ObjectGlobe: Open Distributed Query Processing Services on the Internet[*]

R. Braumandl[⋆] M. Keidl[⋆] A. Kemper[⋆] D. Kossmann[†] A. Kreutz[⋆]
S. Seltzsam[⋆] K. Stocker[⋆]

[⋆]University of Passau
⟨last name⟩@db.fmi.uni-passau.de

[†]TU Munich
kossmann@in.tum.de

1 Vision

The Internet today offers access to a wide range of data sources, for example, in the area of real estate offers, geographical information, product catalogs or travel planning to name just a few. Therefore, it is often stated, that the Internet represents a huge distributed database, but at the moment we cannot use this database in the way we use a database managed by a modern DBMS. We also want to be able to pose queries over *arbitrary* data sources by executing data transformations, aggregate or user-defined functions or join operations. Ultimately, we strive for a query processing system which can perform an ad-hoc composition of completely unrelated *query processing services* which are offered on the Internet in the manner of a free market. These services could be specialized on providing data, resources for the query execution itself (CPU power, storage area) or functions which can be embedded in the execution. Such an open system could vastly ease the interaction in commercial business-to-business and business-to-customer applications like shopping portals, electronic market-places or virtual enterprises. For example, somebody could search for real estate offers which fulfill some constraints on the features of the building, its geographical position and the corresponding ambient data. The respective query should then access data from several commercial realtor databases, a geographical information system and a server with global ambient data. Additionally, the query should use a ranking function specialized for real estate data and provided in the form of *mobile code* by a third party with the appropriate knowledge in that business area.

1.1 The Requirements

The differing demands of *data providers* and users in respect to such a global query processing system show why current architectures for distributed databases and mediator systems are not sufficient. Data providers are interested in

- the **security** of their computers. Thus, some data providers with higher security demands would not be willing to execute mobile code in order to avoid the danger of a hostile system intrusion.
- the **privacy** of their data. Data providers could be interested in restricting and controlling access to their data by the use of *authorization* and *authentication* techniques. Furthermore, they may demand the use of cryptography to avoid that somebody can steal their data during network transmissions.

[*]This research is supported by the German National Research Foundation under contract DFG Ke 401/7-1 and the German Isreali Foundation (GIF).

- the **scalability** of the system. The number of users in a global system could cause overload situations on data providers. Therefore, data providers may allow no other operation to be performed on their machines than a simple scan/index-scan on the data.

Naturally, users have completely different requirements for such a system which also seem to partly contradict the requirements of data providers. Users are interested in

- an **open** system, where service providers can be integrated and instantly be used in queries. As a consequence, there is no need to build several special-purpose data integration systems and a user just has to work with one system.

- a comfortable **service composition**. They just want to state a declarative query and the composition of appropriate services in the form of a *query evaluation plan* is then performed by the *query optimizer*.

- an **extensible** system, where user-defined code could be integrated seamlessly and rather effortless in the query execution. Especially in distributed and heterogeneous systems this is more an issue. It is important there to be able to apply data transformations or user-defined predicates early in order to unify data representations or to reduce the data volume.

- a **quality-of-service (QoS) aware** system. Query execution in a global system can hardly be overlooked by users. Therefore, they should be able to specify quality constraints on the result and the properties of the query execution itself (e.g., time and cost consumption) and the system should try to guarantee these constraints if they are supposed to be feasible.

1.2 A possible Solution

In our ObjectGlobe project we have developed a distributed query processing system which works along the lines stated above. In order to solve the discrepancies between the data provider and user requirements we separated the tasks of a query processing system by introducing also services of separate *cycle* and *function providers*.

- Function providers offer Java byte-code in different standardized forms (query operator, predicate function, ...) which are suited for the incorporation in a cycle provider. For example, a function provider can offer wrappers for accessing data providers, predicate functions specialized on business areas like real estate data or new query operators like join methods for spatial data. We are currently developing a verification and testing environment for such code fragments, which can be used together with a certification infrastructure to establish trust relationships between cycle and function providers.

- A cycle provider runs our Java-based query processing engine. It represents a node in our distributed query processing system which can execute plan fragments of a distributed query evaluation plan if the data providers are not willing or not suited due to their hardware capacities or their position in the network to do so. They provide a core functionality for processing queries but can also load new functionality from function providers, for example, a wrapper for accessing a data provider. A specialized Java ‘sandbox’ is used to secure the cycle provider’s machine against malicious effects of external code.

A distributed lookup service is used for registering and querying meta-data about all known instances of services described above. This meta-data also contains authorization data for all providers which enforce explicit authorization for the usage of their services. Our query optimizer which uses this lookup service

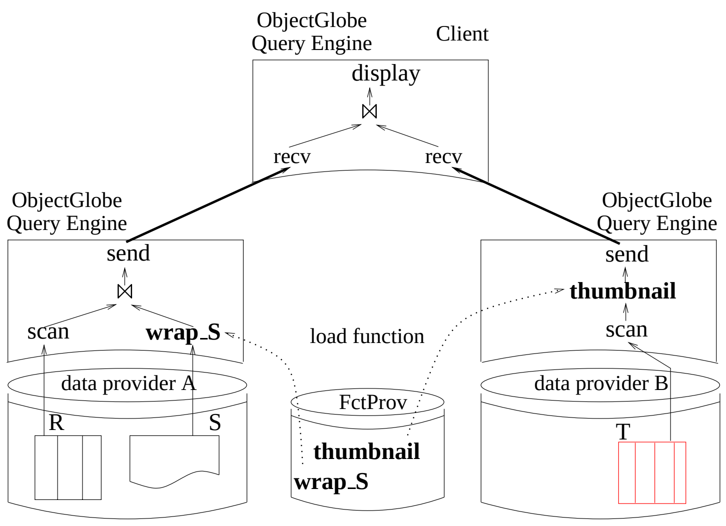

Figure 1: Distributed Query Processing with ObjectGlobe

to retrieve meta-data about services, needs this information together with all the other relevant meta-data for a specific query to compile a valid query evaluation plan. During query optimization and also during the query execution user-defined QoS constraints are considered.

1.3 Query Processing

Processing a query in ObjectGlobe involves four major steps:

1. Lookup: In this phase, the ObjectGlobe lookup service is queried to find relevant data sources, cycle providers, and query operators that might be useful to execute the query. In addition, the lookup service provides the authorization data—mirrored and integrated from the individual providers—to determine what resources may be accessed by the user who initiates the query and what other restrictions apply for processing the query.

2. Optimize: The information obtained from the lookup service, is used by a quality-aware query optimizer to compile a valid (as far as user privileges are concerned) query execution plan, which is believed to fulfill the users' quality constraints. This plan is annotated with site information indicating on which cycle provider each operator is executed and from which function provider the external query operators involved in the plan are loaded.

3. Plug: The generated plan is distributed to the cycle providers and the external query operators are loaded and instantiated at each cycle provider. Furthermore, the communication paths (i.e., sockets) are established.

4. Execute: The plan is executed following an iterator model [Gra93]. In addition to the *external* query operators provided by function providers, ObjectGlobe has *built-in* query operators for selection, projection, join, union, nesting, unnesting, and sending and receiving data. If necessary, communication is encrypted and authenticated. Furthermore, the execution of the plan is monitored in order to detect failures, look for alternatives, and possibly halt the execution of a plan.

To illustrate query processing in ObjectGlobe, let us consider the example shown in Figure 1. In this example, there are two data providers, A and B, and one function provider. We assume that the data providers also operate as cycle providers so that the ObjectGlobe system is installed on the machines of A and B. Furthermore, the client can act as a cycle provider in this example. Data provider A supplies two data collections, a relational table R and some other collection S which needs to be transformed (i.e., wrapped) for query processing. Data provider B has a (nested) relational table T. The function provider supplies two relevant query operators: a wrapper (*wrap_S*) to transform S into nested relational format and a compression algorithm (*thumbnail*) to apply on an image attribute of T.

2 Lookup Service and Optimization

The lookup service plays the same role in ObjectGlobe as the *catalog* or *meta-data management* of a traditional query processor. Providers are registered before they can participate in ObjectGlobe. In this way, the information about available services is incrementally extended as necessary. A similar approach for integrating various business services in B2B e-commerce has been proposed recently in the UDDI standardization effort [UDD00].

The main challenge of the lookup service is to provide global access to the meta-data of all registered providers but also to allow a selective grouping of somehow related meta-data in order to reduce the effort to query the meta-data. Thus, our lookup service uses a distributed two-tier server architecture. Providers register their services by passing an *RDF* [BG99] document to a *local meta-data repository* or a *meta-data provider*. Meta-data registered at a meta-data provider is regarded as globally and publicly available and therefore consistently replicated by all meta-data providers which appear in the meta-data provider backbone. Meta-data registered at a local meta-data repository is private to this repository and not replicated anywhere else. We use a special query language for querying local meta-data repositories by end users for browsing through the meta-data or by an optimizer for finding relevant providers for a query. In contrast, Meta-data providers themselves cannot be queried . They are used to provide the local meta-data repositories with globally available and current meta-data by a *publish and subscribe* mechanism. Local meta-data repositories pass subscription rules to a meta-data provider, which declaratively describe the kind of meta-data this repository is interested in. In the case of updates, inserts or deletions in the meta-data, a meta-data provider evaluates the possibly huge set of subscription rules with the help of a sophisticated prefilter algorithm and forwards the appropriate changes to the corresponding local meta-data repositories which cache these data.

The meta-data in the providers' RDF-documents must conform to our *meta-schema* which defines the structures of the service descriptions of each kind of provider. These descriptions are rather detailed since they are the only information about services the optimizer gets in order to construct a valid and QoS-aware query evaluation plan. For example, for each external query operator offered by a function provider meta-data about its name, category, signature and cost model must be available. The optimizer also needs performance characteristics of cycle providers and schema information and statistics for data collections of data providers. Furthermore, authorization data for the services are used to construct compatibility matrices during the optimization process which represent the information about legal combinations of the services possibly involved in the query execution at a specific position in the query evaluation plan. Due to authorization constraints our optimizer might not be able to find a query evaluation plan although necessary services could be retrieved from the lookup service.

The optimizer enumerates alternative query evaluation plans using a System-R style dynamic programming algorithm. That is, the optimizer builds plans in a bottom-up way: first so-called *access plans* are constructed that specify how each collection is read (i.e., at which cycle provider and with which *scan* or *wrapper* operator). After that, *join plans* are constructed from these *access plans* and (later) from simpler *join plans*. Evidently, the search space can become too large for full dynamic programming to work for complex ObjectGlobe queries. To deal with such queries, we developed another extension that we call *iterative dynamic programming* (`IDP` for short). `IDP` is adaptive; it starts like dynamic programming and if the query is simple enough, then `IDP` behaves exactly like dynamic programming. If the query turns out to be too complex, then `IDP` applies heuristics in order to find an acceptable plan. Details and a complete analysis of `IDP` is given in [KS00].

3 Quality of Service (QoS)

Although the example above is rather small (in order to be illustrative) we expect ObjectGlobe systems to comprise a large number of cycle providers and far more data providers. For example, think of an ObjectGlobe federation which incorporates the online databases of several real estate brokers. A traditional optimizer would produce a plan for a query in this federation that reads all the relevant data (i.e., considers all real-estate data providers). Therefore, the plan produced by a traditional optimizer will consume much more time and cost than an ObjectGlobe user is willing to spend. In such an open query processing system it is essential that a user can specify quality constraints on the execution itself. These constraints can be separated in three different dimensions:

Result: Users may want to restrict the size of the result sets returned by their queries in the form of lower or upper bounds (an upper bound corresponds to a stop after query [CK98]). Constraints on the amount of data used for answering the query (e.g., at least 50% of the data registered for the theme “real estate” should be used for a specific query) and its freshness (e.g., the last update should have happened within the last day) can be used to get results which are based on a current and sufficiently large subset of the available data.

Cost: Since providers can charge for their services in our scenario, a user should be able to specify an upper bound for the respective consumption by a query.

Time: The response time is another important quality parameter of an interactive query execution. A user can be interested first, in a fast production of the first answer tuples and second, in a fast overall execution of the query. A fast production of the first tuples can be important so that the user can look at these tuples while the remainder is computed in the background.

In many cases not all quality parameters will be interesting. As in real-time systems some constraints could be strict (or hard) and others could be soft and handled in a relaxed way.

An overview of processing a query in the context of our QoS management is depicted in Figure 2. The starting point for query processing in our system is given by the description of the query itself, the QoS constraints for it and statistics about the resources (providers and communication links). As shown in the figure, QoS constraints will be treated during all the phases of query processing. The optimizer first generates a query evaluation plan whose estimated quality parameters are believed to fulfill the user-specified quality constraints of the query. For every sub-plan the optimizer states the minimum quality constraints it must obey in order to fulfill the overall quality estimations of the chosen plan and the resource requirements which are believed to be necessary to produce these quality constraints. If, during the plug phase, the resource requirements cannot be satisfied with the available resources, the plan is adapted or aborted. The QoS management reacts in the same way, if during query execution the monitoring component forecasts an eventual violation of the QoS constraints.

4 Security and Privacy Issues

Security obviously is crucial to the success of an open and distributed system like ObjectGlobe. Dependent on the point of view different security interests are important. On the one hand, cycle and data

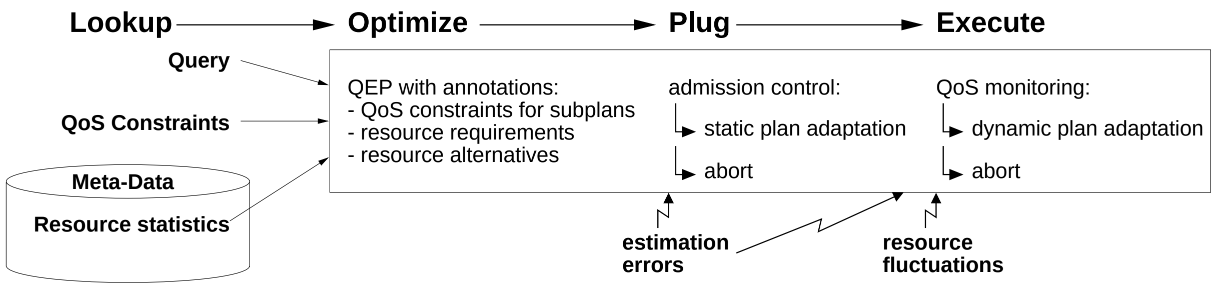

Figure 2: The Interaction of Query Processing and QoS Management

providers need a powerful security system to protect their resources against unauthorized access and attacks of malicious external operators, e.g., denial of service attacks. Besides that cycle and data providers might have a legitimate interest in the identity of users for authorization issues. Users of ObjectGlobe on the other hand want to feel certain about the semantics of external operators to be able to rely upon the results of a query. For that purpose it is also necessary to protect communication channels against tampering. Another interest of users is privacy, i.e., other people must not be able to read confidential data. Below we sketch our conception of the security system of ObjectGlobe. The security measures are classified by the time of application.

Preventive Measures: Preventive measures take place before an operator is actually used for queries and include checking of the results produced by the operator, stress testing, and validation of the cost model. These checkup is done by trustworthy third parties which generate a digitally signed document containing the diagnosis for the tested operator. To support the checkup we developed a validation server which semi-automatically generates test data, runs the operator and compares the results generated by the operator with results acquired from an executable formal specification or a reference implementation of the operator. Additionally, the validation server ensures that execution costs are within the limits given by the cost model of the operator.

The objective of preventive measures is to increase trust in the non-malicious behaviour of external operators, they are optional in ObjectGlobe. Users with a high demand of security will exclusively use certified external operators to ensure that all operators will calculate the result of the query according to the given semantics.

Checks during Plan Distribution: Three security related actions take place during plan distribution: setup of secure communication channels, authentication, and authorization. ObjectGlobe is using the well-established secure communication standards SSL (Secure Sockets Layer) [FKK96] and/or TLS (Transport Layer Security) [DA99, TLS] for encrypting and authenticating (digitally signing) messages. Both protocols can carry out the authentication of ObjectGlobe communication partners via X.509 certificates [HFPS99, PKI]. If users digitally sign plans, such certificates are used for authentication of users, too. Additionally, ObjectGlobe supports the embedding of encrypted passwords into query plans which can be used by wrappers to access legacy systems using password-based authentication. Of course, users can stay anonymous when they use public available resources.

Based on the identity of a user a provider can autonomously decide whether a user is authorized to, e.g., execute operators, access data, or load external operators. Thus providers can (but need not) constrain the access or use of their resources to particular user groups. Additionally, they can constrain the information (resp. function code) flow to ensure that only trusted cycle providers are used during query execution. In order to generate valid query execution plans and avoid failures at execution time the authorization constraints must be known by the lookup service of ObjectGlobe.

Runtime Measures: To prevent malicious actions of external operators, ObjectGlobe is using Java's security infrastructure to isolate external operators by executing them in protected areas, so-called "sandboxes". As a result, cycle providers can prohibit that, e.g., external operators access the filesystem. Another result is that external operators are prevented from leaking confidential data through, for instance, network connections.

Additionally, a runtime monitoring component can take action against denial of service attacks. Therefore the monitoring component evaluates cost models of operators and supervises resource consumption (e.g., memory usage and processor cycles). When an operator uses more resources than the cost model predicted, it is aborted.

5 Conclusion

We sketched the design of ObjectGlobe, an open, distributed and secure query processing system. The goal of ObjectGlobe is to establish an open marketplace in which data, function, and cycle providers can *offer/sell* their services, following some business model which can be implemented on top of ObjectGlobe. End users and applications can use these services with only little overhead.

References

[BG99] D. Brickley and R. V. Guha. Resource Description Framework (RDF) schema specification. Proposed Recommendation `http://www.w3.org/TR/PR-rdf-schema`, WWW-Consortium, March 1999.

[CK98] M. Carey and D. Kossmann. Reducing the braking distance of an SQL query engine. In *Proc. of the Conf. on Very Large Data Bases (VLDB)*, pages 158–169, New York, USA, August 1998.

[DA99] T. Dierks and C. Allen. The TLS Protocol Version 1.0. `ftp://ftp.isi.edu/in-notes/rfc2246.txt`, January 1999.

[FKK96] A. Frier, P. Karlton, and P. Kocher. *The SSL 3.0 Protocol*. Netscape Communications Corp., `http://home.netscape.com/eng/ssl3`, November 1996.

[Gra93] G. Graefe. Query evaluation techniques for large databases. *ACM Computing Surveys*, 25(2):73–170, June 1993.

[HFPS99] R. Housley, W. Ford, W. Polk, and D. Solo. Internet X.509 Public Key Infrastructure Certificate and CRL Profile. `http://www.rfc-editor.org/rfc/rfc2459.txt`, January 1999.

[KS00] D. Kossmann and K. Stocker. Iterative dynamic programming: A new class of query optimization algorithms. *ACM Trans. on Database Systems*, 25(1):43–82, March 2000.

[PKI] Public-Key Infrastructure (X.509) (PKIX). `http://www.ietf.org/html.charters/pkix-charter.html`.

[TLS] Transport Layer Security (TLS). `http://www.ietf.org/html.charters/tls-charter.html`.

[UDD00] Universal Description, Discovery and Integration (UDDI) technical white paper. White Paper, Ariba, Inc., IBM Corp., and Microsoft Corp., September 2000. `http://www.uddi.org/`.