Research Statement – Alan Halverson

We live in an increasingly connected world. Many people have high speed, always-on Internet access at their homes, and Google is a verb in nearly every language on the planet. Our access to multiple streams of information in digital formats has the potential to keep us in touch with world events and far-away loved ones. We have access to this information on multiple devices, such as wired desktop computers at home and work, and wireless laptop computers, PDAs and cell phones. The promise of these connectivity options is simple enough – to enable access to information when and where we need it. While this may be true, I see two additional trends developing. First, high-speed access to data and information has not reduced how much time people spend using the Internet, but rather has allowed them to view more information in the same amount of time. Second, tools for helping users manage interconnected streams of information have lagged behind access to the information itself. A modern web browser provides a history of sites visited, but the list is flat and provides only a day-by-day grouping of sites. Missing is the hierarchical discovery of the information and the information itself.

The Internet is a hierarchy of links and information, and our minds work in a similar fashion. While Google is a good model for cutting across hierarchy to find information quickly, the flat result list model does not match how we discover and remember information. In fact, hierachical data objects are pervasive in information management – a customer and their orders, a grouping of sales figures by continent, country, and region, a computer filesystem, and so on. Developing techniques for efficiently managing information hierarchies is critical to bridging this gap.

My research interests concern optimizations for storage and query processing of hierarchically organized data. I have worked on three projects related to my interests while a graduate student: a novel mixture of structure- and navigation-based approaches for evaluating path expressions in a native XML query system, a system for processing relational queries over XML data, and a comparison of storage optimization techniques for relational schemas in the context of row-oriented and column-oriented storage for a read-mostly query workload. The next three sections briefly describe each project. I conclude with a description of near- and long-term plans for future work.

XML Query Processing

My first exposure to research at the University of Wisconsin-Madison came in the context of the Niagara native XML query system. By the time I became a research assistant on the project, much of the system was already functional. XML is a hierarchical data model, and each document may have an arbitrary schema. We do not know *a priori* whether the document will be shallow or deep, broad or narrow, or some combination thereof. Further, arbitrary path expressions may be evaluated over the data after it is loaded into the Niagara system. Therefore, two kinds of data structures are maintained for each document loaded. First, a depth-first tree structure keyed on the order of elements provides efficient navigation through and reconstruction of the document.

Second, an inverted list for each unique element of the document, also in document order, provide for efficient structural joins.

In my work, I observed that path expressions utilize two primary relationship axes: parent-child and ancestor-decendant. The tree structure tends to be more efficent for parent-child axes, while the inverted lists tend to perform better for ancestor-decendant – but these are heuristics, not rules. I therefore developed a cost model to choose the best combination of structural joins and tree navigation for a given path expression, dependant on several statistics which are easy to gather as the document is loaded by the system. The cost model predictions were validated against the Niagara system and found to be accurate.

Relational Over XML

In the summer of 2003, I had the opportunity to work with Guy Lohman at the IBM Almaden Research Center. At the time, they had also built a prototype of a native XML storage system. For my project, I evaluated a method for evaluating relational queries over XML data stored natively. I called this method ROX, or Relational Over XML. In some circumstances, XML documents will conform to a regular, repeating schema – in fact, the data may have been stored in a traditional relational database and published to conform to the desired XML schema. In this case, defining a virtual table definition, or "nickname", of the data in terms of path expressions over the XML schema is straightforward. ROX works by translating references to these nicknames into XPath expressions, evaluating each expression over the XML data, and translating the XML result back into a relational rowset.

One key observation of this work is that XML allows related data to be stored together in a "de-normalized" fashion. For example, a Customer element may be the parent of several Orders elements. When referential integrity guarantees this one-to-many relationship, we can create a relationship between the nicknames that define the relational view for each Customer and Order based on the structural relationship between the two. In this case, a relational join between Customer and Orders can become a simple, navigation-based scan of the XML document. We evaluated this idea in the XML prototype system using several variations of the TPC-H benchmark schema and discovered that obviating such joins allowed the translated queries to run within 1/2 order of magnitude of the optimized standard relational execution time, despite obvious inefficiencies in the XML prototype.

Storage Optimizations For Read-mostly Query Workloads

Relational data management systems store data as complete rows of information. Such a design provides good performance for query workloads which feature a mixture of reads and writes to the database. Recently, Stonebraker et al. proposed a column-oriented system called C-Store that is optimized for a "read-mostly" query workload. Their evaluation shows incredible performance benefits when compared to a traditional relational DBMS. Motivated by their work, I set out to design a set of storage and query optimizations for a row-oriented system. I designed two storage improvements – "super tuples" and "column abstraction". Super tuples simply pack many relational rows into a single storage block the size of a disk page, and later using a secondary lightweight iterator to extract each logical tuple. Column abstraction utilizes data ordering techniques – both for arbitrary sort columns as well as ordering by the columns in the one side of a one-to-many join in a materialized view – to store repeating data only once to save disk space. I implemented a prototype system that applied these optimizations to both the row- and column-oriented architectures and found that row storage is performance competitive with column storage for most sequential scans. I also developed a cost model which breaks down the total scan cost into disk I/O, iteration cost, and local tuple reconstruction cost. The cost model correctly identifies the trends and relative performance of each storage optimization and storage choice.

Plans For Future Work

The storage optimizations proposed in my most recent work clearly provide benefits for read-mostly query workloads. A natural next step is to design and build a storage selection "wizard". Given an input query workload and a logical relational schema, two approaches are possible. First, hold the choice of storage (row- or column-oriented) and optimizations constant and choose a set of materialized views which cover the query workload while providing the best performance. Second, hold the set of materialized views constant and choose the best combination of storage and optimizations. Note that the best storage choice may be a combination of row- and column-oriented structures. Adding a small update workload into the equation would likely change the decisions made by the wizard.

Hierarchically-organized data is pervasive in data and information flow. I would like to investigate techniques for harnessing the flow of news and information on the Web and providing both a personal and scalable lineage and provenance record for near and long term recall purposes. For example, RSS feeds have become a widely used method for aggregating news and personal interest information. Unfortunately, these views are extremely transient – all of the story links may be gone within a day. Some limited historical archiving is possible with current RSS aggregators, but they cannot answer contextual questions. For example, which sites did I visit as a result of following an RSS link? Which unrelated subjects were interesting to me in the same time frame? At the Internet scale, can we trace the spread of news through RSS by following links back to the source? The provenance of news can provide us with a metric for establishing a "seed-rank" to establish each feeds reputation for either breaking news or simply linking to others. The graph created may also discover cycles which expose sites that cite each other as references to establish credibility. Although developing techniques and systems to answer these questions may be years away, I believe that my techniques for efficiently storing and querying hierarchically-organized data will play an important role.

Conclusion

Mountains of data are being generated every day, and our ability to manage this data continues to be a challenge. My work focusing on the management of hierarchically organized data can help with a subset of the problem. I would like to continue developing techniques for managing data and systems to prove that these techniques are viable.