



Understanding MKDE Internals: System Transactions

Revision 2

November 2002

Pervasive Software Inc.
The Embedded Database

Pervasive Software Inc.
12365 Riata Trace Parkway II
Austin, Texas 78727
Public Relations Contact: Marian Kelley
Telephone: 800-287-4383/ 512-231-6000
Fax: 512-231-6010
Internet: <http://www.pervasive.com>

© 2002 Pervasive Software Inc., Pervasive, Pervasive.SQL, Btrieve, and the Pervasive logo are registered trademarks of Pervasive Software Inc. All other product names are trademarks of their respective companies. All rights reserved worldwide.

INTRODUCTION

Please Note that this document applies to Pervasive.SQL 2000i.

What happens in the Pervasive.SQL engine when an application inserts, updates, or deletes a record? The application may be using the Btrieve API, ODBC, Java, or any other interface, but ultimately the insert, update, or delete request gets to the MicroKernel Database Engine (MKDE), which is responsible for getting that change into the physical Btrieve file on the disk.

In general terms, when an MKDE “change” request (insert, update, or delete operation) comes in, appropriate modifications are made in the MKDE cache to the necessary pages in the Btrieve file. In addition, the change request information is written into a Log Buffer (in memory). This work is all done in the foreground, and when it is complete, a status code is sent back to the calling application. At some point, background processes are kicked off to make these cached changes and Log Buffer information permanent by writing them out to disk.

PROCESSING DETAILS

To look at this process in more detail, we have to understand the jobs performed by the foreground and background threads, and the concept of a system transaction. Some of the behavior is determined by various configuration parameters, which are discussed in more detail at the end.

Threads

Foreground threads are responsible for processing an MKDE request. This involves making all the appropriate changes to any data file’s pages in the MKDE cache and recording that change into the Transaction Log Buffer. There are two kinds of foreground threads: Worker and Communication. Worker threads handle MKDE requests for applications running locally on the same machine as the engine (for example, an application running on the Windows NT Server). A Communication thread handles MKDE requests for applications running remotely (i.e. on a workstation in a client/server configuration).

Background threads are responsible for handling the I/O interaction with the operating system, or, in other words, they are responsible for getting information written out to disk. There are two types of background threads: IOThreads and the LoggerThread. IOThreads (also called “Background Writers”) keep track of pages in cache that need to be written to a Btrieve file on the disk. When a Btrieve file is first opened, it is assigned to an IOThread and all subsequent changes to that file by any users are handled by that assigned IOThread. As more files are opened, they are assigned round-robin to the allocated IOThreads, so a given IOThread may be responsible for writing to multiple Btrieve files.

The LoggerThread is responsible for writing the Log Buffer out to a Log Segment file. The Log Buffer is used to hold information about changes when Transaction Durability is On (which is the default behavior). The LoggerThread sleeps until a foreground thread wakes it up due to the buffer being full or because an application issued an End Transaction operation, or SQL Commit. At first glance, it would seem the Log Buffer is duplicating writes, which would decrease performance. However, the log buffer uses a simple flat-file write mechanism (i.e. Log Segment files are not Btrieve files) to get change information written to disk quickly so that it is “durable” – i.e. it can survive a system failure. Once information is written to a Log Segment file, the change request can be recovered if a system failure occurs before the changed Btrieve file pages make it to disk.

System Transactions

A System Transaction is a unit of work that gets data written to disk. System Transactions are performed on a per-IOThread basis. For further information about System Transactions, please see Chapter 4 of the Pervasive.SQL *Programmer’s Guide* provided with the Pervasive.SQL SDK (also available on-line at http://www.pervasive.com/support/technical/psql2k_docs.asp).

A System Transaction is initiated whenever any of the following occur:

1. The Operation Bundle Limit is reached;
2. The System Transaction Initiation Time Limit is reached;
3. A foreground thread needs more available cache.

The operation bundle limit is a configuration parameter that tells the engine how many change operations on a given file should be processed before they are all written to disk. In general, bundling changes together before writing them out improves performance, because fewer pages are written since some changes affect the same pages in the file.

The Initiation Time Limit is another configuration parameter that defines the maximum number of milliseconds that can elapse before the next System Transaction gets initiated (assuming the bundle limit and foreground thread didn't already initiate one).

When a foreground thread is processing a request and making changes in cache, it may need more cache than is available if the cache is currently full of changed pages waiting to be written to disk. In this case, the foreground thread initiates a system transaction so that some changes can be written out and some cache made available.

Once a system transaction is initiated, four phases occur to complete it:

- all the modified data, key, and variable shadow pages are written to the Btrieve file;
- the File Control Record (FCR) is written to the file with a new usage count and a Recovery Flag set. This flag means that the file is in a state that would need a roll-back if opened this way after a crash;
- the updated PAT (Page Allocation Table) pages are written to the file;
- the FCR is changed so that the Recovery Flag is turned off, and this page is written to the file.

While a system transaction is in progress, the file remains in a consistent state for all users since the pending changes are written on shadow pages and aren't made accessible until phase 4 is complete. If a system failure occurs after phase 2, when the engine is reloaded and the file is next opened, the MKDE will detect the Recovery Flag as set and take appropriate action to roll back any incomplete operations.

For performance reasons, phases 2 & 3 actually take place in a single phase. In addition, starting with the Pervasive.SQL 2000 SP2 release, the Windows NT Server product includes support for Asynchronous I/O by the IOThreads for improved performance. As a result of this implementation, the NT Server engine now combines phases 1, 2 & 3 and writes all shadows, the FCR and the PAT pages out to disk in one step.

After all the phases are complete, a final File Flush function appropriate to the operating system is called to complete the write to disk. This occurs in all environments except the Windows NT server with the Asynchronous I/O implementation; in this environment, a semaphore is signaled when all the pages are written.

When the final open handle to a given Btrieve file is closed, a final system transaction is executed on the IOThread to clear out any remaining changed pages from cache.

CONFIGURATION PARAMETER INFORMATION

Changing the various parameters related to this process is generally a trade-off between performance and the ability to guarantee changes in the database in the event of system failure. The default settings are good for most environments needing good performance with a strong likelihood of changes making it to the disk reliably. In some cases, new defaults have been defined starting with the SP3 release of Pervasive.SQL i. These changes were carefully considered based on customer feedback and internal testing, to try and provide the most appropriate out-of-the-box settings as possible for the majority of our customers.

In general, you should not make drastic changes to these settings unless you are in a test environment and can gauge the results of the change.

Operation Bundle Limit:

Default = 1000 (SP2a and earlier);
Default = 65535 (SP3 and later)
Maximum = 65535

Increasing this value allows for the maximum number of changes to occur before attempting to write to disk. With more changes, the write may take longer, but in general, you want to combine as many operations together as possible to reduce the overall number of page writes to disk.

Initiation Time Limit:

Default = 10,000 milliseconds
Maximum = 1,800,000

Increasing this value allows more time to pass between system transactions, allowing potentially more operations to be bundled together. With more time between writes, the writes may take longer, but in general, you want to combine as many operations together as possible to reduce the overall number of page writes to disk.

Cache Allocation:

Default = Dynamic
Maximum = Limited by memory

In Pervasive.SQL 2000i, the MicroKernel dynamically generates a default cache size the very first time the engine is started after installation. The MicroKernel creates the appropriate entries if they do not already exist and sets the default cache size to a percentage of memory.

- For server engines, this value is 20 percent of the physical memory on the server computer.
- For Workstation and Workgroup engines which run on Windows platforms, the default cache size is 10% of the physical memory on the computer or 8 Mbytes, whichever is less.

Once this initial load occurs, the "Cache Size" entry in the registry or in BTI.CFG is written with the calculated cache value, and the engine does not execute the default cache-sizing code on subsequent loads. The only way for the MicroKernel to execute the default cache-sizing code is if the user deletes the registry key or BTI.CFG entry.

If you change the current cache size setting, that size is used on all subsequent engine startups. No new default cache size is calculated.

Log Buffer Size:

Default = 64KB (SP2a and earlier)
Default = 256KB (SP3 and later)
Maximum = Limited by memory

The Log Buffer Size is the size of the transaction and archival log buffers, which hold information to be written out to the transaction and archival log files. Having a larger log buffer means the information doesn't need to be written out to disk as frequently, which could improve performance. However, for non-user transactions, a larger buffer means more changes could be lost in the event of a system failure.

Transaction Log Size:

Default = 512KB
Maximum = Limited by disk space

The Log Size is the size of each transaction log file written to the transaction log directory by the logger thread. It should be no smaller than the Log Buffer Size. Transaction Log files are automatically created as needed, and deleted when a successful file close operation is issued on the last handle to the file.

Communication Threads:

Default = 3 (SP2a and earlier)

Default = 16 (SP3 and later)

Maximum = 1024

Communication Threads handle MKDE requests coming from remote workstations. On NetWare, they are allocated per supported protocol. For example, if Pervasive.SQL 2000/2000i is configured on a NetWare server for both SPX and TCP/IP, and Communication Threads is set to 16, the engine will allocate 32 total threads. This setting is saved in the BTI.CFG on a NetWare server as the MaxWorkerThreads parameter in the [Btrieve Communications Manager] section.

When a request is received, it is assigned to an available communication thread; if all threads are busy, requests are queued up waiting for the next available thread. If too few threads are available, incoming requests will always have to wait to be processed. On the other hand, if too many threads are allocated and busy at the same time, it may take longer for any individual request to be processed since thread switching between many threads causes a longer delay to complete the work required by any single thread.

The default value of 16 threads is a good starting point; with several hundred concurrent users, you may want to increase this in increments of 16 to see how performance is affected. You should also watch the peak values for Communication Threads on the "MicroKernel Communications Statistics" screen of the Pervasive Monitor utility, to see if you are frequently hitting your configured value. If you are, you may need to increase this parameter. If you only rarely peak on your Communication Threads, and usually stay below your configured value, you probably should not increase it. On NetWare, you will typically not see any performance increase beyond 64 communication threads.

Number of Input/Output Threads:

Default = 4

Maximum = 1024

I/O Threads are the background threads responsible for writing changes out to Btrieve files on disk. You should never set this parameter higher than the number of physical Btrieve files you use concurrently. Typically, it is recommended that this parameter not be set higher than 8 unless you have confirmed any performance improvement gained by a higher setting.

This setting is saved in the BTI.CFG on a NetWare server as the BackgroundThreads parameter in the [MicroKernel] section.

Q & A

Q1: What steps should be taken after a system failure?

A1: In general, you should just proceed as normal after bringing up the server and workstation(s). Starting up the application causes the MKDE to open files that could have been opened when the crash occurred. If the MKDE opens a file and detects an inconsistent state, it either rolls forward changes from the transaction log files, or rolls back incomplete non-durable system transactions. In either case, your files are returned to a consistent state.

Q2: Is the process described for system transactions different if an insert operation is performed inside a user transaction?

A2: An End Transaction operation from an application wakes up the logger thread and tells it to write the contents of the Transaction Log Buffer out to a Transaction Log Segment file, to get the change written to disk. This guarantees that the change is on disk even if a system failure occurs before the normal processing of the background writers occurs to get the changes into the data files. This assumes Transaction Durability is On; if it is off, the user transaction is not guaranteed to make it to disk.

Q3: What happens if a user transaction is ended and receives a successful status code, but a system failure occurs before everything is flushed to the Btrieve file (assuming Transaction Durability is On)?

A3: When the system is brought back up and the application is started, the affected data files are opened. The MKDE detects the log segment files containing a durable transaction, and rolls that information into the data file.

Q4: What changes in this whole process if Transaction Durability is turned off?

A4: If user transactions are performed with transaction durability off, the End Transaction operation does NOT cause the Log Buffer to be flushed to disk in a log segment file; the information remains in cache waiting for the appropriate event to trigger the next system transaction. Therefore, there is no guarantee that the data is written to disk, even though the application may have received a successful return code on the End Transaction operation.

Q5: Why is the maximum value for IOThreads 1024 if it should never/rarely be set above eight?

A5: Currently, the MKDE can accommodate 1024 IOThreads, but values above eight can cause performance degradation in many environments. Future hardware, OS, and/or Pervasive releases may be better able to get higher performance with more IOThreads.

For more information about Pervasive.SQL 2000i, pricing or support, please contact one of our sales offices or visit our web site at <http://www.pervasive.com> .

Corporate Headquarters

Pervasive Software Ltd.

12365 Riata Trace Parkway
Building II
Austin, TX 78727
Phone: 800-287-4383 or 512-231-6000
Fax: 512-231-6010

Pervasive Software SARL

Immueble Atria
21, Avenue Edouard Belin
F-92 500 Rueil Malmaison,
France
Phone: +33-1-55-47-17-00
Fax: +33-1-55-47-17-07

International Offices

Pervasive Software Ltd.

Regus House
Highbridge, Oxford Road
Uxbridge, Middlesex UB8 1HR
UK
Phone: +44-1895-876331
Fax: +44-1895-876331

Pervasive Software GmbH

Frankfurter Strasse 151d
D-63303 Dreieich, Germany
Phone: +49-6103-9622-00
Fax: +49-6103-9622-05

Pervasive Software European

Service and Support
Bessenveldstraat 25A
B-1831 Diegem, Belgium
Phone: +32-2-710-1660
Fax: +32-2-718-0331

Pervasive Software, N.V.

Airport Boulevard Office Park
Bessenveldstraat 25A
B-1831 Diegem, Belgium
Phone: +32-2-718-0330
Fax: +32-2-718-0331

Pervasive Software Co., Ltd.

World Trade Center, 33F
2-4-1 Hamamatsu-cho
Minato-ku, Tokyo 105-6124
Japan
Phone: +81-3-5405-2261
Fax: +81-3-5405-2269

<http://www.pervasive.com>
info@pervasive.com
salesupport@pervasive.com
<http://www.pervasive.com/support/>

© 2002 Pervasive Software Inc., Pervasive, Pervasive.SQL, Btrieve, and the Pervasive logo are registered trademarks of Pervasive Software Inc. All other product names are trademarks of their respective companies. All rights reserved worldwide.

Disclaimer: The performance demonstrated in this document is for reference purposes only and does not constitute a performance guarantee even if under similar configurations.

PERVASIVE®