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

Reinhard Braumandl[⋆] Markus Keidl[⋆] Alfons Kemper[⋆] Donald Kossmann[†]
Stefan Seltzsam[⋆] Konrad Stocker[⋆]

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

[†]TU Munich
kossmann@in.tum.de

1 Vision

Today, there is a vast discrepancy between data publication and data processing capabilities on the Internet. Virtually anybody can publish data, e.g., in HTML- or XML-format. Consequently, a wide range of (different quality) data sources exist on the Internet, ranging from personal documents to real estate offers to product catalogs, to name just a few. In a way, the Internet could be viewed as a large distributed database. However, today's web sites are merely "dumb" page servers which are only capable to send the data sitting behind a particular URL/URI. At best, web sites can process local queries if they are backed by a database system and a query interface is published via, e.g., a forms interface. But true distributed query processing plans as enabled by homogeneous distributed databases with interacting distributed subplans are not supported. Our goal is to create a query processing server that can be deployed throughout the Internet. These query servers can then be used in a federation to execute truly distributed query processing plans composed of completely unrelated *query processing services* which are offered on the Internet in an open 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 business-to-business and business-to-customer applications like shopping portals, electronic marketplaces or virtual enterprises. For example, somebody could search for real estate offers which fulfill some constraints with regard to the building, its location 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 specialized in that particular business area.

1.1 The Requirements

The differing demands of *data providers* and users with respect to such a global query processing system show why current architectures for distributed databases ([CDF$^+$94]) and mediator systems ([HKWY97, PGGMU95, JKR99]) are not sufficient. Data providers are interested in

Bulletin of the IEEE Computer Society Technical Committee on Data Engineering

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

- 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.
- 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 spontaneously 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* dynamically extensible system.
- an automatic **service composition**. Users want to state a declarative query; the composition of appropriate services in the form of a *query evaluation plan* (QEP) should be performed by a *query optimizer*.
- an **extensible** system in which user-defined code can be integrated in a seamless and rather effortless manner. Especially in distributed and heterogeneous systems this is an important issue. In such systems, it is essential to be able to apply data transformations or user-defined predicates early (i.e., close to the data providers) in order to unify data representations or to reduce the data volume.
- a **quality-of-service (QoS) aware** system. Query execution in a widely distributed system can hardly be monitored 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 fulfill these constraints if possible or abort the execution of the query as early as possible. [Wei98] gives a comprehensive motivation for the need to integrate the handling of QoS guarantees in information systems.

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 help both the data providers and the users, we introduced the new services of *cycle* and *function providers*.

- Function providers offer Java byte-code in different standardized forms (query operators, predicate functions, data transformers, etc.) which are suited for the execution by 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 validation 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 to retrieve meta-data

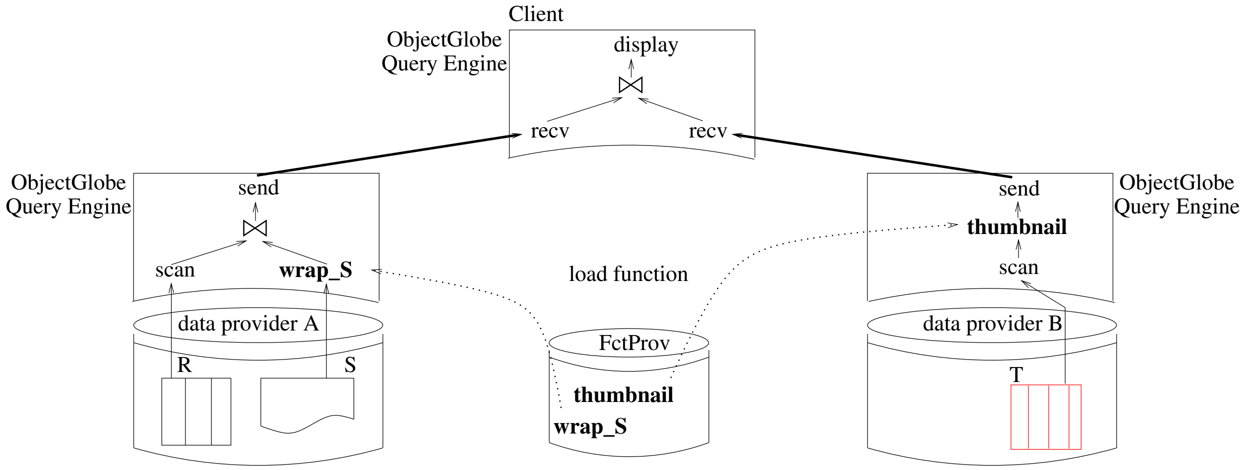

Figure 1: Distributed Query Processing with ObjectGlobe

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 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 the limitations for accessing data, moving data or code to cycle providers, etc.
2. Optimize: The information obtained from the lookup service, is used by a QoS-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 the corresponding cycle providers. Furthermore, the communication links (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 in the case of QoS-violations.

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

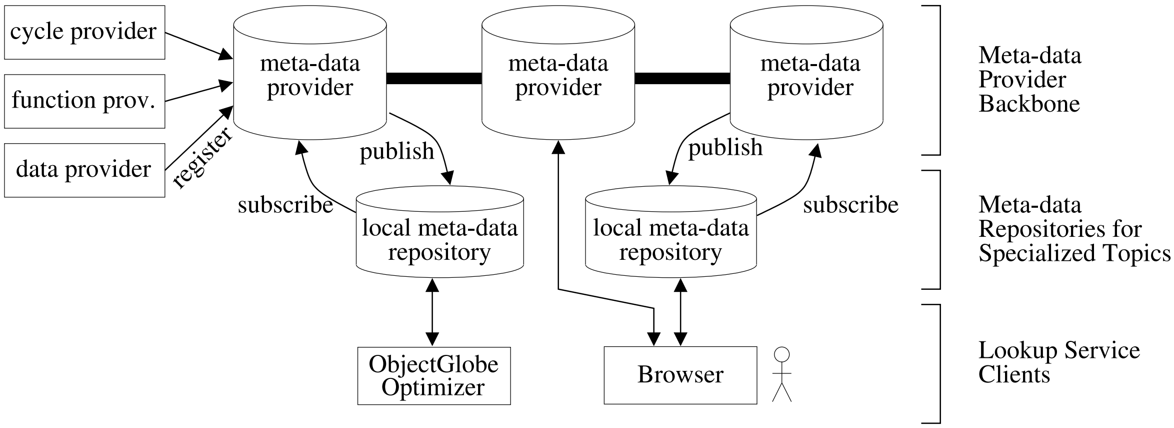

Figure 2: The Architecture of the Lookup Service

wrapper (*wrap_S*) to transform S into nested relational format and a compression algorithm (*thumbnail*) to apply on an image attribute of T. This example query evaluation plan highlights the benefits of placing open distributed query processing servers throughout the Internet. It enables to ship new functionality (*wrap_S* and *thumbnail*) to query processors that are located in the vicinity of the data. That way, data shipping costs are reduced and, furthermore, parallel processing of complex query plans is facilitated. In traditional middleware query processing systems (e.g., Garlic) external operators can only be processed in the centralized middleware system. Those systems can only use the built-in query processing capabilities of the data providers; but they cannot dynamically ship new functionality to the data.

2 Lookup Service

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.

During the optimization of every query in an ObjectGlobe federation, the lookup service is queried for descriptions of useful services for the respective query. Therefore, the main challenge of the lookup service is to provide global access to the meta-data of all registered services without becoming the bottleneck of the whole system. Since the meta-data structures in an open and extensible systems are naturally quite complex, the lookup service offers a sophisticated special-purpose query language which also permits to express joins over meta-data collections. In addition to the network and storage devices, also the computing power of a lookup service machine can limit the throughput of meta-data queries. Thus, our lookup service uses a three-tier architecture as depicted in Figure 2. The purpose of this architecture is to be able to scale in the number of users of the lookup service (real users who browse the meta-data or optimizers which search for specific services) by adding new local meta-data repositories at the hot spots of user activity.

The information at meta-data providers is regarded as globally and publicly available and therefore it is consistently replicated by all meta-data providers which appear in the meta-data provider backbone. For the efficiency reasons stated above, meta-data providers themselves cannot be queried; they only can be browsed in order to detect meta-data which should also be available at a specific local meta-data repository. Only these repositories can be queried for meta-data cached at the repository. They use a *publish/subscribe* mechanism to fetch relevant data from a meta-data provider. For 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. A more

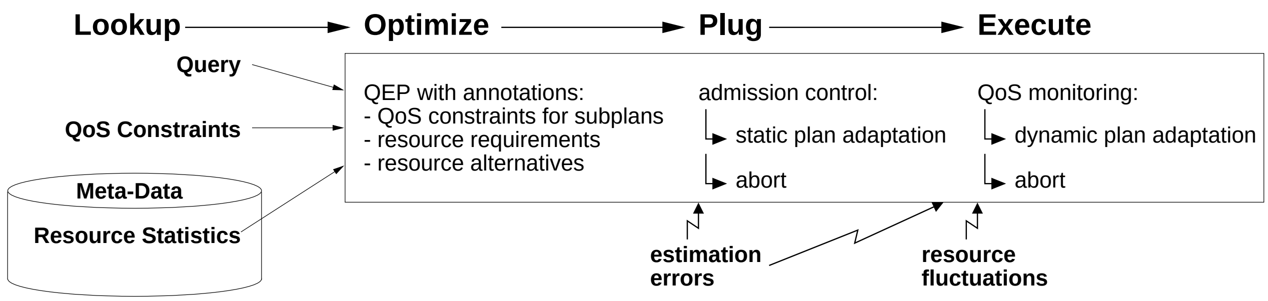

Figure 3: The Interaction of Query Processing and QoS Management

detailed description of the lookup service can be found in [KKKK01].

3 Quality of Service (QoS)

Although the example in Section 1.3 is rather small (in order to be illustrative) we expect ObjectGlobe systems to comprise a large number of cycle providers and 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. 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, users 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. Firstly, users can be interested, in a fast production of the first answer tuples and secondly, in a fast overall execution of the query. A fast production of the first tuples is particularly important for interactive applications so that users can look at these tuples while the remainder is computed in the background.

An overview of processing a query in the context of our QoS management is depicted in Figure 3. The starting point for query processing 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. First, a multi-objective optimizer generates a query evaluation plan whose estimated quality parameters are believed to fulfill the user-specified quality constraints of the query. Each QoS parameter introduces a new optimization criterion which means that alternative query evaluation plans can be incomparable. Thus, a dynamic programming based multi-objective optimization algorithm [GHK92] is used to find query evaluation plans which represent optimal trade-offs between the different optimization criteria. For every sub-plan the optimizer then annotates the 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. Additionally to our adaptation techniques like the movement of plan fragments, the compression of data sent through a network link or the activation of additional sub-plans, it would also be possible to use adaptive query processing technologies already devised, for example, in [HFC+00, IFF+99].

4 Security and Privacy Issues

Obviously, security 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 providers need a powerful security system to protect their resources against unauthorized access and attacks of malicious external operators. 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 parties must not be able to read confidential data. Furthermore users normally want to stay anonymous as far as possible. 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 in test runs, stress testing, and validation of the cost model. These checkups are done by a trustworthy third party which generates a digitally signed document containing the diagnosis for the tested operator. To support the checkups 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.

Preventive measures should increase the trust in the non-malicious behavior of external operators. They are optional in ObjectGlobe, but 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) and/or TLS (Transport Layer Security) [DA99] for encrypting and digitally signing messages. Both protocols can carry out the authentication of ObjectGlobe communication partners via X.509 certificates [HFPS99]. 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 publicly 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 authorization failures at execution time the authorization constraints are integrated into the the lookup service of ObjectGlobe.

Runtime Measures: To prevent malicious actions of external operators, ObjectGlobe is based on 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 external operators access crucial resources, e.g., the filesystem or network sockets. External operators are also prevented from leaking confidential data through, for instance, network connections. Additionally, a runtime monitoring component can react on 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. Applications in the area of electronic marketplaces and virtual enterprises can profit from such an infrastructure. ObjectGlobe provides enabling technology for such applications in the form of an extensible query processor with integrated security and QoS management components and a scalable lookup service.

A more detailed description of the architecture is given in [BKK+99]. Our current implementation is able to run the complete lookup – optimize – plug – execute process automatically given a declarative query. At the moment, we concentrate on benchmarking the QoS component and on fine tuning the lookup and security components of our implementation. Furthermore, we are working on advanced query processing techniques for Internet data sources, e.g., approximate query processing and dynamic query plans.

References

[BKK+99] R. Braumandl, M. Keidl, A. Kemper, D. Kossmann, A. Kreutz, S. Pröls, S. Seltzsam, and K. Stocker. ObjectGlobe: Ubiquitous query processing on the Internet. Tech Rep. MIP-9909, Universität Passau, 1999.

[CDF+94] M. Carey, D. DeWitt, M. Franklin, N. Hall, M. McAuliffe, J. Naughton, D. Schuh, M. Solomon, C. Tan, O. Tsatalos, S. White, and M. Zwilling. Shoring up persistent applications. In *Proc. of the ACM SIGMOD Conf. on Management of Data*, pages 383–394, Minneapolis, MI, USA, May 1994.

[DA99] T. Dierks and C. Allen. The TLS Protocol Version 1.0. RFC 2246, January 1999.

[GHK92] S. Ganguly, W. Hasan, and R. Krishnamurthy. Query optimization for parallel execution. In *Proc. of the ACM SIGMOD Conf. on Management of Data*, pages 9–18, San Diego, CA, USA, June 1992.

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

[HFC+00] J. M. Hellerstein, M. J. Franklin, S. Chandrasekaran, A. Deshpande, K. Hildrum, S. Madden, V. Raman, and M. Shah. Adaptive query processing: Technology in evolution. In A. Y. Levy, editor, *IEEE Data Engineering Bulletin, Vol 23, No 2*, pages 7–18. June 2000.

[HFPS99] R. Housley, W. Ford, W. Polk, and D. Solo. Internet X.509 Public Key Infrastructure Certificate and CRL Profile. RFC 2459, January 1999.

[HKWY97] L. Haas, D. Kossmann, E. Wimmers, and J. Yang. Optimizing queries across diverse data sources. In *Proc. of the Conf. on Very Large Data Bases (VLDB)*, pages 276–285, Athens, Greece, August 1997.

[IFF+99] Z. Ives, D. Florescu, M. Friedman, A. Levy, and D. Weld. An adaptive query execution engine for data integration. In *Proc. of the ACM SIGMOD Conf. on Management of Data*, pages 299–310, Philadelphia, PA, USA, June 1999.

[JKR99] V. Josifovski, T. Katchaounov, and T. Risch. Optimizing queries in distributed and composable mediators. In *Proc. of the IFCIS/CoopIS*, pages 291 – 302, Edinburgh, Scotland, 1999.

[KKKK01] M. Keidl, A. Kreutz, A. Kemper, and D. Kossmann. Distributed Metadata Management on the Internet. 2001. In preparation.

[PGGMU95] Y. Papakonstantinou, A. Gupta, H. Garcia-Molina, and J. Ullman. A query translation scheme for rapid implementation of wrappers. In *Proc. of the Conf. on Deductive and Object-Oriented Databases (DOOD)*, pages 161–186, December 1995.

[Wei98] G. Weikum. On the ubiquity of information services and the absence of guaranteed service quality. In *Proc. of the Intl. Conf. on Extending Database Technology (EDBT)*, volume 1377 of *Lecture Notes in Computer Science (LNCS)*, pages 3–6. Springer, 1998.