Database Technologies for Electronic Commerce

Rakesh Agrawal **Ramakrishnan Srikant** **Yirong Xu**

IBM Almaden Research Center
650 Harry Road, San Jose, CA 95120, USA

1 Introduction

Electronic commerce applications have posed new challenges for database systems. In this demonstration, we present three technologies for electronic commerce that solve the following problems:

- How do you support nameless querying of databases from Google-style search boxes, where queries may not include attribute names?
- How do you efficiently store and query electronic commerce data that may have thousands of very sparse attributes?
- How do you merge catalogs (i.e., set of categorized products) from different sources and with different categorization schemes into a single master catalog?

We encountered the above problems when building an experimental B2B portal called Pangea, whose catalog contained nearly 2 million electronic components and 200,000 data-sheets in more than 2000 categories. We now present solutions to these problems.

2 Querying with Numbers

Electronic commerce applications have made it imperative for databases to support direct querying of database content from the web. However, the most popular web interface is the Google-style search box, and queries submitted from such an interface may include neither attribute names nor units. For example, an user searching for a laptop with a processor rated around 800 MHz and around 150 MB of memory might simply submit {laptop 800 150}. This problem also arises in federated database systems, where the same attribute might be called with different names in different constituent databases.

We conjecture that if we get a close match on the numbers, then it is likely that we have correctly matched the attribute names [AS02]. The extent to which a dataset satisfies the above conjecture can be quantified; we call this value the non-reflectivity of the dataset. For a simple example of a dataset with high non-reflectivity, assume that the records contain only two attributes: 'memory' and 'disk-size'. Further assume that the range of values for memory is 64 to 512 and that for disk-size is 10 to 40. Given a query {20, 128}, the system can correctly retrieve records that have disk-size and memory values close to 20 and 128 respectively. In this example, the attributes have non-overlapping domains. However, high non-reflectivity is not limited to data having such non-overlapping attributes. If memory and disk-size overlapped, but were correlated such that high memory configurations had high disk-size, the data would still have high non-reflectivity.

We now give a formal definition of non-reflectivity. Let $\mathcal{D} = \{x_1, \ldots, x_n\}$ be a set of m-dimensional points (records). Let $\vec{n_i}$ denote the co-ordinates of point x_i. We define the *reflections* of the point x_i to be the set of co-ordinates obtained by permuting $\vec{n_i}$ (including $\vec{n_i}$). For example, if x_i were $\langle 1, 2\rangle$, the reflections of x_i would be $\{\langle 1, 2\rangle, \langle 2, 1\rangle\}$. Let $\theta(\vec{n_i})$ denote the number of points within distance r of $\vec{n_i}$ (in m-dimensional space). The value of r is so chosen that the average value of $\theta(\vec{n_i})$ (over all $x_i \in \mathcal{D}$) is close to the number of top answers that users will be interested in. Let $\rho(\vec{n_i})$ denote the number of points in $\mathcal{D}$ that have at least one reflection within distance r of $\vec{n_i}$. The non-reflectivity of $\mathcal{D}$ in m-dimensional space is then defined as:

$$\text{Non-Reflectivity}(m, r) = \frac{1}{|\mathcal{D}|} \sum_{x_i \in \mathcal{D}} \frac{\theta(\vec{n_i})}{\rho(\vec{n_i})}$$

Non-reflectivity over subsets of attributes is defined in a similar manner (see [AS02]).

We consider nearest neighbor queries where the user is interested in retrieving the top t records containing values close to the query terms. For a given record and query, we find the set of numbers in the record that will minimize the L_p distance between the record and the query. This problem can be mapped to the bipartite graph matching problem and the corresponding algorithms directly apply here. The set of records that are matched is limited by using an adaptation of the Threshold Algorithm (TA) [FLN01].

Demonstration The demonstration compares our approach with existing search engines when searching over

**Proceedings of the 28th VLDB Conference,
Hong Kong, China, 2002**

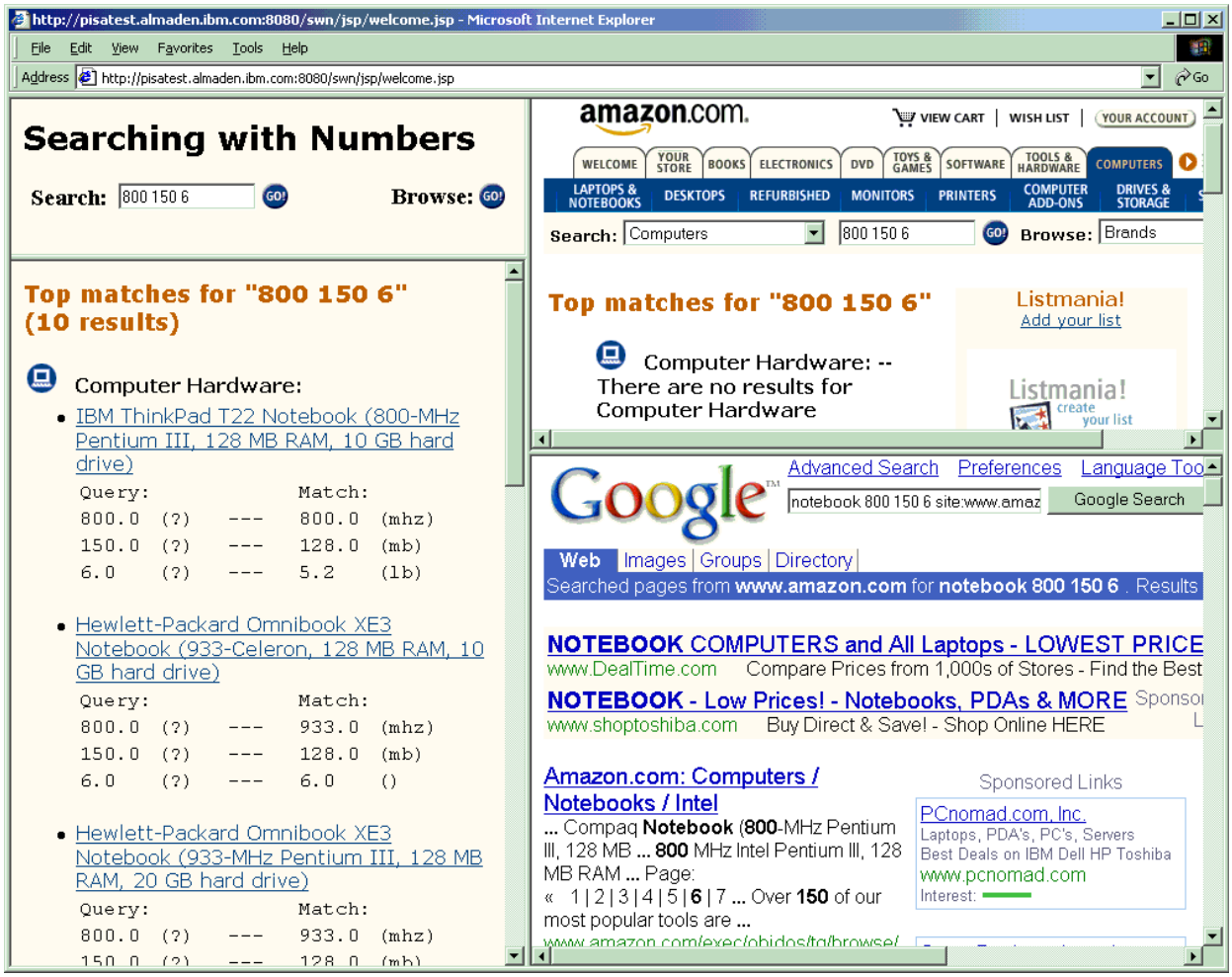

Figure 1: Querying with numbers

database records embedded in text, i.e., we do not know the attribute names either in the data or the query. Our datasets are topics from Amazon (e.g. notebooks currently available at Amazon). The demo system allows users to enter arbitrary search queries. The same query is also automatically submitted to Amazon and Google (the Google search is restricted to the Amazon site to keep the comparison fair). Figure 1 shows a snapshot of the demonstration. This screen shot shows the result of a search for a notebook with a processor speed close to 800 MHz, memory around 150 MB, and weight close to 6 pounds. For this query ("800 150 6"), Amazon does not find any matches since it treats numbers as strings, and none of the laptops have exactly 150 MB of memory. Google also treats the numbers as strings, goes on to match the 150 with the number of tools for each of the top 10 hits; thus none of these hits are correct. In contrast, our approach correctly matched all three attributes in 8 of the top 10 matches and returned notebooks close to the user's preference.

3 Efficient Storage and Querying of Electronic Commerce Data

In relational database systems, data objects are conventionally stored using a horizontal scheme, as shown in Figure 2(a). However, this representation is ill suited for electronic commerce data for the following reasons:

- *Schema Evolution:* We need frequent altering of the table to accommodate new parts and categories. Schema evolution is expensive in the current database systems.
- *Large Number of Columns:* The current database systems do not permit a large numbers of columns in a table. This limit is 1012 columns in DB2 (also in Oracle), whereas in Pangea we had nearly 5000 product attributes across different categories.
- *Sparsity:* Even if the database system were to allow the desired number of columns, we would have had nulls in most of the fields, which creates storage overhead and increases the size of the index.
- *Performance:* A query incurs a large performance penalty if the data records are very wide but only a

(a) Horizontal (*H*)

Oid	A1	A2	A3
1	a	b	⊥
2	⊥	c	d
3	⊥	⊥	a
4	b	⊥	d

(b) Vertical (*V*)

Oid	*Key*	*Val*
1	A1	a
1	A2	b
2	A2	c
2	A3	d
3	A3	a
4	A1	b
4	A3	d

Figure 2: Horizontal and Vertical Table Representations

few columns are used in the query.

We explore an alternative method where objects are stored in a vertical format as a set of tuples. Each tuple consists of an object identifier and attribute name-value pair. Figure 2(b) shows the vertical format corresponding to the horizontal table in Figure 2(a). The symbol ⊥ represents a null value. Schema evolution is now easy, and data storage is much more efficient.

However, once the data is stored in the vertical format, new problems arise. Writing SQL queries against this scheme becomes cumbersome and error-prone. More importantly, the current application development tools designed for the horizontal format no longer work. For example, an original simple query

```
SELECT A1 FROM H
WHERE A1 = 'a'
```

now may have to be written as

```
SELECT Val FROM V
WHERE Key = 'A1' and Val = 'a'
```

To solve this problem, we propose an enablement layer [ASX01] that hides the complexity of the queries over the vertical table and gives a horizontal view of the vertical representation to the user (application). Figure 3 shows the architecture of the enablement layer. The layer accepts user's query against the horizontal view, parses the query, and automatically transforms it to a query against the vertical table. Then the layer submits the transformed query to the underlying database engine, and returns the query results.

We explore two transformation strategies of the enablement layer:

- *VerticalSQL:* The implementation assumes only SQL-92 level of capabilities from the underlying database engine.
- *VerticalUDF:* The implementation exploits object-relational extensions to SQL (e.g. user-defined table functions).

Our performance experiments show that both strategies uniformly outperform the horizontal representation for sparse data [ASX01].

Demonstration We store our sample data in IBM DB2 7.1 using the vertical representation, and define the corresponding horizontal view in the enablement layer. We also

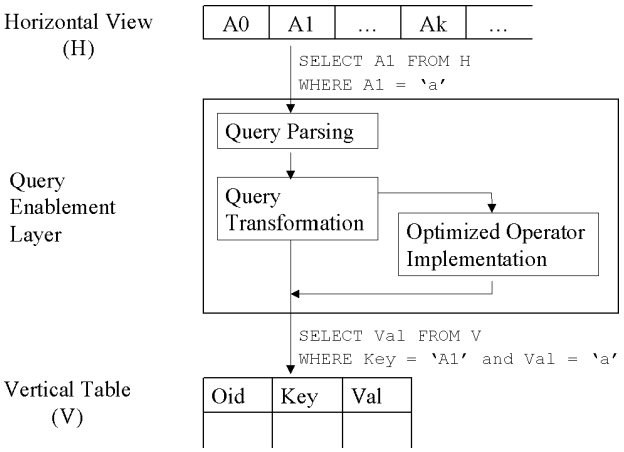

Figure 3: Architecture of query enablement layer

store a copy of the data using the horizontal representation. The demonstration includes a comprehensive set of SQL queries, ranging from basic selection, projection and join operations to complex combinations thereof. We submit each query to the enablement layer to query against the horizontal view. We show the transformed query and the results of the query. We also show that these results are identical to those obtained from directly querying against the horizontal representation.

4 Integrating New Content

Electronic commerce applications such as portals, marketplaces, and online stores (Amazon.com, eBay.com), are often faced with the problem of quickly integrating new catalogs from different sources into their existing catalog (the "master" catalog). This problem can also be found where companies want to integrate internal and external information.

A straightforward approach to attack this problem would be to formulate it as a classification problem. Let $\mathcal{M}$ be a master catalog and $\mathcal{N}$ be a new catalog. Each category in $\mathcal{M}$ is treated as a class, and the products belonging to each category are used as training examples. For this well-posed classification problem we can build a predictive model for classifying products in $\mathcal{N}$ into $\mathcal{M}$'s categories.

Notice, however, in this straightforward approach, we completely ignored $\mathcal{N}$'s categorization. On the other hand, $\mathcal{N}$'s categorization contains valuable implicit information about product similarity. Intuitively, two products belonging to the same category in $\mathcal{N}$ will tend to belong to the same category in $\mathcal{M}$. Suppose the classifier's prediction is such that 98% of the products belonging to some category in $\mathcal{N}$ fall in one category in $\mathcal{M}$ and 2% in a different category. Those 2% predictions are quite possibly errors.

In practice, the categories in $\mathcal{M}$ and $\mathcal{N}$ are likely to be less synchronized. For example, let $\mathcal{M}$ be OpenDirectory and $\mathcal{N}$ the Yahoo directory. For a category under Movies in OpenDirectory, 64% of documents belong to a single category in Yahoo, 20% to a second category, and the distribution tails off: 8%, 4%, 2%, 1%, 0.6% and 0.2%. However,

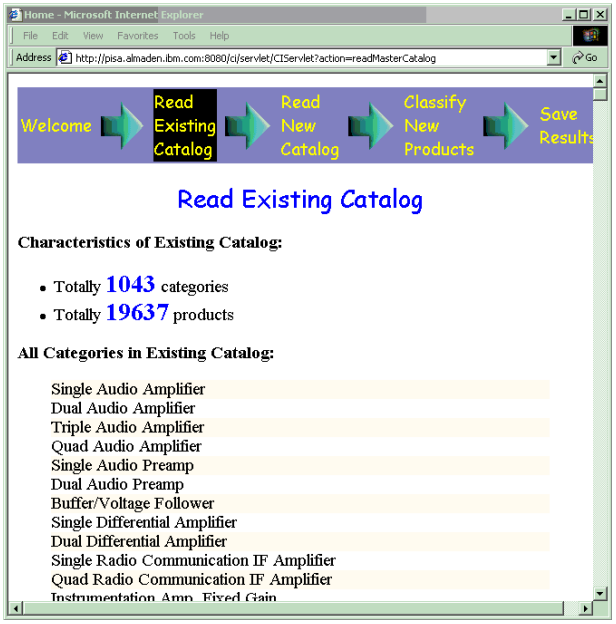

Figure 4: Building prediction model

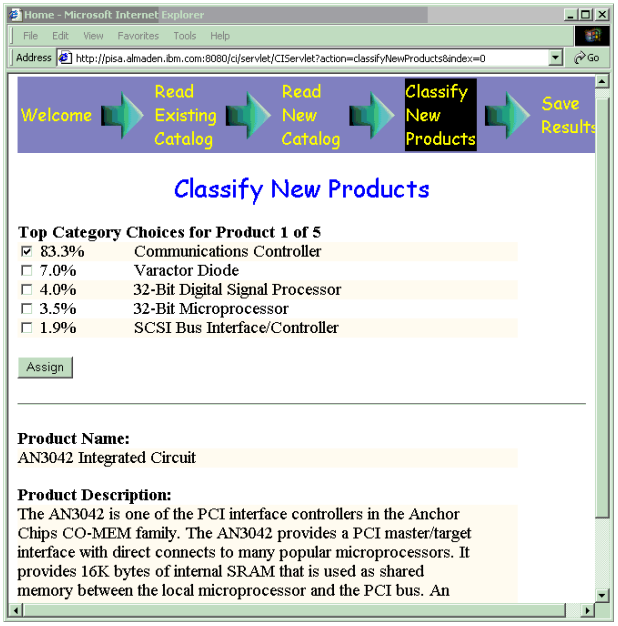

Figure 5: Categorizing a new product

we can still apply the above intuition by biasing the classifier towards the more frequent categories.

Our key contribution is how to incorporate the implicit information contained in the new catalog into the classification process [AS01]. We extend a Naive Bayes classifier as follows. We first classify all the documents in a category N in $\mathcal{N}$ using the standard Naive Bayes classifier. This gives us an estimate of which categories in $\mathcal{M}$ documents from N belong to. We then bias the classifier towards the categories that are more frequent among the documents in N, and away from the categories that are less frequent. The amount of bias is determined (over the entire set of documents in $\mathcal{N}$) using a small tune set of 5 to 10 documents. Notice that this bias will only affect the classification of documents near the border of two categories, and will not affect documents that the classifier was confident about.

The gain in accuracy depends on the match between the two sets of categorizations. Our experiments indicate that we get 26% to 30% fewer errors when going from OpenDirectory to Yahoo (or vice versa).

Demonstration The system is built as an IBM WebSphere Commerce 5.1 module. In the demonstration we use a sample from the Pangea data, stored in WebSphere Commerce's database format. The master catalog contains 1043 categories and 19,637 products. The catalog integration system first loads the master catalog from the database and builds a prediction model (see Figure 4) in around 30 seconds. Next the system loads a new catalog specified in Websphere Commerce's XML format. The system then works in a semi-automatic mode to complete the catalog integration: for each new product in the new catalog, it predicts upto 5 top categories and the user can easily choose from these recommendations (see Figure 5). Finally, the system saves the categorization results.

References

[AS01] Rakesh Agrawal and Ramakrishnan Srikant. On catalog integration. In *Proc. of the Tenth Int'l World Wide Web Conference (WWW10)*, Hong Kong, May 2001.

[AS02] Rakesh Agrawal and Ramakrishnan Srikant. Searching with numbers. In *Proc. of the Eleventh Int'l World Wide Web Conference (WWW)*, Honolulu, Hawaii, May 2002.

[ASX01] Rakesh Agrawal, Amit Somani, and Yirong Xu. Storage and querying of e-commerce data. In *Proc. of the 27th Int'l Conference on Very Large Databases (VLDB)*, pages 149–158, Rome, Italy, September 2001.

[FLN01] Ronald Fagin, Amnon Lotem, and Moni Naor. Optimal aggregation algorithms for middleware. In *Symposium on Principles of Database Systems*, 2001.