The Cornell Jaguar Project: Adding Mobility to PREDATOR

Demonstration

Philippe Bonnet, Kyle Buza, Zhiyuan Chen, Victor Cheng, Randolph Chung, Takako Hickey, Ryan Kennedy, Daniel Mahashin, Tobias Mayr, Ivan Oprencak, Praveen Seshadri, Hubert Siu
http://www.cs.cornell.edu/database/jaguar
Cornell University
Ithaca, NY 14853

ABSTRACT

The Cornell Jaguar Project is exploring a variety of issues related to mobility and query processing. One broad theme is to break down the traditional client and server boundaries, leading to ubiquitous query processing. Another theme is to extend database and query processing techniques to small-scale and mobile devices. The project builds on and extends the Cornell PREDATOR database engine.

Keywords

Mobile computing, ubiquitous query processing.

1. ISSUES

The Cornell PREDATOR system is a full-fledged object-relational DBMS that developed efficient database extensibility mechanisms. PREDATOR was demonstrated at SIGMOD 1997. The Jaguar project extends PREDATOR with support for mobile and portable execution. The demonstration includes the following features:

a) The use of Java user-defined functions (UDFs) that can be developed at the client and shipped to the server where they are executed within SQL queries.

b) The transparent execution of Java functions at the client (instead of the server), along with visualizations of query optimization decisions that direct the choice of execution algorithm [MS99].

c) The use of novel combinations of compression techniques to effectively shrink the size of query results that need to be shipped between the database server and mobile clients [CS99].

d) The incorporation of small-scale devices (sensors, actuators, smartcards, etc.) into the database system, including the ability to execute partial, long-running queries over a mobile and disconnected collection of devices.

e) The use of handheld computers in a data-intensive “telepresence” application.

2. BACKGROUND

2.1 Extensibility in an OR-DBMS

Extensibility is an important goal of object-relational database systems (OR-DBMSs). Two important aspects of extensibility are the ability to add new functions, and the ability to add new types for complex data. Support for complex data types in OR-DBMSs is based on Abstract Data Types (ADTs) and user-defined functions (UDFs).

2.2 Security and Ease of Extension

The extensibility of OR-DBMSs usually comes at the cost of security and reliability. Whenever new code is added into the server, it has the potential to

(a) crash the server,
(b) interact undesirably with other server code,
(c) corrupt the server machine,
(d) monopolize resources, degrading performance,

It has therefore usually been the assumption that database extensions are written by ``database developers'' who are trained and trusted persons. While this assumption may be valid in controlled environments, it is certainly not true when the user community is distributed over the WWW. For instance, consider a database of stock market data, made available to investors across the WWW. Each investor might wish to specify his/her own prediction functions to run in queries against the data. In such a scenario, secure extensibility is an important consideration. Another concern is scalability, and how the database system behaves as hundreds or thousands of users try to use new extensions.

2.3 Overview of PREDATOR

The Cornell PREDATOR system is a client-server OR-DBMS. Data types are modeled as Enhanced ADTs (E-ADTs) [Sesh98]; the enhancements involve the specification of optimization semantics for methods (this was demonstrated at SIGMOD 1997). Several E-ADTs have been implemented, including images, audio, video, documents, and geographic types. Database clients may be implemented in any programming language; currently C++ and Java clients have been implemented. Java clients can run as applets within a browser anywhere on the WWW. Since Java is a portable language with security features, it seems an ideal choice for database extensibility. We have implemented mechanisms for the database server to be extended with Java functions. These functions can be tested on the client, and then migrated to the server.

3. CONTENT OF DEMONSTRATION

3.1 Server Extensibility using Java

We demonstrate the ability to extend PREDATOR with user-defined functions written in Java. The user develops and tests these functions on the client site, and transparently migrates them to the server site, where they are used within queries. This form of extensibility is secure, portable and easy to use, without losing too much by way of performance.

This functionality is demonstrated in the context of a realistic application involving financial data management.

3.2 Client-side Java UDFs

When there are a very large number of users, scalability issues dictate that UDFs stay on the client-site, rather than migrate to the server. UDFs may also use some client resources that should not be shipped to the server. However, we would still like the UDFs to participate in queries. We have developed distributed database algorithms to apply these client-site UDFs, and query optimization algorithms to incorporate these algorithms into the PREDATOR server. There are interesting network utilization tradeoffs between the algorithms. We demonstrate these using a client connected to the server across a slow connection.

3.3 Result compression for clients

Results of queries are typically shipped to clients for display to users. The client applications extract large amounts of data from the database by running queries and apply complex analyses to the query result. We have developed algorithms that compress query results utilizing the semantic information available in the query. These result in compression ratios that are 75% higher than standard compression algorithms like LZW. High compression ratios are critical when network bandwidths act as the bottleneck between client and server, or when the memory resources of the client are limited. We demonstrate these techniques by using a handheld device to access financial data. The compression techniques used vary from query to query, and are determined automatically using a "compression optimizer".

3.4 Device Databases

A new and very interesting form of mobile computing involves small-scale devices like sensors, actuators and smartcards. We have developed a data and system architecture that integrates these devices with a regular object-relational database system like PREDATOR. Our demonstration uses JavaRings and sensor devices. We show queries that run partially over the central database and partly within the devices. The interesting issues deal with the mobility and intermittent connectivity of the devices. We consider devices that are often disconnected as well as devices that provide asynchronous “event” data.

3.5 Handhelds and Scalable Telepresence

We also demonstrate the use of palmtops and handheld computers as agents of telepresence. We imagine a scenario in the not-too-distant future where all our environments are compute-rich. This means that we can interact with computers in the office, walking down the corridor, in the elevator, in the car, on the plane, in the taxi and in the hotel room, without any seeming discontinuity. The key to such seamless interaction is, of course, the ability to transfer state along with the mobile individual. We propose that handheld computers act as the bearers of this state information. We demonstrate this concept using simple applications. In the long term, we expect this to evolve into another variant of ubiquitous query processing, where a query computation “moves” with the end-user across a variety of environments.

4. REFERENCES

[CS99] Z.Chen and P.Seshadri. An Algebraic Compression Framework for Query Results. *Submitted to publication*, 1999.

[MS99] T. Mayr and P.Seshadri. Client-Site Query Extensions. In *Proceedings of* ACM SIGMOD ’99 *International Conference on Management of Data*, Philadelphia, PA, 1999.

[Sesh98] P.Seshadri. Enhanced Abstract Data Types in Object-Relational Databases. In *VLDB Journal* 7(3), 1998.