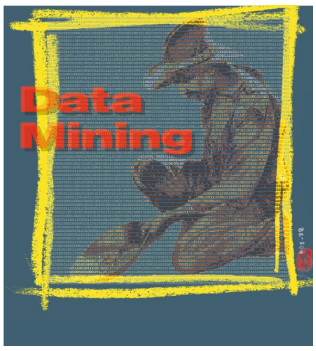
Cover Feature

Interactive Data Analysis: The Control Project



Human insight is crucial for extracting meaning out of massive data sets, yet current user interactions with databases don't allow iterative, intuitive analysis. The Control project looks at ways to give users quicker, more direct interactivity with the data.

Joseph M. Hellerstein
Ron Avnur
Andy Chou
Christian Hidber
Chris Olston
Vijayshankar Raman
Tali Roth
University of California, Berkeley

Peter J. Haas
IBM Almaden Research Center

Data analysis is fundamentally an iterative process in which you issue a query, receive a response, formulate the next query based on the response, and repeat. You usually don't issue a single, perfectly chosen query and get the information you want from a database; indeed, the purpose of data analysis is to extract *unknown* information, and in most situations there is no one perfect query.[1] People naturally start by asking broad, big-picture questions and then continually refine their questions based on feedback and domain knowledge.[2]

Consider repeating this process several times over, sifting through many more results, and you have an idea of why using advanced data analysis tools is so complex. Composing Structured Query Language (SQL) queries for decision-support database management systems (DBMSs) isn't easy, and even users of graphical query tools find it difficult to generate insightful queries.

Although data-mining systems typically don't provide complicated query languages, to use these systems you need to choose a suitable mining algorithm and carefully tune various algorithm-specific parameters such as support and confidence for association rule mining, thresholds for clustering, training sets for classification, and so on. These usability problems increase the number of iterations in the analysis process; you have to try algorithms with different parameters until you find one that produces useful results. In addition, many of these tools require complicated, time-consuming setup phases before they can be used at all.

Most research in the areas of decision support, data visualization, statistics, data mining and knowledge discovery has concentrated on improving a single iteration of the analysis process. Some work has focused on improving the quality of a particular analysis result or on reducing the time it takes for each analysis step or algorithm to provide a complete response.

These fields have progressed greatly, but this research focus ignores a basic invariant in computing: Full-scale data analysis will always be slow. As Greg Papadopoulos, chief technology officer at Sun, points out, the appetite for data collection, storage, and analysis is outstripping Moore's law, meaning that the time required to analyze massive data sets is steadily growing. To date, the result is a worst-case mode of human-computer interaction: Data analysis is a complex process involving multiple, time-consuming steps, and a poor or erroneous choice of inputs is not noticeable until results return at the end of a given step. The long delay and absolute lack of control during individual analysis steps disrupt the user's concentration and hamper the data analysis process. This situation is reminiscent of Herodotus' lament: "Of all men's miseries, the bitterest is this: to know so much and have control over nothing."

In the Control (Continuous Output and Navigation Technology with Refinement Online) project at Berkeley, we are working with collaborators at IBM, Informix, and elsewhere to explore ways to improve human-computer interaction during data analysis. The Control project's goal is to develop interactive, intuitive techniques for analyzing massive data sets. We focus on systems that iteratively refine answers to queries and give users online control of processing, thereby tightening the data analysis process loop. You can use our techniques in diverse software contexts including decision support database systems, data visualization, data mining, and user interface toolkits.

BATCH VERSUS ONLINE PROCESSING

Traditional analysis tools have a black-box interface: The user issues queries, the system processes silently for a significant period, and then the system returns an exact answer. Because of the long processing times, this interaction is reminiscent of the batch processing of the 1960s

0018-9162/99/$10.00 © 1999 IEEE

The challenge is to trade quality and accuracy for interactive response times, minimizing uneventful dead time between outputs.

and 1970s. In contrast, Control systems have an online interaction mode: Users can control the system at all times, and the system provides intermediate feedback in the form of approximate or partial results, which are continually refined as the system processes more of the input.

We use the term "online" in its most traditional, time-sharing sense—the perception that the user is directly connected to the computer and hence can interact with it in real time. In contrast, the theory community, although it uses the term in a somewhat analogous way, understands "online" to imply a specific set of problem constraints that are not necessarily characteristic of our scenarios here. From the exterior, some Control applications (like online aggregation) can be classified as *anytime* algorithms—algorithms that can produce a meaningful approximate result at any time during their execution.[3] Some of our component algorithms are not anytime, however, in that they do not have a well-defined single result; they are lower-level operators that run indefinitely within a data flow, acting on their inputs datum by datum.

Rather than a black box, online systems operate more like a crystal ball: The user "sees into" the processing to get a glimpse of the future final results, and uses that information to change the ongoing processing, either by adjusting the running operation or by issuing further requests. This significantly tightens the data analysis loop: Users can quickly sense if a particular query or mining algorithm reveals anything interesting about the data. They can refine or halt the processing if necessary and issue other queries to investigate further.

Control applications have user interface requirements that demand a fundamental shift in system performance goals and hence system design. Traditional query-processing and data-mining algorithms are optimized to complete as quickly as possible. In contrast, online data analysis processes may never complete—they produce continuously refining approximate answers, and users halt them when answers are good enough. Therefore, Control systems do not attempt to reduce the total time to completion for a single analysis step; instead they quickly provide a rough picture, and they minimize the time that a complete information analysis session takes (typically involving several query iterations).

The challenge is to trade quality and accuracy for interactive response times, minimizing uneventful dead time between outputs. At the same time, we want the system to maximize the rate at which partial or approximate answers approach a correct answer. Optimizing for these new performance constraints requires a judicious mix of techniques from data delivery, query processing, statistical estimation, and user interfaces.

ONLINE PROCESSING SCENARIOS

Many data-analysis applications are designed to provide black-box batch behavior for large data sets. These applications exhibit frustratingly slow behavior that discourages their use. Other analysis tools either avoid batch behavior by limiting the size of the input data sets or introduce an expensive data-preprocessing phase to amortize the cost of batch processing over many queries. Both of these approaches ultimately limit the application's utility. Such problematic scenarios occur in current back-office software (SQL decision-support systems) and in applications for desktop systems (spreadsheets and online analytical processing tools), and analyst workstations (statistics packages and data-mining tools).

As alternatives to these tools, we have developed online processing systems including online aggregation and enumeration, online data visualization, and online data mining.

Online aggregation

An SQL aggregation query consists of a standard relational query—selections, projections, and joins—followed by partitioning of the output into groups and the computation of summary statistics (aggregate functions) per group. In relational database systems, aggregation queries often require scanning and analyzing a significant portion of a database. In current relational systems, these queries are executed in batch mode, requiring the user to wait a long time. Online query processing can make aggregation an interactive process.

Consider a user trying to study grading patterns, starting with the following simple SQL query:

```
SELECT college, AVG(grade)
FROM enroll
GROUP BY college;
```

This query requests that the system partition all records in the enroll table into groups by college, then return the name and average grade for each college. In an online aggregation system, this query's output can be a set of interfaces, one per output group, as shown in Figure 1. The user receives a current estimate of the final answer for each output group. In addition, the system draws a graph showing these estimates and a description of their accuracy. Each estimate is drawn with bars that depict a confidence interval, which says that with x percent probability, the current estimate is within an interval of $\pm\varepsilon$ from the final answer (x is set to 95 in the figure). The confidence slider on the lower left controls the percentage probability, which in turn affects the 2ε width of the bars. In addition, you can use the controls on the upper left of the screen to stop processing on a group or speed up or slow down one group relative to oth-

ers. These controls allow you to dynamically devote more processing to groups of particular interest.

In the example in Figure 1, the analyst realizes early on that the grades for colleges D and R are strangely high. Now more interested in these two colleges, she uses the controls to speed up their processing. After quickly getting accurate results for D and R, she can stop the query and issue another one to "drill-down" on the grades for only these colleges. For example, she can check the average grades in these two colleges in different years to see if there has been any grade inflation. Alternatively, if the first query doesn't reveal anything interesting, the analyst can issue a different query to explore another dimension of the data.

Online aggregation can deliver acceptably accurate results almost immediately—even when traditional techniques take orders of magnitude longer.[4,5] With online aggregation interactivity, users can aggressively explore data in different directions.

The obvious alternative to online aggregation is to precompute aggregation results before using the system. This is the solution of choice in multidimensional online analytic processing (MOLAP) tools. MOLAP is something of a misnomer since multidimensional processing is done offline in batch mode; the user merely navigates the stored results online. While this is a viable solution in some contexts, it is an example of constrained usage because only the precomputed queries are interactive. Moreover, these systems have trouble scaling both because of the storage costs of precomputed answers and the time required to periodically refresh those answers.

More recent work combines precomputation with sampling, storing fixed samples and summaries to provide small storage footprints and interactive performance.[6] However, these techniques don't allow users to choose the query stopping points or time/accuracy trade-offs dynamically, and they don't provide any control over the relative processing resources devoted to different groups. The performance benefits of stored samples can be integrated more flexibly with Control algorithms by means of caching and prefetching schemes.

Online enumeration: user-interface widgets

Instead of aggregating records, an online query processing system—or even a simple file system—can pump records to a client tool like a spreadsheet. For example, a user looking for evidence of discrimination at a university might request to look at all student records, and browse through them looking for correlations between GPAs and ethnicities of names.

Database systems are commonly criticized for being much harder to use than desktop applications such as spreadsheets. Because data analysts are often domain experts and not database experts, they prefer to use

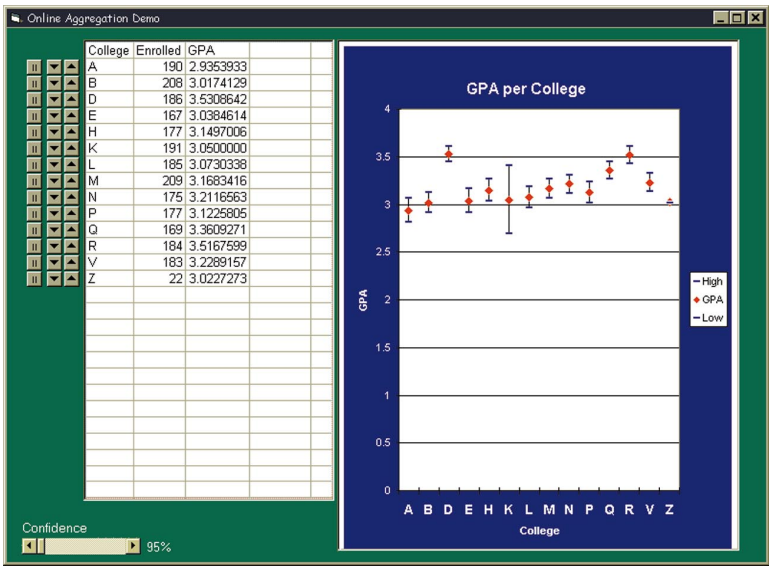


Figure 1. Online aggregation interface.

direct-manipulation interfaces like spreadsheets, in which the data is (at least partially) visible at all times. Analysts can learn a good deal about a data set by browsing a spreadsheet without issuing any complex queries. For instance, "eyeballing" a spreadsheet to find clusters of similar-sounding names is easier than posing a query to find "names of ethnicity X."

A spreadsheet allows more interactive data analysis than a traditional DBMS because it provides responses to fuzzy, imprecise queries. Suppose that a user who is analyzing student grades asks for the records sorted by GPA. The scroll bar position acts as a fuzzy range query on GPA. The user controls the scrollbar position to examine several regions (say the top few, middle few, and bottom few students) without explicitly composing different queries. The main advantage over a traditional querying interface is that the user doesn't need to prespecify a range—panning over a region implicitly specifies the range. This is important because the user doesn't know in advance what regions contain valuable information. Contrast this to the SQL Order By clause and extensions for fetching the top *N* answers, which require a priori order specification, often followed by extensive batch-mode query execution. The minutes to hours response time of large SQL queries limits data browsing and exploratory analysis.

On the other hand, spreadsheets are commonly criticized for being unable to handle large data sets gracefully. Many spreadsheet behaviors are painfully slow on large data sets, if they allow large data sets at all. Microsoft Excel, for example, restricts tables to 65,536 rows or fewer, presumably to ensure interactive behavior. The "speed of thought" response time required in point-and-click user interfaces makes it difficult to guarantee acceptable spreadsheet perfor-

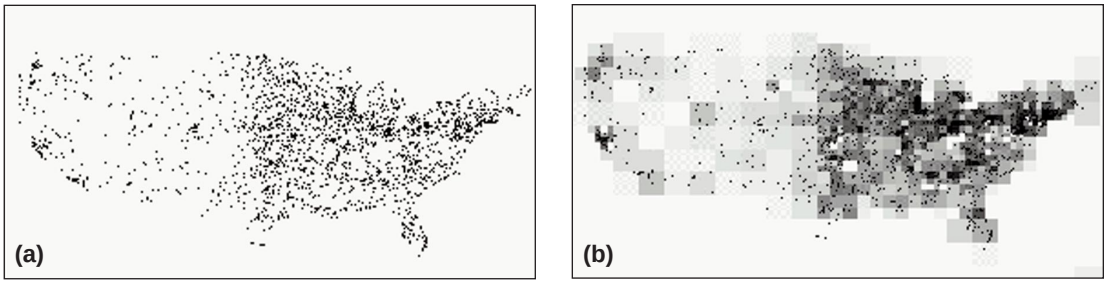


Figure 2. Partially completed visualization of US cities: (a) the result without Clouds, and (b) the effect of adding Clouds. The clouds indicate the density of cities that have not yet been plotted.

mance on large data sets.

We are investigating the integration of user-interface widgets commonly found in desktop applications with online processing algorithms—particularly access methods and reordering techniques. Our goal is to develop a Control widget toolkit to use in constructing graphical applications that have interactive response times even over massive data sets. One widget under development is a table (or "sheet") that allows interactive scrolling, jumping, prefix searching, and sorting. As one example, when a user presses a column heading to re-sort the table on that column, he almost immediately sees a sorted table with the items read so far, and more items are added as they are scanned. Actively displayed items receive preferential treatment: While the rest of the table is sorted in the background at a slow rate, items in the currently displayed range are retrieved more quickly, and they are sorted and displayed as they arrive.

Online data visualization: Clouds

Data visualization is an increasingly active research area. With interactive data exploration tools, users can quickly pan and zoom over visual canvases representing a data set to observe patterns and trends in the data.

A data-visualization system must efficiently present large volumes of information. This involves scanning, aggregating, and rendering large data sets at point-and-click speeds. Typically, these visualization systems don't draw a new screen until the image has been fully computed, resulting in batch-style performance for large data sets. This is particularly egregious for visualization systems that are expressly intended to support browsing significant amounts of data.

We have developed an online visualization technique called Clouds that provides interactivity on large data sets. One obvious solution to batch data visualization is to draw objects online as they are fetched from the database. However, the intermediate pictures this produces don't truly reflect the final image because they are missing data. Clouds alleviates this problem by performing enumeration and aggregation simultaneously: It renders records as they are fetched, and it also uses those records to generate an overlay of shaded regions of color (clouds) that estimate the missing data. The clouds are not an approximation of the image, but rather a compensatory shading that accounts for the difference between the rendered records and the projected final outcome.

During processing, the user sees the picture improve much the same way that images transmitted over a network to a Web browser become refined. However, there is a basic distinction between Clouds and traditional image coding. Images in a visualization system are constructed from ad hoc user requests of the database; therefore we don't know what the final image will be when we begin transmitting data from the database, so traditional coding schemes don't directly apply.

Figure 2 shows an online visualization of US cities with and without Clouds. Each row in the database is displayed as one black point on the screen. The shading in the Clouds version approximates the final density of areas more closely than the other version. Clouds can be particularly useful when a user pans or zooms over the results of an ad hoc query because the accuracy of what is seen is not as important as the rough sense of the moving picture. When the user ceases moving over the visual canvas, there is time to fetch and render more and more data from the visualized region, causing the clouds to gradually "lift."

As with user interface widgets, our data visualization techniques tie into data delivery, and they benefit directly from improved access methods and reordering techniques. DBMS-centric visualization tools need an online query-processing system's full power—joins, aggregations, and so on—on the back end.

Online data mining

Many data-mining algorithms make at least one complete pass over a database before producing answers. In addition, most mining algorithms require a user to tune several parameters that are not adjustable while the algorithm is running. As a result, their execution exhibits batch behavior with a long delay, during which there is no feedback. Recent work has attempted to add mining primitives to core data-

base engines.[7] This may bring the data-mining speed closer to the level of native SQL queries, but the batch behavior is still a basic algorithmic problem.

The much-studied techniques for finding association rules are a good example of the difficulty of using data-mining systems. Before you run standard association rule algorithms, you specify a minimum threshold on the amount of evidence required to produce a set of items (minsupport) and a minimum threshold on the correlation between the items in the set (minconfidence). Although these algorithms can run for hours before producing association rules, they provide no intermediate feedback—if you set these thresholds incorrectly, you have to start over. Setting thresholds too high means that the system returns too few rules. Setting them too low means that the system runs even more slowly and returns an overwhelming amount of information, most of which is useless. In addition, the analyst can't use domain knowledge to prune irrelevant correlations during processing.

The best-known algorithms for association rules use a sequence of aggregation queries, which could be adapted to use online query-processing techniques to present the results of the sequence online. With Carma (Continuous Association Rule Mining Algorithm), an alternative association rule algorithm developed in the Control project,[8] you can control the processing by changing the support threshold dynamically during the first scan of the transaction sequence. This algorithm also provides continually improving deterministic bounds on the support for different itemsets during the processing. Because Carma often has a smaller memory footprint than other algorithms, it frequently produces a final, accurate answer faster than batch algorithms. Thus, Carma has better interactivity, and, in many cases, also excels in overall completion time.

Most other data-mining algorithms (clustering, classification, pattern matching) are similarly time-consuming, but we believe that online mining techniques can be developed for these problems. Control techniques tighten loops in the knowledge discovery process,[1] bringing mining algorithms closer in spirit to data visualization and browsing. Such synergies between user-driven and automated data-analysis techniques seem like a promising direction for cross-pollination between research areas.

NUTS AND BOLTS OF CONTROL

Random data sampling is the fundamental technique for getting approximate answers to queries. Previous work on sampling in databases uses confidence intervals that are specified prior to starting the query processing. The drawback to this approach is that users have to have sophisticated knowledge of statistical methods. Moreover, specifying statistical stopping conditions at the beginning reduces the execution time, but it doesn't make the execution interactive. For example, there is no way to dynamically control the processing rate for different groups of records.

Therefore, with Control systems we continuously fetch new data at random, so that the running result of a partially completed query is computed from a random sample. The precision of the intermediate results continually improves as we scan more data, and we can decide dynamically when to stop or change the query.

Although randomized data access isn't crucial for enumeration scenarios such as spreadsheets, it is still beneficial because users prefer to see a representative sample of the data at any given time.

With Control systems, we continuously fetch new data at random so that the running result of a partially completed query is computed from a random sample.

In addition to randomly and progressively sampling from a data set, Control systems require new techniques for sampling from multiple inputs, reordering data streams interactively, and estimating graphical results.

Sampling from multiple inputs: ripple joins

SQL data-analysis queries often combine data from multiple tables, which typically requires relational join operators (there are several different "flavors" of joins). A join operator combines information from two input tables and produces an output table. Each row in the output table consists of the information of a row from the first input table combined with the information from a row in the second input table. In the simplest case, the information from a pair of rows is combined and appended to the output if the value in a specified column of the first row matches the value in a specified column of the second row. For example, one table contains rows consisting of name and address, and another contains rows consisting of name and item (purchased). These two tables can be joined via the name columns. In general, a pair of rows participates in the join if the column values for the two rows satisfy a specified logical "join predicate" (which is often an equality match). Mathematically speaking, the join of two tables R and S first forms the Cartesian product of R and S, and then selects from the result only the row-pairs that satisfy the join predicate.

Unfortunately, the fastest classical join algorithms, such as Sort-Merge and Hybrid-Hash, are inappropriate for online query processing because they scan a large portion of their input before they start returning records.

The nested-loops join is the only completely pipelining classical algorithm. In the outer loop of a nested-loops join of tables R and S, this algorithm repeatedly reads a record from R, the outer table. During the inner loop of the join, the algorithm completely scans inner table S with the record from R to form the result

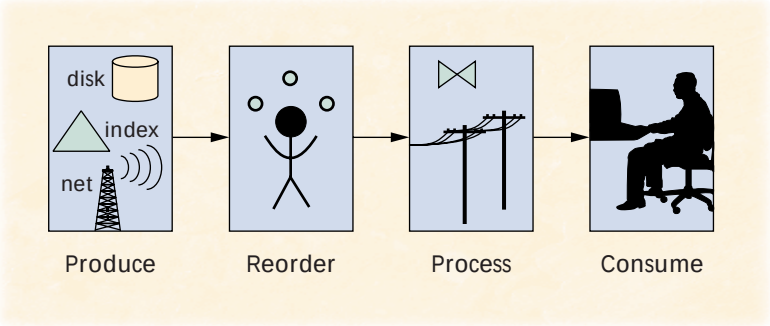


Figure 3. Control inserts a reordering operator into a data flow. The operator reorders data so that more interesting data gets processed first.

table. Although the nested-loops join is pipelining, it is typically quite slow unless there is an index to speed up the inner loop.

In an online aggregation system, a nested-loops join can return an updated aggregate estimate and confidence interval after every scan of inner table S, but its performance is typically unacceptable when the inner table is large (as in decision-support scenarios). Another problem with a nested-loops join is that it samples more quickly from the inner table than from the outer table. Depending on the data values, this sampling-rate discrepancy can result in unnecessarily wide confidence intervals, as in the error bars in Figure 1.

We developed the family of *ripple join* algorithms to address these online query-processing problems (see the "Ripple Join Algorithms" sidebar).[4] The algorithms, which generalize a number of previous join algorithms, can start returning rows of the output immediately upon invocation. The sampling rates for the input tables are dynamically adjusted to shrink the confidence intervals as quickly as possible while still satisfying interactivity requirements.

Ripple join algorithms for enumeration (nonaggregation) queries have benefits similar to sampling: A ripple join's running result arguably represents a subset of the final result because it consists of sizeable random samples from each input relation.

Preferential data delivery: online reordering

A key aspect of an online query-processing system is that users perceive data being processed over time. Thus, an online system should process interesting data early, so users can get satisfactory results quickly, halt processing, and move on to their next request. The online aggregation speed buttons in Figure 1 are one interface for specifying interest preferences for particular records. A spreadsheet's scroll bar is another interface: Items displayed at the current position are of greatest interest, and the likelihood of navigation to other scroll bar positions determines the relative preference of other items.

To support preferential data delivery, we developed an online reordering operator that reorders data on the fly based on user preferences: Interesting items get processed first, and users can dynamically change their definition of "interesting" during a query. The mapping from user preferences to data delivery rates depends on the application's performance goals.[6] Given a statement of preferences, the reordering operator reorders records in a data stream to process interesting items earlier. It aggressively prefetches data from the source and "juggles" interesting and uninteresting records using a buffer and an auxiliary disk, so that interesting items are available when the application wants them.

Since our goal is interactivity, reordering must not involve preprocessing or other overheads that increase a user's wait time. Instead, we do a best effort reordering without slowing down the processing by making the reordering concurrent with the processing. Figure 3 illustrates our scheme for inserting a reorder operator into a data flow. The data flow has four stages:

- *produce* is a source that generates data items,
- *reorder* reorders the items according to the consumer's dynamically changing preferences,
- *process* is the set of operations applied to the records, and
- *consume* captures user think time, if any.

Process involves a variety of per-record processing scenarios: running DBMS operators like joins, shipping data across a slow network, rendering data onto the screen in data visualization, and so on. Consume occurs mainly in interactive interfaces such as spreadsheets or data visualization, in which a user needs some time to absorb a screenful of information before moving to another screenful.

Since all these operations can go on concurrently, we reorder the items by exploiting the difference in throughput between the produce stage and the process or consume stages. For disk-based data sources, produce can run as fast as the sequential read bandwidth, whereas process may involve several random I/Os, which are much slower. While the items sent out so far are being processed/consumed, reorder can permute more items from produce.[6]

Online graphical estimation: Clouds

There are two key issues with using Clouds to account for data that hasn't yet been read in online data visualization. First, we must decide what color to use for the clouds to approximate the missing data. Second, we need to determine how best to partition the visualization into regions where we are going to draw individual clouds.

As we read data from the database, we maintain statistics so that we can predict the eventual number of illuminated pixels (the "density") in a given region and determine the color of those pixels. We use these

Ripple Join Algorithms

Ripple join algorithms address online query-processing problems involving multiple inputs. In the simplest version of the two-table ripple join, the algorithm retrieves one previously unseen random tuple from each of R and S at each sampling step. It then joins these new tuples with the previously seen tuples and with each other, and computes an updated aggregate estimator and confidence interval. Thus, the Cartesian product $R \times S$ is "swept out," as depicted in Figure A.

In each matrix in the figure, the R axis represents tuples of R, the S axis represents tuples of S, each position (r, s) in each matrix represents a corresponding tuple in $R \times S$, and each "×" inside the matrix corresponds to an element of $R \times S$ that has been seen so far. In the figure, the tuples in each of R and S are displayed in the order provided by the access methods; this order is assumed to be random.

The "square" version of this ripple join samples from R and S at the same rate. But sampling the "more variable" relation at a higher rate can be beneficial because it provides the narrowest possible confidence intervals for a given updating rate. Such sampling leads to a "rectangular" version of the ripple join. The optimal sampling rates depend on all of the data in the input tables, and therefore can't be determined in advance; the algorithm adjusts the sampling rates dynamically during the join based on the data seen so far.

Variants of the basic ripple join algorithm use an assortment of techniques to reduce disk accesses and to speed up the process of finding all of the rows in a table that join with a given row from the other table. In a *block ripple join*, rows are retrieved from disk in large blocks to reduce I/O costs. If, say, Table S has an index on the join-column, then the rows of S that join with a given row of R can quickly be found by searching the index; this procedure leads to the *index ripple join*. Finally, *hash ripple join* builds a hash table for each input table from the rows seen so far by using a hash function (the same for each table) to assign each row to a small "hash bucket" based on the join-column value of the row. (In this setting, a hash function is simply a mapping from a join-column value to a bucket number.) For a newly retrieved row r from, say, Table R, the hash function is then applied to the row to locate the hash bucket of S that (by construction) contains all rows of S retrieved so far that join with r.

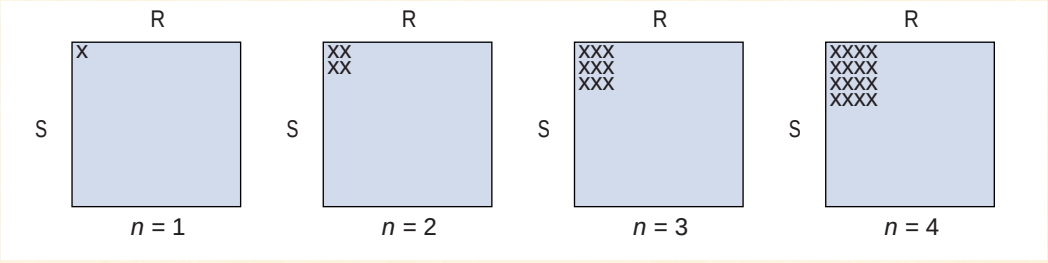


Figure A. Elements of $R \times S$ after n steps of a ripple join.

statistics to choose a cloud color that minimizes the expected error between the clouded visualization and the final visualization. We are currently experimenting with the visual effects of different error metrics such as mean-squared error.

We build and maintain a hierarchical data structure (a quad tree) for choosing granularities that partition the screen into rectangles. For a given region of the screen, Clouds chooses the granularity that incurs the least error. Clouds can choose granularities that aggregate in small or large groups, depending on the local data patterns. On the other hand, since quad-tree boundaries are strictly square and oriented independently from the data, the partition boundaries can have difficulty mirroring some data patterns such as diagonally oriented regions. We intend to experiment with other partitioning techniques such as Voronoi diagrams or image segmentation techniques used in computer vision.

CONTROL TODAY

We have prototyped Control algorithms in several freeware and commercial systems. At IBM, we are investigating the use of online aggregation techniques in the DB2 Universal Database. At Berkeley, we have implemented a number of online query-processing techniques inside the freely available PostgreSQL DBMS.[4,9] In collaborative effort between Berkeley and Informix, we have implemented a full suite of online query-processing algorithms in the Informix Dynamic Server with Universal Data Option version 9.14. We have also implemented Clouds in Berkeley's Tioga

Datasplash visualization system. Our online association rules mining algorithm has been developed as a Java application.[8] We have developed a prototype of a scalable spreadsheet for data analysis and transformation that addresses some of the user-interaction issues in Control. Further information on this project is available on the Web at http://control.cs.berkeley.edu.

LOOKING AHEAD

The Control group's focus to date has been on using continuous sampling and online data reordering to make data analysis interactive. We have just begun to study user interface issues in online data analysis. Much work remains to be done to understand and construct usable applications, especially for online enumeration and visualization.

In the systems arena, we are extending our work to parallel algorithmic and statistical contexts. For example, parallel ripple joins involve stratified sampling techniques, which affect online aggregation estimators and confidence intervals. We are investigating online query-processing optimization issues, and we believe that continuous runtime reoptimization will be especially beneficial in this context. We are also investigating techniques for handling subqueries. Many typical decision-support queries contain subqueries, for example, TPC-D benchmark queries, and some of them can't be rewritten away. Finally, we are investigating middleware approaches to Control that would not require modifying the underlying DBMSs.

We are interested in the way interactive techniques change the data-analysis and knowledge discovery process. More interactive long-running operations blur the distinction between "automated" or "intelligent" techniques such as data mining and user-driven techniques such as data visualization and OLAP. This goes beyond the obvious issues of faster algorithms to suggest more natural human-computer interactions and synergies for data analysis. ❖

References

1. U. Fayyad, G. Piatetsky-Shapiro, and P. Smyth, "The KDD Process for Extracting Useful Knowledge from Volumes of Data," *Comm. ACM*, Nov. 1996, pp. 27-34.
2. V. O'Day and R. Jeffries, "Orienteering in an Information Landscape: How Information Seekers Get from Here to There," *Proc. InterCHI 93*, ACM Press, New York, 1993, pp. 438-445.
3. S. Zilberstein and S.J. Russell, "Optimal Composition of Real-Time Systems," *Artificial Intelligence*, Apr. 1996, pp. 181-213.
4. P.J. Haas and J.M. Hellerstein, "Ripple Joins for Online Aggregation," *Proc. ACM SIGMOD Int'l Conf. Management of Data*, ACM Press, New York, 1999, pp. 287-298.
5. V. Raman, B. Raman, and J.M. Hellerstein, "Online Dynamic Reordering for Interactive Data Processing," to be published in *Proc. 25th Int'l Conf. Very Large Data Bases*, Morgan Kaufmann, San Francisco, 1999.
6. P.B. Gibbons and Y. Matias, "New Sampling-Based Summary Statistics for Improving Approximate Query Answers," *Proc. ACM SIGMOD Int'l Conf. Management of Data*, ACM Press, New York, 1998, pp. 331-342.
7. S. Chaudhuri, "Data Mining and Database Systems: Where Is the Intersection?" *Data Eng. Bull.*, Mar. 1998, pp. 4-8.
8. C. Hidber, "Online Association Rule Mining," *Proc. ACM SIGMOD Int'l Conf. Management of Data*, ACM Press, New York, 1999, pp. 145-156.
9. J.M. Hellerstein, P.J. Haas, and H.J. Wang, "Online Aggregation," *Proc. ACM SIGMOD Int'l Conf. Management of Data*, ACM Press, New York, 1997, pp. 171-182.

Joseph M. Hellerstein is an assistant professor in the Electrical Engineering and Computer Science Department at the University of California, Berkeley. His research interests include interactive data analysis, adaptive and federated computing systems, indexing and indexability, query optimization and execution, and parallel I/O and database systems. Hellerstein received an AB from Harvard University, an MS from the University of California, Berkeley, and a PhD from the University of Wisconsin, Madison. He is a Sloan Foundation Research Fellow, a member of the ACM SIGMOD Advisory Board, a member of the ACM and ACM SIGMOD, and an affiliate of the IEEE Computer Society. Hellerstein is a cofounder and chief scientist of Cohera Corporation, a vendor of data federation systems.

Ron Avnur is a member of the engineering staff of Cohera Corporation. He received an MS in computer science from the University of California, Berkeley. His research interests are in online query processing and optimization. Avnur is a member of the ACM and ACM SIGMOD.

Andy Chou received a BS in computer science from the University of California, Berkeley, and is a graduate student in computer science at Stanford University. His research interests are in database systems, computer languages, and networking. Chou is a member of the IEEE and the ACM.

Christian Hidber received a PhD from the Swiss Federal Institute of Technology, Zurich. He recently completed a postdoctoral internship at the International Computer Science Institute, Berkeley, California. His research interests are data mining, algorithm design, and computer algebra.

Chris Olston received a BS in computer science from the University of California, Berkeley, and is a graduate student in computer science at Stanford University. His research interests are in database and data-visualization systems. Olston received the 1998 Computing Research Association Undergraduate Research Award.

Vijayshankar Raman is a graduate student in the EECS division at the University of California, Berkeley. His research interests are in online data analysis and transformation. Raman received a BS from the Indian Institute of Technology, Madras. He is a member of the ACM and ACM SIGMOD.

Tali Roth is a staff member at Microsoft. She received a BS in computer science from the University of California, Berkeley. Her research interests include user interface specification and database systems.

Peter J. Haas is a research staff member at IBM Almaden Research Center. His research interests center around the application of techniques from applied probability and statistics to the design and performance analysis of computer systems. Haas received an MS in statistics and a PhD in operations research from Stanford University. He is a member of INFORMS and an associate editor of Operations Research.

Contact Hellerstein at the Computer Science Division, University of California, Berkeley, 387 Soda Hall #1776, Berkeley, CA 94720-1776; jmh@cs.berkeley.edu.