Client-Server Semantic Binary Database: Design and Development

Konstantin Beznosov

High Performance Database Research Center

Florida International University

http://www.cs.fiu.edu/˜beznosov

December 9, 1996

Abstract

This paper describes design and implementation of client-server architecture for Semantic Binary Database Management System developed at High Performance Database Research Center[1], Florida International University[2]. We present a conceptual view of the system architecture, give a detailed picture of its layers responsible for client-server interaction, describe implementation issues, and, if time constraints allow[3], present performance tests results.

The document is available in electronic form at http://www.cs.fiu.edu/˜beznosov/client-server-SDBMS.

1 Introduction

The semantics and object-oriented database technologies are emerging in more and more application areas where they are found to be more suitable to problem model, more easier to design, maintain and provide

[1]http://hpdrc.cs.fiu.edu

[2]http://www.fiu.edu

[3]This paper is originally written as a project in "Advanced Database Systems" Fall 1996 class COP6545 lead by Professor Dr. Chen. So, it had to comply with regular school deadlines, etc.

data integrity, than current relational database technology. Advocation of semantics model as well as a detailed description of it can be found in [Ris92]. One of the semantic binary database prototypes is under development at FIU High Performance Database Research Center. The prototype will be referenced further as *DBMS*.

The remainder of this paper is organized as follows. Section 2 describes the architecture of the DBMS. Section 3 presents the design of client-server version of the DBMS. Details of the implementation are explored in Section 4. Performance considerations and test results are discussed in Section 5. Conclusions are contained in Section 6.

2 Semantic Binary Database Architecture

The principal structure of the DBMS is depicted on Figure 1. The user application only interacts with semantic engine through semantic engine interface. The semantic engine, in its turn, uses interface to B-Tree to initiate a session, conduct queries and transactions as well as to end a session. Although B-Tree engine is designed as B-tree with its unique decisions, modifications and optimizations, in this paper, *B-Tree* refers not to the type of data storage structure known as *B-tree* but to the constructive layer of the DBMS. To distinguish these two names, we will refer to the corresponding DBMS layer as B-Tree (with capital “T”) and as B-tree to data storage structure.

2.1 Semantic Engine

2.2 B-Tree Engine and Interface

Each DBMS has a single B-Tree. B-Tree stores all data in its internal representation on local disks, as well as it ”remembers” results of recently served queries in the main memory cache. The current version of B-Tree is a direct implementation of Semantic Binary Model defined in [Ris87] and the data structure closely

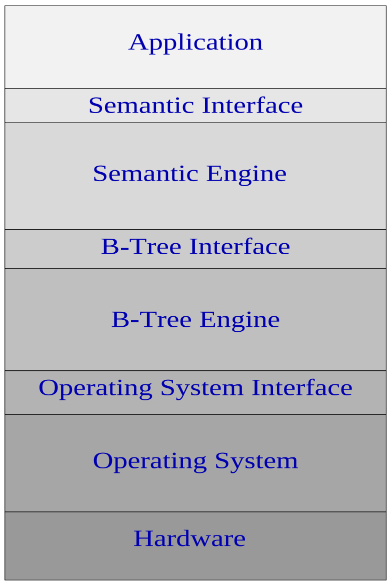

Figure 1: High level structure of semantic database

resembles that described in [Ris91]. A detailed description of B-Tree design is beyond the scope of this paper. The interface to B-Tree is more important in the paper context.

B-Tree interface provides the following simple operations to semantic agent, which acts on behalf of the currently running application[4]:

Login Authorizes the user of the application to interact with the DBMS and begins a *session*[5] by opening specified DBMS in requested mode[6].

Query Retrieves information from the DBMS submitting query in the form of *atomic retrieval operation* described in [Ris91, p.4]. The semantic agent receives blocks of retrieved information.

Transaction Submits transaction by providing a set of facts to be inserted into the DBMS, a set of facts to be deleted from the DBMS, as well as a list of fact ranges needed to verify that there is no interference

[4]Due to software development life-cycles the interface is subject to change as long as it is internal and the database development continues. The main idea of providing service to semantic engine with ability to submit query and to perform transaction will stay the same though.

[5]After session is opened, all queries and transaction are submitted without athorization information.

[6]Creating of new database is just one of possible modes in analogy with Unix operation *open*.

between transactions of concurrent programs. If the transaction failed, the semantic agent receives a list of fact ranges that were interfering with the transaction.

Logout Ends the current session.

3 Client-Server Design

In client-server version, the following partitioning of DBMS functionality has been chosen (Figure 2): semantic engine resides on a client and B-Tree engine resides on a server. In this manner, multiple instances (on each client) of semantic engine interact with one instance of B-Tree[7] (on a server). While performing queries, each client accumulates results of those queries in its own cached image of the server B-Tree called *local B-Tree.*

4 Implementation

Communication engine has been implemented by means of Open Network Connectivity Remote Procedure Call (ONC RPC, further just RPC) industry standard widely available on Unix and most of other platforms in commercial as well as public domain variations.

Each call to B-Tree interface on the client site is translated in to a sequence of calls, each of them manipulates with only small and simple data structures. Every such call is performed remotely on the database server containing actual B-Tree engine via RPC facilities. Results are translated back into complex data-structures of potentially unlimited size, and are returned to Semantic engine.

[7]In case of distributed database, multiple servers will be hidden behind B-Tree interface such that clients will still see one logical server. Although each client might interact with a different server in fact.

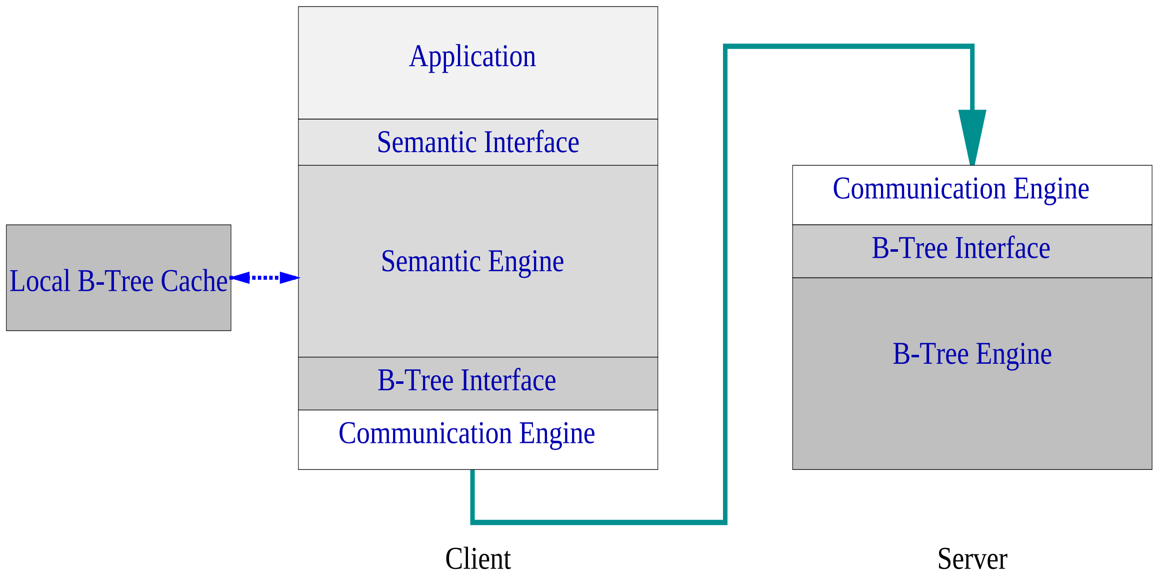

Figure 2: High-level structure of client-server design

5 Performance

To better understand the effectiveness of the current design and implementation, we conducted a number of performance tests for different system configurations.

5.1 Configurations

We set four different configurations of the DBMS:

1. Standalone configuration presented on Figure 1 where database files reside on local disk. This configuration is used as a reference since for this standalone mode and local disks, neither network delays nor communication overhead are presented. We will refer to this architecture as **standalone local** approach.

2. Standalone configuration where database files reside on remote disk[8] and accessed via remote file service such as SUN NFS [Sun88]. It is called **standalone NFS**.

3. Distributed configuration where a client and a server are two different processes and they are located on the same machine so that no network delay (only communication overhead) is introduced. This configuration is referred to as **local client-server**.

4. **Distributed client-server** configuration where a client and a server are located on different hosts of 10 Mbps Ethernet LAN.

5.2 Benchmarks

To compare the performance of described configurations, we used a suite of TPC-C[9] standard business database benchmark developed by benchmark group of SDB project. Since the process of running benchmarks did not comply[10] 100% with official TPC-C specification we call it qTPC-C. We believe that results are still useful because we compared different configurations of the same database, and we avoided comparisons with results of TPC-C benchmark run on different database systems.

Solaris 2.5.x platforms have been choosen for experiments. For each run, total 1000 complex TPC-C specified transactions have being applied to the database created before. Original size of the database is 195 MB. The first and last 50 transaction were ignored to count performance only of system in the "stable" state. During each run, the database grew every time up to about 222 MB. Final size of local B-Tree cache of semantic engine[11] was total 2 KB. In distributed client-server as well as standlone NFS configurations, B-Tree engine were running on computer with more "horse power." The same machine was

[8]This particular test configuration is inspired by [DMFV90].
[9]http://www.tpc.org/bench.descrip.html
[10]The main deviation was ommiting various delays requied by the specification so that the tests could be accomplished in shorter time.
[11]Local B-Tree cache was created for every configuration, even standalone for sake of uniform interface to B-Tree. That is, Semantic layer is not aware wether B-Tree server is on local or remote host.

used for configurations where semantic and B-Tree engines were supposed to be on the same host.

5.3 Result Analysis

This section presents results of qTPC-C benchmark runs. Total performance is represented by Accumulated Transactions Per Minute (TPM) value. Specific types of qTPC-C a represented separately and they show Accumulated Average Responce Time (AART) for each type of transaction in fractions of second. In all cases, ATPM and AART are measurements based on all transactions executed since the beginning of the run, hence Accumulated. CPU time used by process is used to compute results.

5.4 Total Performance

The most important performance characteristic in TPC-C suite is total throughput of the system in terms of accomplished transactions per minute.

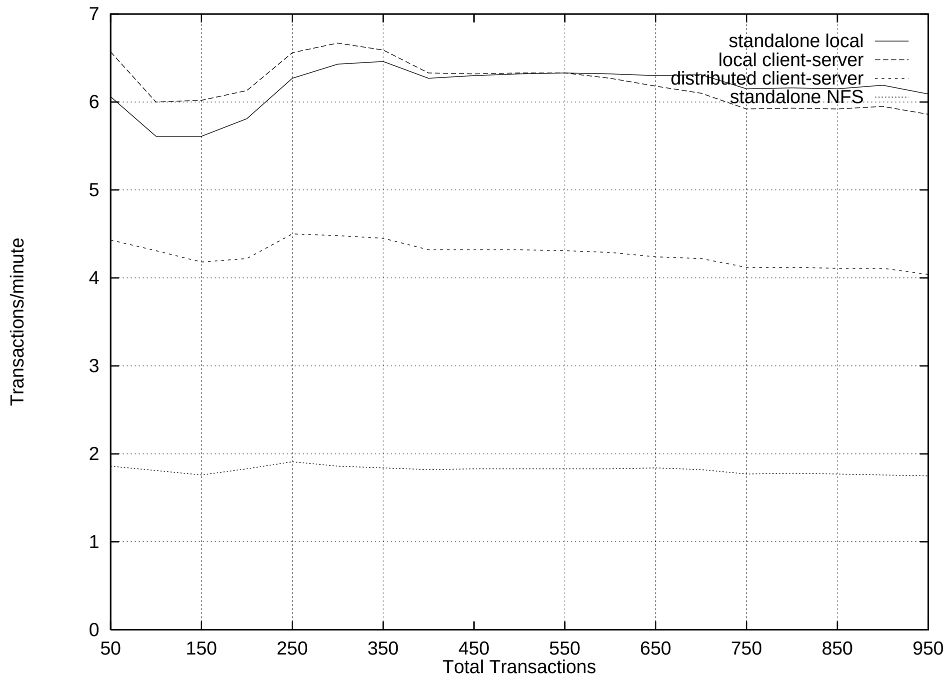

Figure 3: Transactions/minute

As we can see from 3 the performance of the system is the worse when remote file service like SUN NFS is used to provide access to the data. Two-fold better performance is observed for distributed client-server. If network delays are avoided (*local client-server*) client-server configuration is as good as *standalone local* configuration. We will see further local client-server version even out-performs on all types of transactions.

5.5 Transactions By Type

Performance of different system configurations by transaction type was measured in terms of average transaction responce time accumulated over all executed transactions.

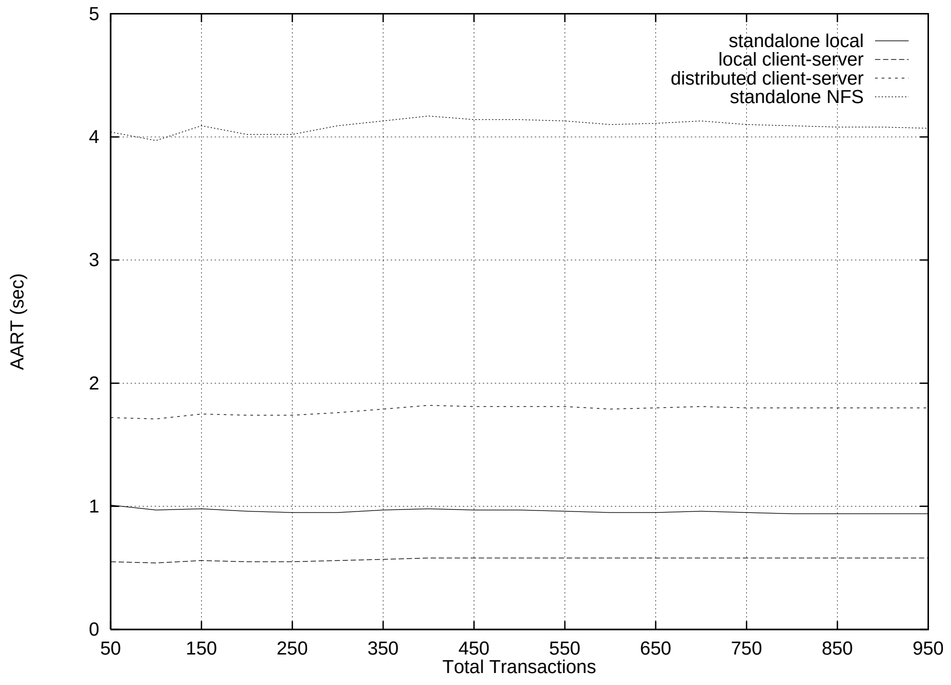

Figure 4: New-Order Transaction

In each of five transaction types, the system demonstrated substantial advantage of *distributed client-server* configuration over *standalone NFS* in response time. Main reason *local client-server* configuration outperforms *standalone local* is that only CPU time spent on client site of the transaction was measured.

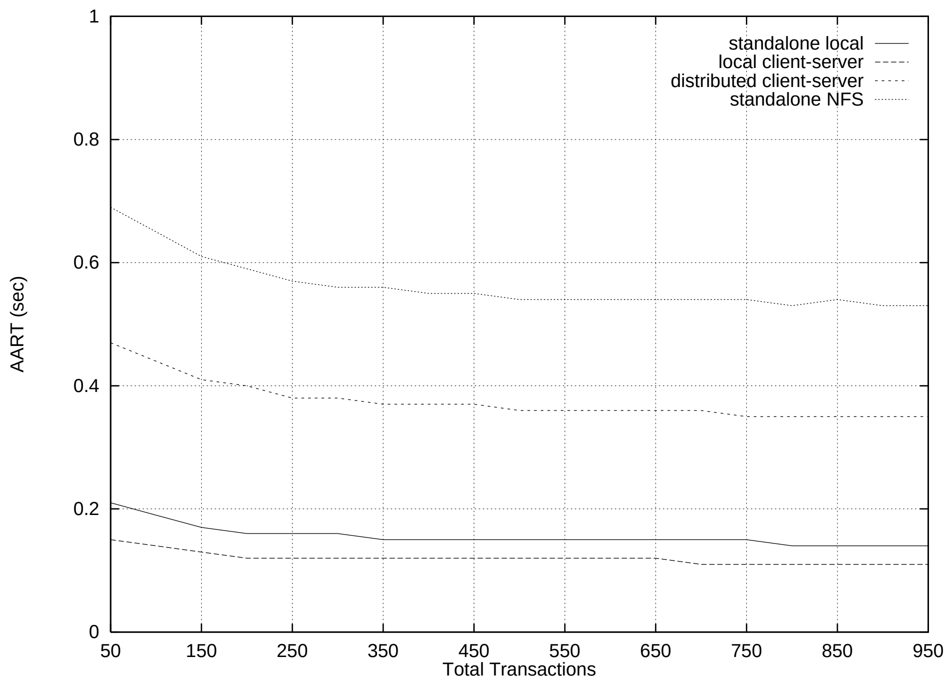

Figure 5: Payment Transaction

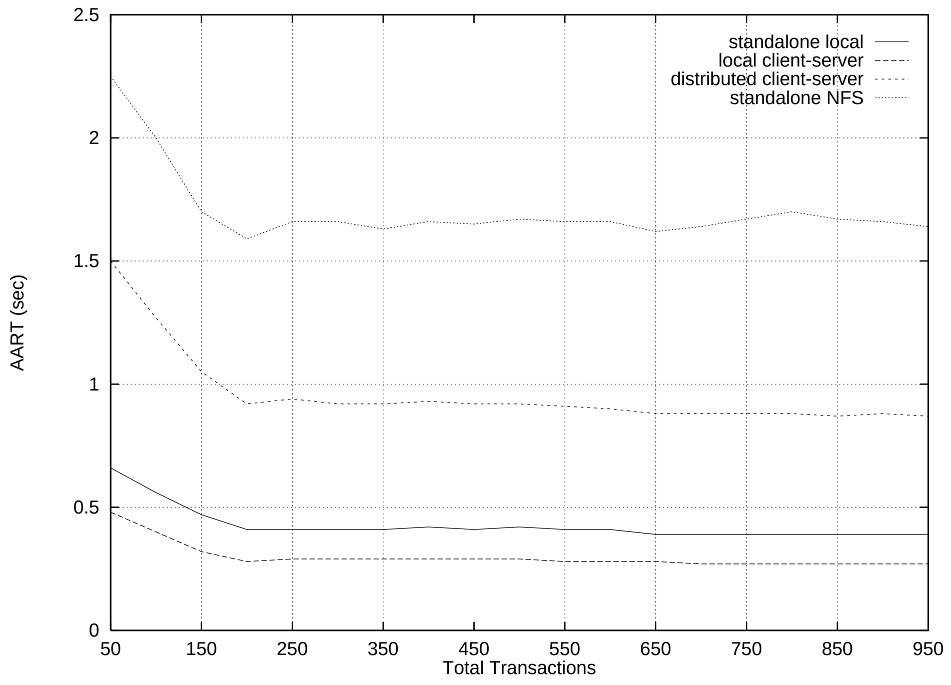

Figure 6: Order-Status Transaction

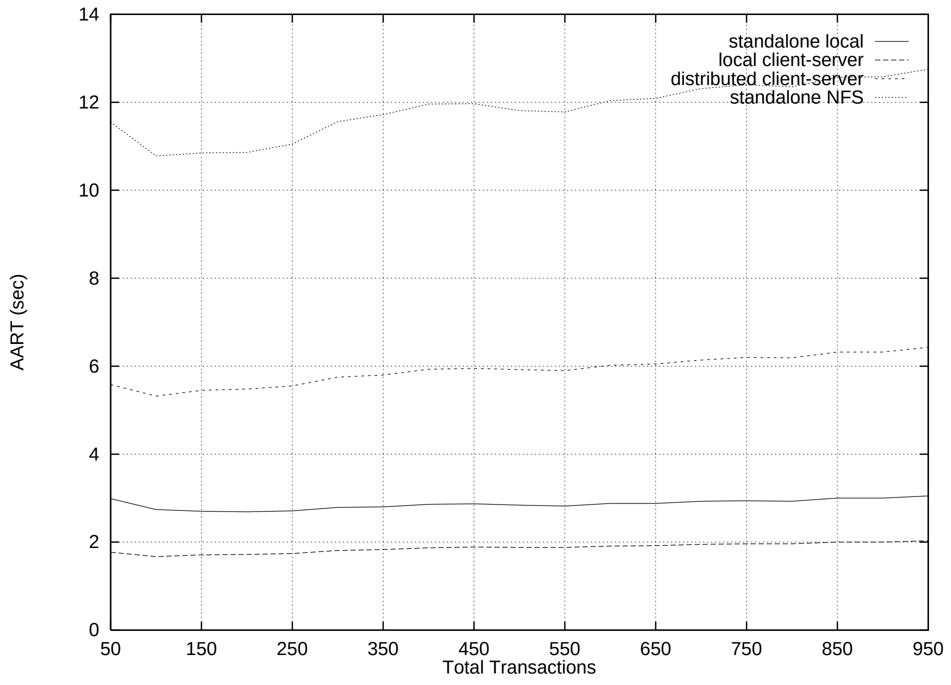

Figure 7: Delivery Transaction

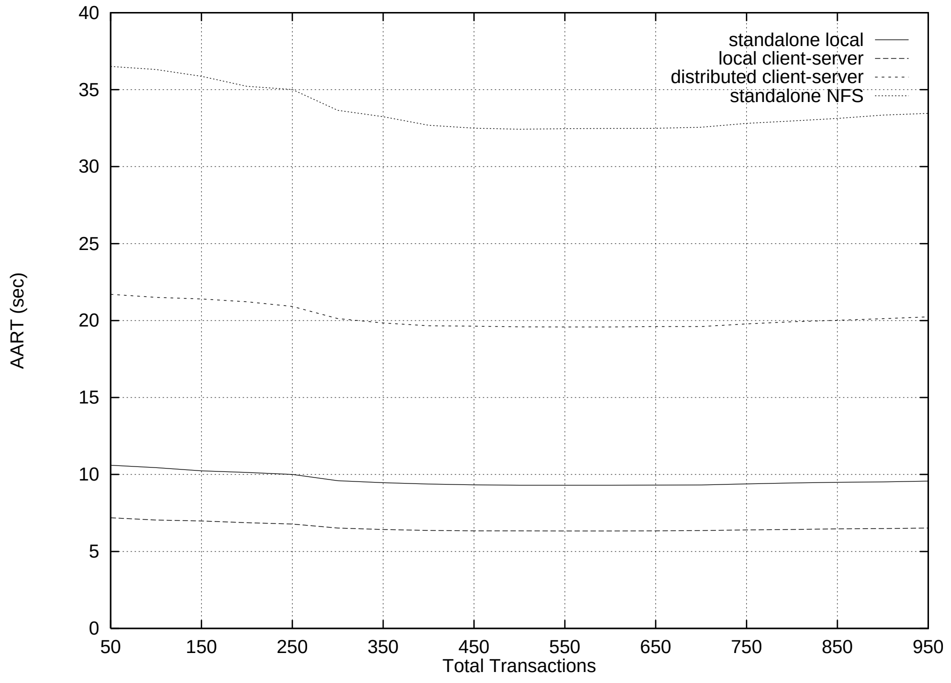

Figure 8: Stock-Level Transaction

6 Conclusion

The client-server version of Semantic Binary Database is implemented and preliminary results of simplified TPC-C benchmark show that the implementation can have good performance when it is optimized and tuned.

6.1 Future Work

The main next step is to conduct more precise tests. For example fully compliant with specification TPC-C benchmark will be a very good aid to analyze system performance objectively. It will be interesting to see the difference in performance via 10 Mb/s Ethernet and 155 Mb/s ATM[12] technologies, as well as emulated by using modem connection WAN. It is necessary to measeer CPU time spent by server on each transaction to have more realistic results. Also its scalability should be measured to appreciate real advantages of client-server implementation.

The second phase in completing client-server implementation should be optimization of communication engine. Feasible directions in this phase will be clearly seen after getting more precise measurements. For now we can state the following possible ways to achieve better performance: minimization number of RPC calls per transaction and query call, minimization of memory based operations might also help in gaining better performance.

References

[DMFV90] David DeWitt, David Maier, Philippe Futtersack, and Fernando Velez. A study of three alternative workstation-server architectures for object-oriented database systems. In *Proceedenings of 16th VLDB Conference*, Australia, August 1990.

[12]Using IP over ATM [Lau94].

[Lau94] M. Laubach. Classical IP and ARP over ATM, RFC#1577, January 1994. The document is available in electronic form at http://fury.nosc.mil/general/atm-rfc/rfc1577.html.

[Ris87] Naphtali Rishe. Database semantics. Technical Report TRCS87-2, Univeristy of California, Santa Barbara 1987.

[Ris91] Naphtali Rishe. A File Structure for Semantic Databases. *Information Systems*, 16(4):375–385, April 1991.

[Ris92] Naphtali Rishe. *Database Design: The Semantic Modeling Approach.* McGraw-Hill, 1992.

[Sun88] Sun Microsystems. *Network Programming Guide*, May 1988. Part Number: 800-1779-10.