Towards a Crystal Ball for Data Retrieval

Joseph M. Hellerstein
U.C. Berkeley
387 Soda Hall #1776
Berkeley, CA, USA 94720-1776
jmh@cs.berkeley.edu

Abstract

Information Systems – both databases and text-search programs – are typically architected as "black boxes": a user submits a request, the system performs an unknown sequences of operations, and after some time an answer set is returned. Two trends are conspiring to make such architectures undesirable. First, users of these systems are often quite naive, and unsure of what they are doing. Second, the queries submitted to these systems are taking increasing amounts of time to complete. These trends together lead to a frustrating experience for users: they are unsure if their inputs are appropriate, and the cost of an inappropriate input is often a long wait followed by a useless or misleading result.

In this paper we propose changing the black-box model to one of a "crystal ball", in which users are given feedback on their queries as they run, so that they can predict the utility of their query results, control the behavior of the queries on the fly, and better understand the operation of the system. We highlight some initial work in this vein, and describe opportunities for similar efforts in new applications.

1 Introduction

Broad sectors of the population want easy access to data; the success of the world-wide web has made this clear. With datasets growing at ever-increasing rates, it is becoming more difficult for users to find the data that interests them. One of the basic problems with existing query facilities – both on the web and in traditional database systems – is that they are "black boxes": users submit requests, and after some time are given responses, with little or no explanation of how or why the system determined that the response fit the query.

This is the legacy of years of research into supporting "declarative" query languages, in which users describe what they want, rather than how to get it. The problem that is emerging with declarative querying is that either (a) most users can not or will not crisply describe what they want (as in database systems), or (b) system designers can not or will not provide an interface in which users can crisply describe what they want (as in text-search engines). The result of this is a frustrating interaction in which users must second-guess the behavior of the system, leading to bizarre and frustrating modes of interaction.

This frustration with black-box architectures is exacerbated by a general increase in processing time per operation. Datasets are growing in size, concurrency levels are increasing (especially on the web), and thanks to ever-better user interfaces, the average query is more and more likely to be non-trivial to evaluate. As a result of all these factors, users are often left waiting for some time before a black box returns an output. The increasing query response time means that an iterative interaction with a black-box system becomes difficult or untenable.

As a solution to this problem, we propose replacing black-box systems with newer "crystal-ball" designs, which allow users to observe and control system behavior *online*, in the middle of processing. Crystal-ball systems can allow users to predict their query results before they complete, control the query processing on the fly, and better understand the operation of the system. Such architectures have been proposed for relational aggregation queries [2], and we believe will be increasingly useful for other relational queries, as well as for information retrieval and data mining.

2 DBMS vs. IR: Motivation from the Distinction

Databases and Information Retrieval systems are philosophically very different. In this section we demonstrate that their difference exposes an important issue which drives the need for crystal-ball systems.

2.1 DBMS: Black Box in the Glass House

Naive users continue to find it difficult to accurately express *ad hoc* queries over relational databases, despite the efforts of projects such as IBM's QBE, or even more recent popularizers like Microsoft Access. The problem is not that the query interfaces are imprecise, but rather that most users are not willing to invest the effort required to understand them even at a basic level. The result is that users who submit *ad hoc* queries are typically unsure if the "semantically correct" answer they receive means anything, since they do not fully understand the semantics of their query to begin with.

2.2 IR: WYGIWIGY Systems

A plausible solution to the complexity of relational queries is to remove the intellectual burden from the users, and encode it into software. This is the approach of the Artificial Intelligence community, which developed the technology behind IR systems. In an IR system, the stored data is usually complex (e.g., documents or images), and users can express queries in a "natural language" (English, almost without exception).

IR systems suffer from the opposite problem of DBMSs. In IR systems, the software can not be sure of the semantics of the data or a user's query, so the answer returned to the user depends entirely on the semantic interpretation which the software gives to dataset and query.[1] These systems may be termed "What You Get Is What I Give You" (WYGIWIGY) systems, since the user has minimal control over the systems' semantic details. An especially frustrating aspect of WYGIWIGY systems is that their semantic interpretation techniques are often proprietary, and even those which are public are difficult to explain. The result is that WYGIWIGY systems have to be used in an iterative black-box fashion: users pose multiple variations of their query, in the hope of both finding (apparently) satisfactory answers, and potentially (though usually only approximately) learning the system's interpretation algorithm.

2.3 Different Approaches, Common Problem

These two types of systems represent extremes on a spectrum of "semantic control". Database systems place all semantic control and responsibility for queries in the hands of users, while IR systems embed this control in the software. These different "Human-Computer Interfaces" represent a fundamental philosophical distinction between the IR and DB communities. However, both of these approaches suffer from the basic problem that *expressing queries accurately is a complex undertaking, and not well-suited to naive users.* This basic problem has stymied the development of a natural interface for querying. The IR and DB communities have chosen different ways to face the problem: the IR community avoids the problem by providing no semantic guarantees, while the DB community ignores the problem by forcing users to express themselves in a mathematical paradigm. Each solution has drawbacks, but these can be ameliorated by a crystal-ball approach in both cases.

3 Beyond the Black Box: Case Studies

In this section, we briefly examine two techniques that provide crystal-ball behavior in the context of a relational database system. These serve as examples of how converting from a black box to a crystal ball can benefit user comprehension and efficiency.

[1]Many IR document systems essentially just do *keyword-search*: they retrieve documents which contain a subset of the words in the query expression. Strict keyword-search is like a limited query language, and such systems have been implemented in DBMSs. However many keyword-based IR document systems attempt to extract more semantics from a query, using techniques like thesauri or vector-based approaches, to find possibly relevant documents which do not contain keywords.

3.1 Online Aggregation

Consider the following simple relational query:

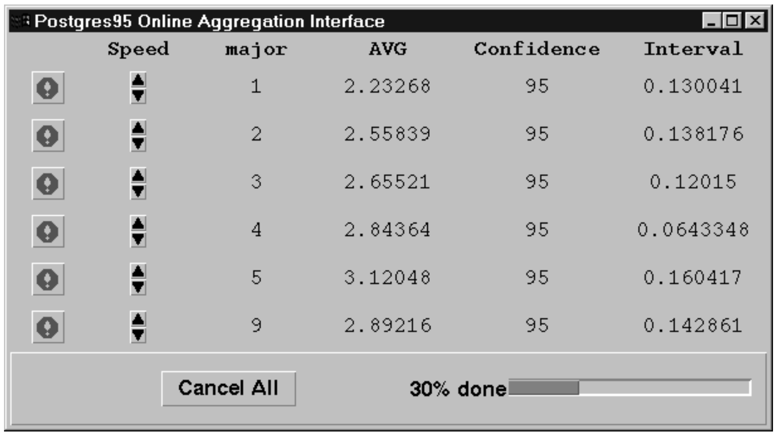
```
Query 2:
SELECT AVG(final_grades) FROM grades
GROUP BY major;
```

The output of this query in an online aggregation system can be a set of interfaces, one per output group, as in Figure 1. For each output group, the user is given a current estimate of the final answer, and a "confidence interval", which says that with $x\%$ confidence, the current estimate is within an interval of size k from the final answer. A status bar at the bottom of the screen shows how much processing time remains. These interfaces expose salient features of the current state of processing, and predict the final outcome in a statistically accurate fashion. In addition, controls are provided to stop processing on a group, or to speed up or slow down the group relative to others. These interfaces are supported by significant modifications to a relational DBMS, as described in [2]. Key system themes in this work include a focus on non-blocking algorithms and new *striding* access methods to support round-robin fetching from different value groups.

The online feedback and control in this system are especially beneficial for OLAP-style data analysis, in which a naive user (who is often an important decision-maker) wishes to quickly traverse through large amounts of information. This sort of user makes high demands on a system, including tolerable accuracy and instantaneous ("speed-of-thought") response time. The crystal-ball approach of online aggregation helps meet these needs in that it allows the user to get instantaneous response time at *some* level of accuracy; if the user is not satisfied with the quick estimate he or she can choose to be more patient. Any compromises in either time or accuracy are placed in the user's hands on a case-by-case basis, another advantage of a crystal-ball system.

3.2 Controlling Query Optimization

A notorious black box within a relational DBMS is the query optimizer. Even experienced MIS professionals are typically unaware of how and why optimizers make the decisions that they do. Years of treating the optimizer as a black box can lead to "solutions" which modify the optimizer's input to produce the desired output. For example, an old trick for "fooling" an optimizer into choosing the right plan is to repeat a Boolean Factor in a query in order to convince the optimizer that a selectivity estimation should be lower (e.g., `SELECT * FROM emp WHERE sal > 100000 AND sal > 100000` may produce the right plan when a more natural query would not.) Such inanities arise from the black-box nature of most query optimizers, and can only be used effectively with significant trial and error. A frustrating side-effect of this solution arises when vendors decide whether or not to fix the underlying problem. If they fix the problem, existing users' previous solutions may no longer produce the desired effect. If they do not fix the problem, new users will be forced to learn the same inane tricks that old users have employed.



	Speed	major	AVG	Confidence	Interval
		1	2.23268	95	0.130041
		2	2.55839	95	0.138176
		3	2.65521	95	0.12015
		4	2.84364	95	0.0643348
		5	3.12048	95	0.160417
		9	2.89216	95	0.142861

Figure 1: A Speed-Controllable Multi-Group Online Aggregation Interface

Some optimizers now allow user "hints" to provide a limited form of control, but typically they do not explain the logic that led to the need for the hints. Newer academic systems (e.g. Wisconsin's Minibase-view [6] and Opt++ [3] projects) provide more feedback and control, displaying portions of the plan-space traversed, allowing pruning of the space followed by differential retraversal, and providing a choice of search strategies. An improvement would be to provide this control in an online fashion, so that for large queries users could see and possibly modify the optimization as it takes place. This sort of online feedback and control will become increasingly important as optimizers are forced to resort to randomized or heuristic search techniques for large queries. This is an example of a crystal-ball architecture being useful even for sophisticated users. For such problems, a crystal-ball system bears a resemblance to a debugger, but works at a higher semantic level.

4 Topics for Future Work

4.1 Online Query Refinement

A key motivation of a crystal-ball system is to allow users to modify their inputs as they are being processed, without requiring the system to start over "from scratch". In the context of relational queries, this may be termed *online query refinement*. Such behavior is possible to a limited extent in the Online Aggregation system described above, where the STOP signs can be thought of as "HAVING" clauses that are added to the query as it runs.

A more powerful online query refinement system would allow users to add arbitrary predicates, SELECT expressions, etc. as a query was running, and the system would make use of the partial results computed so far, changing the processing "just enough" to accomodate the new query efficiently. This is somewhat related to ideas from multiple query optimization [7] and sample views [5], but unifies and extends both ideas.

4.2 Directed Data Mining

"Data Mining" is the inevitable conclusion of the AI approach to querying. In Data Mining applications, not only does the system define the semantics, it actually defines the queries. The user simply says "Go", and the system produces what it believes to be useful answers.

While this has some interesting applications, the idea of Data Mining is fundamentally crippled by having the crudest possible black-box interface. Moreover, data mining algorithms typically run for hours in production environments [1]. Interesting "mining" applications in the future will merge Data Mining with Decision-Support or "OLAP" queries, mixing the AI and DB approaches of system-driven and user-driven discovery. An important aspect of this challenge will be to "open up" this merged technology; these systems will execute long-running analysis steps, and users will need to observe and control this analysis as it runs. At bottom, Data Mining is a sophisticated form of aggregation, in which large amounts of data are processed over a long period of time to produce more cohesive descriptions. The arguments for Online Aggregation apply even more strongly to Data Mining.

4.3 User Interface Issues

The key to a good crystal ball is that it show exactly what the user needs to see. The goal is not to expose the detailed inner workings of the raw algorithms (a "Lucite Watch" architecture), but rather to display a useful precis of the running behavior. For example, in the Online Aggregation system the user is not presented with the status of the hash-table used for grouping, but instead statistical estimators and status bars. It has been noted that status bars alone improve a user's perception of the speed of a system [4], so this is clearly a generic tool that can be used in many crystal-ball applications. Statistical estimators of final outcomes are a similar class of relatively generic tools. Other such tools are clearly needed. In many cases the tools will need to be designed specifically for the system at hand, but one can imagine developing a suite of generic status tools as well. Moreover, since much of the motivation for this work is to improve the user interface, human-factor studies of crystal-ball interfaces are certainly called for.

5 Conclusion

No long-running system should be a black box. This rather basic observation opens up a number of research issues in information systems, both in terms of system architecture and algorithms, and user interface design. The database research community has long called for increased focus on user interfaces, but has been largely reluctant to study user interfaces in the absence of related algorithmic or architectural questions. The challenge of building crystal balls will require unified efforts in this regard – as the Online Aggregation work demonstrates, system support for a crystal-ball interface can require significant research, since the usability goals of a crystal ball often differ from those of a black box. Because of the broad-based modifications required to convert a black box into a crystal ball, this area should spark interest in the user-interface, database systems, and information retrieval communities.

References

[1] Rakesh Agrawal. Personal correspondence, February 1997.

[2] Joseph M. Hellerstein, Peter J. Haas, and Helen J. Wang. Online Aggregation. In *Proc. ACM-SIGMOD International Conference on Management of Data*, Tucson, May 1997.

[3] Navin Kabra and David J. DeWitt. Opt++ – An Object Oriented Implementation for Extensible Database Query Optimization. Submitted for publication. See also: http://www.cs.wisc.edu/ navin/research/opt++.ps, 1996.

[4] B. A. Myers. The Importance of Percent-Done Progress Indicators for Computer-Human Interfaces. In *Proceedings SIGCHI '85: Human Factors in Computing Systems*, pages 11–17, April 1985.

[5] Frank Olken. *Random Sampling from Databases.* PhD thesis, University of California, Berkeley, 1993.

[6] Raghu Ramakrishnan. The Minbase Home Page, 1996. http://www.cs.wisc.edu/coral/minibase/minibase.html.

[7] Timos Sellis. Multiple Query Optimization. *ACM Transactions on Database Systems*, 13(1):23–52, March 1988.