CMSC 818S, 03 March 1998

Performance Tradeoffs for Client-Server Query Processing

Franklin, Jonsson and Kossman

Appears in Proceedings of SIGMOD 96

Existing Database Systems

-RDBMS: high level associate access (using some contraint formulas/predicates) to flat data records

-ODBMS: supports data modelling (objects) & efficient navigation-based access

-Hybrid DBMS (i.e., Ilustra, UNISQL)

+ Integration of object & relational concepts (at language & data model levels)

+ Problem: no support at architectural level (in a distributed environment)

-Architectural support needed b/c each system exploits resources differently for data query processing

Query Execution on RDBMS

Query shipppping: the bulk of query processing is done at the server and results are communicated back to the client. Benefits:

+ reduce communication

+ utilize server resources more efficiently

+ tolerate minimally configured clients (minimum memory, "slow" cpu)

Query Execution on ODBMS

Data shipping: data is *faulted* in (and possibly cached) the to client where query operators are executed. Benefits:

+ exploit client resources (assuming large memories, ”fast” cpus, large & fast disks)

+ reduce communication for large queries & access locality (use cache)

+ client/server systems ”scales” better

Query Execution on Hybrid DBMS

-Paper considers combining the best of both pure policies for *hybrid-shipping*

-Hybrid-shipping's disadvantage is a possible increase in complexity in query optimization:

+ larger search spaces (to find an acceptable query plan)

+ sensitive to optimizer's cost model & changes to query environment state

Query Execution Plan

-Represented by a binary tree where nodes are *operators*, edges represent *producer-consumer relationships*

-Specifies ordering of operators, placement of operators at sites (client, server) and methods for executing operators (i.e., join-type, etc.)

-Each execution policy (data-,query-,hybrid-shipping) differs in placement of operators at sites: *site-selection*

Operators

Annotated with site (logical site: "producer", "consumer", "client", etc.).

Bound at run-time.

Types and annotations:

Display: root of query plan tree

Join: annotated with *consumer*, *inner-relation* and *outer-relation*

Select: annotated with *client* (site: parent), *producer* (site: child)

Scan: annotated with *primary copy* (server), *client* (client cache)

Note: operators with logical sites can execute on "same" physical site.

Query Optimizer

Study uses 2-phase randomized query optimizer (2PO) of Ionaddis, et al.

2PO chosen based on:

+ ability to successfully find good join orderings

+ simplicity for easy implementation

+ small computation overhead for complex queries

+ ability to generate "reasonable" plans, not "optimal" ones

Query Optimizer (cont'd)

2PO selects a random plan and uses *simulated annealing* and *iterative improvement* to improve plan

Possible transformations:

+ Changes join orderings of base or temporary relations
+ Change join *site* annotation to *cons*, *out rel*, *inn rel*
+ Change select *site* annotation to *cons*, *prod* or vice-versa
+ Change scan *site* annotation from *client* to *primary* or vice-versa

Cost Model for Optimizer

-For a query plan and given system configuration, estimates:

+ response time (based on model in Ganguly, et al.)

+ total cost: function of method used to execute plan

  sequential (response time = total cost)

  parallelism - independent subtrees

  pipelined

Experimental Platform & Parameters

-Simulation based on CSIM; models heterogeneous peer-to-peer db (SHORE)

+ includes detailed query processing costs

+ configured to model client-server architecture (one client, mult. servers)

Clients vs. servers

+ possses similar resources (disk, mem, cpu, buffer manager & query processing engine)

+ all displaying done on client

+ client has disk cache; no primary copies of data (managed by servers only)

Experimental Platform & Parameters (cont'd)

-Query operation based on iterator model in Graefe: *open-next-close*

open: initialization

next: retrieve tuples from children/yield result tuples

close: cleanup

-Demand driven dataflow

Workload & Metrics

-Workload:

+ Benchmark queries: 2-way and 10-way chain joins w/ moderate selectivity

+ Selectivity: join relation produced is same cardinality of base relation

-Metrics (minimized by optimizer): 1) communication cost (pages xmitted), 2) response time

Comments on Experimental Environment

-Total amount of memory allocated to joins varies from minimum to maximum

+ minimum alloc: inner & outer relations are partitioned

+ maximum alloc: build entire inner relation hash table in memory

-Memory buffers at clients are empty (no data cached across queries for max alloc)

-Disk I/O required at client & server to read base relation

-Data cached at client initially resident on client's local disk

-Disk utilization on server & client is varied by issuing random I/Os

Results for 2-way Joins

-Communication Cost: HS minimized communication volume (best of both pure policies)

-Response (min. alloc):

+ QS: scan I/Os interfere with join I/Os (on server)

+ DS: as more data cached at client, disk I/Os become a factor

+ HS: able to minimize disk contention (cache & scan I/Os)

-Response (vary disk util)

+ HS & DS: caching in client can be beneficial when server's disk util is high (client contention)

Results (cont'd)

-Response (max alloc.)

+ QS: constant response time; overlap of join processing and communication at server

+ DS: response time decreases as amt of data cached increases

+ HS: poor planning by optimizer prevented best response time of pure policies

-Response (vary disk util)

+ Similar results for min allocation no interference at disk from joins & scans

Results for 10-way Joins

-Multiple servers entailing server-to-server communication

-Inter-operator parallelism (independent parallelism) & pipelined parallelism exploited

-Randomly placed relations

-Communication Cost (no caching)

+ DS: Constant; must fault in all pages

+ QS: Varies as #servers change; cost of inter-server comm. increases

+ HS: Best of both policies (at 10 servers, communication costs dominate)

-Nonlinearity in HS & QS due to join order being independent of relation placement

Results for 10-way Join (cont'd)

Communication Cost (5 relations cached):

+ DS: need only fault in remaining relations

+ QS: no caching; same as no-caching case

+ HS: exploits cached relations to minimize communication; joins can be done at sites w/ common join attribute

-Response (min alloc, no caching):

+ DS: constant; single server better than QS (disk contention)

+ QS: able to exploit disk parallelism;

+ HS: able to distribute join processing across both client & server for smaller configurations

Query Optimization

-Costs for optimizing plans per operator execution is expensive

-Pre-compiled plan can be sub-optimal w/o knowledge of run-time dynamics

-Paper considers using:

static optimizer

2-step optimizer

-Investigate performance w/ lack of runtime information (caching, etc.)

-Quality of plans is focus, not site-selection overhead

Communication Cost Example

-At compile time: relation A, B at Server 1, relation C, D at Server 2

-At runtime, C migrates to Server 1, A migrates to Server 2

-A static, compiled plan would need to do 2x more communication for 4-way join

-A 2-step plan could account for migrating relations and dynamically change operator sites; still incurs 50% more communication

-A 2-step can potentially take advantage of cached copies of data (at runtime); since migration is less likely, could be beneficial

Response Time

-Attempt to exploit multiple servers

-Join graph shape must be taken into account: *left-deep* and *bushy*

Results (min. alloc, 10-way join):

+ Deep static pays high join penalty (executed on single site)

+ Deep 2-step can distribute join processing at runtime; cannot exploit inter-operator parallelism

+ Bushy static suffers for low #servers & fails to redistribute load evenly

+ Bushy 2-step performs close to ideal across all server configurations

Response Time

-For high-selectivity join-query, bushy plans suffer for small number of server

-Exploits parallelism as number of servers increase

Conclusions

HS matches performance of pure policies and outperforms them in some instances

HS accomplishes this by making use of client resources for caching or operator execution

HS introduces challenges for query optimization; a 2-step optimizer was investigated

2-step was shown to perform well for bushy and deep join graphs vs. static plans