OpenEdge® RDBMS Performance Tuning Made Simple

EXCHANGE 2006 EDITION, 4 JUNE 2006

GUS BJÖRKLUND, WIZARD,
PROGRESS SOFTWARE CORPORATION

PROGRESS
SOFTWARE

OPENEDGE RDBMS PERFORMANCE TUNING MADE SIMPLE
GUS BJÖRKLUND

Contents

1 Introduction

This monograph is a short guide to performance optimisation techniques for the OpenEdge RDBMS. It is intended to be a useful collection of tips and techniques, not an exhaustive treatise on the subject. Many of the recommendations are quite easy to implement and will provide you significant "bang for the euro" without doing too much work or too much thinking[1]. Some of the recommendations are much easier to implement if you are creating a new database than if you have an existing database. For example, if you have a large existing database, redesigning the storage layout may be too time consuming to be convenient.

To illustrate what a little performance tuning can easily accomplish, we used a multi-user benchmark with a highly update-intensive workload. When we used an "out of the box" configuration of OpenEdge 9.1D with no tuning, throughput was about 30 transactions per second. By implementing just eight of the suggestions presented here, we obtained over an order of magnitude improvement – almost 600 transactions per second![2]

Because every database, computer, operating system configuration, application, and workload is different, there is no universal database configuration that will work best, or even equally well, for everyone. While several of the suggestions presented in this monograph can be useful for most systems, they may not be appropriate for *your* system. Therefore, when you implement them, you should always make measurements before and after making changes so you can be be sure the effects you get are the ones you wanted. An old carpenter's adage "measure twice, cut once" is apropos– but the second measurement should come afterward.

In several examples in the following sections we have used "foo" as the database name. You should of course substitute your own database name.

If you notice any mistakes or have suggestions for improvements to this monograph, please email gus@progress.com.

2 Measuring Performance

To know if your efforts at optimising performance are producing the results you require, or even if they are needed at all, you must have some means of measuring the effects the changes you make have on your installation's per-

[1] As Ted Williams said, "If you don't think too good, don't think too much.". Even if you do "think good", thinking is work and unnecessary work should always be avoided.

[2] Your mileage may vary, transportation, meals, and accomodations not included.

formance. The metrics you use should be specific and reproducible. For example, you might measure the elapsed time required for performing some function of your application such as adding a new customer, running jobs like month-end closing, or generating a particular report. There are a variety of tools you can use to examine system and database activity counters and timers that provide hints toward areas that may need improvement. Here's a short list of widely available tools:

- The OpenEdge RDBMS `promon` program, which displays a host of information about a database's status, activity, I/O rates, transactions, etc on more than 50 different screens. `promon` is included with the OpenEdge RDBMS. It connects to local databases through shared-memory

- Virtual System Tables are "magic" tables, present in every database instance, which you can query to get information similar to that displayed by the `promon` program. With little effort, you can write small 4GL programs to monitor specific values that are of interest. VST's can be accessed remotely from 4GL or SQL-based applications.

- The OpenEdge RDBMS database log file (the ".lg" file) records a variety of information when a database instance is started and stopped, when users and applications connect and disconnect, as well as error messages that may be generated while the system is operating. One set of information in particular is worth pointing out: most of the configuration parameters are recorded in the log file when a database instance is started. These can be quite useful when diagnosing problems, performance related and otherwise.

- OpenEdge Management (née Fathom Management), is a system management and monitoring console that gives you information about one or more database instances, actively monitors its operation, and generates alerts when operational metrics are "out of spec". Unlike the other tools, OpenEdge Management can also remember the past so you can examine trends in the data it collects. OpenEdge Management is available from Progress Software Corporation.

- ProTop is an excellent free monitoring tool written in the 4GL by Tom Bascom of Greenfield Technologies. It reads the Virtual System Tables to get the data it displays. ProTop monitors databases with an eye towards presenting the data like the UNIX "top" program–hence the

name. You can find ProTop on the web at
`http://www.greenfieldtech.com`

- Pro Monitor, from BravePoint, Inc. is another excellent monitoring tool written in the 4GL. It also reads data from the Virtual System Tables. It is designed to give you a simple, real-time view of your Progress OpenEdge system. Performance statistics are recorded daily in a format that is easy to read and understand. You can find it on the web at `http://bravepoint.com/products`.

- White Star Software's Adam Backman has written a set of UNIX DBA Scripts. This is collection of useful scripts for database monitoring and administration. They are available on the web at
`http://peg.com/utilities`

- Operating systems come with a variety of monitoring tools like `top`, `sar`, `iostat`, `vmstat`, `pstat`, `ps`, `perfmon` etc, that provide information about the operational metrics of the host operating system and the processes running on it. While many of these tools are available on most operating systems, the collection on each system varies somewhat and there are often tools specific to one system (such as HP's Glance) that are not available on the others. Consult the documentation for your system to see what is available.

- On Linux, the `/proc` filesystem contains a wealth of useful information about the system.

- Among other things, Solaris provides the pmap, proc, and truss commands.

- Operating system logs are another useful source of information. What information is available varies by operating system, but you should always consult the system logs when diagnosing errors.

- Last, but not least, is the independent Progress Email Group, run by the indefatigable Greg Higgins. You can find it on the web at
`http://www.peg.com`. The PEG has very lively and active discussions about all sorts of things Progress related. This is a great place to learn, ask questions, and get help with solving problems.

None of the aforementioned tools will tell you several important things, like:

- Are your users satisfied?
- Are there specific functions your users think are taking too long?
- Are there times when users are not getting acceptable response times?

The tools mentioned above will be useful in helping you to figure out how to solve their problems, but you should also use your users opinions as one of your monitoring tools. Another useful technique for measuring performance is to instrument your application to capture activity and timing data for those operations you think are worth measuring.

3 Perform Regular Backups!

When a system failure occurs (and one eventually will) and you do not have a recent backup to restore from, your database performance will be zero. It will be zero for a long time if you don't have a current backup. You may have to reconstruct your database from pieces or re-enter a lot of data by hand.

Once you have established reasonable and proper backup procedures, then you can worry about improving performance.

4 Easy Optimisations

This section contains recommendations which should be very simple for anyone to implement and they can be done quickly with only a few minutes of downtime. However, you should note that the easy ones are not necessarily optimal—but they are significantly better than the defaults and should get you close. Note that the configuration parameter default values mentioned in this monograph are those of Progress Version 9.1. Later releases may have different defaults.

4.1 Increase Buffer Pool Size

The most effective thing you can do to improve performance is to give the database more memory to work with, starting with the size of the buffer pool. The database buffer pool is a region of memory that is designed to speed up many database operations. The speedup comes from reducing the number of disk read operations by keeping copies of recently used data blocks in memory. Another speedup comes from reducing the number of

disk write operations because data blocks in memory can be updated multiple times before they are eventually written back to disk storage. Blocks that are already in memory can be accessed *much* faster than if they have to be read from disk first. The database buffer pool is normally the largest part of a multi-user database instance's shared memory region (SMR). By increasing its size you can (usually) improve performance.

You set the number of buffers in the database buffer pool with the `-B` configuration parameter. The buffer size is the same as the database's data block size.

The default size is quite low and will not be optimal for anyone. For single-user mode, the default is 10 buffers and for multi-user mode it is 8 times the value of `-n`. Since `-n` defaults to 20, the default number of buffers is 160. While the optimal value is dependent on your database and application workload, here are two very rough approximate values to start with:

- 10 % of database size
- 2 to 4 megabytes per user

In the `promon` program, you will find the value of the "Buffer Pool Hit Ratio" displayed on several screens. This value tells you the percentage of database accesses that could be satisfied from data already in memory instead of reading from disk. The higher the value, the better. If the value is 98 %, that means only 1 of every 50 database accesses required reading from disk. If the value is below 95 % and performance is insufficient, you should definitely consider increasing the buffer pool size. Try to get as much above 98 % as you can. Tom Bascom's ProTop program, mentioned earlier, has a buffer pool size "guesstimator" that may be helpful.

You can raise the size of the buffer pool as long as you have free memory available in the system. But there are limits–if you make it too large, then the system may start paging which can cause the entire system's performance to plummet. Many systems will have enough memory so you can set the buffer pool to 100,000 or more buffers. Note that for a database with an 8192 byte data block size, 100,000 buffers will consume approximately 900 megabytes of memory.

With 64-bit versions of Progress OpenEdge, if you have enough real memory in the system, you can have a total shared memory region size of about 116 gigabytes.

With 32-bit versions of Progress OpenEdge, the total size of your shared memory region must be smaller than 2 gigabytes, more or less. The exact

limit varies by operating system, but for most systems, 32-bit programs can use *roughly* 2 gigabytes[3] of database shared memory *under ideal conditions*. Often the actual limit will be less, depending on many different factors[4].

If your application connects to multiple databases *in self-serving mode*, the total space consumed by the shared-memory regions of all the databases you connect at the same time must be cumulatively no greater than the limit described previously. In addition, you must have enough other resources, for example file handles for all the database related files, and semaphores for process scheduling.

The more databases your application connects to at once, the smaller each one's shared memory region must be in order that they will all fit into the available address space of applications using self-serving conections.

4.2 More Before-Image Log Buffers

A set of before-Image log buffers are used to hold new transaction log records before they are written to the transaction log. Every transaction state change and database change is recorded in one or more transaction log records, which are first spooled into the current before-image log buffer, and then written to the before-image log, preferably by the before-image writer process. Having multiple buffers provides some extra "give" so the before-image writer has adequate time to do the writes. When the current log buffer is full, and there are no more empty log buffers, a transaction that wants to spool a log record has to wait until an empty buffer becomes available. By raising the number of buffers, you will reduce the probability of having to wait to complete a database change.

The default number of before-image log buffers is 5 and is set with the `-bibufs` configuration parameter. You should increase the number to 25 or so. Note that this parameter is not a "speed control". You only need to have a enough buffers to smooth out temporary variations in the amount of log data that needs to be written. With the `promon` program, you can see how many *Empty buffer waits* occur for the before-image log. If the value is more than a few per second, you should consider increasing the number of buffers. However, if the disk that contains the before-image log is saturated and unable to perform additional write operations, then increasing the number of buffers will have no effect.

If you do not use the Before-Image Writer, then increasing the number of buffers will have little or no effect.

[3] on recent Linux versions, about 2.7 gigabytes is the maximum

[4] which are outside the scope of this monograph

4.3 More After-Image Log Buffers

When the After-Image Journalling facility is enabled, a set of after-image log buffers are used to hold new transaction log records before they are written to the current after-image journal extent. Every transaction log record written to the before-image log[5] will also be written to the after-image journal, preferably by the after-image writer process. Like the before-image log buffers, when the current after-image log buffer is full, and there are no more empty after-image log buffers, a transaction that wants to spool a log record has to wait until an empty after-image buffer becomes available. By raising the number of buffers, you will reduce the probability of having to wait to complete a database change.

You should set the number of after-image log buffers to a value slightly higher than the number of before-image log buffers. 25 to 50 percent higher seems to work well in most cases.

If you do not use the After-Image Writer, then increasing the number of buffers will have little or no effect.

If you do not use the After-Image Journalling facility, then increasing the number of buffers will have no effect and just wastes memory.

4.4 Use The `-spin` Option

The database can use *spinlocks* as a fast and low overhead method for synchronising the activities of multiple processes that access and operate on memory-resident shared data, such as the buffer pool. If the system hosting your database has more than one processor, you should set a value for the `-spin` configuration parameter.

Finding the *optimal* value for spin is time consuming but finding a *good enough* value is not–just use 50,000. With modern systems, we suggest using the number of processors times 20,000 as a rough starting point, but simply using the value 50,000 is easier to remember and usually sufficient. In most cases, determining the exact best value is not that important and not worth your time. In our benchmarks with 150 users, we saw very little difference in throughput when the `-spin` value was varied from 10,000 to 90,000, but the difference between values of 2,000 and 40,000 was about 50% higher (436 vs 680 transactions per second). Generally, the lower the number of users accessing the database, the less difference adjusting the spin value makes.

[5] Actually, there are one or two exceptions, but they are not worth worrying about

Provided you started the database with a nonzero value, you can use the `promon` program to change the setting of the `-spin` configuration parameter while the database is running. This makes it easier to determine a good enough setting. Don't forget to update your database startup configuration file or script if you do this.

We have seen one large system with many processors where a value of 2,000,000 worked pretty well, but that is somewhat unusual.

Note that the Workgroup edition of the OpenEdge RDBMS does not support the use of the `-spin` configuration parameter.

4.5 Raise Before-Image Log Cluster Size

In multi-user mode, the database performs a continuous process, called *checkpointing*, for writing modified data blocks to disk in the background. A new checkpoint cycle is started each time a new before-image log cluster is opened.

Larger before-image log cluster sizes increase the duration of checkpoint cycles and allow more time for modified database blocks to be written to disk in an orderly fashion by the page writers. Durations for the last 8 checkpoints are displayed in promon's Checkpoints display. If the checkpoints are about a minute long, or longer, and the number of buffers flushed is zero or close to zero, you are fine. Longer durations are acceptable but not needed. If they are shorter than a minute, you should increase the cluster size. If the number of buffers flushed is higher than 10, you will need to increase the cluster size, or increase the number of page writers, or increase your disk write capacity.

You can set the before-image cluster size with the `proutil` program using the following command:

```
proutil foo -C truncate bi -bi 4096
```

You specify the cluster size in units of kilobytes. A good size to use as a starting value is 4 megabytes, but if your database has very high levels of update activity you may need to use a higher value. The database remembers the cluster size after you set it

If you have the Workgroup edition of the OpenEdge RDBMS, keep the cluster size small. 512 k or less will be better than larger cluster sizes because there will not be any page writers to write modified database blocks to disk and this can result in periodics "freezes" during the time all modified buffers are written to disk.

4.6 Set Before-Image Log Block Size

By increasing the before-image block size, writing of the before-image log becomes more efficient. On UNIX systems (Solaris, AIX, HP-UX, Tru64, UnixWare, etc) you should use 8k. On Linux systems you should use 4k. On Windows systems you should use 4k unless you have increased the NTFS cluster size as described in the section on Windows-specific topics.

You can set the before-image block size with the `proutil` program using the following command:

```
proutil foo -C truncate bi -biblocksize 8
```

You specify the block size in units of kilobytes.

4.7 Set After-Image Log Block Size

By increasing the after-image block size, writing of the after-image log becomes more efficient. Preferably, the after-image block size should be the same as the before-image log block size. On UNIX systems (Solaris, AIX, HP-UX, Tru64, UnixWare, etc), you should use 8k. On Linux systems, you should use 4k. On Windows, you should use 4k unless you have increased the NTFS cluster size as described in the section on Windows-specific topics.

Before you change the after-image block size, you must end after-image journalling (if it is active) and truncate the *before-image* log as follows:

```
rfutil foo -C aimage end
proutil foo -C truncate bi
```

You can set the after-image block size with the `rfutil` program using the following command:

```
rfutil foo -C aimage truncate -aiblocksize 8
```

You specify the block size in units of kilobytes.

4.8 Use Before-Image Writer (biw)

The before-image writer is an i/o process whose purpose is to write the contents of newly filled before-image log buffers to disk asynchronously and in the background so the server does not have to do it. This gives the server more time to do useful work. When using self-serving clients, the

server and the client are in the same process, but you will still want the server part to do useful work.

Note that the Workgroup edition of the OpenEdge RDBMS does not have a before-image writer.

You can start the before-image writer with the following command:

```
probiw foo
```

You can start only one before-image writer.

4.9 Use After-Image Writer (aiw)

If you use after-image journalling (and you should, but not to improve performance), run the after-image writer (aiw) process.

The after-image writer is an i/o process whose purpose is to write the contents of newly filled after-image log buffers to disk asynchronously and in the background so the server does not have to do it. This gives the server more time to do useful work. When using self-serving clients, the server and the client are in the same process, but you will still want the server to do useful work.

Note that the Workgroup edition of the OpenEdge RDBMS does not have a after-image writer.

You can start the after-image writer with the following command:

```
proaiw foo
```

You can start only one after-image writer.

4.10 Use Asynchronous Page Writers (apws)

You should always run at least one asynchronous page writer (apw) process.

The asynchronous page writer is an asynchronous background i/o process whose purpose is to write recently changed database blocks to the data extents on disk in an orderly fashion. In this way, the server does not have to do it, and modified data blocks do not have to be written to disk all at the very end of a checkpoint cycle. The `promon` program reports these writes as *buffers flushed* in several places. You want that number to be less than 10 for most checkpoints.

In most cases, one or two page writers will be sufficient. Having too many generally doesn't hurt, but having a very large number like 50 might

reduce performance a little because buffer pool contention will be increased slightly as the page writers examine it.

Note that the Workgroup edition of the OpenEdge RDBMS does not have page writers.

You can start an asynchronous page writer with the following command:

```
proapw foo
```

4.11 Avoid Single-user Mode

Nearly all the recommendations in this chapter assume you are running the database in multi-user mode. You will usually experience significantly worse performance using the database in single-user mode.

When the database is being used in single-user mode, there is one and only one connection to the database and there is no shared-memory region (all the data structures are allocated in process-private memory instead). Single-user mode has the following disadvantages over multi-user mode:

- There are no asynchronous background processes to write data to the data extents and transaction logs. This limits performance.
- You cannot use the `promon` program or other monitoring programs to monitor database activity.
- You cannot use any of the online utility programs. This includes functions like switching the active after-image journal extent, archiving full extents, or performing online backups.

5 Optimising Disk Layout

This section provides suggestions for optimising the disk layout of your data areas. For existing databases, because moving or dumping and reloading large amounts of data can be time consuming, you may find them impractical. If you are creating a new database, they will be easy.

5.1 Use More Than One Disk For The Database

One disk drive can do one data transfer at a time. Two disk drives can do two transfers at the same time. The more disk drives you have for the

database, the higher your performance can be. Storage capacity is unrelated to disk performance and several smaller drives will give you better performance than a single large drive.

5.2 Stripe and Mirror Disks

For best performance, you will want to have the IO workload evenly balanced over the available drives so that they all contribute equally to handling the workload. If one disk drive is doing much more work than the others, it can slow down the system if it is unable to keep up with its workload.

The most effective way to balance the load across multiple drives is to create a stripe set that combines all the drives into one logical drive with the data evenly spread across them. Striping alone has a serious disadvantage: if you lose one drive because of a failure, you will lose them all. To correct this problem, you have to combine striping with *mirroring* to get sufficient reliability. You should mirror pairs (or even triples) of drives and then stripe over the pairs. This is more reliable than the reverse because such a configuration can tolerate more disk failures.

In the tests we have performed on Linux and Windows, we found that larger stripe sizes provide better performance than smaller. The maximum stripe size you can use depends on your operating system or disk subsystem. We have not tested stripe sizes larger than 256 kilobytes because that was the maximum size we had available to us.

5.3 Avoid RAID 5

When using RAID-5 disk configuration instead of striping and mirroring, you are likely to achieve a 45 percent performance loss[6] in normal operation and more when doing maintenance or recovery operations. The performance loss is caused by the write overhead inherent in the RAID-5 design–each block must be written to two drives, one for the data, and one for the recovery data. In an array with four drives, a write always consumes *50 percent* of the total available disk bandwidth. The performance penalty is lower the more drives there are in the array. In a RAID-5 array with 20 drives, the write penalty can be as low as only 5 percent extra. But: if you have to replace a failed disk drive, then all of the data on all of the remaining drives has to be read while the failed drive is being recon-

[6] observed in a benchmark using 6 drives

structed. This can be time consuming and causes a very severe degradation in performance until the reconstruction has been completed.

5.4 Choosing Data Block Size

Larger block sizes provide greater disk IO efficiency and make more efficient use of database storage. Many UNIX filesystems have a fundamental block size or page size that is 4 kilobytes or 8 kilobytes. You will get the best performance when the database block size matches the filesystem's page size or is a multiple of the filesystem page size.

You choose the data block size at the time you create a new database. You do this by making a copy of an empty database with the desired block size. Once created, you can load data into it. You cannot change the data block size of an existing database.

5.4.1 Type I Data Area Block Size

When the data block size is larger than the filesystem page size, there is a small chance that a data block write can be only partially completed in the event of a failure such as a power failure. In most cases, a data block that is written to two contiguous filesystem pages will be performed by a single disk write operation and both pages will be written together. However, if the two filesystem pages are on different tracks, a seek operation might be required to write the two pages. If the power failure occurs just as the first page transfer begins, the disk may shut down before the second page can be written. In this case, only half of the data block will be written to disk, resulting in a *torn page*, Crash recovery will not detect this situation and cannot not repair it.

On Linux systems, to match the system page size, you should use a data block size of 4 kilobytes. The current Linux virtual memory architecture does not allow for larger page sizes. This may change with some future Linux release.

On Windows systems, you should either use a data block size of 4 kilobytes or set the NTFS cluster size to 8 kilobytes and then use an 8 kilobyte data block size.

5.4.2 Type II Data Area Block Size

With type II data areas, the database computes block checksums and stores them in all data block headers, enabling detection of torn pages. If all of

your data is in type II data areas, you can use a block size that is a multiple of the filesystem page size.

With Type II areas, set the data block size to 8 kilobytes. Future versions of the OpenEdge RDBMS will support larger data block sizes.

5.5 Use Type II Data Areas

Type II data areas provide higher performance and improved space allocation with less fragmentation than type I data areas. Most utility programs will complete their work in significantly less time. For these reasons, you should always use type II data areas.

5.6 Use Fixed-Length Data Extents

Fixed-length extents are formatted at the time they are created. Variable-length extents are formatted dynamically as they are expanded at runtime when the database needs more space to store data. Fixed-length extents provide moderately higher performance because the extents do not need to be expanded during space allocation for index or table data. The performance difference can vary significantly depending on your particular operating system and filesystem and how much and how often your database grows. The difference is greatest when the data block size is smaller than 4096. There is no difference in read performance between fixed and variable extents.

5.7 Separate Before-Image Log From Data Extents

Locate the before-image log extents on disks separate from your data extents.

You want writing the before-image log to be as efficient as possible and no interference from other activity.

If you have many databases on the same machine, then you should put the before-image logs in with the data extents. You did stripe the data extents, didn't you?

5.8 Disk Array Subsystems

There are a great variety of disk storage subsystems on the market, ranging from various types of disk controllers such as the ones from 3Ware to large storage systems from EMC, IBM, and others. Most of them can be used successfully with the OpenEdge RDBMS.

6 Advanced Topics

The optimisations described in this section may be a bit more difficult or inconvenient to implement than the ones described earlier. Some of them may be altogether impractical if your database is very large or if your system is underpowered. Doing the work might require more downtime than you can afford.

6.1 Don't Forget The Application

Since this monograph is about database tuning, we won't go into application writing here but it is a topic you should not ignore. Proper database configuration can have a significant impact on performance, but proper application design and coding can have a much larger impact. For example, if you application is using queries that require a table scan, improving the speed of the table scan will never be as effective as not doing it at all.

6.2 Creating Records

Each table in the database has a *template record* which holds default values for the columns in the table. When new rows are created, the row is initialized by making a copy of the template record. This happens when you execute a `CREATE` statement in the 4GL. Most applications set column values shortly afterward. You should immediately assign values to all the columns after creating a row and this should be done in a single `ASSIGN` statement. This way, when the row is created in the database, the initial space allocation for the row will include all the column values. Otherwise, when the row is created with only some column values filled in and the rest defaulting, perhaps to the unknown value, then space will be allocated for a fairly short row and later, when the remaining column values are filled in, the row will be updated. If the row changes size a lot between the time it is created and when it is updated, then the data block it was assigned to might not have enough free space to store the larger row. In that case, the row will be divided into two or more fragments, each stored in a different data block, increasing the overhead when it is updated and also each time it is fetched thereafter.

6.3 Make Sure You Have the Right Indexes

No amount of database tuning and reconfiguration will make up for a poorly written application so do not neglect to look into it. If you are under pressure to improve performance immediately, by all means do the obvious database configuration and tuning first, and then come back to the application. Another application-related factor that can have enormous effect on performance is having the right indexes to be able to satisfy the queries of your application without the need to perform full index range scans.

You can determine if your 4GL application is using full index range scans by compiling your 4GL code with the `XREF` compile option. The compiler will then annotate all database accesses with information about which tables are being accessed and what index(es) are being used. When a full index range-scan is performed, the listing will include the notation `WHOLE-INDEX`.

For SQL queries, you can look at the query plans that are generated for specific queries in the application. Note that SQL query processing often benefit from additional indexes beyond the ones you need for your typical well-written 4GL application. This is because many SQL queries are ad-hoc and they vary quite a bit in complexity and the amount of data they access. In many cases, creating additional indexes can improve SQL query performance significantly. If you execute a SQL query with the NO-EXECUTE option, you can then examine the query plan without actually having to run the query.

6.4 Outer Edge of Disk is Fastest

Disk drives rotate at a constant angular velocity, and the tracks at the outer edge are significantly longer than the ones on the inner edge. This means there is more data per track at the edge so the transfer rate is higher by about 20 percent. By constructing disk partitions in a suitable manner and storing the data extents in the outermost partition, you can squeeze a few more percentage points of performance from the system.

6.5 Manual Striping

If your disk controllers or operating system do not support striping, you can use a technique called *manual striping*. The idea is to create many equal-sized data extents that each contain a small piece of the whole database. You then spread these over multiple disks in a round-robin fashion. For

example, if you have four data drives, you could create 32 extents and put eight on each drive. Put extent 1 on the first drive, extent 2 on the second, extent 3 on the third, extent 4 on the fourth, extent 5 on the first, and so on. If your database has 10 gigabytes of data and you have 4 drives, and you use an extent size of 250 megabytes, you would need 40 extents and could place 10 on each drive.

Drawbacks to this manual striping technique are:

- Results are not as good as with operating system, disk controller or storage subsystem striping.
- When the database is composed of many small files rather than a few large ones, you will use significantly more file handles and other system resources.
- As the database grows larger and you add extents, maintaining a balanced disk workload becomes progressively more difficult,
- Most operating systems inbuilt striping capability can do a better job.
- It does not provide any mirroring capability.
- The balance is likely to change over time as the workload and data change

6.6 Use 64-bit Versions

All currently available Sun, IBM, and HP computers that run UNIX have 64-bit processors. They haven't made any 32-bit processors for quite a few years. These systems can run both 32-bit and 64-bit versions of OpenEdge.

If a 64-bit version of OpenEdge is available for your operating system, you might consider using it instead of the 32-bit version. The main benefit of doing so is that you can use much larger database buffer pools. On the 32-bit versions, the shared-memory region cannot be larger than roughly 2 gigabytes, but on 64-bit versions it can be as large as 116 gigabytes.

6.7 Lock Shared-Memory Region In Memory

On most operating systems, the shared-memory region is created so that it is pageable by the operating system's paging mechanism. If the shared-memory region is actually being paged (due to insufficient real memory), database performance will decline steeply when database reads and data

buffer accesses incur extra disk I/O because to the shared-memory paging. You can prevent this by locking the shared-memory region into real memory by using the `-pinshm` configuration parameter.

Using `-pinshm` can improve performance, but it can also make things worse. If you have insufficient real memory, many database instances running on one server, and one or more of them is seldom used, locking all the shared memory regions in memory might cause paging to increase for the remaining virtual memory used by the applications and other programs.

On Solaris systems, you do not need to use the `-pinshm` option because the database always uses the Solaris ISM ("Initmate Shared Memory") option, which also causes the shared region to be locked in memory.

`-pinshm` is not available on Windows and AIX systems.

6.8 About The `-directio` Option

The normal way the OpenEdge RDBMS performs database reads and writes on UNIX is to use buffered i/o using the `pread()` and `pwrite()` system calls, combined with `fdatasync()` calls (or `sync()` calls in versions prior to 10.0) at the end of each checkpoint cycle to ensure that data written to the filesystem buffers will be forced to disk. *On some filesystems of some operating systems*, when the database is using buffered i/o, the overall disk-write workload can be quite bursty. This occurs because as the page writers write database blocks to disk, the data just go into the filesystem buffers and are written to disk later, possibly not until after the next fdatasync() call. So all the careful work the page writers do to plan their activities to smooth out database writes is somewhat ineffective.

When you use the `-directio` configuration parameter, the database performs what is known as *synchronous i/o*, and uses either the `O_DSYNC` or the `O_SYNC` option when it opens the data extent files. In this mode, all data block reads and writes use the operating system's buffers in the usual way[7], but a `pwrite()` system call does not return until after the data have been sent to the disk. As a result, the page writers take a little longer to perform writes, but the overall i/o rate is smoother and the disk workload will have fewer spikes.

When the database is using synchronous i/o, it can also avoid using time consuming `fdatasync()` calls[8], which can block the calling thread until all the buffers associated with the file have been flushed to disk.

[7] you may have heard it said that the `-directio` option bypasses the operating system's buffers, but this is not true

[8] or `sync()` calls in versions earlier than 10

By using the -directio configuration parameter, you can gain the following beneficial effects:

- The database does not have to issue time consuming fdatasync() or sync() calls, and in the case of sync(), unnecessary i/o caused by flushing data that has nothing to do with the database instance completing a checkpoint is eliminated.
- Overall disk write scheduling is much more even because writes occur when the page writers want them to, and they try to organize their activities to provide as even a write rate as they can.

If you use -directio, you may also have to increase the number of page writers beyond the one or two that are normally needed.

Note that using the -directio parameter has little effect on some operating systems and may not be useful in your situation. It has been found to be most useful on AIX systems.

You may also find -directio helpful in Linux systems in situations where there are periodic "freezes" under heavy load. You can recognize these situations by using the vmstat program to monitor disk IO. If you see high rates of disk IO followed by periods of several seconds where there is no disk IO at all taking place and the system is heavily loaded, you *may* be able to resolve the problem by using -directio.

6.9 Multiple Semaphore Sets

The OpenEdge database uses the operating system-provided semaphores for several purposes, such as managing certain internal locks and for blocking processes that must wait for row, table, or schema locks. On some UNIX systems, semaphore performance can be improved by using the -semsets configuration parameter. By default, the OpenEdge RDBMS uses a single large semaphore set for each database instance. The size of this semaphore set is determined by the maximum number of users that can connect to the database. The -semsets parameter tells the database to use the specified number of separate smaller semaphore sets. Groups of users are allocated separate semaphore sets. This can improve performance *if* there is internal contention in the system kernel when several users are performing operations on the same semaphore set at the same time. The performance improvement will be most noticeable in systems with more than about 150 users and can provide up to five percent higher throughput[9].

[9] Your mileage may vary.

The best value to use for the number of semaphore sets depends on the specific workload and the operating system being used. We suggest you try using one semaphore set for every 20 to 50 users.

On Linux systems, the `-semsets` configuration parameter has no effect. The Windows operating system does not have semaphore sets.

6.10 Spread After-Image Log Extents On Two Drives

The purpose of after-image journalling is to provide for a way to recover if the drive(s) holding your database fail. Therefore you MUST NOT store any after-image extents on the same drives as the data extents.

Alternate after-image log extents between the two drives, placing the first on drive 1, the second on drive 2, the third on drive 1, and so on.

Alternating the extents between two disk drives allows the filled extents to be read and archived without slowing down the writing of new data to the current extent. If you do not have enough disk drives to dedicate two for after-image journalling, put all the after-image extents on the same drive.

6.11 Filesystems

Most operating systems have a variety of different filesystems options to choose from. In many cases, the particular filesystem you use does not affect database performance all that much because database systems generally do not create and delete large quantities of files. Many filesystems are quite usable for database storage, but there are some exceptions. The table below lists a few filesystems that are known to work well for databases, for a variety of operating systems.

Operating system	Filesystem
Linux	ext3 or JFS
AIX	JFS or JFS2
Solaris	UFS or VxFS
Tru64	AFS
Windows	NTFS

The table below lists some filesystems that you should never use.

Operating system	Filesystem
Windows	FAT16 and FAT32
HP-UX	HFS
Linux	ReiserFS

6.12 The `noatime` mount option

Filesystems on Linux and UNIX have a variety of options that can be specified when the filesystem is mounted. One of the these is called `noatime`. It dispenses with updating files' last access time. By specifying this mount option, you can reduce filesystem overhead somewhat. Your mileage may vary, depending on the operating system.

6.13 Increase SHMMAX

On many operating systems, the maximum shared memory segment size is configured to be smaller than the maximum size the OpenEdge RDBMS can use. When that is the case, the database will create many smaller segments. You should set the maximum size to 128 megabytes (134,217,728). In benchmarks on Linux, this also increased throughput by a modest 2%.

6.14 Linux I/O Schedulers

Linux systems based on the 2.6 kernel have several disk i/o schedulers you can choose from. Benchmark data shows that some of them should be avoided. We recommend using the "deadline" scheduler. Avoid the CFQ and anticipatory schedulers. The CFQ scheduler gave especially bad results in benchmarks. The deadline scheduler gives about 10% higher throughput than the anticipatory scheduler.

You specify the scheduler you want to use by adding a line to the boot loader configuration file. For example, to use the deadline scheduler, add the line

```
"elevator=deadline"
```

6.15 Use a Dedicated Database Server

The more stuff you run on the machine that has the database on it, the more resources you take away from database performance.

That means no print serving, file serving, mail serving, screen savers, Microsoft Office, and so on.

7 Networking-specific Topics

- Keep a local copy of the schema cache. Use the command

```
SAVE CACHE COMPLETE foo TO foo.cache
```

to save a local copy of the database's schema information. Connecting to the database will then be faster because less information is read from the database at connect time. Don't forget to update the schema cache when you change table or index definitions or add new ones.

- Use Gigabit Ethernet. By using Gigabit Ethernet instead of 10mbit for network clients, you can achieve close to the same performance as with self-serving clients that are running on the same machine as the database.

- Configure network interfaces for FDX. Most modern network devices, such as NIC's and switches, support full-duplex Ethernet. By using it, you can noticeably improve network speed. Note that all devices on the same network segment must be configured for full-duplex.

- Increase maximum message size (the `-Mm` configuration parameter). By using a larger message buffer size, database access over a network connection can be improved because fewer messages are sent and received. The default message buffer size is 1024 bytes, which is quite small. 16k is a good value to use. Note that both client and server must be configured to use the same value. Note also that only some messages need to be sent with a maximum size buffer. When messages are shorter than the maximum size, only as many bytes as needed are transmitted.

- Decrease clients per server. By decreasing the maximum number of clients per server configuration parameter, each server process will operate on behalf of fewer clients and will be able to provide better response because when a new client request arrives, the server is less likely to be busy already. If you decrease the number of clients per server, you should also increase the maximum number of servers.

- Use multiple brokers. By using multiple brokers, you can provide different qualities of service for different users. For example, you might configure one broker for use by batch jobs and another for interactive users.

8 SQL-specific Topics

If you are using the OpenEdge SQL Server with ODBC or JDBC interfaces, there are a number of things you can do to improve performance.

- Add indexes - Interactive SQL queries, whether generated by a tool such as Crystal Reports or EasyAsk, or directly entered by a user, tend to be more unpredictable than the ones built into applications. You may benefit from adding more indexes to suit the queries that are being generated by such tools.
- Use separate broker for SQL - By using a separate broker for SQL connections, you can isolate them from 4GL connections and manage them separately. This allows you to use different configuration parameters for things like the maximum number of servers and clients per server.
- Use type II data areas - By placing your tables and indexes in type ii data areas, you enable the SQL server to take advantage of the improved performance and other capabilities offered by type ii data areas.
- Update statistics - The SQL query optimizer relies on statistics about the data in the database to decide which of the many possible query execution plans it will use. You should periodically use the `UPDATE STATISTICS` command to make sure the statistics are current. The index statistics are the most important ones to update.
- WITH clause of SELECT - The `WITH` clause of the SELECT statement provides finer control over lock behaviour and allows adjustment of the lock-wait timeout.
- NO REORDER - You can use the `NO REORDER` option with the `FROM` clause in `SELECT` statements to force table join order to be from left to right as written in the query text. This may sometimes be useful for improving the speed of specific joins if the optimiser makes the wrong choice.
- Use the API's effectively - The DataDirect developer web site has several technical articles that discuss how to use the ODBC and JDBC API's to get the best performance from you SQL application.

- Use OpenEdge 10.1, which has many performance improvements to the SQL Server as well ODBC and JDBC drivers that utilize the new and faster SQL wire-protocol.

9 Windows-specific Topics

This section lists a few things you can do on Windows servers to make them work better.

- Use NTFS filesystems. Windows has several different filesystem types to choose from. Use NTFS. Never use FAT-16 or FAT-32.
- Set NTFS cluster size to 8k - the NTFS file system's default cluster size is 4k. By increasing it, you will get better performance for large files, and you can also use block sizes of 8k for the data extents and transaction logs when the cluster size is 8k. Use the following command to initialize a new NTFS partition:

  ```
  format <drive>:/fs:ntfs /A:8k
  ```

- Configure memory management for applications, not file serving. To do this, perform the following actions:
  - Right-click on "My Network Places"
  - Select "Properties".
  - When the "Network Dial-up and Connections" dialog box is displayed, right-click on "Local Area Connections"
  - Select "Properties" again.
  - In the "Components" pane, click on "File and Printer Sharing for Microsoft Networks".
  - Click on "Properties", again!
  - Choose "Maximize Throughput for Network Applications".
  - Click "Ok".
- Eliminate crapware - Windows systems are particularly susceptible to unwanted software being installed without your knowledge. All you have to do is visit a hostile web site with Internet Explorer. Such software can severely reduce system performance, reliability, and security. You should use a tool like Ad-Aware or Spybot Search and Destroy to remove these infestations. Windows database servers should not be used to access the Internet at all.

- Short filenames - Modern Windows systems don't need to use the 8 character filenames that DOS used to have, but they are still generated for compatibility. You can get a little performance boost from disabling the creation of the short filenames. Directory lookups and file creation will be slightly faster. Then again, for best database performance, you shouldn't be creating and deleting many files on the database server and once the database has been started, it doesn't need to open more files very often.

- Turn off unneeded services - Most Windows systems have many system services turned on by default. By turning off the ones you don't need, you can get back some memory and CPU cycles that can then be put to better use. Microsoft's TechNet web site (at http://www.microsoft.com/technet) has an article entitled "System Services for the Windows Server 2003 Family and Windows XP Operating Systems" which describes all the services. This web page: http://majorgeeks.com/page.php?id=12 explains which ones could be disabled.

- Turn off unneeded startup items - Many Windows systems are configured to start a variety of programs when the system is started. These programs consume memory and cpu time. You should turn of all the ones you don't need.

- Turn off GUI decorations. The various options for fancy appearance of windows and dialog boxes consume memory and cpu cycles.

- Turn off file sharing. The database server should not be used for other purposes such as print or file serving because they consume memory, cpu cycles, and disk bandwidth.

- Remove Office - Microsoft Office has a variety of background processes that use memory and processor time. The worst offender is the file indexer.

- Don't use screensavers - Many of the Windows screensavers consume a lot of CPU time. Set the screen saver to solid black for best performance.

- Don't run virus scan software - virus scanning software can consume large quantities of cpu cycles and disk bandwidth. You may then ask "but how will I prevent my Windows system from becoming infested then?". Install Linux, grasshopper.

10 Conclusion

In the introduction, we claimed that we improved throughput from 30 to almost 600 transactions per second by performing just a few simple tuning tasks. The changes we made to the default settings were:

- Set the database block size to 8192
- Set the buffer pool size (`-B`) to 64,000
- Stripe the data extents over 8 disks
- Locate the bi log on a separate disk
- Start 4 page writers[10]
- Start the bi writer
- Set the bi cluster size to 16 megabytes
- Set -spin to 50,000
- Increase the number of bi buffers (`-bibufs`) to 25

The suggestions we've given in this monograph should get you a long way toward the best performance but remember three things:

- All systems are different. "Your mileage may vary. Transportation, meals, and accomodations not included."
- Keep notes on what you have done and why you did it. later, when you have forgotten, you will have a way to find out. Write comments in your scripts and .pf files to document them.
- Stop when when you've accomplished enough. Most people can always find another thing to improve or still have something to do that they haven't gotten around to yet. The more tuning you do, the smaller the return on your investment usually becomes. You can accomplish a lot by doing just a few things. Adding memory and spreading the disk workload across as many disks as you can will have the most effect.

What are you waiting for?

[10] This is probably more than you need. One or two should be sufficient.