IBPhoenix Research

Conceptual Architecture for InterBase/Firebird

Hubert Chan and Dmytro Yashkir
Faculty of Mathematical, Statistical and Computer Sciences
University of Waterloo, Canada.
January 29, 2002

Abstract

In this paper, we investigate the conceptual architecture for the InterBase/Firebird database. We identify the major components of the database system and their interactions. Some of these components themselves have their own architectures, which we will also discuss. We present some scenarios which illustrate the interactions between the components. We show that the top-level architecture is styled after a pipe and filter architecture, while each component may have its own architectural style.

Table of Contents

Introduction

Software architecture presents developers with a tool for organizing large, complex software systems into various components, allowing the developers to better understand the system as a whole by reducing the number of components which must be kept in mind. It allows each developer to focus on a smaller part of the system, and enables the development team to integrate its work together by specifying the interfaces between the various parts in a clear and well defined manner.

There are several different types of architectural views. The conceptual architecture is a high-level view, with relatively few details, and is particularly well suited for describing the functionality of a system.

In this paper, we investigate the conceptual architecture for the InterBase/Firebird database. Interbase is developed by Borland, and an open source version (Interbase 6.0) was released in July 2000 under the InterBase Public License (derived from the

Mozilla Public License). However, due to Borland's attitude towards community developers, the Firebird project was created, developing its own database based on Borland's source code. However, due to their common lineage, InterBase and Firebird share a common architecture, so for the purposes of this report, we will consider them to be the same.

InterBase includes many tools and utilities for developing and administering a database, and the code base includes code from older versions, as well as some failed attempts. We will not be considering these in our analysis.

We begin by discussing the overall architecture of the system. We then look at the architectures of some of the subsystems, and we finally present scenarios which illustrate the behaviour of the system in response to queries generated by clients.

Top-level architecture

InterBase can be divided into four major components (see Figure 1): the remote connection system, the SQL translator, the relational database engine, and the lock manager. The arrows in the diagram indicate the flow of data. We include the clients in the diagram to illustrate their relationship with the rest of the system.

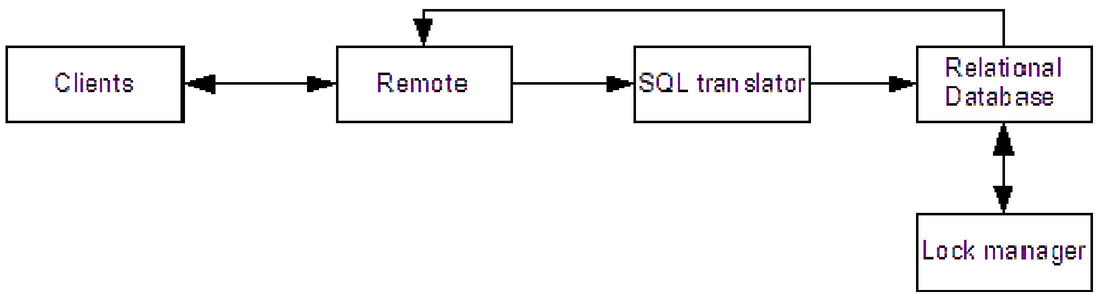

Figure 1: Top-level architecture

remote connection system (REMOTE): This subsystem allows remote clients to connect to the database over different network protocols. It is composed of two parts: a client-side component and a server-side component.

SQL translator (DSQL): This subsystem translates requests from SQL into BLR, the native language of the database.

relational database engine (JRD): This subsystem performs the actual queries.

lock manager (LOCK): This subsystem handles synchronization among transactions.

The arrangement of the remote connection system, SQL translator, and relational database engine, can be seen as a pipe and filter style of architecture: a request flows through the remote connection system, to the SQL translator, where it is converted to BLR. The BLR goes through the relational database, which returns a request through the remote connection system.

Remote communication

The remote communication subsystem, REMOTE, allows clients to communicate remotely, or locally, with the server. It enables communication over several different network protocols, such as TCP/IP, MS LanManager, and SPX. The subsystem is split roughly into two parts: a server side, and a client side. It contains generic code for client-server communication, as well as protocol-specific code.

It can be viewed as a layered system: conceptually, the client sends requests to the server, through a generic communication layer. The generic layers communicate through a protocol-specific layer, and the protocol-specific layers communicate through the operating system's network stack. This is much like to a two-layered version of the OSI reference network model.

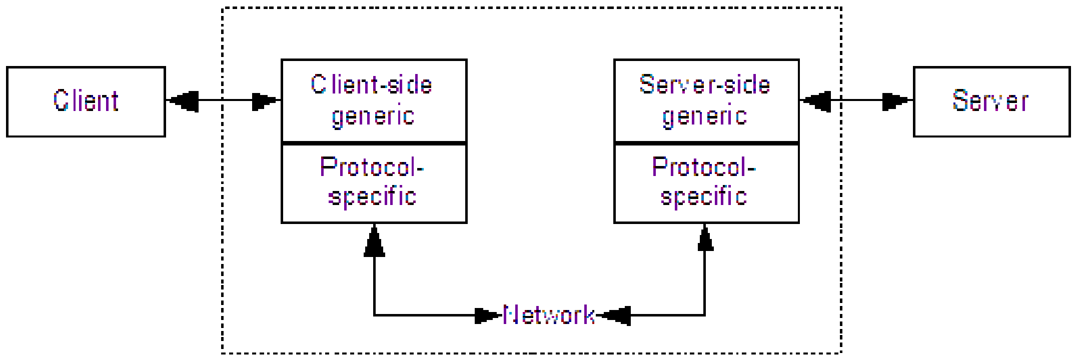

Figure 2: Remote connection

The client can also communicate with the server locally, using a module which emulates a network connection by using shared memory.

SQL translator

The SQL translator,>DSQL, converts SQL queries into the native BLR language. Its architecture is like that of a simplified compiler: it contains a lexer, parser, symbol table, and code generator.

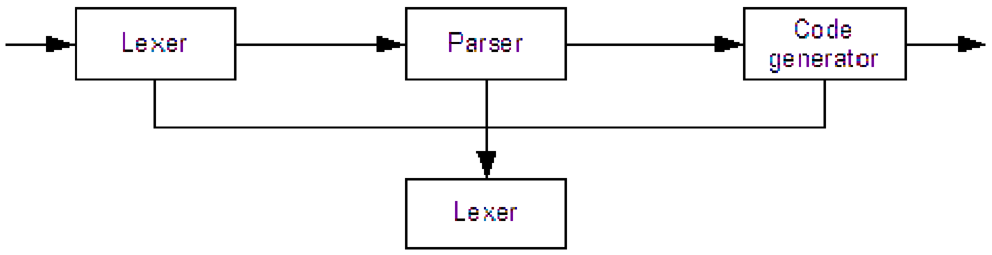

Figure 3: SQL translator

As in all compilers, the lexer, parser, and code generator are arranged, conceptually, as a pipeline. The lexer divides the input into tokens, the parser determines the meaning of the input, based on the tokens, and the code generator emits BLR code which is equivalent to the original SQL.

JRD

The JRD subsystem executes requests and returns their results. It handles the access to the disk through a virtual IO system, verifies that security constraints are followed, and ensures that transactions are handled atomically.

Requests first pass through a compiler, which translates from BLR into an internal representation of the request. It calls the metadata subsystem, MET, to get metadata pertaining to the request, and to ensure that the requested tables are present.

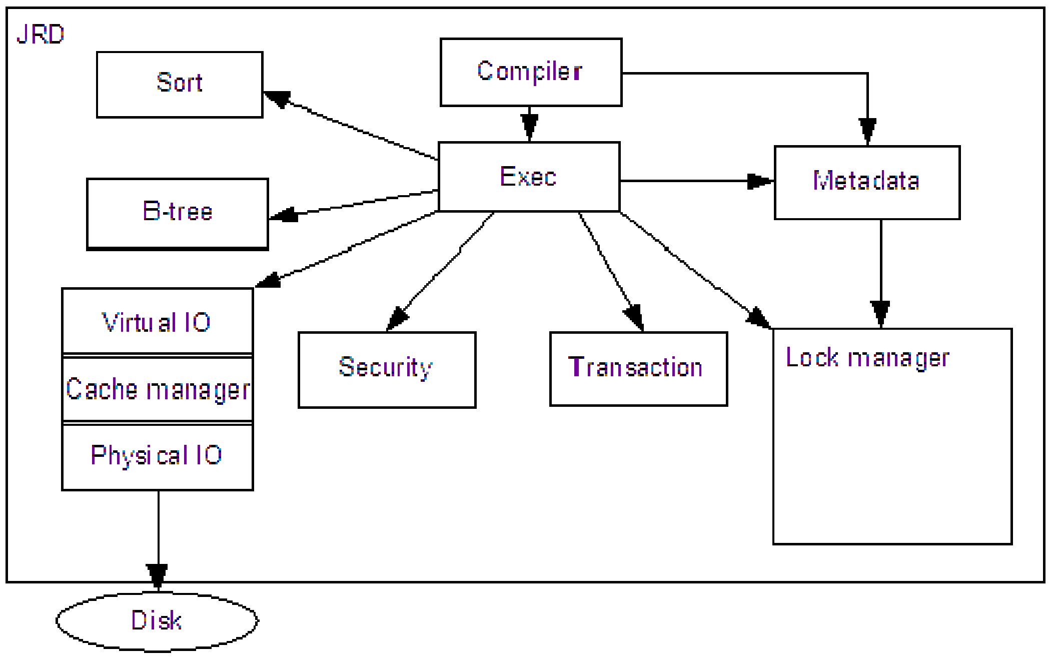

Figure 4: Relational database engine

The Exec subsystem then processes the requests, using the B-tree subsystem for indexing, the security subsystem to check user priviledges, the transation subsystem to ensure atomic execution of a series of requests, the sort subsystem to sort, and the lock manager to ensure concurrency. It uses a virtual IO system to access the disk, and it uses the metadata subsystem to operate on metadata.

The virtual IO subsystem is a layered system with three layers. The top layer presents an abstract method for accessing the data on the disk. The second layer, the cache manager, handes caching of the data, to speed up data access. It is the last layer which has a concept of structured data in the database. The final layer is a physical IO layer which is specific to the operating system on which the engine is run, and makes the system calls to access the disk.

JRD and Lock module

The main purpose of the lock module is concurrency control, when multiple users are accessing same database file simultaneously. Such situations are a common occurrence during the normal operation of any DBMS.

Lock handling is separated into two major parts: the lock handler sub-module inside the JRD and the Lock module that handles concurrent access to the lock table. Figure 5 illustrates relationship between these parts.

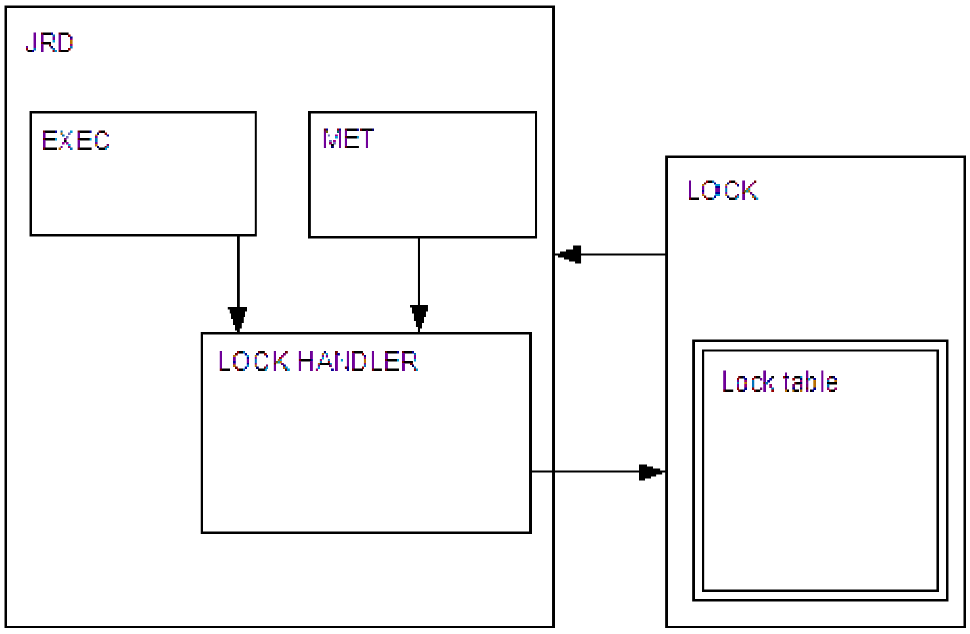

Figure 5: JRD and Lock module

Requests that need access to the lock mechanism are divided into two major categories, metadata modification requests and usual data requests. Both of these categories use lock the handler to be able to access the lock mechanism.

Normally when modification is needed, a lock on the appropriate data is requested. Then modification is performed and the lock is released. If it is impossible to gain a lock, the lock handler can wait certain amount of time for the other lock handler to release the lock and then resume normal execution.

The Lock module itself waits for the requests from lock handlers. From their requests it performs modifications to the lock table.

Use scenarios

This section describes two common scenarios of the InterBase operation. These help better understanding of the DBMS architecture.

Scenario 1: New table creation

This scenario describes the creation of a new table in the database. When reading through the following description, please refer to the appropriate diagrams above.

1. Request originates from the user application, and is passed to Remote module client side
2. Requests is packaged depending on the network used and passed through the

network to the Remote server side
3. DSQL is called to transform request from SQL to BLR
4. JRD is called, CMP is called to compile the request
5. Exec is called to execute the request, MET is called
6. Request begins execution in MET since it modifies metadata for the DB
7. Lock handler is called to obtain a lock on the metadata of the DB
8. Lock handler calls Lock, which adds appropriate lock to the lock table
9. MET calls virtual IO library to commit changes to the disk
10. Appropriate disk handling routine is called depending on the file system
11. MET calls Lock handler, which calls Lock to remove lock from the Lock table
12. JRD calls Remote module to return success message to the user program
13. Remote moves the message through the network and get it to the user application

Scenario 2: Search for a row by index field

This scenario describes simple search request to the database.

1. Request is created in the user application and passed to the Remote client side
2. Remote moves request to the server side and calls DSQL
3. DSQL transform SQL of the request into BLR
4. JRD is called and CMP compiles the request
5. EXEC starts executing the request
6. Virtual IO is called to get appropriate table
7. Cache is checked for the data requested
8. Search routine in B-tree module is called to find row with appropriate index
9. Remote is called and returns result of the search to the user application

Extensibility of InterBase

A good test for any system architecture is how it can accommodate future changes and expansions. InterBase existed under different names since 1985; obviously large number of changes and enhancements was implemented since then. However the basic conceptual architecture did not change.

First, let's examine some changes made in the Remote module. With the appearance of a large number of new LAN standards, Remote was modified to allow it to work with them. Since all of the network communication routines are isolated in the remote module, these changes do not affect other parts of the system. An example of such a modification is the addition of the IPSERVER module inside Remote. It was added to allow users to run InterBase on a single Windows-based machine where client and server share one machine. This modification was relatively simple, even though there is no LAN involved whatsoever in this implementation.

A second example of modifications is the addition of the Windows file system to the Virtual IO module. To add the ability to run InterBase on a machine using the Windows file system, the only modification required was to add another subsystem in the IO system that enables operation with the Windows file system.

Conclusion

In this paper, we discuss the conceptual architecture for InterBase. The system is composed of four major compenents, arranged in a pipe and filter style. We also look at the architecture within some of the subsystems. We provide scenarios illustrating the behaviour of the system in response to different client requests and the interactions between the subsystems, and we demonstrate how the architecture allows for extensions to the system, such as new network protocols or file systems.

References

Garlan, David and Shaw, Mary ,‟ An Introduction to Software Architecture”, *Advances in Software Engineering and Knowledge Engineering*, Volume 1, World Scientific Publishing Co., 1993.
Harrison, Ann and Beach, Paul; ‟ InterBase Source Code Guide”; IBPhoenix; October 2001.
Harrison, Ann; ‟ High-level Description of the InterBase 6.0 Source Code”; IBPhoenix.
Kruchten, P.B., The 4+1 Views Model of Architecture, IEEE Software, Nov 95, pp 42-50.