Firebird Classic server versus Superserver

Many thanks to Paul Beach of https://www.IBPhoenix.com for this article.

InterBase® Superserver architecture

Superserver is a multi-client, multi-threaded implementation of the InterBase® server process. This implementation replaces the “Classic” implementation used for previous versions of InterBase®.

Superserver serves many clients at the same time using threads instead of separate server processes for each client. Multiple threads share access to a single server process. The benefits of Superserver architecture include:

Having a single server process eliminates bottlenecks resulting from arbitration for shared database pages and reduces the overhead required for multiple process startups and database queries. Superserver improves message interaction performance because a shared library call is always faster than an interprocess communication request to a server process.

Superserver improves database integrity because only one server process has write access to the database, rather than one process for each client. All database engine functionality is encapsulated into a unified, protected subsystem that is isolated from user application error.

SuperServer allows for the collection of database statistics and user information that InterBase's tools can use for performance monitoring and administrative tasks.

Superserver is more cost-effective than the Classic architecture. All operating systems have limits on the number of OS processes that can run concurrently. Superserver allows for a fixed number of database threads to be multiplexed over a potentially large number of concurrent database connections. Since these threads are not hard-wired to any specific database connection, Superserver can support a larger number of users with minimum resources use.

InterBase® Classic architecture

Classic architecture, the design in InterBase® 4.0 and earlier, was process-based. For every client connection, a separate server process was started to execute the database engine, and each server process had a dedicated database cache. The server processes contended for access to the database, so a Lock Manager subsystem was required to arbitrate and synchronize concurrent page access among the processes.

Invoking the Classic Server

The InterBase® Classic server runs on demand as multiple processes. When a client attempts to connect to an InterBase® database, one instance of the gds_inet_server executable runs and remains dedicated to that client connection for the duration of the connection.

The initiator of gds_inet_server is inetd, the UNIX service turnkey process. It has a configuration file,

/etc/inetd.conf, which associates services with the executable that is to receive the connection. When inetd receives a connection request for a given service, it looks up the appropriate program in /etc/inetd.conf, executes it, and transfers the network connection to the service program.

When the client chooses to disconnect, gds_inet_server closes its connection to the database and any other files, and then exits. When there are no clients connected to any database, there should be no invocations of gds_inet_server running.

Lock management

Lock management is taken care of by another process, gds_lock_mgr. This program is started when the second client attaches to a given database. The job of the lock manager is to serve (metaphorically) as a traffic cop. It grants locks on database resources to clients. It also requests that clients relinquish locks on a resource when that resource is in demand by other clients. The gds_lock_mgr remains running even after the last client disconnects. The next time a client connects, it can avoid the slight overhead of starting the lock manager process. For further information regarding locking, refer to Firebird for the database expert: Episode 5 - Locking and Record Versions.

Use of Posix signals

The gds_lock_mgr process communicates with each client process by using a shared memory area, and a signaling mechanism using the POSIX signals SIGUSR1 and SIGUSR2. Signals are caught in signal handling routines in libgdslib.a, and for this reason user applications should not perform signal handling or any modification to the signal mask. Applications which need to use POSIX signals must compile with an alternate InterBase® library, libgds.a. This library functions identically to libgdslib.a, but it handles signals sent by the lock manager in a child process called gds_pipe. All client applications are compiled with libgds.a automatically run with this child process.

No changes to application code are needed, only a different linking option.

Resource use

Each instance of gds_inet_server keeps a cache of database pages in its memory space, which is likely to result in some duplication of cached data across the system. While the resource use per client is greater than in Superserver, classic uses less overall resources when the number of concurrent connections is low.

Local access method

The Classic architecture permits application processes to perform I/O on database files directly, whereas the Superserver architecture requires applications to request the IBServer I/O operations by proxy, using a network method. The local access method is faster than the network access method, but is only usable by applications that run on the same host as the database.

Monitoring

The database information call for active connections always reports exactly one connection on a Classic server, no matter how many clients are connected to databases on that server. The reason for this is that every client connection has its own gds_inet_server process on the server, and each instance of that program knows only about its own connection. Only in Superserver does the server process have the ability to report all client connections on the server.

Security

In order for InterBase® Classic to work with a mixture of local and remote clients running as different user ID's, the server executables gds_inet_server and gds_lock_mgr must run as root.

The processes must run with a real uid of root to set their effective uid to that of the client uid. The lock manager must have the superuser privilege to send signals to the processes. In some IT environments, the presence of executables with setuid bits turned on raises concerns about security. Nevertheless, do not change the runtime configuration of the InterBase® server. The setuid root configuration of the Classic software is important to its function.

Because applications can run as any uid, database files must be writable by all uids that access the databases. To simplify maintenance, database files are created writable by the whole world.

With care, you can restrict these file permissions, so that the database files are safe from accidental or deliberate damage. Make sure you understand file permissions completely before attempting this, because all local and remote clients need write access to the database, even if they intend only to read data.

Classic versus Superserver

Invoking Superserver

Superserver runs as a single process, an invocation of the ibserver executable. ibserver is started once by the system administrator or by a system boot script. This process runs always, waiting for connection requests. Even when no client is connected to a database on the server, ibserver continues to run quietly.

The Superserver process is not dependant on inetd; it waits for connection requests to the gds_db service itself.

The Superserver process is a multi-threaded application. Different threads within the process are dedicated to different tasks. For instance, one thread waits on the gds_db service port for incoming connection requests. Other threads are analogous to individual gds_inet_server processes in the Classic model, serving client queries. Another thread serves as the lock manager, replacing the gds_lock_mgr process from the Classic model.

Lock management

The lock manager in Superserver is implemented as a thread in the ibserver executable. Therefore InterBase® does not use the gds_lock_mgr process. Likewise, POSIX signals are not used by the lock manager thread in Superserver; interthread communication mechanisms are used.

Resource use

The Superserver implementation has less overhead and uses fewer system resources per client connection than the Classic model. Superserver has one cache space for all client attachments, allowing more efficient use of cache memory. For these and other reasons, Superserver has demonstrated an ability to efficiently serve a higher number of concurrent clients.

Threaded server & UDFs

User-Defined Functions (UDFs) are libraries of functions that you can add to extend the set of functions that the InterBase® server supports. The functions in your UDF library execute within the process context of the InterBase® server. Due to the threaded implementation of Superserver, there are issues with UDFs that require that you write UDF functions more carefully than when writing UDFs for a Classic server.

You must design UDFs for Superserver as thread-safe functions. You cannot use global [02-ibexpert:02-03-database-objects:stored-procedure:#Local variables / DECLARE VARIABLE statement|variables]] in your UDF library, because if two clients run the UDF simultaneously, they conflict in their use of the global variables.

Do not use thread-local global variables to simulate global variables. Superserver implements a sort of thread pooling mechanism, to share threads among all the client connections. It is likely that if a given client executes a UDF twice, that each execution is not executed in the context of the same thread. Therefore, you cannot depend on thread-local variables keeping values from one execution of the UDF to the next for a given client.

UDFs that allocate memory dynamically run the risk of creating a memory leak. Because Superserver is supposed to stay up and running indefinitely, not just for the duration of the client connection, memory leaks can be more damaging in Superserver than in Classic. If your UDFs return dynamically allocated objects, then you must use malloc() to allocate the memory for these objects (on Win32, you must use ib_util_malloc() or the malloc() that is part of the Microsoft Visual C++ runtime library). Do not use new or globalalloc() or the Borland malloc().

Finally, such functions must be declared in databases with the FREE_IT option of the DECLARE EXTERNAL FUNCTION statement.

By contrast, in Classic, there is a separate process for each client connection, so the UDFs are guaranteed not to conflict. Global variables are safe to use. Also, memory leaks are not as dangerous, because any leaked memory is released when the client disconnects. InterBase® recommends that you design UDFs for Superserver, the more restrictive model, even if you use a version of InterBase® implemented with the Classic model. Eventually InterBase® will be implemented with Superserver on the platform you use. If you design UDFs with this assumption, you can upgrade to a later version of

InterBase® without the risk that your UDFs must be redesigned to work with Superserver.

Security

Superserver can be configured to run as a non-root uid, for enhanced security. In Superserver, you can restrict the permissions on database files to allow only the InterBase® server uid to access the database.

Why two implementations?

The Classic implementation predates the Superserver implementation, and the Superserver implementation is the future of InterBase®. Classic configuration is used on operating systems that currently don't have the technology for threaded applications, which is required for Superserver. InterBase® also distributes the Classic version on platforms that have threading technology, but which benefit from the low-profile implementation.

Superserver has a greater ability to meet the demands of a growing multi-user system, while retaining good performance and efficiency. Superserver is implemented in InterBase® product on all platforms where it is technically practical. It is the intention that Superserver is the future direction of InterBase® on all platforms.

Changing server to solve undefined crashes

September 2004. Many thanks to Gerhard Behnke at dpa (Deutsche Presse Agentur) for this contribution.

We managed to solve our problem with undefined Firebird crashes in the following way:

W2003/Superserver

It is essential to check Firebird's memory requirements using the Task Manager. If the requirements are approaching 2 GB, there is a danger of Firebird crashing, e.g. if more than 2 GB is required when submitting a long and detailed query.

Solution

1. Equip your server with at least 3 GB, and ensure the 3GB switch is set in the Boot.ini. In order to handle this 3 GB address space, it is necessary to use the appropriate Firebird version (when the normal Firebird version is only linked with a different link flag). I think we may be the only company to currently be in possession of such a Firebird version (Paul Reeves performed the linking for us).

2. The best solution is however to change to the Firebird Classic Server, together with sufficient RAM and more that one CPU. This certainly puts life back into the database!

From:
http://ibexpert.com/docu/ - **IBExpert**

Permanent link:
http://ibexpert.com/docu/doku.php?id=01-documentation:01-05-database-technology:database-technology-articles:firebird-interbase-server:firebird-classic-server-versus-superserver

Last update: **2023/06/10 18:50**