Towards a Framework for Knowledge Discovery:

An Architecture for Distributed Inductive Databases

Jeroen S. de Bruin, Joost N. Kok
Universiteit Leiden, Leiden Institute of Advanced Computer Science (LIACS), Niels Bohrweg 1, 2333 CA Leiden, The Netherlands,
jdebruin@liacs.nl

Abstract. We discuss how data mining, patternbases and databases can be integrated into inductive databases, which make data mining an inductive query process. We propose a software architecture for such inductive databases, and extend this architecture to support the clustering of inductive databases and to make them suitable for data mining on the grid.

1 Introduction

The size and variety of machine-readable data sets have increased dramatically and the problem of *data explosion* has become apparent. Scientific disciplines are starting to assemble primary source data for use by researchers and are assembling data grids for the management of data collections. The data are typically organized into collections that are distributed across multiple administration domains and are stored on heterogeneous storage systems. Recent developments in computing have provided the basic infrastructure for fast data access as well as many advanced computational methods for extracting patterns from large quantities of data.

These collections provide excellent opportunities for *data mining*. Data mining refers to the process of analyzing data in databases hoping to find patterns that are novel, interesting, and useful. In a way it is comparable to statistics since it uses techniques based on statistics, but takes it a bit further in the sense that where statistics aims at validating given hypotheses, in data mining often millions of potential patterns are generated and tested, aiming at potentially finding some that are prove to be useful. This is however a much more computationally intensive process. Examples of well-known data mining techniques are discovery of association rules (which properties of individuals are typically associated with each other?); building predictive models (decision trees, rules, neural nets) that can be used to predict unknown properties of individuals; building probabilistic models that summarize the statistical properties of a database, etc.

The enormous amount of data generated from scientific experiments, together with the developments in data mining and data warehousing, have led to a paradigm-shift in scientific research from hypothesis-driven science to discovery-driven science. No longer need experiments be conducted in a hypothesis-driven fashion, where the experimenter has an idea, and tries to validate by experimenting. Rather, the trend is to collect as much data as possible on a specific function or system, look for emerging *patterns*, interpret those patterns, and relate them to the current knowledge.

The *challenge* is to provide a persistent and consistent environment for the discovering, storing, organizing, maintaining, analyzing *patterns*, possibly across *distributed* environments.

2 Inductive Databases

With respect to such pattern bases, the important question arises how the existing methods and algorithms can be elegantly integrated into current database management systems. In order to meet this reqirement, Imielinsky and Mannila proposed the concept of so-called inductive databases [3]. In an inductive database it is possible to get answers about the collected data in the database as well as answers to questions about inductively gathered "knowledge" in the form of patterns concerning that data.

Inductive databases have been studied extensively within the European cInQ project and also play a central role in its recently founded successor IQ (acronym for: Inductive Queries for Mining Patterns and Models, EU IST-FET). In order to efficiently and effectively deal with patterns, researchers from scientific domains would greatly benefit from adopting a Pattern-Base Management System (PBMS) in which patterns are made first-class citizens. This provides the researcher with a meaningful abstraction of the data.

The general idea is to modify existing databases to support efficient pattern storage, and extend databases with an implementation of an inductive query language and in this manner transforming a DataBase Management System (DBMS) into a DataBase Knowledge Discovery System (DBKDS). Since inductive databases provide architecture for pattern discovery as well as a means to discover and use those patterns through the inductive query language, data mining becomes in essence an interactive querying process. Some of these queries, however, will not be efficient despite query optimizations. Therefore, some data mining primitives must be built into the database system itself, and must serve as primitive functions within the inductive query language.

The efficiency of the data mining process also depends on the way that data is represented within the database, so a compromise must be made between efficient storage and efficient discovery. Since a gigabyte is becoming cheaper and cheaper every day, we are inclined to prioritize a representation that facilitates efficient discovery over efficient storage.

Over the past few years much research has been done on (efficient) pattern representation and pattern storage issues [2, 5]. The studies in the PANDA project (http://dke.cti.gr/panda/) have shown that the relational way of storing patterns is very inefficient, and proves to be too rigid to efficiently and effectively store patterns, since patterns often have a more semi-structured nature. To be able to support a wide variety of patterns and pattern classes, XML or variations have been explored and the results were encouraging. [6, 7]

3 Distributed Knowledge Discovery

Over the last few years grid computing – the use of the memory and/or processing resources of many computers connected with each other by a network to solve computation problems – has received much attention, and not without reason. The research community - universities, academic research institutes, and industrial research laboratories - is becoming ever more dependent on previous research outcomes from third parties. The complexity of modern experiments, usually requiring the combination of heterogeneous data from many fields (physics, astronomy, chemistry, biology, medicine), requires multidisciplinary efforts. This implies that this community is becoming increasingly dependent on the quality of the *e-Science infrastructure*. Such an infrastructure allows scientists to collaborate with colleagues world-wide and to perform experiments by utilizing resources of other organizations. A common infrastructure for experimentation also stimulates community building and the dissemination of research results. These developments apply to pure as well as applied sciences, including data mining.

Data used in knowledge discovery is often distributed over a multiple of resources, which in their turn can be spread among several different logical or physical places. Of course, the same can be true for patterns over that data. It is therefore important to see how standard data mining algorithms can be adapted to cope with these distributions to make *data mining on the grid* possible.

The problem stated above can be addressed in two ways. One way is to adapt current mining algorithms to cope with distributed data and pattern sources. Current data mining algorithms usually address problems on a single resource, and require a somewhat rigid structure for the data. Relational mining algorithms, thus mining algorithms specifically developed for relational databases and thus able to work with several tables within such a database could prove to be a good basis for such adaptation.

The second way is through an architecture that supports a distributed environment, allowing the database or patternbase itself to support and internalize remote connections to other databases and patternbases. In this case, the client is unaware of the fact that is mining in fact scheduled and executed on different databases or patternbases, since to the user there appears to be only one location of data storage. It is the task of the database or patternbase itself to keep track of all connections and remote access protocols.

Another advantage of data mining on the grid is the ability to process data mining requests on a location other than the client or the data server(s). This poses some implications on the inductive query language supported by the inductive database, since it must be able to evaluate and segment queries into subqueries that can be simultaneously processed by multiple (distinct and/or remote) processing locations.

To be able to support such remote query processing, it should be addressed and internalized in the architecture of the inductive database itself. The architecture should support load balancing algorithms that are efficient enough to dynamically and continuously check whether or not a (sub)query should be handled locally or be outsourced to another grid node.

4 Knowledge Discovery Architecture

In this section we propose an architecture that addresses the challenges posed in sections 2 and 3. We have based this architectural design on component technology, which is commonly seen as the evolution of object oriented technology. The reason for choosing the component-oriented paradigm lies in the fact that it is well suited for the description of architectures and that it decentralizes the development effort. Especially the last property is important for an inductive database, since a user must be able to outfit the inductive query language with its own custom developed data mining algorithms. Component software i.e. software made out of software components, is designed for extensibility, and therefore the use of component technology seems an obvious one.

The remainder of this section will be used to discuss four different architectural levels that together form the Distributed Inductive Database Architecture. These levels are:

- The Fusion level
- The Query level
- The DataNet level
- The Delegate level

The Fusion level

At the heart of our architecture lies the inductive database. As stated earlier, we view an inductive database as a traditional (relational) database extended with a patternbase. Of course, for the sake of simplicity and security, the user should remain oblivious of the separation between patternbase and database, creating the need for automatic, transparent updating of patterns whenever changes are made in the patternbase or database.

In order to explain the fusion architecture, we first need to specify some constraints on a pattern, and define the relationship between patterns and their related underlying data. A *pattern* is a semantically rich representation of a collection of data. A pattern, as defined in this architecture, has at least the following properties: A unique id or name, the data collection that it applies to, a set of measurements, and the component and function that calculates these measurements.

Now consider the (simplified) architecture in Figure 1, which functions as a fusion component between patternbase and database, thereby using the component catalog of data mining primitives that are supported by the inductive query language. Notice that there is a notifier present in the fusion architecture, namely *Notify_Table_Update*, which is used by the database to signal that a certain table has changed. The fusion component is a passive component that listens for update notifications, and remains idle if there are none.

To illustrate what happens when an update notification is received, let us consider the case that the data in a set of tables *T* has changed. In this case, the database sends a notification to the Fusion component specifying *T*. The Fusion component then requests the pattern collection *P*, which consists of all patterns in the patternbase that have $t \in T$ in their data collection specification. At the same time, the Fusion component requests all the data from *T*, and specifies both sets *P* and *T* in a call to a component in the Data Mining Primitives catalog. It knows exactly what components and routines to call, since this is part of the pattern's specification. After the calculations have been completed, the Fusion component returns the updated patterns to the patternbase via the *Update_Patterns* interface.

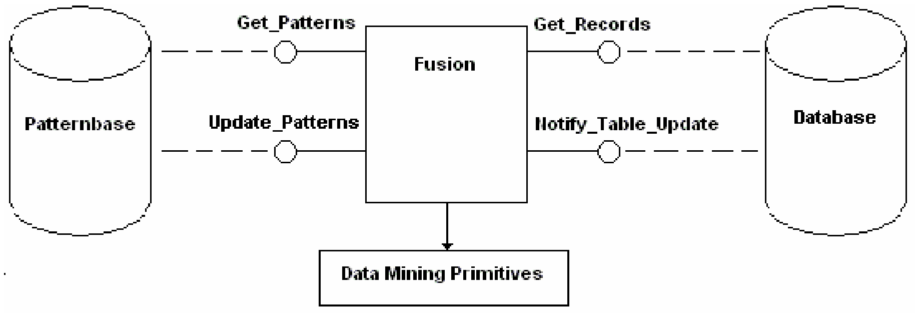

Figure 1: The Fusion Level

The situation described above shows how updating of patterns can be done in an automatic, transparent way without interference from the client.

The Query level

The Query architecture addresses the issues related to the inductive query language built on top of the inductive database. Currently, a number of inductive query languages that have been implemented, i.e. MSQL [4] and MINE RULE [1]. For now, we will not address their individual properties, advantages and disadvantages, nor the effects they might have on the proposed architecture. Instead, we will present a global architectural framework, where the choice of query language could be considered an implementation detail that becomes an issue when the architecture becomes a blueprint for a more application-specific design as part of the product family that the architecture constitutes.

As is illustrated in Figure 2, this architecture provides a layered approach to the analysis, optimization and execution of a query. This is pretty straightforward if the query concerns only patterns or only data, but it becomes complex if both are addressed in the same query, via so called crossover operations.

The top layer of the architecture consists of the query parser, which parses the entire query and assigns meaning to each identifier in the query string. It subsequently passes these tokens on to the query analyzer and the query optimizer. Both operate closely together, and have an almost synergetic relationship: The query analyzer checks what parts of the query can be handled in parallel or can be handled more efficiently, and passes this on to the optimizer. The optimizer, in turn, optimizes these queries, which may yield in a new set of (sub)queries, which are passed back to the query analyzer for further analysis, until no more optimizations are possible. We are fortunate that lately inductive query decomposition and optimizing such (sub)queries have been given a great deal of attention [8] [9].

After the query has been analyzed and optimized, it is passed on to the query scheduler, which schedules the subqueries for execution, either locally or remotely. The query scheduler is also responsible for parallel or sequential scheduling of queries.

At the lowest level of the Query architecture the query executor is found. This layer is responsible for delegating the queries to the correct data storage facility at the fusion level and performing the correct data mining primitives on them, such as crossover operations or other data mining operations. After the queries have been executed their individual results are passed back to the query scheduler, which passes the final result to the client.

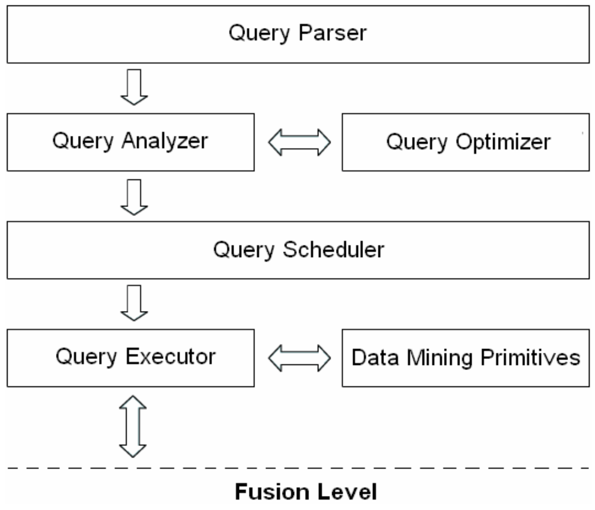

Figure 2: The Query Level

The DataNet level

Up to this point we have modeled a database and a patternbase as a single entity. However, we mentioned earlier that data mining over multiple data sources could be accomplished in an architectural manner by having databases keep track of remote connections with other databases. The architecture we foresee here is based on a peer2peer network architectural style; each data server also functions as a router in a private data net.

The use of an internal data net could have many advantages. For the user, it appears that there is only one database that they can query, and depending on the user permissions, the database dispatches his or her requests to the right database or patternbase in the data net, resulting in a greater ease of use and increased security.

There are some drawbacks to this method as well, especially in the area of overhead management. For each user of the database his or her status must be checked for access rights, operational rights, etc. It is a huge challenge to solve this in an efficient and scalable way.

The DataNet architecture in Figure 3 addresses some of the issues and functionalities discussed above. Each database (or patternbase) contains three catalogs: the Data Catalog, which is used by the database to specify which tables it contains; the User Catalog, which specifies which users or user groups have access to (part of) its content; and finally, the Connection catalog, which contains (private) information on all the other databases that are part of the data net. The DataNet architecture is built on top of the Query architecture, but the separation is not strict: After all, the query analyzer and scheduler check whether or not the data queried is available in the database. If not, the subquery addressing that data will be forwarded by the query scheduler via the DataNet level to the database that does contain the data, which can be found out by broadcasting a request on all data catalogs in the data net.

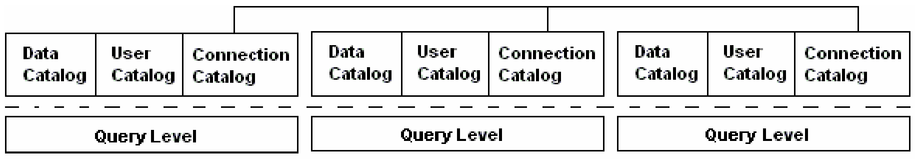

Figure 3: The DataNet Level

The Delegate level

The Delegate Architecture is the part of the architecture that interacts with the data grid to which the database is possibly attached. This architectural part concerns itself with intelligent structures and components that perform load-balancing algorithms, remote method invocations and procedure migration to processing nodes.

The Delegate architecture, as depicted in Figure 4, displays a possible framework for implementation. Notice that this architectural level is also connected to the Query level, since the query scheduler needs the routines the Delegate level to make a decision whether or not the (sub)query execution should be performed locally, or on a remote processing node in the grid.

As can be seen, there are two major parts in this layer. The first part is the load-balancer, which contains algorithms that check whether the task should be performed locally or on a grid node. The decision depends on a range of factors, such as the type of query, the load of the local node, the load of the external nodes, etc. Apart from providing the optimal solutions at any given time, these algorithms must also be fast enough to provide the solution in a reasonable amount of time.

The second part of the Delegate architecture is the grid driver, which wraps the remote method invocation requests of the database so they are compatible with the grid software. To execute the query on a remote location, that remote location will need the data and the procedure code of the query, which are delivered in one package by the grid driver.

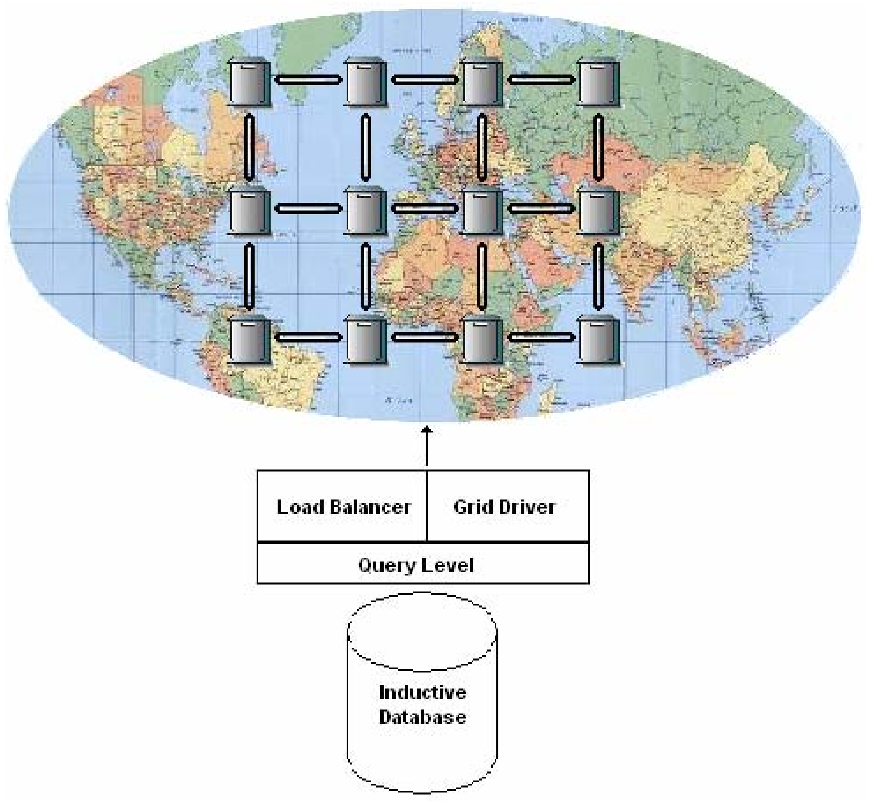

Figure 4: The Delegate Level

5 Use Case

In this section we will present a use case scenario that illustrates the workings of the diverse parts of the architecture. Suppose we have a database that contains transaction information on product sales for a supermarket and we want to perform some market-basket analysis. Normally we would use the *apriori-algorithm* to uncover frequent item sets and association rules in the data collection. In this example we will illustrate how this can be done in an inductive database.

First, consider a query that tries to find frequent patterns in the transaction data using a data mining primitive *FREQ_ITEM*. When the user poses this query to the inductive database, the query scheduler uses the query analyzer and optimizer to receive an optimized set of (sub)queries.

The next task of the query scheduler is to verify whether the tables addressed in the (sub)queries are available locally or if it's somewhere else in the data net.. When they are available locally, the query scheduler uses the data mining primitives to execute the (sub)queries using the data retrieved from the tables. If some of the data is not available locally, the (sub)queries involving those data are forwarded to the DataNet level, where it sends out a broadcast to discover the location of the required data. After the location has been received and the user has been verified, the subquery is sent to the query scheduler at that location. In this example, let the required data be available locally, so the query scheduler executes the query using the data mining primitive *FREQ_ITEM*, which results in a collection *P* containing frequent patterns over the transaction data set.

Let the collection *P* be stored in a pattern table *T*. Now that we have all frequent patterns over the data set, we can use those patterns to find association rules in them using a second data mining primitive *ASSOC_PATTERN*. This example illustrates the benefits of storing patterns as well as data. The reuse of patterns could prove to be an enormous optimization in data mining.

Now consider the case where we want to check if the same frequent item sets hold for another transaction database. To do this, we could either formulate the same query again over the second dataset, or we can use patternset *P* of the last query and apply a *CHECK_PATTERN* crossover data mining primitive operation to it. This illustrates the potential power of the inductive database: intuitively, the patternset *P* describes a subset of the data and thus it is intuitively more efficient to operate on those patterns instead of the whole dataset. While it is true that you can derive all patterns from the underlying data, but sometimes it might be more efficient to gain patterns from the patterns already available, as is the case in the example described above.

6 Conclusions and Future Work

In this paper we have discussed the topics of inductive databases and distributed knowledge discovery. We introduced a global architecture for the implementation of an inductive database that is suited for distributed computing and querying over the grid and discussed its various components. We have also described how a query would be handled by the architecture.

Our plan is to implement this architecture using the inductive database SINDBAD (acronym for: structured inductive database development) [10] which has been developed by Stefan Kramer and his team. By using an existing inductive database we can see how modifications in the architecture affect the efficiency of storage and querying, allowing us to maximize its performance.

Another goal is to provide a complete set of data mining primitives which are suitable for distributed data mining. However, research needs to be done on how existing algorithms can be modified for distributed computing. Furthermore, a thorough investigation must be done on which algorithms should be in this set.

Acknowledgements

This work is part of the BioRange programme of the Netherlands Bioinformatics Centre (NBIC), which is supported by BSIK grant BSIK03013 through the Netherlands Genomics Initiative (NGI). The authors also gratefully acknowledge support from the Leiden Universiteits Fonds. Finally, the authors would like to thank Stefan Kramer for allowing us to get involved in the SINDBAD project, and Dr. Fons Verbeek for proofreading this paper.

References

[1] J-F Boulicaut, M. Klemettinen, H. Mannila, *Querying Inductive Databases: A Case Study on the MINE RULE Operator*. PKDD 1998: 194-202

[2] L. de Raedt, *A perspective on inductive databases*. SIGKDD Explorations 4, pp. 69–77, 2003.

[3] T. Imielinski, H. Mannila, *A database perspective on knowledge discovery*, Communications of the ACM, v. 39 n. 11, pp. 58-64, Nov. 1996

[4] T. Imielinski, A. Virmani, MSQL: *A Query Language for Database Mining*, Data Mining and Knowledge Discovery, Vol. 2(4), pp. 373-408, 1999.

[5] R. Meo, *Inductive Databases: Towards a New Generation of Databases for Knowledge Discovery*, invited paper at First International Workshop on Integrating Data Mining, Database and Information Retrieval (IDDI), at DEXA, Copenhagen, Denmark, August, 22, 2005.

[6] R.Meo, G.Psaila, *Toward XML-Based Knowledge Discovery Systems*, Proc. of the IEEE International Conference on Data Mining, pp. 665-668, 9-12 December, 2002, Maebashi City, Japan.

[7] E. Bertino, B. Catania, E. Kotsifakos, A. Maddalena, I. Ntoutsi, Y. Theodoridis, *PBMS Querying and Storage Issues*, PANDA Technical Report PANDA-TR-2004-02, Feb. 2004

[8] C. Masson, C.Robardet, J-F. Boulicaut: *Optimizing subset queries: a step towards SQL-based inductive databases for itemsets*. SAC 2004: pp. 535-539

[9] L. de Raedt , M. Jaeger , S. D. Lee , H. Mannila, *A Theory of Inductive Query Answering*, Proceedings of the 2002 IEEE International Conference on Data Mining (ICDM'02), p.123, December 09-12, 2002

[10] S. Kramer, V. Aufschild, A. Hapfelmeier, A. Jarasch, K. Kessler, S. Reckow, J. Wicker, L. Richter: *Inductive Databases in the Relational Model: The Data as the Bridge*. In Knowledge Discovery in Inductive Databases: 4th International Workshop, KDID 2005, Porto, Portugal, October 3 , 2005.