Hybrid Shipping Architectures:
A Survey

Ivan Bowman
itbowman@acm.org
http://plg.uwaterloo.ca/~itbowman

CS748T
14 Feb 2000
Outline

• Partitioning query processing
• Partitioning client code
• Optimization of query plans
• Mobile code approaches
• Conclusions and future work
Introduction

• RDBMSs partition applications into a relational portion and procedural portion
• Recent advances have shown how to distribute the relational query processing
• It may also make sense to distribute procedural processing of client programs
• This is possible today, but it is very hard!
Partitioning System Functionality

- Split relational and procedural processing
  - Relational processing can be optimized and executed efficiently
  - Procedural code is flexible and more powerful
  - This gives a clean architecture, with nice separation of concerns

- Relational system provides transparency to client application
Single Processor System

Program One -> Access Module -> Storage Access

Access Module -> Storage Access

Access Module

Program Two
Using Multiple Computers

• An early idea: Client/Server
  – Data and DBMS on one machine (Server)
  – Client application on another (Client)
• Client sends queries to Server, gets results
• Client resources really only used for client application
• Server is a bottleneck
Query Shipping Client Server

Shared Data

Server DBMS

Communications Software

Network

Communications Software

Client Application

Client

Queries

Tuples
Distributed Client Server

• To support more clients, we can:
  – Buy a bigger server
  – Add more servers and partition the data
• Adding more servers is likely cheaper
• Requires data partitioning, distributed query processing, distributed concurrency, …
  – (all of which we’ve talked about in class)
Distributed Client Server
Heterogeneous Processing

- Servers may be heterogeneous
- *Wrappers* can provide a uniform view of servers
- Leads to distributed query processing with some processing on client (at a minimum, what the servers could not handle)
Mediator Architectures
Data Shipping

- Do all query processing at client
- Cache data on clients
- Ship data in from server as needed:
  - Object
  - Disk pages
  - Groups of objects
  - Hybrid of the above
- Typically used by OODBMS
Data Shipping Client Server

*Delis et al.*

![Diagram of Data Shipping Client Server]

Objects, Pages, or Clusters → Local Cache → Client DBMS → App. → Fetch Requests → Network → Server

Shared Data

Comm. Software

Comm. Software

Server

Client
Hybrid Shipping

- Query shipping under-uses client resources
- Data shipping under-uses server resources
- Instead, use *Hybrid Shipping*
- Query processing at client *and* at server
  - Gives a form load balancing
  - Can reduce data movement for data inflating or reducing operators
  - Client caching can be used effectively
Data Shipping Client Server

*Delis et al.*
Splitting Client Code

• Original systems partitioned apps into relational queries and application code
  – This provides a clean architecture

• Distributed systems originally split execution sites along these dimensions
  – This non-optimal splitting has been addressed for queries
  – The problem remains for application code
Why Move Client Code?

• To take advantage of a powerful server
• To reduce query processing costs (e.g. with selective user functions)
• To minimize network communication costs by executing code closer to the data
sql = "SELECT emp_id, emp_name "
    "FROM EMP";
rs = stmt.executeQuery( sql );
while( rs.next() ) {
    if( isPrime( rs.getInt(1) ) ) {
        System.out.println(rs.getString(2));
    }
}

User Defined Functions

sql = "SELECT emp_id, emp_name "
    "FROM EMP WHERE isPrime(emp_id)";
rs = stmt.executeQuery( sql );
while( rs.next() ) {
    System.out.println(rs.getString(2));
}
Client Code on the Server

- DBMS vendors allow:
  - User-defined functions used in queries
  - Stored procedures allow arbitrarily complex procedural code to be executed at the server

- So, what’s the problem?

It's TOO HARD!
Getting Code On the Server

- Must be ‘multi-lingual’
  - Server environment doesn’t match client’s
    (Java in the database simplifies this somewhat)
- Partitioning decision made by the developer
  - Must decide early on (design phase)
  - Programmer intuition is often wrong
  - Cannot easily tune to new systems
  - Cannot adapt to dynamic workload
Partitioning Code: Past Experience

- ICOPS (Brown) and CAGES (North Carolina) automatically partitioned graphics applications between a mainframe host and a ‘satellite’ graphical terminal
CAGES and ICOPS

- Coding for Host/Satellite systems required bilingualism
- Programmers often made incorrect partitioning choices in the design stage
- Design-time partitioning led to vendor lock-in
- Configurable programs addressed these issues
  - Provided a run time that could *monitor* costs
  - Allowed either run-time or compile-time partitioning
Code Shipping: MOCHA

- Allow relational operators and client code to be executed either at client or any server
- Code that is not present is shipped to the appropriate location
- All code that may be shipped implemented as a static function and described in XML
- Optimized only for network costs
Optimizing Distributed Plans

• Optimizer must choose:
  – Access path (e.g. sequential, or an index?)
  – Algorithm for physical operators
  – Join order
  – Expensive predicate placement
  – Intra-query parallelism
  – Execution site for each operator
Optimizing Expensive Predicates

• Rank-order approaches: selectivity and cost
• Some approaches increase join degree
  – Dynamic programming can not handle high join degree (say, higher than 15)
  – Randomized, greedy, or branch-and-bound algorithms may be more effective up to 50-100
• Existing approaches do not support site selection for expensive predicates
Mobile Code

- User defined functions and stored procedures are not mobile
- We would like an approach that can place code dynamically based on system statistics
- Load and run, or on-the-fly mobility
Coign

• Coign distributes binary applications written using COM

• Classifies components using training runs
  – Components with similar access paths will be placed near each other

• A commodity flow network is used
  – Max-flow, min-cut graph cutting optimization
Component Communication*
Coign’s Experience

- It’s possible to optimize distribution of existing systems
- Non-distributable components constrain the optimization
- Applications designed for distribution were optimized better
Abacus

• Abacus project at CMU also used ADP
  – Provides primitives for data-intensive, distributable components in C++
  – Partitioned during execution based on statistics
  – Moved components using checkpoint/restore

• Partitioning is resistant to bad access plans, since it dynamically adapts to system load
Languages

• We want:
  – Mobility
  – Safety
  – Security
  – Portability
  – Efficiency

• Java is promising; but, is it fast enough?
• Software fault isolation may be faster
• Many other languages are available
Conclusions

• Recent advances have shown good ways to partition the cost of relational processing
• Similar work can be done for user code
  – Automatically detect code that can benefit from partitioning
  – Optimize the partitioning of many functions
  – Execute user code on server efficiently, safely, securely, and transparently
Questions?