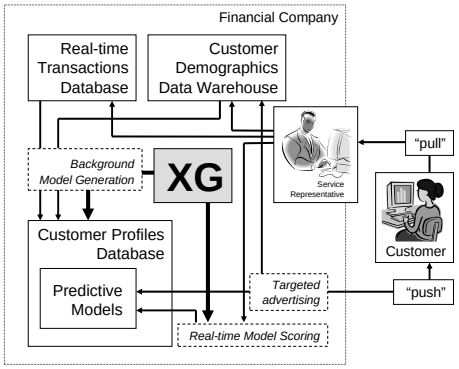

Fig. 1. Push/pull customer interaction scenario.

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) [17] to be satisfied. Due to their customer-initiated nature, these interactions could be (intuitively) quite valuable, thus additional resources and care should be taken in satisfying real-time and quality aspects. 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. Scalability issues need to be carefully considered and sized to the potentially large number of close-timed events. Higher predictive model accuracy can be attained by enabling speedups in the model computation and then performing these more often and with increasing accuracy constraints (more CPU cycles required). Often, the per-customer nature of profiling naturally allows model generation tasks to be out-sourced to a computation grid, thus providing opportunities for scalability.

In an *incoming* 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. It would benefit from an ability to outsource different simultaneous model scoring tasks to a computation grid.

To understand the size of this problem, let us quantify some of the previous statements. To do so we outline an actual deployment case business scenario for which the orders of magnitude of the values have been preserved. In this scenario, incoming customer calls and online system accesses are event-triggering. 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 Proposed Solution

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

3.1 Data Layer Overview

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.

We propose a design that (i) tightly integrates the grid with the data layer and (ii) hides its management complexity by transparently performing behind the scenes. Our 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: (i) a grid-data layer interface and (ii) data placement/replication mechanisms. Due to scope and space constraints here we are detailing (i).

Job submission is initiated in the database and forwarded to the main computation grid through a web-service 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 [11], Oracle [14] and SQL Server [13]).

Present in a majority of big-vendor DBMS solutions including DB2 [11], Oracle [14] and SQL Server [13], UDFs are SQL-extensions that allow interaction with host-language functions for performing customized tasks. The UDF concept presents a host of advantages, including the ability to modularize an application, to provide custom functionality not offered within the database, to reuse and share code etc.

The grid scheduler controls are exposed through a webservice interface. Through the XML Extender [12] and its SOAP messaging capabilities, DB2 provides the ability to create UDFs that interact with webservices. This allows the invocation of job submission methods exposed by the schedulers in the compute grid layer (see Figure 2 (a)).

This invocation is asynchronous so as to not block the calling thread and to allow actual parallelism in data processing. Incidentally, due to DB2's internal query handling, in this particular case, additional advantages are obtained through the use of the GROUP BY clause which can also benefit from potential SMP (symmetric multi-processor) parallelism behind the scenes.

While extensive details are out of the current scope, for illustration purposes, let us discuss here the following query:

```
SELECT c_id, reg_analysis_grid(reg_agg(c_assets))
FROM customer_tx WHERE c_age < 21 GROUP BY c_id
```

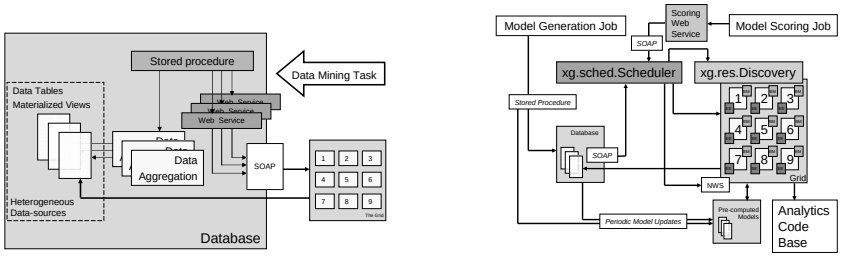

Fig. 2. (a) Data Layer Overview: The grid is leveraged transparently from the data side. Mining tasks can execute normally within the query processor (e.g., as stored procedures). (b) Computation Grid Overview: Single compute grid cluster shown.

After the initial aggregation step (performed by $reg\_agg()$) the resulting computations are outsourced to the grid (grouped by customer, c_id) through the $reg\_analysis\_grid()$ 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 3) 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 [10]) if the data is 'far" (see Figure 3).

3.2 Computation Layer Overview

XG is our experimental grid management solution custom 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 3), 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 (Figure 2 (b)) a scheduler (xg.sched.Scheduler) is managing a set of *computation nodes*. A node is composed (among others) of an Execution Engine (xg.blade.ExecutionEngine) and a monitor (xg.blade.BladeMonitor).

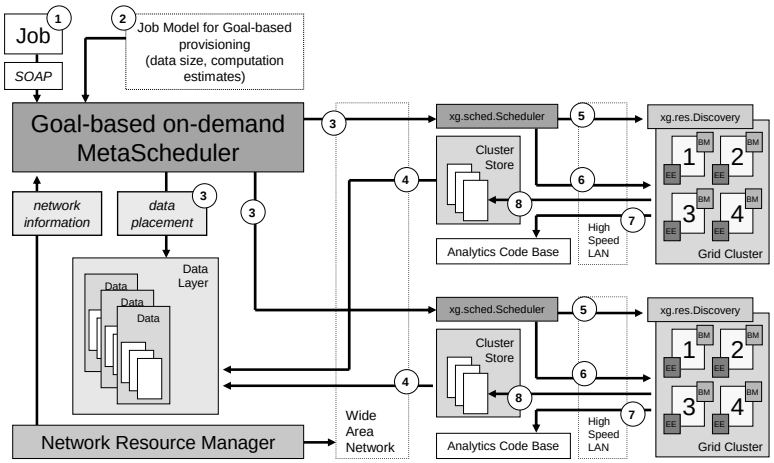

Fig. 3. Both data replication/placement (to cluster stores) and job scheduling in the hierarchical grid infra-structure is controlled by a meta scheduler.

The scheduler deploys a discovery protocol (BladesDiscovery) for automatic discovery of available compute resources, a polling mechanism (BladesPoller) 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, the schedulers are designed be invoked (e.g. for job scheduling) through a web-service interface. It allows for job submission (with both inline and by-reference data), job monitoring and result retrieval when the persistence of results was requested in the job submission step.

4 Experimental Results

We performed experiments on a grid cluster composed of 70 general purpose 1.2GHz Linux boxes with approximately 256MB of RAM each. The data layer used deployed IBM DB2 ver. 8.2. with the XML Extender [12] enabled.

4.1 Linear Regression Model

We implemented code for linear regression modeling. In order to be able to easily assess *actual* job computation times (required for the next experiments), we first evaluated its behavior to varying input size (number of tuples of the input data set).

As can be seen from Figure 4 (a) a naturally linear dependency was observed. The proof-of-concept un-optimized code can handle 73k tuples/sec in the considered setup. We estimate $1-2$ orders of magnitude speed-up in an optimized industry-level version.

This linearity allows the construction of a natural metric evaluating the amount of computation associated with a certain job. In the following we are going to use the term

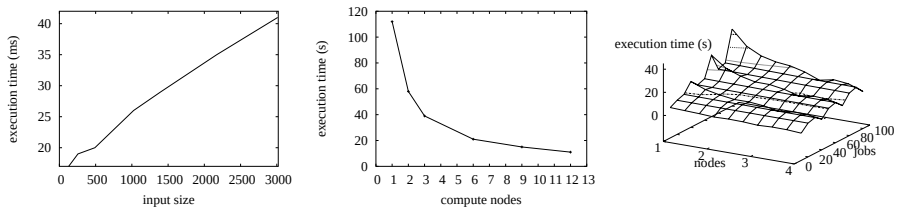

Fig. 4. (a) The considered regression analysis behavior to data input size is virtually linear. (b) Grid speed-up. With increasing number of nodes, the overall execution time decreases naturally. The data points shown are for a job load of 100 jobs with an input size of roughly 90k tuples each. (c) Overall observed execution time with varying number of jobs and grid nodes. Notice the (intuitive) upper-left to lower-right tilt of the surface. The considered input size per job was roughly 30 k tuples.

"job size" to denote the amount of input data corresponding to the considered job(s). This metric naturally identifies the amount of CPU cycles a certain job will require and is arguably independent from the speed of the deployed CPU. Another related metric could be defined by a translation to actual MIPS numbers but we feel this would be out of scope and of no additional benefit in this framework.

4.2 Grid Speed-ups

Once a "job size" metric was established, we proceeded by evaluating the actual speed-ups of the model generation process with increasing number of grid nodes. The mining job load was generated by issuing a GROUP BY query (as described in Section 3.1) that resulted in 100 regression analysis jobs (e.g., one model for each of 100 customers).

In Figure 4 (b) it can be seen that the solution naturally scales. In a scenario with 100 jobs of input size 90k, execution time went down from roughly 112 seconds for one compute node, to about 11 seconds when 12 nodes where deployed.

In Figure 4 (c) the grid speed-up is analyzed from a 3-dimensional perspective. The question answered here is: does the design scale in *both* the number of jobs *and* the number of available compute resources? Noting the upper-left to lower-right tilt of the execution time surface is providing the answer. It basically confirms that as the number of jobs goes up, the implemented scheduling algorithm discovers and utilizes available compute resources properly.

4.3 Overheads

Another question of importance is whether the scheduling and overheads are stable and allow for arbitrary scaling in the number of jobs. In Figure 5 (a) it can be seen that the overall scheduling overhead is linear in the number of jobs, as expected. It averages about 40ms per job in the experimental proof-of-concept Java release. Significant speed-ups can be obtained in an optimized version.

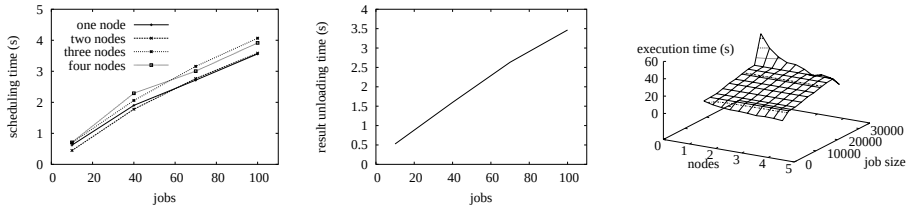

Fig. 5. (a) Overall scheduling overhead is linear in the number of jobs (averaging at about 40 ms per job). There is a very small (increasing) dependency of the number of nodes. The considered input size per job was also roughly 30 k tuples. (b) In the case of inline result transfer, its associated total overhead is (naturally) linear in the number of jobs and, in this scenario averaged 35 ms per job (mainly network costs). The experiment was performed with job input sizes of roughly 30 k tuples. (c) Overall execution time for 100 jobs with varying number of grid nodes and job size ranging from 1500 to 30 k input tuples per job, corresponding to roughly 20 to 400 ms individual times. Observe the natural upper-left to lower-right tilt.

Additionally we observed (see also Figure 5 (a)) a slight, (arguably) insignificant, increase in scheduling times with increasing number of nodes. The causes for this are two fold: (i) resource matching, blade and runtime job monitoring all take additional time, likely directly proportional to the size of the resource pool (i.e., number of nodes in this case) and (ii) we believe a certain "collision" factor (e.g., due to using the same network bus) is also associated with migrating jobs to a larger number of nodes.

Figure 5 (b) shows a result depicting direct inline result transfer overheads for the considered scenario. It can be seen that the behavior is also stable and linear, averaging about 35 ms per job. If the data is passed by reference these overheads are not occurring directly, but are rather hidden in the process of result transfer back to the data sources, directly from the grid to the data layer.

4.4 Scalability

Figure 5 (c) explores whether the solution actually scales for varying job sizes. In other words, what happens if the jobs become so "small" that their execution time is comparable to or even lower than the observed overheads? Does deploying such a solution still make sense? It can be observed that, intuitively, the most benefits are reaped in the case of large jobs. However, the upper-left to lower-right tilt again confirms that overall, the grid scales favorably and execution times are reduced even in the case of multiple small jobs.

5 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. While in this initial effort we dealt with mostly independently executing jobs, increased capabilities and expression power can be achieved by the integration of a message passing interface solution in the compute grid, allowing jobs to communicate and synchronize. Failure recovery is currently based solely on job re-scheduling to different compute nodes. Check-pointing would increase the ability to better deal with large jobs in a failure-prone environment.

Security is of paramount importance. Privacy-preserving primitives for data processing in (possibly hostile, eavesdropping) compute grid-environments should be addressed and deployed. We believe it is essential to take resource scheduling to a new level and treat both data placement and computation scheduling as first class citizens. 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. DiskRouter. Online at http://www.cs.wisc.edu/condor/diskrouter.
2. Kangaroo. Online at http://www.cs.wisc.edu/condor/kangaroo.
3. Parrot. Online at http://www.cs.wisc.edu/condor/parrot.
4. STORK: A Scheduler for Data Placement Activities in the Grid. Online at http://www.cs.wisc.edu/condor/stork.
5. The Condor Project. Online at http://www.cs.wisc.edu/condor.
6. The Global Grid Forum. Online at http://www.gridforum.org.
7. The Globus Alliance. Online at http://www.globus.org.
8. The Globus Data Grid Effort. Online at http://www.globus.org/datagrid.
9. The Grid Physics Network. Online at http://www.griphyn.org.
10. The IBM DB2 Information Integrator. Online at http://www.ibm.com/software/data/integration.
11. The IBM DB2 Universal Database. Online at http://www.ibm.com/software/data/db2.
12. The IBM DB2 XML Extender. Online at http://www.ibm.com/software/data/db2/extenders/xmlext.
13. The Microsoft SQL Server. Online at http://www.microsoft.com/sql.
14. The Oracle Database. Online at http://www.oracle.com/database.
15. The Particle Physics Data Grid. Online at http://www.ppdg.net.
16. John Bent, Douglas Thain, Andrea C. Arpaci-Dusseau, Remzi H. Arpaci-Dusseau, and Miron Livny. Explicit Control in a Batch-Aware Distributed File System. In *Proceedings of the First USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, San Francisco, CA, March 2004.
17. Julie Ratner. *Human Factors and Web Development, Second Edition*. Lawrence Erlbaum Associates, 2002.