Architecture and Performance of Large and Disperse Distributed Object Base Systems*

Donald Kossmann Michael J. Franklin

Department of Computer Science and UMIACS
University of Maryland
College Park, MD 20742
{*kossmann*|*franklin*}@*cs.umd.edu*

1 Introduction

Most object-oriented database systems and persistent object stores have been constructed in a workstation-server environment in which client workstations are connected to the server machines by a local-area network (e.g., an Ethernet). The servers are responsible for the persistent and consistent storage of the object base and for preventing unauthorized access to the data. Application programs are carried out on the client workstations to exploit the resources of the workstations and to avoid computation on servers, the potential bottlenecks of the system. Typically, object-oriented databases have been used in very computation-intensive and/or highly interactive applications (e.g., engineering applications), and therefore, techniques such as pointer swizzling [Mos92, KK95] have been used to speed up the processing of persistent objects at workstations. In this environment, scalabilty means that an arbitrary number of client workstations can be connected to the servers.

The current client-server architectures, however, are not flexible enough to meet the requirements of large, disperse systems that are now becoming reality. Break-through advances in communication technology and the establishment of inter-operability standards such as CORBA make it possible for applications to process data stored on a very large number of servers all around the world. Today, systems grow very quickly: in addition to client workstations, new servers are permanently added to systems, and slow, old servers are replaced by new servers with improved hardware characteristics. At no time is it possible to freeze or stop a system to reconfigure the servers or reorganize the database. In addition, new applications (e.g., multi-media) and the wider acceptance of object-oriented database technology [Cat94] has pushed researchers and vendors to create object-oriented database systems that support data-intensive applications that browse through large collections of objects or process very large objects (e.g., image data). Clearly, neither the distributed relational systems of the late seventies nor the current client-server architectures are going to be appropriate for this scenario. As a consequence, the design of and experimentation with new database architectures for distributed systems is being promoted; the most known research projects are SHORE [CDF+94] and Mariposa [SAD+94, SDK+94].

We believe that the following two concepts are going to be vitally important to achieve acceptable performance in very large distributed object base systems and, therefore, will greatly influence

*This work was partially supported by NSF Grant IRI-94-09575 and by the Humboldt-Stiftung.

the design of new database architectures. Both concepts are currently being investigated in the DIMSUM project[1] at the University of Maryland:

1. **Flexible Execution Policies.** The execution policy determines where database operations are carried out. The more flexibility an execution policy provides, the better the resources of a distributed system can be exploited.

2. **Dynamic Query Optimization.** To capture changes in the state of a system, dynamic query optimization defers some or all optimization decisions to the time when a query is prepared for execution.

In this position paper, we will concentrate on analyzing the response time of queries in systems with different execution policies and approaches to dynamic query optimization. The response time of a query is composed of the time to *activate* the query (for example, establishing connections to servers, and generating or refining a query evaluation plan) plus the time to *execute* the query. The execution time of a query is determined by the following three factors:

1. **Communication Cost.** The communication costs depend on the data allocation and on the effectiveness of caching; i.e., the availability of objects at different sites. In addition, some communication links of a wide-area network can be very slow, and others can be very fast. Also, the communication costs can change dynamically due to congestion, router and/or link failures, etc. As a consequence, communication costs can be negligible to process some queries, but they can dominate the execution time of other queries.

2. **Local Processing Cost.** A database operation should be carried out at the site at which it can be executed most efficiently. For example, a join should be carried out at a site with a large main memory, and complex computations should be carried out at sites with fast CPUs. In addition to the hardware characteristics, the availability of indexes can have an influence on how efficient an operation (e.g., a join) can be processed at a site. Furthermore, operations should be carried out on idle or lightly loaded machines.

3. **Parallel Execution.** The response time of a complex query can be reduced significantly by executing parts of the query at different sites in parallel (thereby, possibly increasing communication costs). The degree of parallelism that can be exploited depends on the availability of copies of objects at different sites (data allocation and caching) and on the query: most relational queries can be parallelized fairly easily; methods such as the traversals of the OO7 benchmark sometimes can and sometimes cannot be parallelized well [DNSV95].

The remainder of this position paper is organized as follows: Section 2 discusses alternative execution policies and approaches to dynamic query optimization, the two concepts we believe are going to be very important for distributed object base systems. Section 3 presents the results of some performance experiments, and Section 4 contains a summary.

2 Architectures of Distributed Object Bases

2.1 Flexible Execution Policies

State-of-the-art client-server database systems employ either a data-shipping or a query-shipping execution policy [Fra93]. Using *data shipping*, all application-level database operations (e.g., methods or joins) are executed at the client workstation at which the operation was initiated. Upon

[1] http://www.cs.umd.edu/projects/dimsum/index.html

request of a client, data in the granularity of objects or database pages are shipped from a server to the client. In a system with many servers, *yellow pages* or name servers keep the information which servers store copies of objects [EGK95].

Using *query shipping*, database operations are carried out on servers. For example, a method is carried out at a server that owns a copy of the object to which the method is bound, and a (functional) join can be carried out at a server that owns copies of objects of the inner class of objects or of the outer class of objects.

Both query shipping and data shipping impose strong restrictions upon where operations can be carried out, and therefore, sometimes do not fully exploit the resources of a distributed system. In [KF95], we identified a more flexible execution policy referred to as *hybrid shipping*. Combining the options provided by data and query shipping, hybrid shipping allows operations to be carried out at servers and at client workstations. For example, a method can be carried out at a client or at a server that owns a copy of the object to which the method is bound. Similarly, a join can be carried out at the site of the inner relation, of the outer relation, or of the consumer operator that processes the result of the join (possibly, the client), as observed in [MGS+94].

	Communication Cost	*Local Processing Cost*	*Parallel Execution*
Data Shipping	low, if high locality high, if low locality	sometimes high (fast servers, indexes)	not possible
Query Shipping	sometimes high (navigation, no caching)	high, if all servers are heavily loaded	possible
Hybrid Shipping	best possible	best possible	possible

Table 1: Performance of Execution Policies

Table 1 assesses how well the three execution policies perform in terms of the performance factors identified in the introduction. For caching data at clients, data shipping has relatively low communication costs if the same objects are repeatedly referenced at the client. To select a couple of objects from a large collection of objects, however, data shipping will unnecessarily move the whole collection of objects to the client. Another disadvantage is that data shipping does not utilize fast servers or indexes that might be available on servers, but not on a client workstation. Executing all the database operations on one site, data shipping cannot benefit significantly from parallelism; only techniques such as prefetching can be exploited to overlap some computation at the client with the fetching of objects from servers.

In the way query-shipping systems are built today[2], objects are not cached at clients, and therefore, query-shipping systems are not appropriate for applications that navigate through the object base and require a light-weight, fine-grained interaction with the database system. In addition, query shipping shows poor performance if the servers are heavily loaded. Query shipping, however, supports well the parallel execution of a query on several servers.

Obviously, hybrid shipping has the potential to match at least the best performance of data or query shipping; it shows better performance than both in many situations. However, the increased flexibility to carry out database operations must be paid for with an increased complexity in query optimization: in addition to classical query optimization (e.g., join ordering), the optimizer must decide where to execute an operation; this process is referred to as *site selection*. Site selection must also be carried out for data shipping and for query shipping: for example, the system must decide which copy of an object to use, if objects are replicated. But, site selection is most complex for hybrid shipping.

[2]One exception is ADMS [RK86], which caches query results at clients.

2.2 Dynamic Query Optimization

Query optimization can be carried out in one of the following three ways[3]:

Static: At compile-time, the query is parsed and a query evaluation plan is generated specifying the join order and where every operator is to be executed (site selection).

2-Step: Classical query optimization (join ordering, etc.) is carried out at compile-time thereby generating an *incomplete* plan. At activation-time, the *incomplete* plan is refined carrying out site selection depending on the location of copies of objects and the load of the machines. A similar approach was first proposed in [CL86].

Dynamic: Only parsing is carried out at compile-time, and the optimization of a query (join ordering and site selection) is completely carried out while the query is activated.

Potentially, a query can be executed many times, and the state of the system can change significantly between the compile and an execution-time of a query. Static query optimization, therefore, cannot generate optimal plans in most situations: plans generated by static query optimization can have high communication costs (if objects migrate), high local processing costs (if operations are executed on loaded machines), and possibly may not exploit parallelism very well (ignoring new machines). Carrying out site selection dynamically, 2-step optimization can exploit the client's cache and minimize local processing costs by carrying out the operators on idle or lightly loaded sites. If objects migrate between compile and execution-time, however, sometimes unnecessary communication overhead is required to carry out a query in the join order specified at compile-time. In addition, the degree of inter-operator parallelism is limited, if a left-deep join tree is generated at compile-time. Whereas static query optimization requires virtually no overhead when a query is activated, dynamic site selection can be expensive and must be carried out every time the query is activated.

Obviously, full dynamic query optimization generates the best plans given precise information of the state of the system at activation-time. However, full query optimization can be very complex, and therefore, generating a query plan when a query is activated can be very expensive. In summary, Table 2 lists the trade-offs of the query optimization techniques.

	Communication Cost	*Local Processing Cost*	*Parallel Execution*	*Activation*
Static	sometimes high (migration of objects)	sometimes high (loaded machines)	sometimes not exploited (new servers)	very cheap
2-Step	sometimes high (poor join order)	best possible	sometimes not exploited (left-deep join order)	expensive
Dynamic	best possible	best possible	best possible	very expensive

Table 2: Trade-Offs of Static, 2-Step, and Dynamic Query Optimization

3 Performance Results

To give an impression on the significance of the trade-offs, we present the results of performance experiments that were obtained using a response-time and cost model. The execution and activation times of a 10-way join query (with 10 sets of 10,000 objects of 100 bytes each) were measured. All

[3]For centralized database systems in which the search space of a query optimizer is much smaller, dynamic query optimization has been studied in, e.g., [CG94].

the joins were functional joins that, for example, occur in path expressions. Every tuple of a relation matched exactly one tuple of another relation. The whole experimental environment is described in detail in [KF95].

Table 3 lists the execution time of the 10-way join query in 4 different system configurations using the three execution policies described in Section 2.1 and using full dynamic query optimization to generate the plans. Table 4 lists the activation and execution times of the same query in the same system configurations; here, hybrid-shipping plans were generated by static, 2-step, and full dynamic query optimization. The header of every column indicates which performance factor had the largest impact on the performance of a technique: the communication cost factor was analyzed in a slow, wide-area network; the influence of local processing costs was studied measuring the execution time of the query in a system with one heavily loaded server; and the effect of parallelism was investigated comparing the execution time of the query in a system with 1 and 5 servers. The activation time for static, 2-step, and dynamic query optimization was determined by measuring the time from the initiation of a query at the client (by the user) until the execution of the query began, and it was virtually independent of the number of servers and the network of the system.

	Comm. Cost	*Local Proc. Cost*	*Parallel Execution*	
	Slow Network 5 Servers	Fast Network 1 Loaded Server	Fast Network 1 Unloaded Server	Fast Network 5 Unloaded Servers
Data Shipping	2787	400	165	165
Query Shipping	1417	1709	214	60
Hybrid Shipping	1398	400	113	55

Table 3: Query Execution Time in Seconds (Dynamic Query Optimization)

	Comm. Cost	*Local Proc. Cost*	*Parallel Execution*		*Activation*
	Slow Network 5 Servers	Fast Network 1 Loaded Server	Fast Network 1 Unloaded Server	Fast Network 5 Unloaded Servers	All Systems
Static	2500	1709	113	115	0
2-Step	2222	400	113	56	8
Dynamic	1398	400	113	55	40

Table 4: Query Activation and Execution Times in Seconds (Hybrid Shipping)

Going into the details of these performance experiments is beyond the scope of this position paper. Nevertheless, it can easily be seen that the experiments confirm some of the effects pointed out in Tables 1 and 2 (for benchmarking only one query, not all the effects can be observed). Clearly, a hybrid-shipping plan generated by dynamic query optimization always has the lowest execution time—compared to a data or a query-shipping plan or a plan generated by static or 2-step optimization. The increased activation time to optimize a query just before it is executed, however, does not always pay off.

4 Summary

Due to the wider acceptance of object-oriented database technology, object base systems will have to support a large diversity of applications: in addition to computation-intensive applications, data-intensive applications will have to be carried out efficiently. Furthermore, object bases are going to be partioned over a large number of autonomous servers all around the world. The heterogeneous nature of the sites and communication links and the dynamic expansion of systems require the use of new architectures for distributed query processing. In many situations, dynamic query optimization and the use of a flexible execution policy are going to be essential to achieve acceptable performance. A great deal of research, however, is still required to fully understand all the performance trade-offs and to develop robust methods to process queries with low activation and execution times in distributed object base systems.

References

[Cat94] R. G. G. Cattell, editor. *Object Database Standard.* Morgan-Kaufmann Publishers, San Mateo, CA, USA, 1994.

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

[CG94] R. L. Cole and G. Graefe. Optimization of dynamic query evaluation plans. In *Proc. of the ACM SIGMOD Conf. on Management of Data*, pages 150–160, Minneapolis, MI, USA, May 1994.

[CL86] M. Carey and H. Lu. Load balancing in a locally distributed database system. In *Proc. of the ACM SIGMOD Conf. on Management of Data*, pages 108–119, Washington, USA, 1986.

[DNSV95] D. DeWitt, J. Naughton, J. Shafer, and S. Venkataraman. Parallelizing OODBMS traversals: A performance evaluation, May 1995. Submitted for Publication.

[EGK95] A. Eickler, C. A. Gerlhof, and D. Kossmann. A performance evaluation of OID mapping techniques. In *Proc. of the Conf. on Very Large Data Bases (VLDB)*, Zürich, Switzerland, September 1995.

[Fra93] M. Franklin. *Caching and Memory Management in Client-Server Database Systems.* PhD thesis, University of Wisconsin, Madison, Wisconsin, June 1993.

[KF95] D. Kossmann and M. Franklin. A study of query execution strategies for client-server database systems. Technical Report CS-TR-3512, University of Maryland, College Park, MD 20742, August 1995.

[KK95] A. Kemper and D. Kossmann. Adaptable pointer swizzling strategies in object bases: Design, realization, and quantitative analysis. *The VLDB Journal*, 4(3):519–566, August 1995.

[MGS+94] D. Maier, G. Graefe, L. Shapiro, S. Daniels, T. Keller, and B. Vance. Issues in distributed object assembly. In T. Özsu, U. Dayal, and P. Valduriez, editors, *Distributed Object Management*, pages 165–181, San Mateo, CA, USA, May 1994. Morgan-Kaufmann Publishers. International Workshop on Distributed Object Management.

[Mos92] J. E. B. Moss. Working with persistent objects: To swizzle or not to swizzle. *IEEE Trans. Software Eng.*, 18(8):657–673, August 1992.

[RK86] N. Roussopoulos and H. Kang. Principles and techniques in the design of ADMS. *IEEE Computer*, 19:19–25, December 1986.

[SAD+94] M. Stonebraker, P. M. Aoki, R. Devine, W. Litwin, and M. Olson. Mariposa: A new architecture for distributed data. In *Proc. IEEE Conf. on Data Engineering*, pages 54–65, Houston, TX, 1994.

[SDK+94] M. Stonebraker, R. Devine, M. Kornacker, W. Litwin, A. Pfeffer, A. Sah, and C. Staelin. An economic paradigm for query processing and data migration in Mariposa. In *Proc. of the IEEE Conf. on Parallel and Distributed Information Systems*, pages 58–67, September 1994.