

[Firebird Documentation Index](#) → [Firebird Enterprise Whitepaper](#) → Factors Impacting Scalability



Factors Impacting Scalability

[User base](#)

[Hardware, software and network limits](#)

[Database and Software Design](#)

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 150 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 Lamborghini syndrome": if you drive to work in the CBD at peak-hour in a Lamborghini, 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.



[Firebird Documentation Index](#) → [Firebird Enterprise Whitepaper](#) → Factors Impacting Scalability