CS856

ObjectGlobe: Ubiquitous query processing on the Internet

R. Braumandl, M. Keidl, A. Kemper, D. Kossmann, A. Kreutz, S. Seltzsam, K. Stocker

Presented by: Yasemin Ugur

02/16/05

OUTLINE

- Introduction
- ObjectGlobe system overview
- Generating query plans
  - Lookup Service
  - Parser
  - Optimizer
- Executing query plans
  - Distributing and executing
  - Monitoring
- Experiment Results
- Summary
- Comments

Introduction – Motivation

- Electronic marketplaces and virtual enterprises on internet have become very important applications for query processing

- The companies can sell or offer their services (not only data but also query processing capabilities) in open electronic marketplace

- Building this type of environments requires a highly flexible/open and distributed query processing

Introduction

- Challenges for query processing and optimization
  - Open environment (easy to add a new participating enterprise) with large number of participants with different capabilities
  - Schema integration (and maintenance of integrated schema) over various heterogeneous data sources
  - Provision of privacy and security of the data
  - Management of quality of services so that user can constrain cost and time over large sources

ObjectGlobe System

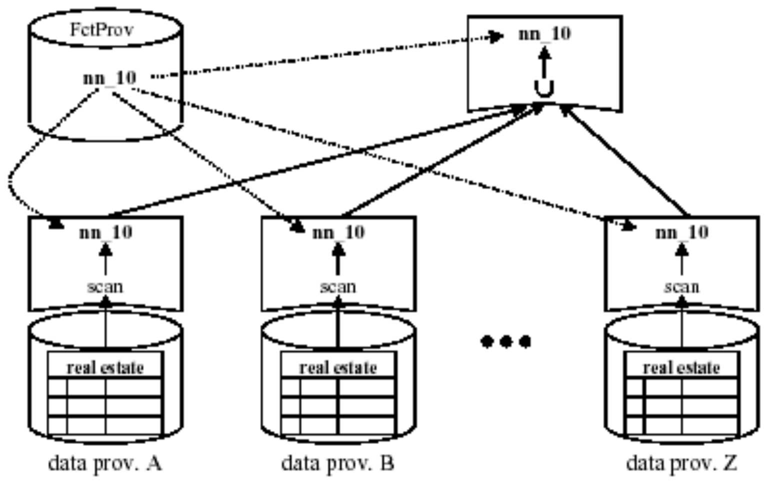
- ObjectGlobe system aims to provide an infrastructure that can :
  - Enable clients to execute complex distribute query processing across the participating internet sites
  - Enable participating sites to offer not only their data provision capabilities, but also query processing capabilities (e.g. implementation of a specific query operator, hardware resources)
- This paper does not talk about the schema integration in distributed and heterogeneous environment, and assumes the existence of
  - A meta-schema that specifies all relevant properties of services (e.g. data sources/collections)
  - Wrappers/access-methods to access data collections and their distribution and other statistics

OUTLINE

- Introduction
- ObjectGlobe system overview
- Generating query plans
  - Lookup Service
  - Parser
  - Optimizer
- Executing query plans
  - Distributing and executing
  - Monitoring
- Experiment Results
- Summary
- Comments

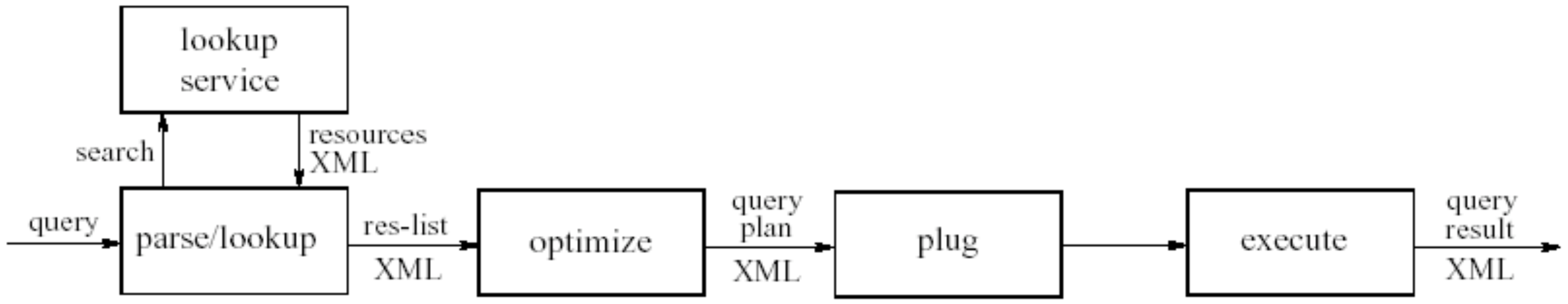
ObjectGlobe System Overview

- ObjectGlobe system support three different type of supplier participants
  - Data providers supply data collections
  - Cycle Provider supply hardware resources to execute the query (e.g. CPU, memory)
  - Functions provider offer query operators to process data



System Overview

- ObjectGlobe system consists of 4 major components to process a query:
  - Lookup Service
  - Parse/Optimize
  - Plug
  - Execute/Monitor



System Overview

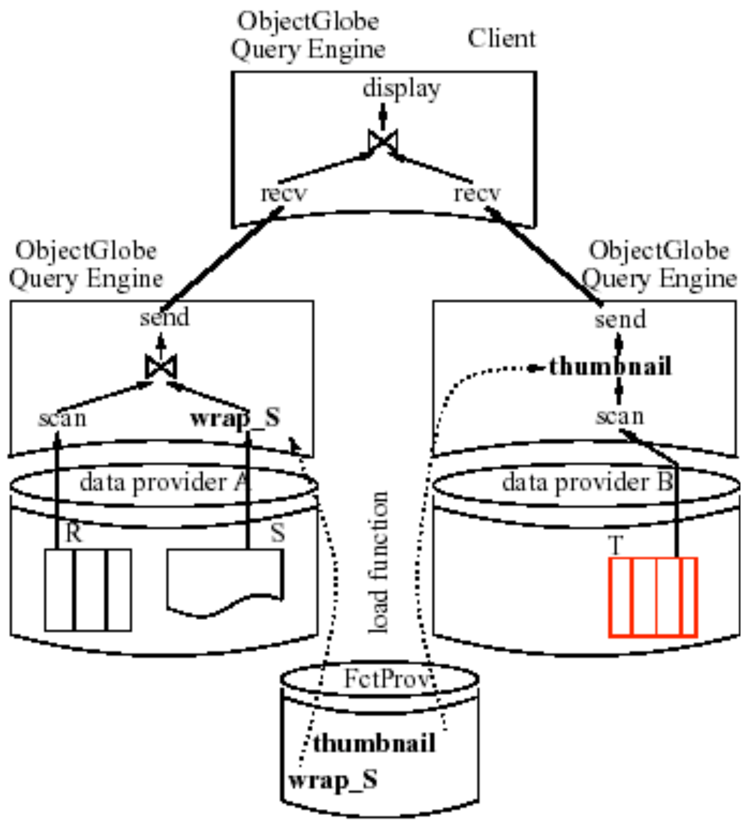
- Lookup Service:
  - Used by providers to register meta-schema that describes services
  - Used by parser to find
    - Relevant data sources, cycle providers, and operators for the given query
    - Authorization information to determine what restrictions apply for processing the given query
- Parse/Optimize:
  - Parser reads the query (written in a subset of SQL), and find the required resources and authorization information from the lookup service
  - Based on the information obtained from parser, the optimizer tries to find a "good" execution plan for the given query that meets user's quality requirement

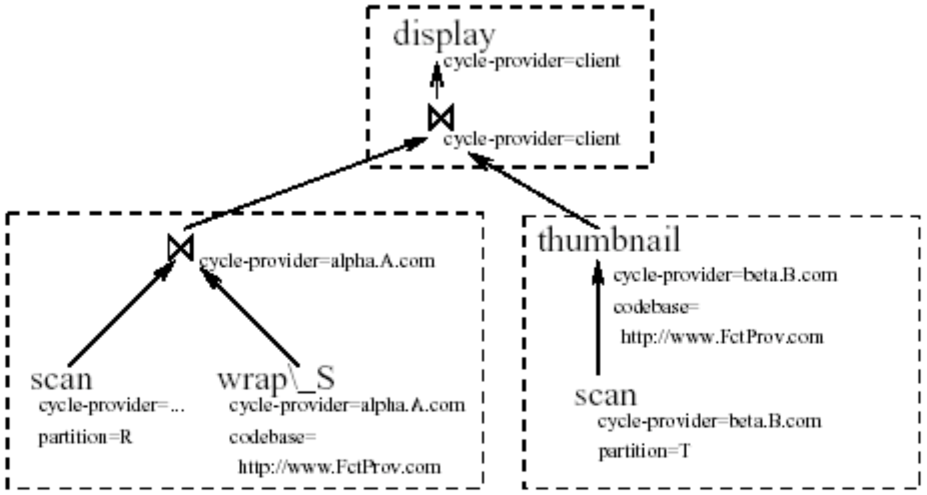
System Overview

- Plug:
  - Using the query plan generated by the optimizer, distributes sub-queries to cycle providers, loads the external query operator, and establishes –secure- communication channels between cycle-providers, before a query can be executed

- Execute/Monitor:
  - Executes the plan using external operators, and built-in query operators (e.g. selection, join, union, etc) by following a iterator model
  - Monitors the failures, quality requirements violations

Example


ObjectGlobe Query Engine
Client
display
recv
recv
ObjectGlobe Query Engine
ObjectGlobe Query Engine
send
send
thumbnail
scan
wrap_S
scan
data provider A
data provider B
R
S
T
load function
FctProv
thumbnail
wrap_S


display
cycle-provider=client
cycle-provider=client
cycle-provider=alpha.A.com
thumbnail
cycle-provider=beta.B.com
codebase=
http://www.FctProv.com
scan
cycle-provider=...
partition=R
wrap_S
cycle-provider=alpha.A.com
codebase=
http://www.FctProv.com
scan
cycle-provider=beta.B.com
partition=T

Query processing requirements

- QoS parameters: The nature of the environment, i.e. large number of data, cycle and data providers, results in a need for user to control :
  - Cost: Upper bound for the charge
  - Time: Upper bound for overall execution time, fast production of partial results, ..
  - Result: Lower and/or upper bound of result set size (cardinality), freshness of the result

- Security and Privacy: E.g. Protecting data and cycle providers from malicious external operator, securing the communication between ObjectGlobe servers, protecting confidential data or operator code by enforcing an authorization and authentication scheme, restrict the access and use of resources to particular set of users (identified through authentication schemes)

Other system architectures

System Arhitecture	Comparison
DDBMS	Participating data sites are database data-resources, and can only use built-in query operators
Centralized/Distributed Middleware Architecture	Can only exploit query capabilities of data sources (e.g. can not load dynamic operators to the data source to reduce the network cost)

OUTLINE

- Introduction
- ObjectGlobe system overview
- Generating query plans
  - Lookup Service
  - Parser
  - Optimizer
- Executing query plans
  - Distributing and executing
  - Monitoring
- Experiment Results
- Summary
- Comments

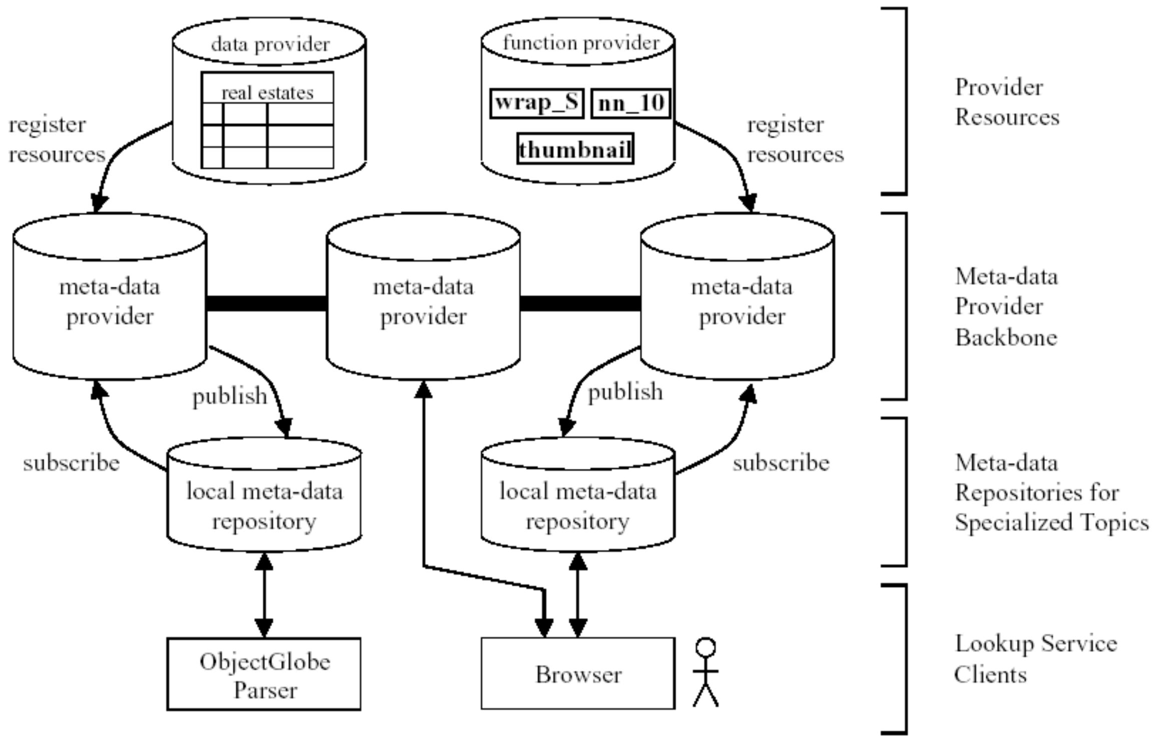
Generating Query Plans

LOOKUP SERVICE

- Similar to database catalog or metadata, used to find the relevant resources to execute the query (also statistic regarding to the resources to optimize the query plan, and meet the QoS requirements)
- Data providers
    - Register the attributes of their data collections (either materialized collection or virtual collection in nested relational format) to the lookup service
    - Associate each collection with a theme (query is performed against themes) in the lookup service (The attributes registered by a new data collection must be a subset of theme's attributes)
    - Registers statistics about collections (for estimating the selectivity, ..)
    - Registers authorization information to control the access to the collections
- Cycle providers
    - Stores hardware resource details (e.g. CPU power), latency and bandwidth of the connection between cycle providers, authorization, ..
- Function provider
    - Name and signature of query operator, formulas to estimate consumption of resource by the operator, selectivity of operator

Lookup Service

- Keep the metadata up-to-date: Enforce participants to update their meta-data
- Registration of all this information is done using RDF documents, and the lookup service can be queries through a declarative query language
- Implementation details: RDF is mapped to tables and stored in RDBMS. Search on the lookup service is translated in SQL queries:



Parser

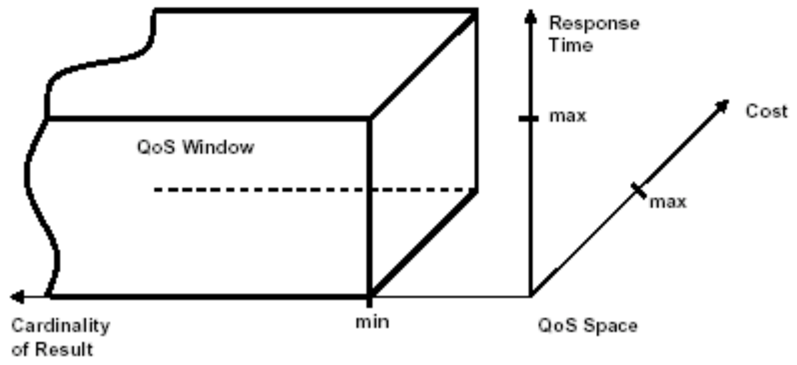
- Simplified Query: *SELECT <list_of_attributes> FROM <list_of_themes> WHERE ..*

- Parser searches lookup service:
  - Find all the collections and their providers using themes and attributes (extracted from SELECT, and FROM subclause)
  - Find the information about function providers if the query contains external operator (e.g. thumbnail)
  - Cycle providers (Data or function providers might have registered preferred cycle providers in the lookup service. It also can get user-preferred cycle providers.)
  - Statistics and authorization information

- Parser generates a XML with all the discovered information for the query and passed it to the optimizer

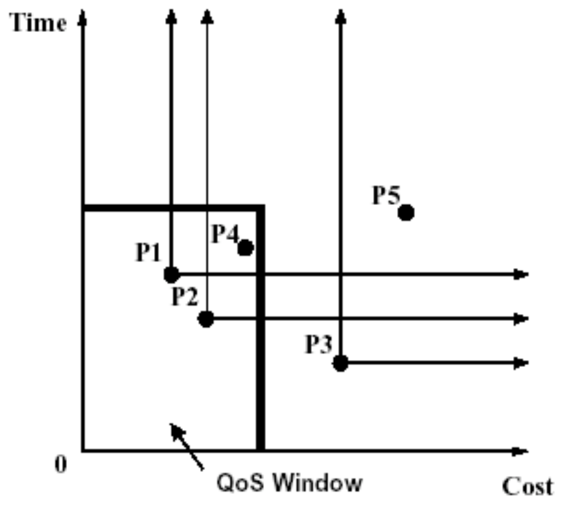
Optimizer

- Aim: Find a good execution plan, if a plan exists
  - The plan might not exist because of authorization and quality parameter constraints
- Enumerates the query plan using System-R style dynamic programming
  - Bottom-up: first construct access plans to read collections, then joins plans from these access plans, and so on.
- Pruning search space:
  - Cost of each plan is calculated on a separate dimension for each quality parameter (the space of these dimensions is called QoS space)
  - Every enumerated plan in each level of query plan enumeration is mapped to the QoS space.



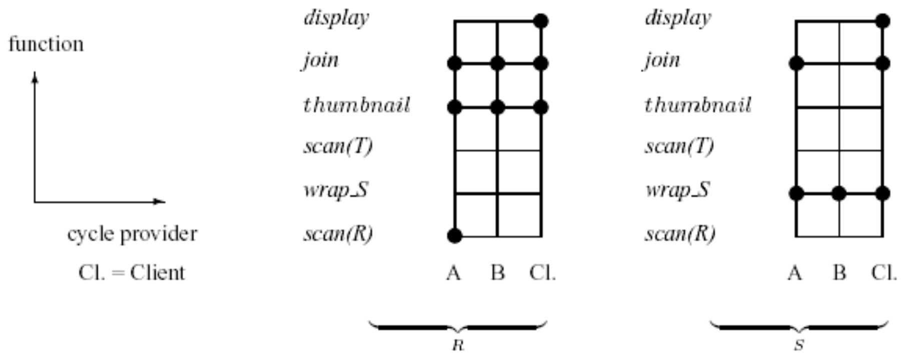
Optimizer

- Pruning search space:
  - The comparison between some plans is straightforward (e.g. Plan 1 is superior than Plan 5 on each dimension). Inferior plan is pruned from the search space
  - Some of plans are not comparable (e.g. Plan 1 is superior than Plan 2 only on sub-set of dimensions). Heuristic is used: the plan with largest, min-normalized distance to any QoS Window border is picked, and others get pruned.



Optimizer

- Optimizer explicitly states demands in terms of resource requirements (estimated resource at the optimization time) from a provider and quality constraints in the query plan.
- (Pruning Space) Compatibility Matrix : Every plan is annotated with a compatibility matrix.
  - The compatibility matrix of access plan is same as the one generated by parse
  - NDind two aceess plans



  - Prunes the plan if it does not match with compatibility matrix
- (Enumaration) UNIONs are enumerated similar to joins enumarations in QueryGlobe
- Using Iterative Dynamic programming to deal with the large search space

OUTLINE

- Introduction
- ObjectGlobe system overview
- Generating query plans
  - Lookup Service
  - Parser
  - Optimizer
- Executing query plans
  - Distributing and executing
  - Monitoring
- Experiment Results
- Summary
- Comments

Executing Query Plans

Distributing and Executing Query Evaluation Plans:

- Distributes sub-plans using cycle-provider annotations in the plan, loads external operators using codebase annotations, etc.

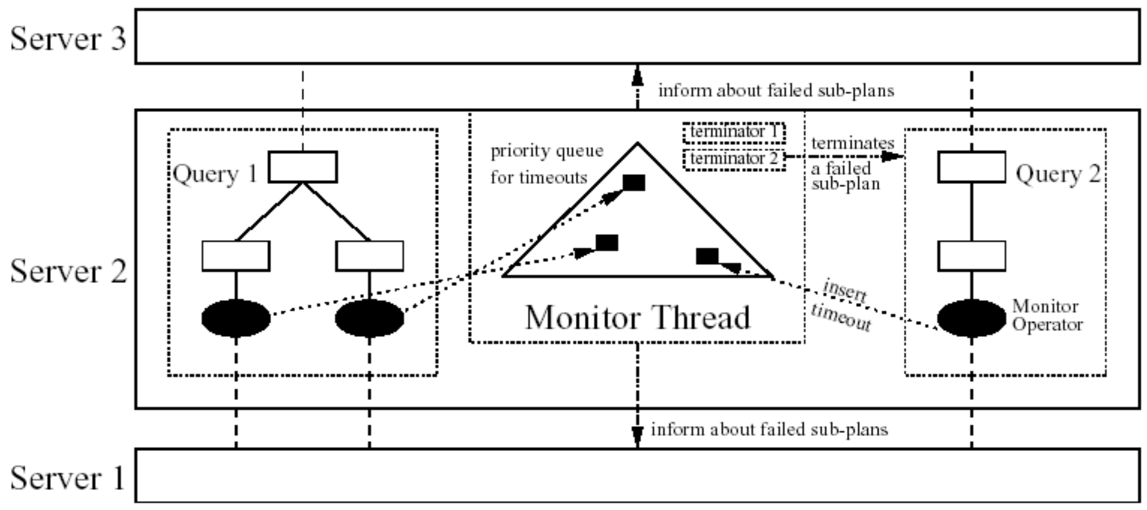
- Authentication information will be part of query plan; authorization is carried out by the provider (or ObjectGlobe security provider) when the query is instantiated

- To support extendibility, the open() method for iterators returns the type specification of the object that will retrieved in next():
  - retrieve up-to-date list of attributes (in ase lookup service is outdate),
  - Outer union of two collections if the collections had different attributes (e.g. SELECT * from hotel ..)

- Utilized Java's 5-layer security model to guarantee the security of ObjectGlobe servers while executing external operators

Monitoring query execution

- Monitoring the query execution progress
  - Monitoring the aliveness: Time-out query if a specified event in the query did not occur until the specified PIT



Monitoring query execution

- Monitoring the QoS, and adaptations
  - Monitor the current status of quality parameters through monitor operators: number of tuples produced, time and cost consumption, and rates like cost and time rate per produced tuple
  - These rates are used to estimate that total cost or time of the executing query based on the past, and see if the quality constraints are likely to be violated
  - Quality constraints are jeopardized by inaccurate estimates during the optimization or resource changes
  - Adaptations using event-condition-action rules
    - Prevention of Response time violation: E.g. Moving subplan with all its state information to another cycle provider if CPU is a bottleneck
    - Prevention of cardinality or completeness violation: E.g. use union operator to integrate additional data source (-> requires the consideration of this possibility at the optimization time and having the resources appended to the query plan)
    - Prevention of cost violation: E.g. stopping input plans before they finish, or move the sub-plan to another cycle provider, which charges less for the same execution

OUTLINE

- Introduction
- ObjectGlobe system overview
- Generating query plans
  - Lookup Service
  - Parser
  - Optimizer
- Executing query plans
  - Distributing and executing
  - Monitoring
- Experiment Results
- Summary
- Comments

Experiment Results

Plan generation:

Setup: 5-way join query, 6 cycle provider, optimizer considers 3-join type (nested-loop, hash and sort-merge)

Scenario-I: All joins can be executed on any cycle provider,

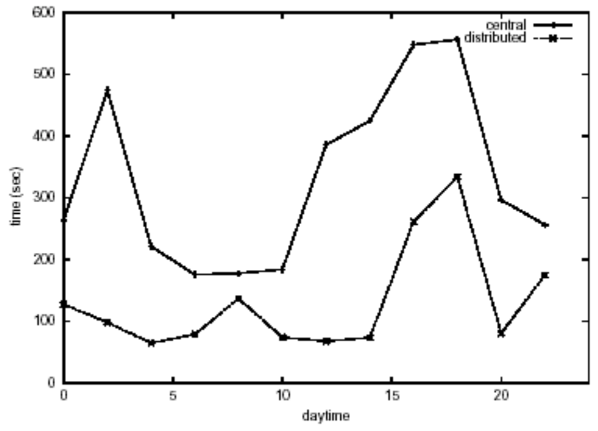
Scenario-II: Two join can be only executed on specific cycle provider

Table 1. Overheads of plan generation

	Total lookup time	Avg. time per search	Optimization time
Scenario I	5.64 s	0.47 s	0.83 s
Scenario II	5.64 s	0.47 s	0.07 s

Operator Mobility:

- Central: The wrappers and group operator (hotel with largest room number) is executed on the client
- Distributed: The wrapper and intermediate group operator is loaded to cycle providers that are close to the data provider.



Summary

- Goal of ObjectGlobe is to establish open marketplace on internet in which data, cycle and function providers offer/sell the services
- Provides an distributed query processing framework for this type of environment
  - Each participant can provide data or query capabilities
  - Each participant can have heterogonous data resources (wrappers are required to map to nested relational model, and register this information)
- Attempts to address the following challenges:
  - Keeping the system open and scalable
  - Providing data and code security and privacy in the query processing
  - Quality-aware query optimization and processing
  - Finding optimized query plan in large search space (great number of cycle, data and function providers, quality requirements parameters, ..)

Comments