IBPhoenix

WHITE PAPER

Prepared
by IBPhoenix Consultants

Author: Helen Borrie

Firebird Databases as the Back-end to Enterprise Software Systems

Contents

What is Firebird?......................................2

Software Release History.........................2

Initial Release..2

Current Release Version................................3

In Development..3

Firebird 2...

"Vulcan"...

Firebird 3...

Is Firebird "Enterprise Capable"? 3

Stability..4

"ACID compliance"..4

Track Record and Support.............................4

Future Development.......................................4

Scalability...5

Availability...5

Capacity...6

Interoperability...6

Independence..6

Connectivity..7

Autonomy...7

"Autonomic Features"...................................7

Complexity of Migration and Update...............7

Migration...7

Upgrade Compatibility....................................8

Sub-releases..8

ACID Compliance and Firebird...............8

Atomicity...8

Consistency..8

Isolation...9

Durability...9

Transaction Logging.......................................9

Firebird's Multi-generational Architecture........10

Transaction Logging and Firebird...................10

Who Uses Firebird?................................10

Deployment Samplings................................11

Distributel, Telco Services Provider.................11

Prague Municipal Library................................11

OneDomain, niche BIS....................................12

Moscow Interbank Currency Exchange............13

Factors Inhibiting Scalability..................14

User Base...14

Hardware, Software and Network Limits......14

Superserver...14

Classic...15

64-bit Addressing...15

Connection Pooling..15

Competition for RAM.....................................16

SMP Support...16

"Windows Networking"..................... 16

Database and Software Design.....................17

Garbage Collection...17

Commit Retaining and "Autocommit"..............17

Who Owns and Manages Firebird?........18

Management..18

Code Maintenance..18

Release Policy..18

Open Source Licensing...................................18

Funding..19

Firebird Databases as the Back-end to Enterprise Software Systems

A White Paper compiled by IBPhoenix consultants

What is Firebird?

Firebird is relational database management software, similar in purpose to products such as DB2 by IBM, Oracle, SQL Server by Microsoft and the open source PostGreSQL. The software has two main components: the database server, which lives on the same host machine as the databases, and the application interface, commonly referred to as "the client library". The client library is a run-time component—a DLL on Windows or a shared object (.so) on other platforms—that two-tier deployments need on each client workstation. For multi-tier deployments, where users access databases through middleware from a web browser or other "thin" interface, the Firebird client library is not deployed to end-users at all but is incorporated into the middleware.

The Firebird server boasts a very small "footprint" on the filesystem when installed on a host server. The server's executable is less than 1.5 Mb and a full server installation, including all tools and documentation, takes up less than 10 Mb. The memory footprint will vary according to the scale of the deployment, which can range from a single user running an application over a single database to hundreds of concurrent connections to multiple databases servicing thousands of users on wide-area networks.

Firebird is maintained and developed by a community of developers from around the globe. It is a non-commercial, open source software project owned by the developers. Because the software is distributed completely free of any fees, licensing is not a revenue source to anyone.

Software Release History

Production versions of Firebird have been in in distribution since the beginning of 2002, but the ancestry of the software goes much further back. It began as InterBase in 1985, for the Unix-like VMS platform of that period. It passed into the hands of Borland Software Corporation in the early 1990s through acquisition and evolved through the years to version 5.6. Late in 1999, financial conditions in Borland caused its version 6 development to be stopped. The following year, the InterBase 6 source code was released to the public under an open source licence in July 2000. Two Australian developers downloaded the newly released source code and set up the Firebird Project at Sourceforge, a huge server farm that provides sophisticated facilities free of charge to open source developers.

Initial Release

The months between the collapse of IB development at Borland and the source code release consolidated the former InterBase developer base into a lively, enthusiastic Firebird community of designers, testers, tools and interface developers and support gurus. By the time the source code was released, a substantial team was ready to get stuck in. Firebird never looked back. Firebird 1.0—essentially a code cleanup of the IB 6 C-language code with some important fixes to the build system and some long-standing bugs—was released in 2002 and ran to four sub-releases.

Current Release Version

Firebird 1.5, initially released in March, 2004, was a complete revamping of the codebase from C to C++ to prepare the way for essential architectural enhancements planned for Firebird 2. The most recent production release, Firebird 1.5.3, is highly stable and has had the benefit of some important back-porting from the Firebird 2 development.

In Development

Firebird 2

Firebird 2, which features significant enhancements to many of its subsystems, including the SQL optimizer, is currently in its second beta testing cycle. The full release is expected by mid-year.

"Vulcan"

A "fork" was taken from the early Firebird 2 alpha code in December 2003 for the purpose of redesigning the threading architecture of the database engine. The project, implemented by the original developer of InterBase (Jim Starkey) was commissioned for SAS Institute, the world's largest vendor of business and medical statistics application software. SAS had made the decision in 2003 to move many of its business applications over from Oracle to Firebird. The sources, code-named "Vulcan", were formally handed over to the Firebird project in 2005 and continue to be developed in parallel with Firebird 2. The first public beta-testing cycle of Vulcan is due to begin about now.

Firebird 3

Merging Firebird 2 and the Vulcan code has started, with the objective of releasing Firebird 3 early in 2007. Firebird 3 will have full support for fine-grained thread management on multi-core and multiple-CPU hosts, full utilisation of 64-bit system features and a raft of configuration options for customising both server and database-level security.

Is Firebird "Enterprise Capable"?

Putting a rope around the term "enterprise capable" is a lot harder than counting the hits it gets on Google. The on-line press uses the term as if it were a given, like "newborn" or "duty-free". The inference is that this ephemeral thing is something everyone wants in database software, is either present or absent, and can only be obtained in commercial products.

To address the question in a meaningful way, one first needs to find a context for "enterprise" that fits the business model being addressed. The two rational questions to ask are: What needs, present and future, does Enterprise X have that must be satisfied by the capabilities of the database software? Can DBMS Y satisfy them?

Those of us who sit in the users' gallery rather than the press gallery would put the capabilities for our enterprise context under at least six general spotlights, viz., stability, scalability, availability, capacity, interoperability and autonomy.

Stability

We want a database management system that stores the data that our business needs to store and protects it from corruption from both environmental mishaps and human error. Our system must be able to deliver data to applications in flexible conformations consistently, accurately and with acceptable speed and it must be able to handle conflicting conditions. It must do all of these things at once, without interruptions, stalling, crashing or excessive waiting..

"ACID Compliance"

Such intrinsic stability in a DBMS is an obvious requirement, regardless of any other factors in the wish-list of the enterprise. A set of interleaved principles has evolved over the years defining four essentials that no database management system should lack if it is to be taken seriously. "ACID" is an acronym for these four principles: Atomicity, Consistency, Isolation, Durability.

Firebird's design and architecture are fully "ACID-compliant". The ACID concepts are described below, in the section *"ACID Compliance and Firebird"*, with comments about how Firebird meets each principle. Everything in Firebird is done within the context of an ACID transaction. A Firebird transaction may involve many complex, inter-dependent steps of an enterprise task but it will never permit violation of any ACID principle.

Track Record and Support

With the stability of our production environment in mind, we may not always want to choose a brand-new DBMS to be the heart of our enterprise systems. In general, we prefer to begin with a product that has established its capabilities and whose sweet and sour spots are known and well understood. If end-users are to be responsible for dealing with problems that might occur, we want to know whom they should call on for assistance. If an application software vendor is the one who takes the "000" call, does that vendor have adequate support on hand?

The Firebird end-user community is well-served by highly-capable tool and application developers and vendors. Typically, these developers have been close to the developing source code for six years. Much of their expertise extends much further back, to the earliest days of the Borland development tools, which have always shipped with InterBase developer versions in the enterprise packages. The Firebird developer community is famous for its support forums, where expertise is constantly shared. End-user support contracts, while offered by several companies, are rarely demanded.

Contract support for developers is available in many countries, including Australia. However, it must be said that demand from developers for support, beyond perhaps the installation stage on an unfamiliar hardware platform, is low in most locations.

Future Development

On the other hand, because business demands change and hardware capabilities increase rapidly, we may also want to know whether the DBMS is under active development or is near the end of its development life. We expect regular sub-releases and signs that the next major release is on the way. When we are ready, will it be easy to upgrade? Can our existing databases be migrated painlessly to a new release or will it be a lengthy, do-or-die logistical exercise?

Now in its sixth year, the Firebird Project team continues confidently on its track towards planned releases that will implement architectural enhancements to meet hardware advances and satisfy a demanding community of developers, users and supporters. The openness of the code and the renowned willingness of the community to share what they know ensures a continual supply of up-to-date knowledge about the code modules and the software's capabilities. The project's independence from commercial owners frees end-users from concerns about continuity and ensures that technical discipline prevails over the drive to get releases "to market".

Scalability

Scalability refers to the ability of the DBMS not just to manage the database needs for the current range of enterprise needs but also its capacity to cope with growth (upward scalability) and minimalisation (scaling "down" to single-user, briefcase or embedded appliance models, for example). Scalability considerations address factors such as minimum and maximum numbers of active end-users, whether database growth and rising user numbers affect performance or stability, whether migration of data is involved in ascending or descending the scale of possible requirements. For some enterprises, scalability factors may overlap with other areas of capability, such as availability and interoperability (q.v.).

Scalability in any direction is one of Firebird's strengths. Unlike many software systems competing in the same space, Firebird, since its earliest days as InterBase, was always network software by design, and the on-disk structure of databases has always been managed independently of the host's filesystem, by the database engine itself. In contrast with some heavily marketed, supposedly "enterprise-capable" DBMS offerings that respond to demand for upscaling capability by adding weight and ever more prolific file-bound mechanisms, Firebird's upscaling is merely a question of adapting the environment. The same engine comfortably handles anything from being embedded in a stand-alone client application, through to a classical two-tier client/server LAN of around 750 potential users, to incorporation in a multi-tier solution for thousands of potential clients. Database growth is effectively limited only by the disk storage available and can be split across multiple hard disks.

Through smart replication and good connection management in the access layers, the workload of a busy system can be distributed across multiple servers. For example, a well-resourced central server can handle the interactive demands of LAN, intranet or extranet (or all together) while a replicated server takes care of long-running jobs that need to isolate a snapshot of data for lengthy periods.

Availability

Maximum availability is sometimes measured by the "five nines" criterium: can the server be relied on to keep databases available for 99.999 per cent of the required operating hours? Among its users, Firebird has a reputation for being bomb-proof. Some high-profile data management systems require costly expertise on site for all of the hours when the databases have to be working. The typical Firebird deployment is to companies that do not have any database staff on the payroll.

As with any complex system, a Firebird deployment needs to be well planned and well designed, but an appropriately configured Firebird deployment on a reliable network infrastructure "just works". It uses optimistic locking at record level, drastically reducing the wait-time overheads in comparison to others where read-write transactions lock entire sets, even tables, pre-emptively. No tuning is ever required to facilitate handling varying workloads through the day or week. A database does not have to be shut down for backups. It can be replicated or shadowed for almost instantaneous cutover in the event of disk failure. It is robust and recovers immediately from power failure, without loss of database integrity.

Firebird is a popular choice for enterprises needing continuity of service around the clock. Command-line tools are distributed with the software for all administrative activities, allowing regular housekeeping to be automated as scheduled or on-demand jobs. A Services API is also available to wrap admin tasks into a program or service application.

On-line backup (gbak) creates not a database but a platform-neutral file containing metadata and data saved separately in a compressed text format. Firebird 2 also has optional on-line incremental backup, which can be scheduled throughout the day to suit the loads.

Capacity

The largest Firebird database we have heard of is about 11 Terabytes and growing. Tables are limited to about 2,000,000,000 rows and, up to version 1.5.x, a maximum of about 30 Gigabytes per table. The maximum table byte-size limit disappears in v.2.0, due in mid-2006.

Interoperability

Standards Compliance

Firebird is the most compliant of all the RDBMS offerings in its implementation of the international (ISO) and U.S. (ANSI) standards for the the SQL query language, exceeding even its InterBase cousin. Standards evolve continually and, as they do, the Firebird developers keep the language implementations from release to release aligned to the latest published versions. Existing implementations of language features that become subject to new standards are retained as “deprecated options” to ensure backward compatibility.

Independence

Firebird is designed for interoperability as a “back-end” in the absolute sense. It is not bound to any “integrated solution” that ties databases to a specific environment. A decision to move Firebird databases from a Windows to a Linux or Unix host, or vice versa, can literally happen overnight. All it takes is a transportable backup at the old host and a restore at the new and you are back in business. Application vendors who are mindful of the ease of such transitions are careful to write software that is “aware” of the ways Firebird servers can be configured so that software running on any client operating system can access database servers running on any other supported operating system.

Firebird is capable of running in heterogeneous networks. Application software written for one operating system happily connects to databases running on another, without modification. An application can connect to multiple database servers on mixed hosts simultaneously and can run multi-database transactions across mixed hosts. It is common, for example, for demanding sites to run several Firebird servers, replicating between a high-cost main server and cheaper boxes running a sparsely-configured Linux with fast disk systems for load-balancing and failover.

Firebird servers can also participate in heterogeneous transaction server environments like MTS and Citrix, in pass-through environments and in secure virtual networks.

Connectivity

Firebird has no requirement for suites of special modules to bind it to external applications. All data access layers converse with the server through its two published application programming interfaces (APIs), one for database-level operations and the other for server-level services such as backup and user authentication.

Driver support is available for a large number of programming and standard interface environments, including Java/JDBC, ODBC, .NET, Delphi, Python, PHP and Perl.

Autonomy

Autonomy refers to the degree to which data stored and managed in a database has "an internal life of its own", i.e., its capability to exist and to remain consistent, independently of hardware and operating system environment, external application programs, specific programming environment.

"Autonomic Features"

"Autonomic features" include such things as

- file-system independence
- declarative referential integrity
- the ability to implement validation and/or business rules via triggers and CHECK constraints
- stored procedures, to encapsulate business procedures and processes internally, independently of any application language or user interface
- database-resident user access control
- the ability to query external data as "virtual" structures without needing to store that data
- the ability to output programmatically constructed temporary data sets without the need to materialise temporary tables
- retention of deprecated features to ensure backward compatibility

Firebird supports all of these features.

Complexity of Migration and Upgrade

A further aspect of database autonomy that has become an issue for users of many high-profile DBMS products in recent times is the degree of transparency with which legacy databases can be managed and migrated when the server software is upgraded or databases are ported to a different host hardware or operating system platform. With these issues, the common experience is that they are high-cost in terms of time and logistics.

Migration

In contrast to any other significant RDBMS except Firebird's cousin, InterBase, migration of databases is seamless. Unlike others, which have architectually different on-disk structures for multiple platforms, or which bind database structures to a single platform, Firebird database structures are stable across platforms. This means, for example, that if choosing one operating system as a host turns out to be a sub-optimal decision, you may switch to another host in the time it takes to perform a backup and restore.

Upgrade Compatibility

A new release of a Firebird server connects to any database that ran under a previous version and a transportable database backup made on one hardware/OS platform can be restored, ready-to-go, on any other supported host platform.

When upgrading to a new major release, it is highly recommended, although not essential, to put databases through the backup/restore cycle, in order to upgrade the on-disk structure of the database and make new features available. The question of whether to do this really depends on whether the application vendor has already updated applications to take advantage of new and enhanced capabilities.

Sub-releases

Sub-releases do not normally change the on-disk structure, although it may be worth performing the backup/restore cycle if a sub-release enhances an existing feature.

ACID Compliance and Firebird

The four ACID principles are atomicity, consistency, isolation and durability.

Atomicity

Atomicity guarantees that there are only two possible outcomes from a task (known as a transaction) that involves changing multiple sets of interdependent data: either every step in the task completes successfully and can be committed to the database as a single unit or, if one step should fail, all other steps can be undone ("rolled back"), leaving the global state of the database unchanged.

Atomicity is obviously of extreme importance in financial systems, where imbalances caused by partial failures could be catastrophic. Spreadsheets, and database systems that do not support transactions, cannot be atomic.

Firebird is completely "transaction-driven": nothing occurs outside a transaction context.

Consistency

Consistency is the capability of the database engine to protect a database from any change of state that could leave any set of data in out-of-synch with any other set. For example, the system has to be capable of recognising that an invoice depends on a customer and is itself depended on by invoice line items. It must be able to prevent, for example, deletion of a customer record if there are invoices stored for that customer, and deletion of an invoice that has line items associated with it.

The practical implementation of such dependencies is by means of declarative referential integrity constraints ("foreign keys") enforced by automatically-generated triggers. (Triggers are automatically-generated or user-defined blocks of executable code that run whenever a record is inserted, modified or deleted.) Database systems that rely on application code to enforce consistency rules do not comply with the consistency principle. Firebird supports the full range of referential integrity rules defined by the standard.

Firebird also guarantees consistency when a single transaction is performing changes across multiple databases, by means of what is termed "two-phase commit". Systems that can access mutliple databases concurrently without the ability synchronise changes in all of them will fail on the consistency principle.

Note: Firebird does not support declarative referential integrity across database boundaries. Each database involved in multi-database transactions is required to be responsible for its own RI.

Isolation

Isolation refers to the capability of the database to allow each user (or transaction) to operate as though it were the only user (or transaction). The isolation mechanism must be capable of keeping each transaction's view of database state consistent as long as that transaction is running, regardless of any changes that are performed by other transactions. Database management systems that comply with this principle usually offer a range of isolation levels, the rules for which are defined in the ISO/ANSI SQL standards.

In addition to the level described above (*Concurrency* or *Repeatable Read* isolation), which must be supported, Firebird supports *Read Committed* (where a transaction's view of database state is kept up-to-date with work committed by other transactions) and, on the other side, *Consistency* or *Table Stability* (where a transaction that is capable of writing to the database blocks other write-capable transactions from accessing the tables that it is reading).

Durability

Durability guarantees that the database will keep track of pending changes in such a way that the state of the database is not affected if a transaction is interrupted. Hence, even if the database server is unplugged in the middle of a transaction, ACID-compliant database servers must return databases to a consistent state when restarted.

Transaction Logging

Durability is one of the hardest principles to comply with. Other database management systems that claim ACID support have traditionally dealt with it by storing uncommitted transactions in a transaction log. However, logging never totally guarantees durability, since the log file itself may be logically or physically corrupted by the event that interrupts the transaction.

Some DBMSs that rely on logging to achieve durability try to reduce that risk by using a "write-ahead log" to log requests to disk before attempting to post changes. If the write-ahead log survives undamaged, it may be possible to retrieve uncommitted work when the system recovers and use it to reconcile database state and restore it as it was before the event. Such systems are characterised by the need for a lengthy "recovery procedure" after network or power failures.

Certain high-profile DBMS products are notorious for log-related breakages resulting from interrupted transactions. The instability of these database engines is such that, even on sites with moderate requirements, it becomes a necessity to employ staff to babysit the server around the clock to keep it out of trouble and fix breakages before problems propagate too deeply to save data integrity.

Firebird's Multi-generational Architecture

Firebird's architecture avoids the need for recovery logging by literally retaining the preceding version of every deleted or changed record, not just for the duration of the transaction but until all transactions that were "interested" in that record, for any reason at all, have ended. The term for this is "multi-generational architecture", or MGA. MGA was unique to InterBase for about 10 years until it was imitated by Oracle. Once the Firebird sourcecode was available, PostGreSQL copied it. More recently, Microsoft has introduced MGA in the latest evolution of SQL Server.

Transaction Logging and Firebird

Firebird does not need transaction logging for recovery purposes and it does not include any transaction logging facilities in the engine. However, where enterprises require logging for auditing purposes, some excellent logging service software is available from third party vendors.

Who Uses Firebird?

Because Firebird is free, there are no licences to count, no beans to count. It is known, from reputable enterprise surveys, that Firebird is chugging away on hundreds of thousands of production sites around the world. The following is a selection of companies and organisations that are publicly known to be using Firebird:

Broadview Software Ltd, Toronto, Canada, vendor of information and control systems and online services for broadcasters worldwide

Morfik P/L, Hobart, Tas., developers and vendors of WebOS development suite for construction and maintenance of interactive websites, stores web objects in a Firebird meta-layer (system database) as well as Firebird user data.

Communicare Systems Pty Ltd, Perth, WA, vendor of patient management and medical records software for hospitals, clinics, medical practices and mobile health units across Australia.

"The Examiner" newspaper, Launceston, Tas., high availability(24/7) business, information, production and news systems.

U.S. Navy, broad range of management and logistical systems

Frontrange Solutions USA Inc., Colorado Springs, U.S.A, as the back-end of the highly scalable, award-winning integrated CRM, service management and business systems "Goldmine" software suite.

British Rail, U.K., timetabling, bookings, accounting and information systems for national railway passenger network.

Deutsche Presse-Agentur GmbH, HQ in Hamburg, Germany, largest press agency in Germany, provides a worldwide service to newspapers, magazines, TV and radio news networks.

KIMData, Munich, Germany, business intelligence systems and data warehousing for German hospitals.

Deployment Samplings

Distributel, Telco Services Provider

Location: Canada

Contact: Dalton Calford (CTO)

Distributel is a long-distance telephone services provider with three corporate offices spanning three cities and two provinces. It uses Firebird as the back-end to corporate information systems, servicing an average staffing level of around 500 users. However, in-house information systems is not the area where we use Firebird most heavily.

The real stress test comes from our customer load. We provide a wide variety of services, servicing hundreds of thousands of customers, who process 2 million transactions a day. The whole shebang is handled by one database, effectively.

We have three unmanned network Points of Presence ("PoPs"), each in a different city, plus a development PoP in our testing lab. Each PoP has a dual telco switch system that performs load balancing and acts as failover support in case one fails.

That equipment is very specific to our needs, but we control it all using Firebird. Each switch is connected to two signal control processors (SCPs), which are slimline computers running Firebird. That means that each PoP has four SCPs. Each SCP hosts two independent databases, meaning we have 32 separate databases, all containing identical metadata and data.

If we lose a PoP, the other PoPs can take over the call, according to the "call state". Of a possible 96 call states, only four are not recoverable. Each city keeps redundant dialogs with the SCPs from other cities and the response dialogs are live-audited in real time.

For an indication of how fast the response has to be, when you pick up the phone, a signal goes back to a telco switch, which then asks a SCP what to do. The general response is 'play dial tone and wait for digits'. Dial tone is not automatic—it is the response of a query into the Firebird database that checks line IDs, customer status, service notices (such as call answer) as well as governmental warrant and privacy needs.

By law, we are only allowed one failure in 99,999 phone calls, the so-called "five nines" that is said to define "enterprise-capable availability".

Our customer service reps are also involved in updating the databases. Those databases report real-time call information to the reps, as well as to the billing system, a Firebird database. Besides storing and generating our Accounts Receivable, as one would expect, the database is also concurrently queried by our interactive voice response units, the service personnel whom customers call to inquire about their account status and their call history.

Prague Municipal Library

Location: Czech Republic

Contact: Ondrej Cerny, IT department manager

Prague Municipal Library oversees about 3,000,000 publications and has about 120,000 regular (registered) users. The central library has many branches all over the city, of which about 20 are currently connected to the IT center at the central library. Deployment is ongoing, with two or three more branches going on-line each month, each adding about 5-15 new users using 15-50 new attachments.

The library runs about 20 applications over a single Firebird database. Five are considered as core applications used by most users, handling normal library operations, public-access terminals (in libraries) and public Internet access to book collections. These applications are used by 300-350 concurrent users during working hours, using between 400 and 600 connections to the single database, which is currently about 30GB in size. Firebird handles 3-5 million transactions each day.

Other applications are specialised or daemon-like watchdog applications (sending out e-mails about requested books etc.).

They run Linux Classic Firebird 1.5.2 on Red Hat 9 with a kernel tailored to handle 400-600 classic instances. Hardware is a 4-CPU Xeon machine with 16Gb RAM and a 120Gb RAID 10 storage array. They also have an 8-CPU Xeon with 20Gb RAM and a 500Gb RAID 10 to handle future needs—recall that this is an ongoing deployment, so scale is rising steadily all the time.

The library does not operate 24/7. They have a maintenance window from midnight to 5 a.m. each day, but the system must run without outages the rest of the time. Outages are not life-threatening so they have a failover plan in place to restore operations in less than two hours in case the primary system suffers a fatal failure.

Although they are very satisfied with Firebird, they do have problems with old applications still in use that use the Borland Database Engine (BDE), an obsolete data access layer designed for use with desktop databases such as Access, that never scaled well on networks and is particularly dumb about transaction-driven data management. The biggest problem is blocked garbage collection that forces them to do a backup/restore every night and to throw a lot of hardware into it to compensate for the performance degradation as garbage accumulates throughout the day.

Despite the current not-so-healthy state of the application architecture, the 4x Xeon machine is about 50 per cent utilised. Those old BDE apps are due for replacement this year with a new application set that uses three-tier architecture and connection pooling, using a direct data access interface in the Delphi middleware. They expect that when this is sorted out, they will get a substantial amount of hardware reserve with current equipment, enough to handle all their growing needs for the next five years.

OneDomain, niche business intelligence systems

Location: Birmingham, Al, U.S.A.

Contact: Ed Salgardo Snr

OneDomain is a rapidly growing Birmingham, Alabama-based software company that develops and markets media planning, research, and business intelligence software to television stations across the United States. The primary product, called ClearView, allows broadcast salespeople to analyze TV ratings and target their approaches to selling time slots to advertisers. Following incorporation in October 2001, our first two years were spent in product development. Only 24 months after our first major software release in November 2003, OneDomain earned a 20% market share and it is more like 30% now.

Conceptually, our application architecture is really client/server but we use Citrix [Metaframe Terminal Server] to make it kind of "thin" for the clients by running the application in a chunk of memory at the server and not at the client.

We have a server with the Firebird 1.5.2 database and several (upwards of six) Citrix servers hitting it. We use a thick Win32 client, written in Delphi, that runs on the regular Citrix servers and provides the users with an application that acesses the database hosted in the Citrix server machine, a four-processor machine, using hyperthreading to look like eight processors, and with some 3.5Gb of RAM available.

We started with Firebird running as SuperServer but quickly changed to Classic for several reasons, including the memory usage limitations that seriously downgraded performance when things started getting hot.

With some 800-900 theoretical users, of whom fewer than 300 are likely to be simultaneously logged in, we have been able to handle the load so far. The good thing about this set up is you can scale easily if you need to, just by adding more servers.

Moscow Interbank Currency Exchange Bank

Location: Russia

Contact: Sergey Korotkikh

Despite its name, MICEX is the largest stock, foreign currency and derivatives exchange in Russia. Average daily turnover exceeds $6 billion. As a fully electronic exchange, MICEX has been using InterBase and, latterly, Firebird, databases since 1994 as the main storage for market data, orders and trades.

The MICEX Trading System serves more than 2000 users located throughout the whole territory of Russia in eight time zones. It trades in real time with an average payload of more than a quarter of a million orders per day, with more than 180,000 concluded trades daily.

In addition, more than 300 electronic broker systems connect to the Trading System via a special bridge providing a connectivity API. The Trading System itself is “semi-detached” from the database backend, effectively providing a very fat middleware layer to the database, from and to which it retrieves and maintains all necessary market and trading information

Apart from its trading functions, the Firebird database is heavily used by our clearing activities and reporting. We generate daily trading and clearing reports for all of our almost 1000 members and send them via e-mail.

Being a key Russian financial exchange, MICEX is obliged to maintain a high level of reliability. According to the results of an audit done by the Gartner Group, we maintain an availability level of 99.999 per cent.

Factors Inhibiting Scalability

Several factors must be considered when plans to upscale involve increasing the number of concurrent connections to the server.

User base

First one needs to identify the actual concurrency requirement: are we talking about actual connections (the number of users who are expected to make a connection to a database at peak load times) or about defining the limits for the size of the user base?

Hardware, software and network limits

Firebird can be deployed in a range of "models". Of interest here are the two "full server" models, Superserver and Classic. Both models are available on Windows, Linux and several other platforms. Although the server engine is identical, regardless of model, the choice of model ascends in importance as the potential number of logged-in users increases. Hardware and OS platform limitations also come into play here.

Superserver

Each Firebird Superserver is limited to a practical maximum of, at most, around 250 to 400 concurrent connections. The reason for this is that a Superserver and all of its connections to all databases are wrapped in a single, 32-bit process. A 32-bit process cannot address more than 2 Gb of memory. Each connection to a database causes the database engine to start one or more threads in which to process requests from that connection; the Superserver also runs process threads of its own for background garbage collection and other on-line tasks.

The Superserver also needs RAM for other purposes, of course.

A configurable page cache—one for each database that has active connections—is maintained to keep frequently-used database and index pages in memory. On Superserver, this cache is shared by all connections to that database. The size of the page cache defaults to 2048 pages so, for a database having the default page size of 4Kb, 8 Mb of RAM needs to be available for efficient use of the cache. (If the cache needed to be paged out to virtual memory, its purpose would be defeated!).

Typically, databases are created with an 8Kb page size and it is a common source of resource starvation when developers and DBAs over-configure the cache drastically, in the belief that "more is better" (what the author refers to as "the Lambourghini syndrome": if you drive to work in the CBD at peak-hour in a Lambourghini, you're going to get there faster than I do in my Mazda 121!).

Sometimes this measure is taken to try to speed up response time that becomes degraded by poor transaction management in client code and by careless connection management. It is pointless. Such problems have nothing to do with the size of the cache and everything to do with inattention to resource cleanup. Ramping up the cache simply exacerbates the problem.

Consider the effect on resources when, for example, the cache for a typical 8Kb page size is ramped up to 20,000 pages. That represents 160 Mb that must be preserved in memory to make caching useful. The server maintains its lockfiles in RAM and these will grow dynamically (up to a configurable maximum) as more connections come online. Firebird 1.5 will use RAM for sort operations if it is available. This is great for response time, but many poorly-written applications hold open ordered output sets comprising thousands of rows for long periods.

Database servers love RAM. The more that is available to the server, the faster it goes. However, that 2Gb barrier kills Superserver in an overpopulated environment because each Superserver connection uses about 2Mb of RAM to instantiate its process thread and maintain its resources. So 500 Superserver connections will use a gigabyte, leaving somewhat less than a gigabyte of total addressable RAM available to the database server for processing, sorting, transaction accounting, lock tables, in-memory storage and new connections.

Classic

The Classic "server" is in fact not a database server at all, but a background service (or, on Linux, an xinetd daemon) that listens for connection requests. For each successful connection the service instantiates a single Firebird server process that is owned exclusively by that connection. The result is that each connection uses more resources, but the number of connections becomes a question of the amount of RAM available on the host machine. In addition to the ~2Mb to instantiate its connection, each also maintains its own, non-shared page cache. All connections get the same starting cache and, since the cache does not need to cater for shared use, it can (and should) be smaller. The recommended size is around 2Mb (512 pages for a 4Kb page size) on a 4Gb dedicated machine, but it can be much less. It is important to keep enough RAM available for the lock manager's table to "grow into" for peak connection loads.

64-bit Addressing

A way out of the 32-bit RAM limitations is to build a 64-bit Superserver. A 64-bit Superserver is possible on POSIX platforms where the OS has stable support for 64-bit hardware. An experimental 64-bit Superserver for Firebird 1.5.2 on AMD64/Linux i64 was released more than a year ago but it proved unstable. A 64-bit AMD64/Linux i64 build is included in the Firebird 2.0 beta 2 field test armoury. On Windows, it is not yet possible to build 64-bit capable Superserver because of issues with the current releases of the Microsoft Visual Studio 7 C++ compiler, the release-standard build environment for Firebird 2 on Windows. Private builds of both v.1.5.x and v.2.0 on non-Microsoft compilers are known to be working stably in production environments and the core development team has signalled its intention to keep a 64-bit Windows release in sight for Firebird 2.0.

Connection Pooling

Using middleware to hold and manage pre-allocated resources to keep a finite number of connections ready for incoming connection requests is known as connection pooling. There are several ways to implement connection pooling and compelling reasons to do so when a growing user base must be catered for. It would be unrealistic to set out to build an extensible multi-tier system without it. Connection pooling may be combined with a "queuing" mechanism if resources are inadequate to cope with peak loads.

The middleware developer needs to take special care to monitor attachments and detachments, especially from "stateless" clients like web browsers, to avoid resource leakages. Middleware should guard against recursive threaded workflows that could monopolise the pool and, for architectural reasons, should prevent threading across connections totally.

Competition for RAM

Trying to run a database server on a system that is running competing services, such as a web server, Exchange server, domain server, etc., will risk having those other services steal resources (RAM and CPU time) from the database server, denying Superserver even the 2 Gb that it is capable of using or limiting the number of connections to Classic, and slowing down database response time.

SMP Support

The incompatibility of the Superserver's threading implementation with the Windows implementation of symmetric multiprocessing (SMP) could be regarded as a “scaling limitation”, because of the arbitrary way Windows switches the entire affinity of a process between CPUs, resulting in a “see-saw effect”, whereby performance will continually come to a standstill at busy times, while the system waits for the OS to shift all of the Superserver's active memory resources from one CPU to another when it detects unevenness in CPU utilisation. Superserver is configured by default to be pinned to a single CPU to avoid this.

On Linux, multiprocessor handling does not cause this see-saw effect on 2.6 kernels and higher, although it has been reported on some 2.4 kernels on older SMP hardware. However, SMP does not gain any significant performance benefit for Superserver on Linux, either.

Support for configurable levels of SMP-aware, fine-grained multi-threading has been architected into the Vulcan engine and will become a feature of Firebird 3. It has already demonstrated sufficiently useful effects on memory-bound operations to make it something to look forward to.

At the same time, given the inherent importance of disk I/O to responsive online transaction processing, much of the “noise” about SMP capability is attributable to the Lambourghini syndrome. In the appropriate environment, Superserver is an efficient uniprocessor system!

Classic—being a single process per connection—does not suffer from SMP inhibition, even on Windows. Until Firebird 3, when the distinctions between the two resource models will disappear, to be replaced by configurable resource management, Classic is the better choice for any upscaling plan that is able to accommodate increasing demand by augmenting system resources.

Note: there have been reports of kernel-level problems with SMP on some Linux versions with 8 CPUs and hyperthreading enabled, that do not appear on a 4X system. Since full support for setting CPU affinity on Linux did not arrive until kernel v.2.6, setting CPU affinity via firebird.conf is not supported for Linux in the current Superserver release.

“Windows Networking”

An adequately resourced TCP/IP network using the appropriate Firebird server model is amenable to upscaling. A Linux host with no competing applications (including Xserver!) running on it will deliver the optimum performance at the enterprise end of the scale. As a database server, a Windows host suffers from some inherent overheads that will have a noticeable impact on performance under conditions of high demand.

However, there is another “gotcha” attached to Windows networking. The native Windows Named Pipes transport (“NetBEUI”), which Firebird's network layer supports for largely historical reasons, has an absolute limit that cannot exceed 930 connections to any server. Named Pipes should be avoided for environments other than small workgroup LANs anyway, since it is a “noisy” transport that gets noisier as contention rises.

Incidentally, databases on Windows should always be stored on NTFS partitions which, if properly protected, offer better performance and less risk than FAT32. Older hard disks that have been retained on systems that previously ran NT 4.0 should be also be regarded as risky, since the older NTFS does not reliably support filesystem protection measures added in later implementations.

Database and Software Design

The availability and scalability of any software system can be negatively affected by poor database design, careless query specification, inappropriate workflows and, especially, poor transaction management.

The multi-generational architecture ensures robustness, excellent, optimistic task isolation and highly efficient throughput. Well-performing databases result from careful design, clean normalization, good indexing and, on the client side, well-planned queries and careful attention to timely completion of transactions. On the contrary, poor database design results in complicated queries and careless indexing that can cripple the capability of the optimizer to make effective query plans. Slow queries are generally the result of a combination of these flaws.

Garbage Collection

MGA accumulates "garbage", in the form of old record versions. The engine performs in-line garbage collection. However, it will not permit garbage collection of old record versions that are still "interesting" to some transaction. Transactions that remain interesting for long periods trap record versions pertinent to any transactions that started later, causing garbage to build up to an unmanageable level. It is essential that Firebird developers understand how this could happen and write software that avoids it, keeping the level of garbage low enough for the garbage collection subsystem to cope with.

Commit Retaining and "Autocommit"

Commit Retaining is a "feature" of Firebird that was inherited from the ancestor, InterBase. It was implemented as a means to retain server-side resources and keep them available for developers using Borland's rapid application development tools and the BDE, the generic data access layer that was used to connect Windows graphical applications to databases. Its purpose was to present a level playing field for RAD development, regardless of the database running at the back-end.

Autocommit—a client-side mechanism that rolls the statement-level Post and the transaction-level Commit phases into a single step—was provided to enable developers to avoid transactions altogether. In the RAD components, Commit Retaining and Autocommit were bound together. This fitted well with desktop databases like dBase and Paradox (whose engine is, in fact, the BDE!), which do not have transactions.

With Firebird (and InterBase), Commit Retaining causes transactions to remain interesting indefinitely. Garbage collection effectively ceases on the "standard" Borland RAD tools database application and any other applications that make use of Commit Retaining. Such systems are fraught with problems of progressively degrading performance that cannot be resolved except by shutting down the database and allowing these old transactions to die.

Autocommit and Commit Retaining are not restricted to the Borland tools, of course. They are supported by most data access interfaces and Commit Retaining is available in SQL, so it behoves the application developer to understand the effects and to use these features with extreme care and control.

Who Owns and Manages Firebird?

No commercial organisation “owns” Firebird. It is the common property of the project members. Its legal and financial interests are handled by the Firebird Foundation, a non-profit organisation incorporated and administered in New South Wales, Australia.

Management

The development team is self-managing. Decisions about what goes into a release are made by consensus in a closed forum made up the active developers, the build managers (these are the people who compile the release kits and installers), the documentation coordinator and a release manager.

The Firebird Foundation, per se, is not involved in directing or managing the development effort at all, although there is a degree of overlap (project members who are also Foundation members).

Code Maintenance

Firebird has behind it a solid, well-maintained codebase with many eyes constantly upon it at all levels, from architecture to design to implementation and testing.

Release Policy

The development team holds fast to the principle of not making a production release until and unless all outstanding issues in QA testing are resolved to the satisfaction of the project's administrators. Releases and sub-releases go through multiple beta and pre-release testing cycles. The downside of such tight discipline is that production release dates are not firm.

Releases represent major enhancements and new features. Sub-releases occur periodically between major releases to implement bug fixes and minor enhancements to existing features and tools, or to complete features that were partly implemented in a previous release or sub-release.

For client applications, the only essential requirement is to upgrade the run-time client library to the version matching the major release. Application vendors may also wish to incorporate new features into their software and synchronise the installation of a new Firebird release with installing new modules or versions of client and application server software.

Open Source Licensing

The Firebird source code is licensed under two derivatives of the Mozilla Public License v.1.1. (MPL). Modules that originated from the InterBase codebase are maintained under the InterBase Public License v.1.0, while the Initial Developer's Public License (IDPL) applies to new modules.

The IPL and the IDPL differ from MPL 1.1 only insofar as they remove some implied proprietary rights that the Netscape Corporation built into the Mozilla licences. The IDPL differs from the IPL in that it excludes “Inprise Corporation” from the severality of the source code copyright.

The MPL-style licences differ from the well-known Gnu Public Licenses (GPL) in that they are not “viral”. Viral is a term that describes the GPL restriction of forbidding code under that license from being compiled with other source code modules unless those modules are themselves made open source under the GPL. A non-viral licence permits modules to be compiled with closed, proprietary code, making it more useable for commercial software development.

However, like the GPL and most open source licences, the MPL-style licenses require that all changes to the licensed code itself be made freely available to the public and that the licensed code not be distributed under any other licence.

Funding

The designers and programmers who develop Firebird are volunteers, mainly self-employed people or people with other jobs who work part-time on the Firebird code. Some are funded directly or indirectly by their employers, by being allocated company time to work on Firebird.

A number of the most active "core" developers (those working on the Firebird engine itself) are assisted by regular grants from funds raised by the Firebird Foundation, thus enabling them to commit to a minimum number of hours per month. The project has one designer/coder whose grant is sufficient to enable him to work full-time on the project. Grant assistance is also available to project members developing and supporting interface drivers.

Funding sources are Foundation membership subscriptions, company sponsorships and company and private donations. The Foundation draws its membership, sponsorships and donations from the end-user community and vendor companies that incorporate Firebird in their commercial products, as a way to put something back into the development.