The Drill Down Benchmark

Peter A. Boncz, Tim Rühl, Fred Kwakkel
Data Distilleries B.V.
{boncz,tim,fred}@ddi.nl

Abstract

Data Mining places specific requirements on DBMS query performance that cannot be evaluated satisfactorily using existing OLAP benchmarks. The DD Benchmark – defined here – provides a practical case and yardstick to explore how well a DBMS is able to support Data Mining applications. It was derived from real-life data mining tasks performed by our Data Surveyor™ tool running on a variety of DBMS backends. We describe initial results obtained using both the Monet system and a relational DBMS product as backend.

1 Introduction

Data Mining is the process of automated extraction of knowledge from databases, and involves building models that explain observed phenomena in vast amounts of – historic – data. Being one of the most promising and profitable Data Warehousing applications, it has attracted much interest from both the commercial and research communities.

Data Mining causes a DBMS query load that differs from OLAP in various ways. A mining task generates a search process through a large hypotheses space and checks the validity of possibly thousands of hypotheses – where each check corresponds with a DBMS query. One mining task therefore corresponds to up to thousands of DBMS queries! These queries are run against one table, called the *mining table*, that contains all objects of interest for the mining problem. These objects typically have a large number of attributes.

Since most useful models for human understanding are simple ones, data mining algorithms tend to *drill down* to interesting subgroups in the mining table that can be described with just a few characteristics. For these reasons, the generated DBMS queries typically involve only a few attributes, on which selection, grouping, and aggregation operations are performed, but no complex join operations.

This very specific system load makes other analytical DBMS benchmarks, like TPC-D [Tra95] and SET [Gra93] inadequate for evaluating how well a DBMS is suited to support Data Mining. The test set often used by Agrawal [AS94] is specifically targeted to association rule algorithms and does not translate well to DBMS queries.

As we use benchmarking to assure quality control and to steer product development, we decided to define the Drill Down Benchmark (or: **DD Benchmark**). It is based on real-life experience that Data Distilleries has gathered since 1995 with the Data Surveyor tool [HKS95] on large-scale applications at clients from the financial sector in The Netherlands.

We ran the DD Benchmark with Data Surveyor using two different backend systems for DBMS query execution. One is a relational DBMS product, and the other is Monet [BWK98], a novel system licensed to Data Distilleries, which uses full vertical table fragmentation to avoid I/O bottlenecks and employs efficient main-memory techniques.

2 Data Surveyor Architecture

In order to meet the interactive performance requirements of Data Mining, most existing tools employ specialized data structures and algorithms to manipulate mass data outside the DBMS. This stand-alone approach defeats one of the prime purposes of a DBMS, which is integration of the data of an organization in one consistent, reliable and protected store.

The *Data Surveyor* tool of Data Distilleries – in contrast – has a 3-tier architecture, that facilitates integration of Data Mining with the DBMS:

GUIs taking the form of Java applets. Apart from a powerful expert data mining interface, we provide pre-cooked user interfaces targeted to end-users, that solve one subproblem (so called 'vertical applications'), like selecting the top mailing

Proceedings of the 24th VLDB Conference
New York, USA, 1998

addresses of the week for database marketing.

Data Mining Kernel containing the data-mining specific algorithms. This component directs the Data Mining operations and translates a data mining task into multiple DBMS queries. Multi-query optimization and parallelization are employed to improve performance.

DBMS backend can be all SQL-speaking commercial (parallel) DBMSs. Data Distilleries also licensed the novel high-performance Monet system for use as backend.

The Data Mining Kernel uses a unique algorithmic framework that decomposes data mining algorithms in three orthogonal dimensions:

- a *modeling language* for expressing hypotheses,
- a *quality function* for testing the quality of a hypothesis, and
- a *search strategy* for looking for interesting hypotheses.

This decomposition facilitates the translation of a data mining task into DBMS queries, which – thanks to its background in database research – is one of the strengths of Data Distilleries.

3 Drill Down Benchmark

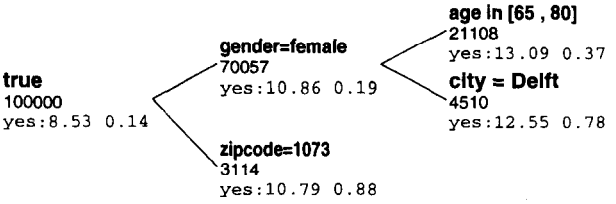
The goal of the DD Benchmark is to measure DBMS performance on a typical Data Mining query load. Since SQL-like query languages are unsuitable to express Data Mining algorithms, we chose to formulate the benchmark in terms of our algorithmic framework on an typical data mining task. Benchmark implementors must translate this task into DBMS queries, while adhering to a number of implementation rules.[1]

3.1 Benchmark Case Study

The task chosen for the DD Benchmark is a **customer loyalty** application, a common and prototypical data mining problem. In this task, a company wants to find profiles for (un)reliable groups of customers.

The DD Benchmark uses *decision rules* as the modeling language to describe such customers, where rules are simple conjunctions of selection expressions on the attributes of the mining table. The quality of such rules is expressed with a *confidence-interval(P)* with $P = .95$, and the employed search strategy is *beam-search(W,D)*. Beam-search is a form of breadth-first search, bounded both in width W and depth D. We use parameters $D = 4$ to find descriptions involving at most 4 conjunctions, and $W = 10$ to use only the best 10 hypotheses for refinement in the next level.

[1]The full benchmark specifications, including database content, implementation rules, query scripts and detailed benchmark results can be obtained at `http://www.ddi.nl/ddbench`



The above figure shows an example decision rule model as a tree, where each node at depth $X \in \{1,2\}$ identifies a group of customers – characterized by a conjunction of X selections – and for each group its size and the *probability*(`reliable = yes`), with a 95% confidence interval. More specifically, this tree states with a confidence of 95% that the probability of any client to be a reliable customer is between [8.53-0.14,8.53+0.14] percent. This range lies higher for the subgroup with zipcode=1073 and for females. There are two extra-interesting subgroups of the females: those in Delft and those of old age, which are even more reliable customers.

3.2 Benchmark Data Definition

The data mining task modeled in the DD Benchmark is scalable both in size and complexity. This translates into mining on more objects or mining on more characteristics. The first scaling dimension corresponds to the number of rows in the mining table (*vertical scaling*), whereas the second corresponds to the number of columns (*horizontal scaling*).

The mining table of the DD Benchmark contains one tuple for each customer, and has 1*VF million rows of 100*HF attributes, of which 6*HF play a role in the queries. VF and HF indicate the vertical and horizontal scaling factors, respectively. The distribution of the values in these 6*HF attributes is non-uniform, and their cardinalities vary:

Attributes in the Mining Table		
attribute	domain	card.
target attribute		
reliable	{yes,no}	2
query attributes $\forall i \in \{1,..,HF\}$		
age$_i$	{18,..95}	77
zipcode$_i$	{1000,7999}	7000
marital$_i$	{ married,single,divorced,widowed }	4
town$_i$	{ Amsterdam,..,Rotterdam }	15
spendings$_i$	floating point values	100
gender$_i$	{male,female}	2

The `reliable` attribute is the so-called 'target attribute'; the others are called 'query attributes'. Though we only use the above attributes, no a priori knowledge of this may be assumed by implementors. The mining task corresponds to *one mouse click* made by a Data Surveyor user after randomly selecting 6*HF attributes from the 100*HF available for detailed exploration. This means that before the mining task starts, it is not allowed to treat the 6*HF query attributes in any way different than the other attributes.

3.3 Benchmark Queries

The data mining algorithm executes 5 *query batches* that contain data cube requests. We require each cube request to produce an ASCII dump of its table representation. Below we will outline the cube requests in SQL, but we do not pose specific requirements on the query syntax, or even on the language used.

Each query batch corresponds to one level searched by the beamsearch algorithm. Based on the results of the batch, the data mining algorithm determines which subgroups are most interesting. As these groups are used for further exploration in the next level, the content of the next batch directly depends on the outcome of the previous.

The Drill Down Benchmark measures DBMS performance solely, hence the data mining algorithm (i.e., choosing the best groups for the next batch on the basis of results from the current batch), is **not** part of the benchmark. The benchmark therefore consists of five fixed query batches, but we mimick the dependencies of real-life data mining by requiring that the current batch must be completed before the next may start.

batch B_0 computes a histogram on each query attribute and the target attribute. In SQL: `SELECT count(*) FROM miningtable GROUP BY attr`.

With such a histogram, the data mining algorithm can make a preliminary decision whether this attribute is interesting, or not. For instance, an attribute like 'client-number' has unique values for each customer. Knowing this, the Data Mining algorithm can already conclude that this attribute is not useful for characterizing groups of customers. In the case of the DD Benchmark, none of the attributes is discarded.

batch B_1 computes a histogram on each query attribute with respect to the target attribute `reliable`. In SQL, this becomes: `SELECT count(*) FROM miningtable GROUP BY attr1, reliable`.

The output of this query is then used by the Data Mining Algorithm to find groups of values of the query attribute that have a significantly deviating distribution with respect to the target attribute (`reliable`). The `beam-search(W=10,D=4)` algorithm keeps the 10 most deviating groups formed by a selection on this attribute (e.g., `gender=female`).

batch B_2 further explores each group. As each group is a selection on one attribute, 5 query attributes are still left for exploration. Hence, B_2 consists of 10*5=50 queries that compute the distribution with respect to `reliable` for each group and for each attribute left. In SQL, B_2 consists of a series of `SELECT count(*) FROM miningtable WHERE cond1(attr1) GROUP BY attr2, reliable`.

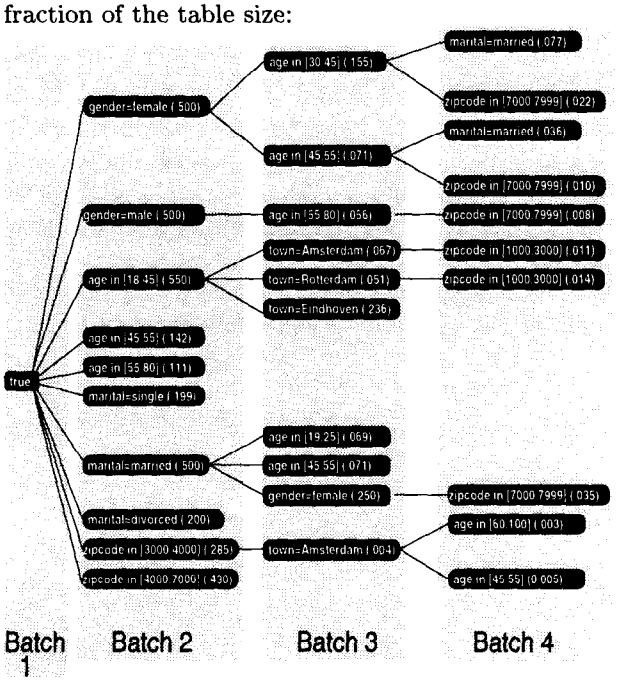
With this information, new subgroups can be identified in terms of selections on two query attributes (e.g., `gender=female AND` $30 \leq$ `age` ≤ 45). The Data Mining Algorithm again keeps the 10 most significantly deviating groups for further exploration.

batch B_3 further explores these groups. Now there are 4 attributes left for exploration. This turns into SQL as: `SELECT count(*) FROM miningtable WHERE cond1(attr1) AND cond2(attr2) GROUP BY attr3, reliable` queries.

The Data Mining Algorithm again keeps the 10 most significantly deviating groups – now based on selections on 3 attributes.

batch B_4 This last batch explores for each subgroup the 3 query attributes left. The SQL queries now have selections on a conjunction of tertiary conditions. Note that using these results, the mining algorithms can identify groups with four conditions; which ends the search process (D=4).

Below we display the groups explored by the DD Benchmark; with between parentheses their size as a fraction of the table size:



In total, when HF=1, the five batches of the DD Benchmark represent $7 + 6 + 50 + 40 + 30 = 133$ SQL queries.[2] Each such query needs a scan over the mining table. As the mining tool requires an interactive response, it may be clear that Data Mining is a very high performance DBMS application area.

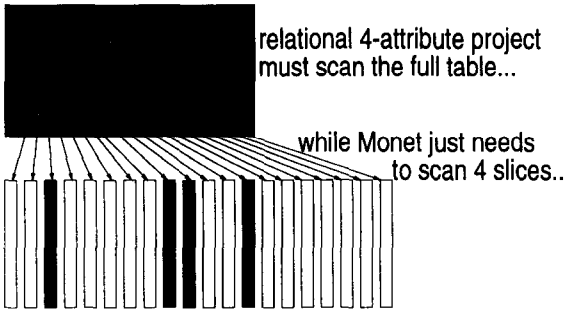
[2]When the database is scaled horizontally, this number grows. The benchmark definition document defines exactly how its query set is generated.

4 Experiments

We have carried out DD Benchmark experiments with Data Surveyor version 2.0 using both Monet and a relational DBMS product as backend system.

4.1 Monet

Monet is a new DBMS developed at the CWI [BWK98] oriented towards the OLAP and Data Mining application areas. It uses full vertical data fragmentation, by column-wise slicing of each relational table:



The DD Benchmark is especially heavy as each of its 133 queries requires a full table scan. In a standard query execution strategy, these table scans would make I/O the bottleneck in query execution. It is clear that Monet avoids a lot of I/O on the DD Benchmark, as it only has to scan the 6*HF slices of the query attributes, instead of the whole relational table.

The downside of vertical fragmentation, however, is that extra effort (joins) must be done to recombine fragmented data. The novel aspect of the Monet system is that its database kernel is specifically designed to cope with sliced tables. Table slices are interrelated, as each slice contains the same tuple sequence. This inter-relatedness is exploited by Monet and hence it is able to prevent doing extra work introduced by the vertical fragmentation.

The second important characteristic of the Monet system is that its code is heavily optimized for main-memory query execution. If vertical fragmentation is successful in avoiding unnecessary I/O, the balance of query processing cost shifts to CPU cycles and memory access time. Past research done on main-memory databases [GMS92] has shown that main-memory execution favors totally different optimization criteria than those important in systems where the dominant cost is I/O.

The increased complexity of modern hardware has added to the importance of main-memory specific optimizations in the software architecture: all computer systems now come with a 3-level memory hierarchy and an extensive collection of register sets, whose optimal use is crucial to leveraging the power of the CPU. Programs that do not take advantage of the memory cache or do deeply nested procedure calls that lead to register trashing, now suffer comparatively more performance loss than in the past [SKN94].

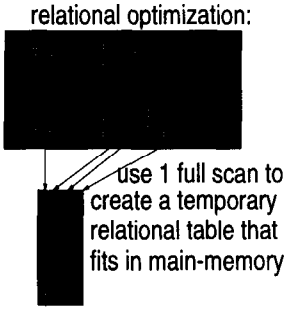
While Monet is designed to exploit main memory when abundant, it is not an all-or-nothing main-memory system. That is, if the database hot-set exceeds main memory, the system relies on operating system support for managing virtual memory. Access to virtual memory causes page faults, and in this way I/O does play its role in the system. Working with such a *single-level store*, and relying on the OS to manage I/O, has the advantage that algorithms and data structures can stay simple, thus not compromising performance when the hot-set does fit in main memory.

4.2 Relational DBMS

Most relational DBMS products stem from a design line that originated in the 1970s, and hence were carefully designed and tuned to the application requirements and hardware characteristics of that time.

Technically speaking, their storage infrastructure is still optimized towards the needs of OLTP, which requires quick performance on large amounts of small updates. Query-intensive applications like OLAP and Data Mining, however, have an entirely different access pattern, as they condense large volumes of data into small and meaningful results. As mentioned earlier, this leads to table scans that use only a fraction of the generated I/O. Naive query execution of the benchmark on the relational DBMS product showed execution times of several hours, I/O being the bottleneck.

In order to speed things up, we decided to use the DBMS product in a pure main-memory situation, by introducing an additional run-time step. In this step, which takes just one table scan, all 6*HF query attributes are selected and put into a newly created temporary table:



We increased the buffer size of the DBMS product so that this temporary table would then fit into memory. The 133 SQL queries of the DD Benchmark are then performed on this temporary table; which is destroyed afterwards.

Summarized Results of the DD Benchmark										
database size			platform		elapsed time					
	HF	VF	DBMS	#cpus	total	B_0	B_1	B_2	B_3	B_4
small	1	1	relational	2	22m35s	1m22s	1m28s	7m48s	5m47s	3m9s
(400MB)			monet	2	39s	4.6s	4.7s	17.9s	9.1s	2.3s
				1	1m10s	9.1s	8.9s	31.1s	17.0s	4.1s
big	1	10	monet	2	8m32s	50.0s	54.0s	4m23s	1m52s	33s
(4GB)				1	13m58s	1m27s	1m38s	6m43s	3m14s	57s
wide	10	1	monet	2	8m46s	1m5s	53.0s	3m41	2m29s	37s
(4GB)				1	15m25s	1m39s	1m38s	6m33s	4m29s	1m6s

4.3 Results

The above table displays the performance results obtained by running the DD Benchmark on a Sun Ultra 2 Creator3D machine with two 168 MHz CPUs, 512 MB of memory, 2 MB L2 cache, and 12 MB/s of real disk throughput.

The first row of the table contains the result of the relational DBMS using both CPUs on the small DD Benchmark (VF=HF=1). It shows that our optimization with the temporary table worked; creating this in-memory table cost 3 minutes; after which no I/O activity was observed during the remaining queries. In all, the relational DBMS was considerable sped up by our optimization (from several hours to 22 minutes). This made the query execution CPU-bound instead of I/O bound.

The other lines describe the various Monet experiments. The parallel experiment of Monet on the small DD Benchmark takes 39 seconds of elapsed time; more than an order of magnitude faster than the relational DBMS. The sequential experiments take almost twice as much time; showing that Monet is able to use parallelism of both CPUs effectively. The wide (HF=10) and big (VF=10) experiments scale reasonably well. The main memory of our hardware configuration was not large enough to try these experiments on the relational system.

These results indicate that a system like Monet that is optimized towards main-memory access in terms of data structures and algorithms, clearly outperforms a general-purpose relational system under conditions like the DD Benchmark. One might ask the question how relational DBMS products could be improved to perform better under conditions like the DD Benchmark. In our view, this would mean introducing data structures and algorithms to relational DBMS products that are better suited to main-memory execution.

While this might seem an unrealistic option; we regard the emergence of object-relational DBMS products as an opportunity to achieve this goal, as an object-relational DBMS gives the possibility to extend the database system with new structures and primitives, packaged in a so-called *data-blade* or *data-cartridge*.

5 Conclusions

We have defined a new, useful, and public benchmark that measures Data Mining performance of DBMS products, and sets a yardstick for future developments in this area. The results of our benchmarking effort using Data Surveyor indicate that interactive Data Mining on large data sets is possible, but requires specific algorithms and data structures to be integrated in DBMS technology. We are looking forward to results produced by other DBMS platforms to help identify new efficient DBMS techniques for supporting Data Mining applications.

References

[AS94] R. Agrawal and R. Srikant. Fast algorithms for mining association rules. In *Proc. VLDB Conf.*, 1994.

[BWK98] P. A. Boncz, A. Wilschut, and M. L. Kersten. Flattening an object algebra to provide performance. In *Proc. ICDE Conf.*, 1998.

[GMS92] H. Garcia-Molina and K. Salem. Main memory database systems: An overview. *IEEE TKDE*, 4(6):509, 1992.

[Gra93] J. Gray. *The Benchmark Handbook For Databases and Transaction Processing Systems.* Morgan Kaufman, 1993.

[HKS95] M. Holsheimer, M. L. Kersten, and A. Siebes. Data Surveyor: Searching for nuggets in parallel. In *Knowledge Discovery in Databases.* MIT Press, 1995.

[SKN94] A. Shatdahl, C. Kant, and J.F. Naughton. Cache conscious algorithms for relational query processing. In *Proc. VLDB Conf*, 1994.

[Tra95] Transaction Processing Performance Council. *TPC Benchmark D*, 1.2.3 edition, 1995.