Infor LN Performance Tracing and Tuning Guide

Publication Information

Release: Infor LN 10.x

Publication date: December 3, 2012

Document code: U9357E US

Contents

About this guide

This document provides guidelines to improve the Infor LN performance by tracing and tuning the environment. The following chapters deal with the processes to improve the operating system and the Infor LN application, not the database.

For database specific tuning and tracing information, refer to the database specific performance, tuning and tracing guides listed in the related documents section.

The following preconditions apply to this document:

- All information is based on the use of the Infor LN software. If you require information about other versions, read the relevant documentation.
- The database-related information mentioned in this document is described in more detail in database specific guides.

Note: This document is a comprehensive compilation; however there may be instances wherein relevant information or procedures may have been omitted. Therefore, we strongly recommend verifying the proposed changes in a test environment before moving to production. The information provided may not hold true for future versions of the Operating System, database and application software.

Intended audience

This document is intended for intermediate to expert Infor LN and database Administrators and Technical Consultants in order to get optimal performance out of an Infor LN system.

Organization

This table shows the chapters of the guide:

Section	Description
Chapter 1, Introduction	Introduces the scope of this document.

Section	Description
Chapter 2, The Infor LN Architecture	Describes how the Infor LN application can be traced with the standard Infor LN tools.
Chapter 3, System Configuration and Tuning	How to configure a UNIX, Linux or Windows system for optimal Infor LN Performance.
Chapter 4, Tuning Infor LN	Discusses Infor LN optimization parameters.
Chapter 5, Batch Performance Optimization	Explains the special settings to be considered for running Infor LN batches.
Chapter 6, Tracing Infor LN	How to identify performance problems within Infor LN.
Chapter 7, Performance Wizard	Helps to find the bottlenecks and how to solve them.

Related documents

Certain sections in this document are described in more detail in other documents. The following documents help to extend the knowledge in particular areas.

- *Infor LN - Performance, Tracing and Tuning Guide for DB2 (B0077 US)*
- *Infor LN - Performance, Tracing and Tuning Guide for Oracle (B0078 US)*
- *Infor LN - Performance, Tracing and Tuning Guide for SQL Server (B0079 US)*
- *Infor Enterprise Server - Technical Reference Guide for DB2 Database Driver (U7829 US)*
- *Infor Enterprise Server - Technical Reference Guide for Oracle Database Driver (U7076 US)*
- *Infor Enterprise Server - Technical Reference Guide for Microsoft SQL Server Database Driver (U8173 US)*
- *Infor Enterprise Server - Technical Manual (U8172 US)*
- *Infor Enterprise Server - Administrator's Guide (U8854 US)*
- *Infor LN – Deployment in a Virtualized Environment (B0073 US)*
- *Infor LN - Application Response Time Measurement (U7379 US)*
- *Infor LN - Performance Guidelines (U9502 US)*
- *Infor LN - Sizing guide (B0045 US)*
- *Infor LN - Installation Guide (U9498 US)*

You can find the documents in the product documentation section of the Infor Xtreme Support portal, as described in "Contacting Infor" on page 11.

Abbreviations and Terminology

Abbreviation	Description
OLTP	Online Transaction Processing. A normal user behavior.
Batch	Applications that run to process data without user interaction.
CGP	Call Graph Profiler. The Infor LN call graph profiling tool.
DDL	Data Definition Language, such as CREATE and DROP TABLE commands.
DML	Data Manipulation Language, such as INSERTS, UPDATE, and DELETE queries.
DLL	Dynamic Load Library. A library that will be loaded in the application when required.
Named or licensed user	A user who can potentially log on to the Infor application
Connected or logged-on user	A user who is logged on to the Infor application
Active or concurrent user	A connected user who is actively using the Infor application
2-Tier Server	System that runs both the applications from both the application server and the database server
3-Tier Application Server	System that runs the bshell (Virtual Machine) and the database driver
3-Tier Database Server	System running the database
DS	DS protocol, used for communication between the Infor Web UI Web server and the Infor LN server
JVM	Java Virtual Machine
ION	Integrated Open Network platform
DAS	Direct Attached Storage
SAN	Storage Attached Network
IOPS	Input/Output Operations Per Second

Contacting Infor

If you have questions about Infor products, go to the Infor Xtreme Support portal at www.infor.com/inforxtreme.

If we update this document after the product release, we will post the new version on this Web site. We recommend that you check this Web site periodically for updated documentation.

If you have comments about Infor documentation, contact documentation@infor.com.

Chapter 1 Introduction

1

After the installation of the Infor LN software, we recommend that you start a continuous process of monitoring and improving the environment. Infor LN strives for optimal default performance settings, but tuning is still required for optimal performance. It is still not possible to expect an out-performing Infor LN application without any performance knowledge. Optimization starts during the installation, and continues when users become more experienced with the processes, and expect more from the system when the database grows and more functionality is added. This document describes how performance bottlenecks can be identified and total performance can be improved.

Note that most of the recommendations provided are proven by high-end benchmarks. Smaller customers should take into the account the administrative overhead of deviating from default settings, as not every tunable will increase performance in small Infor LN implementations.

An application's performance is based on the following factors:

- Performance

  Performance is the speed of the application, which is usually measured in:

  - Response time (OLTP)
  - Run time (batch)
  - Transactions per unit of time (batch).

  The performance aspect will receive the most attention from end users as it will influence their day-to-day work.

- Effectiveness

  Effectiveness deals with the efficient use of the available recourses. To increase the speed, it is maybe easier to add faster CPUs, more memory, or faster disks, but the cost can be higher. Therefore, adding hardware does not always solve a performance problem. In case of a badly performing SQL query, adding more or faster CPUs might help, but it is more effective to change the query or query plan.

- Scalability

  This aspect looks into the impact when the following happen:

  - More users are added: Will the increase in the number of users from 100 to 200 impact the performance? Locking can become a problem here.
  - More data is added: Will we still be able to process 200 orders per minute?
  - Scale-in: What happens when more CPUs are added?
  - Scale-out: What happens when more systems are used in parallel?

This document covers all these scenarios and helps to find a solution for better performance.

Chapter 2 The Infor LN Architecture

2

In order to get a well performing Infor LN environment, it is important to understand the Infor LN architecture. This chapter provides an overview of the most important components.

The components in the architecture model are as follows:

- The user interface (UI) is the Infor workspace and Web UI interface to connect to the Infor LN environment.
- The Infor LN application, which consists of:
  - A platform dependent layer called portingset. The main component of the portingset is the virtual machine, also known as bshell, which runs the platform independent applications / sessions. A bshell process is started for each user. The bshell loads the database driver. The database driver translates Infor LN queries into database native queries. The database driver can also run as a separate process.
  - A platform independent layer with the application sessions. This layer exists of the Enterprise Server with the 4GL engine and administration sessions. On top of that, the Infor LN application runs, executing the business logic.
- The database is used for storage of data and can be any of the Infor supported databases for Infor LN.

A model of the Infor LN architecture is shown in figure 2-1:

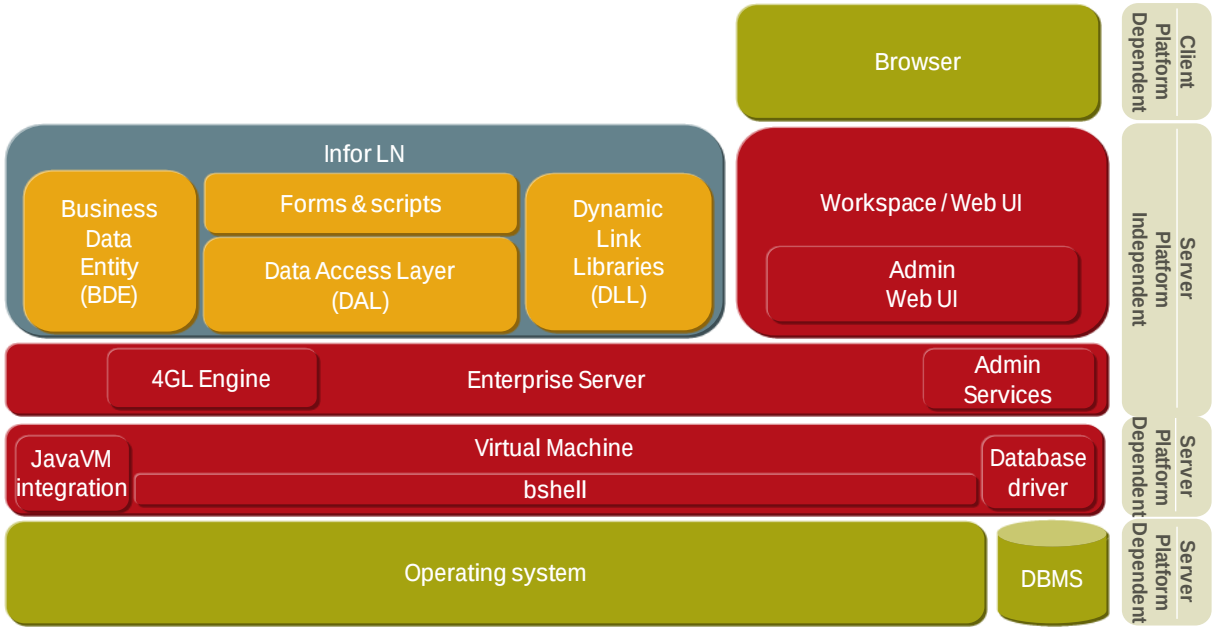


Figure 2-1 Infor LN architecture

If an application requires interaction with the database, an application command is sent to the database driver. The driver translates the command into a database-specific command and sends it to the database. Retrieved results from the database are translated to data understood by the bshell.

This architecture has the following characteristics:

- Although multiple sessions can be started in one bshell, only one session can be active. Time sharing is used to give each session some processing power.
- The communication between the different components is synchronous, which means that only the bshell, database driver, or database can be active at a certain time.

The Infor LN bshell has a single threaded architecture. Therefore no more than one session must be run simultaneously in one bshell. More than one session can be opened simultaneously, but only one session can process at a time.

See *Infor LN - Sizing Guide (B0045 US)* for more detailed information about the Infor LN architecture.

Deployment

The components discussed in the previous section are called tiers and can be deployed on separate servers. Which deployment should be chosen, depends on performance, security and functional requirements. See *Infor Workspace with Infor Web UI - Sizing and Deployment Guide (B0032 US)* for more information about possible deployment scenarios.

You can run the Infor LN application and database on the same server and the UI tier on separate servers (2-tier scenario). It is also possible to run every tier on separate servers (3-tier scenario).

Figure 2-2 shows the 2-tier deployment scenario in a graphical way.

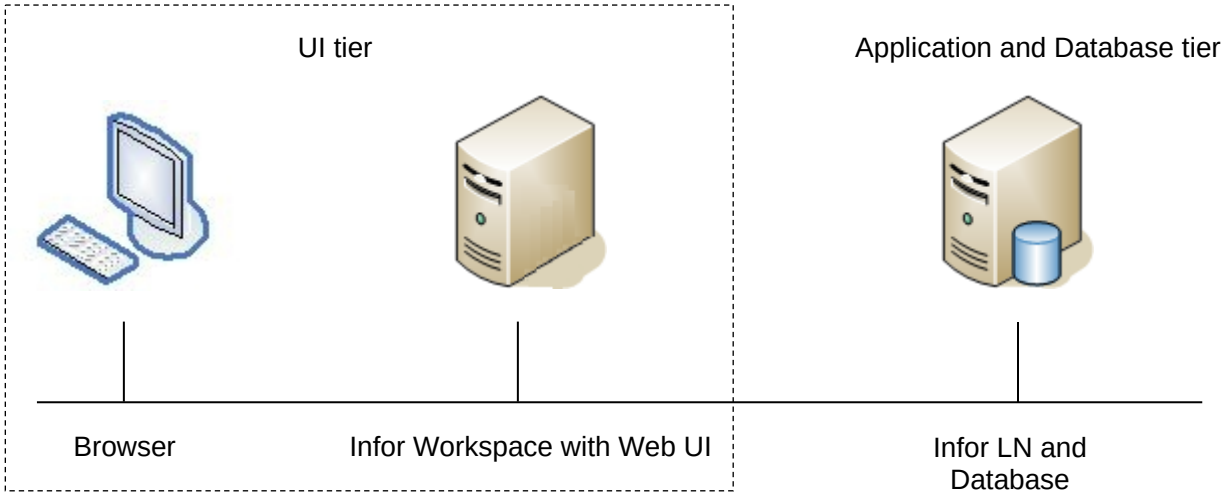


Figure 2-2 Infor LN 2-tier deployment scenario

Figure 2-3 shows the 3-tier deployment scenario in a graphical way.

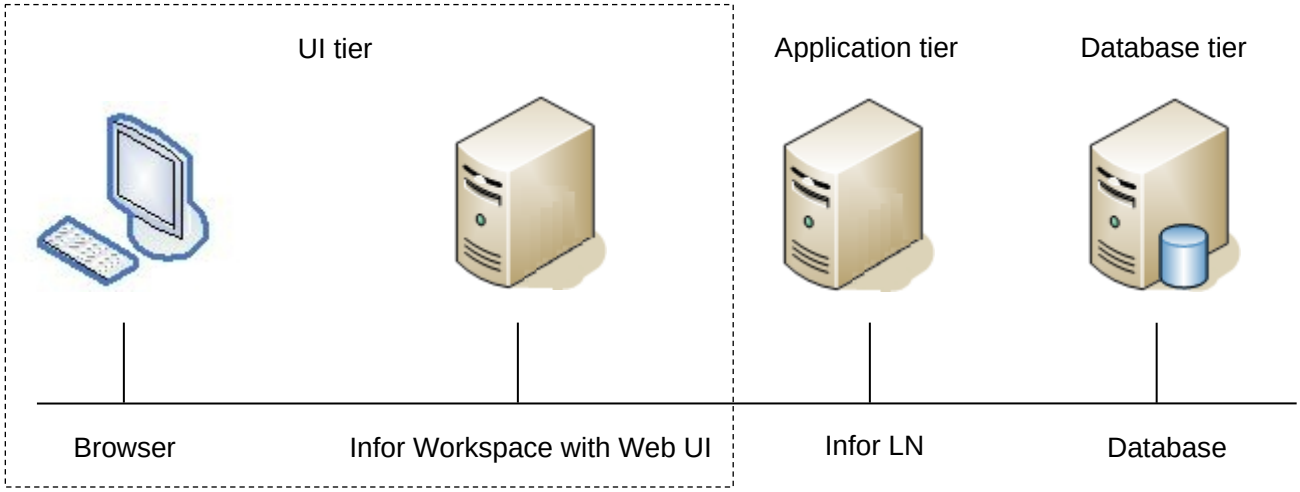


Figure 2-3 Infor LN 3-tier deployment

Chapter 3 System Configuration and Tuning

3

Before you install Infor LN, check the OS configuration and kernel. Necessary changes must be made to avoid performance issues. This chapter explains the OS configuration settings and kernel parameters for various platforms.

General

Some of the generic performance related information applicable for all operating systems and databases is discussed below.

Hardware selection

No extensive rules for hardware selection will be described in this paragraph. The message being, for a high performance Infor LN system, you must buy the correct system and hardware that suits your business needs. See *Infor LN - Sizing whitepaper (B0050 US)* for more information or consider sizing by Infor.

I/O Setup

The key to get a well performing Infor LN system is the correct I/O setup, which is valid for any type of I/O setup, such as Direct Attached Storage (DAS) and Storage Attached Network (SAN) solutions. Nowadays, the performance of the storage subsystem not only depends on the number of disk spindles, but storage vendors are introducing more advanced technologies like flash-based read caching, by combining SSD's and SAS drives into the same SAN, optimize data patterns to prevent random write access, compression techniques, etc. For the Infor LN application, it is important that the total number of IOPS requested can be managed by the storage subsystem. Refer to the sizing for the required number of Input/Output Operations Per Second (IOPS).

The following guidelines help design and implement a correct I/O setup:

- For smaller implementations using a DAS solution, configure enough drives to keep the transaction log volume to less than 225 IOPS per disk and keep the data and/or index volumes to less than 85 IOPS per disk. Regularly monitor the number of IOPS per disk. If disks are nearing their capacity limit, add more disk drives.

- For optimal performance, the I/O subsystems must be configured in RAID 1, RAID 0+1 or RAID 10 volumes. When performance of the database data files is not so important, RAID 5 or RAID 6 can also be used.
- For larger implementations (200+ Concurrent users) or systems with more than 10 disk spindles, we recommend isolating the database transaction log onto its own RAID 10 volume. I/Os to the transaction log are mostly sequential writes. For optimal performance, configure transaction log I/O cache such as cache in SCSI Controller, to 100 percent write. Do not use this volume for any other purpose.
- The database temp table space will not heavily be used in an Infor LN environment. However, for larger implementations, or for other applications using the Infor LN database, it is advised to give temp its own volume
- Try to keep disk fragmentation as low as possible on the database disks.

Storage setup

A good storage setup is important for performance. The following guidelines will help design and implement the storage setup and are applicable for every database:

- The Infor LN database must be created with the appropriate size. If 150 GB is the requirement for a year, create the database with this size. Do not let it grow and cause additional fragmentation in the database and the files on disk. If auto growth is needed, use big chunks, like 10 GB, to limit the number of extents.
- Raw devices still give the best performance, but are difficult to manage. However, a correct implementation of Direct I/O and Concurrent I/O will give a performance up to 97% of raw devices.

For database specific storage recommendations, see the database specific Performance, Tracing and Tuning Guide.

Mount options

To get the optimal performance on your UNIX system, it is very important that the file systems are mounted with the correct mount options. The file systems must be split based on data access patterns:

- Keep commit heavy data away from data that does not have to be synchronous.
- Keep streaming writes and reads on different spindles rather than random I/O

The mount option recommendation for the different file types are shown in this table.

File type	Access type
Database Redo log and Archive log	Direct I/O
Database Data files	Concurrent I/O

File type	Access type
Binaries	Cached I/O

Direct I/O

Direct I/O can be more efficient than using cached I/O and provides the benefit for writing the Database Redo log and archive log files.

Concurrent I/O

Concurrent I/O allows multiple processes to read from or write to the same file without blocking other read(2) or write(2) calls. This increases the performance, due to the huge decrease of JFS inode contention. Concurrent I/O performance range is 93-99% of raw logical volumes.

Cached I/O

Cached I/O will use the OS file system buffer cache, which is ideal for binaries, as they will not change often and kept in memory as much as possible. Cached I/O is enabled by default.

Binaries must never be mounted for direct I/O as data blocks are often accessed multiple times. Using cached I/O on the database and application binaries will eliminate unnecessary I/O.

Large pages

Large pages are very useful for large database shared memory (for example Oracle SGA) sizes and in general for systems with large amount of physical memory. Benchmarks showed that Infor LN running on a database using large pages will benefit from the use of large pages. Using large pages optimizes the use of Translation Lookaside Buffers (TLB) and the system has less bookkeeping to do for that part of virtual memory, due to the larger page sizes.

Physical memory is partitioned into pages which are the basic unit of memory management. When a process accesses a virtual address, the CPU must translate it into a physical address. Therefore, for each process, the kernel maintains a page table that is used by the CPU to translate virtual addresses into physical addresses. Before the translation, the CPU has to perform several physical memory reads to retrieve page table information. To speed up this translation process, the CPU saves information for recently accessed virtual addresses for future references, in its Translation Lookaside Buffers (TLB), a small but very fast cache in the CPU. The use of this cache speeds up the virtual memory access. Since TLB misses are expensive, TLB hits can be improved by mapping large contiguous physical memory regions by a small number of pages. So, fewer TLB entries are required to cover larger virtual address ranges. A reduced page table size also means a reduction in memory management overhead.

IBM AIX

The following parameters can be changed to optimize the performance of the Infor LN application running on IBM AIX 6.1 and higher.

Large pages

AIX maintains separate 4 KB and 16 MB physical memory pools. You can specify the amount of physical memory in the 16 MB memory pool using the vmo command. Starting with AIX 5.3, the large page pool is dynamic, so the amount of physical memory that you specify takes effect immediately and does not require a system reboot. The remaining physical memory backs the 4 KB virtual pages.

AIX treats large pages as pinned memory. AIX does not provide paging support for large pages. The data of an application that is backed by large pages remains in physical memory until the application has been shut down. A security access control mechanism prevents unauthorized applications from using large pages or large page physical memory. The security access control mechanism also prevents unauthorized users from using large pages for their applications. For non-root user ids, such as database owners like oracle or db2inst1, you must enable the **CAP_BYPASS_RAC_VMM** and **CAP_PROPAGATE** capability with the chuser command in order to use large pages.

Configure the AIX large page pool by calculating the number of large pages required for the database shared memory (for example Oracle SGA):

```
num_of_large_pages = INT(total_db_mem_size GB / Large page size) + 1
```

If the database shared memory is 20 GB, use 20 GB Large pages + 1 extra page. For example:

```
num_of_large_pages = INT(21474836480) / 16777216) + 1 = 1281
```

The following steps allocate 20 GB of RAM as large pages (16 MB):

1. Run as user root, these commands to reserve 20 GB of large page:

   ```
   # vmo –p –o lgpg_size=16777216 –o lgpg_regions=1281
   # vmo -p -o v_pinshm=1
   ```

2. Reboot the system

3. As user root, add these capabilities for the user:

   ```
   # chuser capabilities=CAP_BYPASS_RAC_VMM,CAP_PROPAGATE <user id>
   ```

4. Validate large page support is used with this command:

   ```
   # vmstat –P ALL

   System configuration: mem=112640MB
   pgsz            memory                           page
   ----- -------------------------- ------------------------------------
              siz      avm      fre    re    pi    po    fr     sr    cy
     4K 23515136 13955711  9307982     0     0     0     0      0     0
    16M     1281      785      496     0     0     0     0      0     0
   ```

For more information about large pages, refer to the IBM Info center at http://publib.boulder.ibm.com/infocenter/aix/v7r1 and search for “large pages”.

Mount options

The following table shows the AIX mount option recommendation for the different file types.

File type	Access type	Mount options
Database Redo log and Archive log	Direct I/O	dio (JFS2 + agblksize=512) For Archive Log Files, the rbrw mount option can be advantageous.
Database Data files	Concurrent I/O	cio (JFS2 + agblksize=<DB block size>)
Binaries	Cached I/O	Optional: noatime

Note 1: When using DIO/CIO, I/O requests made by Oracle must be aligned with the JFS2 block size to avoid a demoted I/O (Return to normal I/O after a Direct I/O Failure).

If block size is >=4096, use a file system block size of 4096, else use 2048.

Oracle Redo logs are always written in 512B block; So the JFS2 block size must be set to 512:

```
# mkfs –o agblksize=512 <filesystem>
```

Note 2: When using the **rbrw** mount option, the file system will be mounted with both release-behind-when-reading and release-behind-when-writing capabilities. When sequential reading of a file in this file system is detected, the real memory pages used by the file will be released once the pages are copied to internal buffers. When sequential writing of a file is detected in this file system, the real memory pages used by the file will be released after the pages are written to disk. This can be advantageous for database redo log and archive log file systems. For more information, see http://pic.dhe.ibm.com/infocenter/aix/v7r1/topic/com.ibm.aix.prftungd/doc/prftungd/multiple_page_size_support.htm

Note 3: Use of Concurrent I/O when running on AIX 5.3 or higher in combination with DB2.

The database manager already prevents caching of most DB2 data, except temporary data and LOBs on AIX, by invalidating the pages from the cache. For more information, see: http://pic.dhe.ibm.com/infocenter/db2luw/v10r1/topic/com.ibm.db2.luw.admin.dbobj.doc/doc/c0051304.html

Direct I/O

To enable direct I/O, use the following command:

```
# mount –o dio <mount_point>
```

Concurrent I/O

To enable concurrent I/O, use the following command:

```
# mount –o cio <mount_point>
```

Cached I/O

No specific mount options are needed for file systems using cached I/O. However, it is possible to specify the Noatime option, to turn off access-time updates. Using this option can improve performance on file systems where a large number of files are read frequently and seldom updated. If you use this option, the last access time for a file cannot be determined.

```
# mount –o noatime <mount_point>
```

Network

In order to optimize the network for Infor LN, especially in a 3-tier environment, the following parameters must be changed.

- The tcp_recvspace tunable specifies how many bytes of data the receiving system can buffer in the kernel on the receiving sockets queue. The tcp_sendspace tunable specifies how much data the application can buffer in the kernel before the application is blocked on a send call. Set "tcp_recvspace" and "tcp_sendspace" to larger values in case of 1 Gb or higher network:

  ```
  # no -p -o tcp_recvspace=262144
  # no -p -o tcp_sendspace=262144
  ```

- "tcp_nodelayack" turns off delaying the acknowledgement. This can reduce latency and allow the sender to receive the acknowledgement and thus send the next partial segment sooner.
  To turn off delaying the acknowledgement, use the following command:

  ```
  # no -p -o tcp_nodelayack=1
  ```

- The rfc1323 tunable enables the TCP window scaling option. Use the following command to set this option:

  ```
  # no -p -o rfc1323=1
  ```

Restart the system in order to effectuate these settings.

Thread tuning

Setting the following 2 options influences the kernel thread behavior:

- The AIXTHREAD_MNRATIO variable controls the scaling factor of the library. Changing this variable makes the kernel thread support more efficient by setting one kernel thread for each user thread.

This ratio is used when creating and terminating pthreads. It may be useful for applications with a very large number of threads. By default, one kernel thread serves 8 user threads. (8:1). As Infor LN is a single threaded application a ratio of 1:1 will be beneficial.

To set this variable permanent, add the following line to /etc/environment and restart the system:

```
AIXTHREAD_MNRATIO=1:1
```

- The AIXTHREAD_SCOPE variable controls the AIX thread contention scope. The P option signifies a process-wide contention scope (M:N) while the S option signifies a system wide contention scope (1:1). One of these options must be specified; the default is P.

  If a user thread is created with system-wide scope (S), it is bound to a kernel thread and it is scheduled by the kernel. The underlying kernel thread is not shared with any other user thread.

  If a user thread is created with process-wide scope (P), it is subject to the user scheduler. This means that the user thread:

  - does not have a dedicated kernel thread.
  - sleeps in user mode.
  - is placed on the user run queue when it is waiting for a processor.
  - is subjected to time slicing by the user scheduler.

  Benchmarks showed that user thread created with a system wide scope is beneficial for Infor LN. To place this variable as permanent, add the following line to /etc/environment and restart the system:

  ```
  AIXTHREAD_SCOPE=S
  ```

Fiber channel parameters

In order to support high disk I/O, some fiber channel adapter parameters can be changed. To obtain a list of the current hard disk parameters, enter the following command:

```
lsattr -El fcsX
```

Change the value as follows:

- The lg_term_dma parameter controls the memory area to store I/O commands and data. This can be increased to 8 KB using the following command:

  ```
  # chdev –l fcsX –a lg_term_dma=0x800000
  ```

- The max_xfer_size parameter controls the maximum I/O that can be issued. This can be increased to 2 KB using the following command:

  ```
  # chdev –l fcsX –a max_xfer_size=0x200000
  ```

- The num_cmd_elem parameter controls the maximum number of simultaneous I/Os the fiber channel adapter can handle. This can be increased to 400 using the following command:

  ```
  # chdev –l fcsX –a num_cmd_elem=400
  ```

Hard disk parameters

On high-end systems it may be useful to make some changes to the hard disk parameters. To obtain a list of the current hard disk parameters, enter the following command:

```
lsattr -El hdiskX
```

Change the value of every hard disk in the same volume group as follows. The volume group needs to be varied off, before applying the changes.

- The queue_depth parameter controls the maximum number of I/O commands that can be queued on each hdisk. Default is 20. This can be increased to 40 using the following command:

  ```
  # chdev –l hdiskX –a queue_depth=40
  ```

- The max_transfer parameter controls the maximum disk I/O on each hdisk. This can be increased to 4 Mb using the following command:

  ```
  # chdev –l hdiskX –a max_transfer=0x400000
  ```

- The algorithm and reserve_policy parameters are set to support multipath I/O using AIX MPIO capabilities. MPIO provides better I/O throughput and path redundancy.

  ```
  # chdev –l hdiskX –a algorithm=round_robin
  # chdev –l hdiskX –a reserve_policy=no_reserve
  ```

Additional information about AIX performance

For more information, see these documents:

- The AIX 7.1 Information Center
  http://pic.dhe.ibm.com/infocenter/aix/v7r1/index.jsp
- Optimizing AIX 7 performance
  https://www.ibm.com/developerworks/aix/library/au-aix7optimize1

Hewlett Packard – HPUX – IA64

The following parameters can be changed to optimize the performance of the Infor LN application, running on HP-UX 11.31 and higher.

Enable hyper threading

The default behavior for HP-UX based systems is to have the hyper threading feature disabled and the default Processor Set's LCPU (Logical processor) attribute disabled. As hyper threading will be beneficial for Infor LN performance, it is recommended that you enable the feature.

To turn the LCPU attribute (HT) on for the default Processor Set (PSET), use the following command:

```
# kctune lcpu_attr=1
```

File-system tuning

As performance of the application and database is very I/O dependent, it is important that the file system is configured optimally. Following are some guidelines to tune the file system.

Mount options

The following table shows the recommended HP-UX mount options for the different file types.

File type	Access type	Mount options
Database Redo log and Archive log	Direct I/O	delaylog,mincache=direct,convosync=direct
Database Data files	Concurrent I/O	delaylog,cio
Binaries	Cached I/O	delaylog,nodatainlog

Direct I/O

To enable direct I/O, use the following command:

```
# mount -F vxfs –o delaylog,mincache=direct,convosync=direct <device_special_file>
<mount_point>
```

Concurrent I/O

To enable concurrent I/O, use the licensed VxFS CIO mount option:

```
# mount -F vxfs -o delaylog,cio <device_special_file> <mount_point>
```

Notes:

- The remount command/option must not be used when using the “cio” mount option.
- Do not use “cio” and “mincache=direct, convosync=direct” together as it may cause degradation. Use either Direct I/O or Concurrent I/O.

Cached I/O

Use the following mount options for cached I/O:

```
# mount -F vxfs –o delaylog,nodatainlog <device_special_file> <mount_point>
```

Kernel parameters

The following HP-UX kernel parameters are important for Infor LN performance. In case the recommended value is lower than the default value, keep the default value.

Tune inode cache

Tune a static VxFS inode cache. This will save some CPU:

```
# kctune vxfs_ifree_timelag=-1
# kctune -h vx_ninode=64000
```

FILECACHE_MIN / FILECACHE_MAX

Specifies the minimum and maximum percentage of memory used for file system buffer pages. These parameters will be considered for a lower and upper boundary for the buffer pages.

We recommend a minimum size of 2% and maximum of 10% of the total internal memory for the file system cache:

```
# kctune filecache_min=2%
# kctune filecache_max=10%
```

NPROC

The NPROC parameter specifies the maximum number of processes running at the same time. It depends on the configuration as to how you tune the NPROC parameter. Some variables are used in the formulas. Their definitions are as follows:

- **#connected user**: Maximum number of connected users on the system. The users do not have to be active, but still consume system resources. It is assumed each end user logs on once. Increase the number of connected users with extra logins for each end user.
- **#additional database drivers**: Extra database drivers needed to access multiple database instances at the same time. Each connected user normally uses one database instance and one database driver. In certain situations, this value is larger than zero.

2-tier	3-tier application server	3-tier database server
6 * #connected users + 512	4 * #connected users + 512	2 * #connected users + 512

These settings are based on the following:

- Each connection contains a bshell, (incl database driver in case of combo mode), an audit driver, a database backend process, and one additional process. In case of exceptional circumstances, the number of processes must be increased.
- Each user connects to Infor LN through Web UI or the Worktop.
- Each user runs one database driver. In case of multiple database drivers, the number of processes must be **increased** with the following:

2-tier	3-tier application server	3-tier database server
2 * #additional database drivers * # connected users	#additional database drivers * # connected users	#additional database drivers * # connected users

Note: The 512 additional processes are based on a common environment with a database. In case of additional products, the total amount of processes must be increased.

When UNIX error 11 “No more processes” occurs, the value of NPROC must be increased.
The current and maximum number of processes can be found in the column proc-sz, from the **sar -v** output.

Note: On some UNIX systems the following extra command is started by the inetd:

```
sh –c /.../ipc_boot6.2/...
```

To prevent this, use a Korn shell **(/usr/bin/ksh)** as a login shell in **/etc/passwd**, instead of a Bourne shell **(/bin/sh)**.

MAXFILES

This parameter specifies the maximum number of files that can be opened per process. Unlike the database, Infor LN does not use numerous files at the same time. A save value is:

```
MAXFILES >= 256
```

The number of this parameter must be increased when error 24 appears: *Too many open files.*

SHMMAX

SHMMAX specifies the maximum size of a shared memory segment that can be created in bytes. Several shared memory segments of this size can be created per process in an environment.

We recommend setting SHMMAX equal to the size of the internal memory. For example (32 GB):

```
SHMMAX = 34359738368
```

In case a database also runs on the same system, take the recommended value from the database vendor. Usually, it will be between 60% and 100% of internal memory.

Increase this value in case of error 22.

Messages

Messages can be used for communication between processes. Communication between processes by message queues is not the most optimal performing method. Therefore, skip configuring message queues in the kernel. If message queues (m) are used in the $BSE/lib/ipc_info file, try to change them into pipes (p).

MSGMNB

This parameter specifies the maximum length, in bytes, of a message queue. The maximum number of bytes in a message queue is 262144. Therefore, set MSGMNB to at least this value:

```
MSGMNB >= 262144
```

Check the number of bytes in a message queue with **ipcs –qo** command:

MSGMNI

Specifies the maximum number of message queues that can be used system-wide.

The number of message queues required for Infor LN is relatively small. MSGMNI must be 64 or more.

```
MSGMNI = 64
```

Increase the number when error 28 occurs.

MSGTQL

Specifies the maximum number of message headers that is the maximum number of open messages.

Keep this parameter equal to MSGMNI:

```
MSGTQL = MSGMNI
```

MAXUPRC

This parameter describes the maximum number of concurrent processes per user id.

The value depends on the configuration. Usually, the default value is fine. However, there are some exceptions when this parameter must be increased, such as follows:

- Multiple users run under the same userid. This often occurs before going live. Several users logon under the id of bsp. Together, they generate a lot of processes.
- If **BSE_REM** is used, customers prefer to run all users as one particular remote user, such as the user **bsp**. In that case, the number of processes per user must be increased.
- On a 3-tier database server, all processes run with the process ID of the database owner on several databases. Therefore, increase the number of processes per user.

The above situations can cause problems on the MAXUPRC parameter. If any of these situations occur, calculate the maximum number of needed processes per user and tune the parameter.

If you run out on processes, increase the value of NPROC, or lower the value of MAXUPRC when possible.

Additional information about HP-UX performance

For additional reading, see these documents:

- HP-UX VxFS mount options for Oracle Database environments
  http://h20195.www2.hp.com/V2/GetDocument.aspx?docname=4AA1-9839ENW&cc=us&lc=en
- Common Misconfigured HP-UX Resources
  http://bizsupport1.austin.hp.com/bc/docs/support/SupportManual/c01920394/c01920394.pdf
- Veritas™ File System 5.0.1 Administrator's Guide (HP-UX 11i v3)
  http://h20000.www2.hp.com/bc/docs/support/SupportManual/c02220689/c02220689.pdf
- Performance Improvements using Concurrent I/O on HP-UX 11i v3 with OnlineJFS 5.0.1 and the HP-UX Logical Volume Manager
  http://h20195.www2.hp.com/V2/GetDocument.aspx?docname=4AA1-5719ENW&cc=us&lc=en

Oracle Solaris

The following parameters can be changed to optimize the performance of the Infor LN application running on Oracle Solaris. Infor tested below settings on Oracle Solaris 10.

Fixed-Priority (FX) Scheduling Class

The FX scheduler provides a scheduling policy for processes that require user or application control of scheduling priorities. The priorities of processes that run under FX are fixed. These priorities are not dynamically adjusted by the system. Benchmarks showed that Infor LN benefits from fixed priorities on the Application- or 2-tier server.

Use the dispadmin command to set or get the scheduling class.

Multiple Page Size Support

To gain the maximum CPU performance for your Infor LN server use Multiple Page Size Support (MPSS) for the database. To set up MPSS with optimal configuration of 4 MB heap size, set the following in the profile for the Oracle users; for example:

```
set LD_PRELOAD=$LD_PRELOAD:mpss.so.1
set MPSSHEAP=4M
export LD_PRELOAD MPSSHEAP
```

For more information about MPSS, see
http://www.solarisinternals.com/wiki/index.php/Multiple_Page_Size_Support

Intimate Shared memory

On Solaris, Oracle can use (Dynamic) Intimate Shared Memory (DISM). Dynamic ISM allows a database to dynamically extend or reduce the size of the shared data segment. This feature eliminates the misconfiguration problem and denial-of-service security vulnerability of Intimate Shared Memory.

The ISM is a shared memory segment that consists of large locked memory pages. The ISM number of locked pages remains constant or unchanged. Dynamic ISM is pageable ISM shared memory, where the number of locked pages is variable or changeable. Therefore, the DISM supports releasing or adding more physical memory to the system during dynamic reconfiguration. The size of the DISM can span available physical memory plus disk swap.

For more information about DISM see: http://www.oracle.com/technetwork/articles/systems-hardware-architecture/using-dynamic-intimate-memory-sparc-168402.pdf

Linux

This section describes parameters applicable for SuSE Linux and Redhat kernel 2.6.

Mount option

This mount option can be used, to increase performance:

- noatime, nodiratime
  Prevent the access time from being updated unless the access involves a modification of a file's or directory's metadata or content. Avoiding the writes associated with updating the access time can result in measurable performance gains. If you use the option, the last access time for a file cannot be determined. To disable the access time use the following mount options:

```
# mount -o noatime,nodiratime <mount point>
```

Connections

By default, no more than 32 connections to the xinet deamon can be made on Linux. When connecting more than 32 users, rexec login failures will occur. To increase the number of connections, edit the file /etc/xinetd.conf. The "instances = 32" must be replaced with for example "instances = 1000". After that, restart the xinetd using "/etc/init.d/xinetd restart":

```
instances       = 1000
```

Kernel parameters

Kernel parameters are important for performance. To list the kernel parameters, use this command:

```
# sysctl -a
```

To set a parameter, use the “sysctl –w” command. For example:

```
# sysctl -w kernel.shmmax=2147483648
```

To make a change permanent, add a line to the file /etc/sysctl.conf in the format:

```
<parameter>=<value>
```

The file /etc/sysctl.conf is used during the boot process.

Swappiness

This parameter will influence the priority paging mechanism. After changing this parameter, a process memory page gets higher priority than a file system buffer page.

The threshold when processes should be swapped can be tuned via /proc/sys/vm/swappiness. The default value is 60, which works well if you want to swap out daemons or programs which have not done a lot lately. Higher values will provide more buffer/page cache; lower values will wait longer to swap out idle processes. We recommend changing swappiness to 10, in order to keep the bshell processes as much as possible in memory:

```
vm.swappiness=10
```

Read ahead buffer

To speed up streaming reads, increase the size of the block read ahead buffer for fast storage (SCSI disks or RAID). The default is 128. We recommend increasing the block read ahead buffer for database data disks to 512:

```
# echo 512 > /sys/block/<sdX/hdX>/queue/read_ahead_kb
```

Number of requests

The length of request queue has some influence on the IO performance. It increases throughput at minor latency expense. The default is 128. We recommend increasing the number of requests to 256 with CFQ scheduler for fast storage:

```
echo 256 > /sys/block/<sdX/hdX>/queue/nr_requests
```

Huge Pages

To use larger page sizes for shared memory, Huge Pages must be enabled on Linux which also locks these pages in physical memory. The default page size in Linux for x86-64 is 2MB.

Configure the Linux huge page pool by calculating the number of huge pages required for the database shared memory (e.g. Oracle SGA):

```
num_of_huge_pages = INT(total_db_mem_size / Huge page size) + 1
```

If the database shared memory is 12 GB, use 12 GB Large pages + 1 extra page. For example:

```
num_of_huge_pages = INT(12884901888) / 2097152) + 1 = 6145
```

The following steps allocate 12 GB of RAM as huge pages (2 MB):

1. Set the vm.nr_hugepages kernel parameter to a value a little bit more than used by the database shared memory. In this case, the database shared memory is 12 GB, so decided to use 12 GB and set the parameter to 6145.

   ```
   vm.nr_hugepages=6145
   ```

2. The database userid needs to be able to lock a greater amount of memory. So, the file "/etc/security/limits.conf" must be updated to increase soft and hard memlock values for the database userid. In case of an oracle database, add the following lines:

   ```
   oracle          soft    memlock        12582912
   oracle          hard    memlock        12582912
   ```

3. After setting this up, we need to make sure that the database shared memory (e.g. Oracle SGA) is indeed using HugePages. The value of ((HugePages_Total - HugePages_Free) * 2MB) will be bigger than the size of the database shared memory (or it will equal the shared memory segment shown in the output of ipcs -ma).

   ```
   # grep -i huge /proc/meminfo
   HugePages_Total:  6145             #The size of the pool of HugePages.
   HugePages_Free:   1656             #The HugePages that are not yet allocated.
   HugePages_Rsvd:      0             #The number of HugePages for which a commitment
                                      # to allocate from the pool has been made, but
                                      # no allocation has yet been made.
   Hugepagesize:     2048 kB          #HugePage size
   ```

Caution: Some experts do NOT recommend using Automatic Shared Memory Management (AMM, for example setting *memory_target*) with Linux HugePages. See Oracle note 749851.1 "HugePages and Oracle Database 11g Automatic Memory Management (AMM) on Linux". Also see notes 361323.1 and 361468.1

SHMMAX

Shared memory allows processes to access common structures and data by placing them in shared memory segments. It's the fastest form of Interprocess Communication (IPC) available since no kernel involvement occurs when data is passed between the processes. In fact, data does not need to be copied between the processes.

To see all shared memory settings, run:

```
# ipcs -lm
```

The SHMMAX parameter defines the maximum size in bytes of a single shared memory segment that a Linux process can allocate in its virtual address space.

Since the database shared memory size (for example Oracle SGA) is comprised of shared memory, SHMMAX can potentially limit the size of the database shared memory. SHMMAX must be slightly larger than the database shared memory size to prevent that this will be scattered. We recommend using a SHMMAX size equal to the amount of internal memory.

We recommended that the shared memory fits into the Huge Pages pool, see Huge Pages section of this chapter.
To determine the current maximum size of a shared memory segment, run:

```
# cat /proc/sys/kernel/shmmax
2147483648
```

Change the SHMMAX size to the size of the internal memory. For example:

```
kernel.shmmax=34359738368
```

SHMALL

The SHMALL parameter sets the total amount of shared memory pages that can be used system wide. Therefore, SHMALL should always be at least SHMMAX / PAGE_SIZE. The page size can be determined via

```
getconf PAGE_SIZE
```

When the SHMMAX parameter has been set to 34359738368 and the page size is 8192, the SHMALL parameter will be 4194304:

```
kernel.shmall=4194304
```

Microsoft Windows

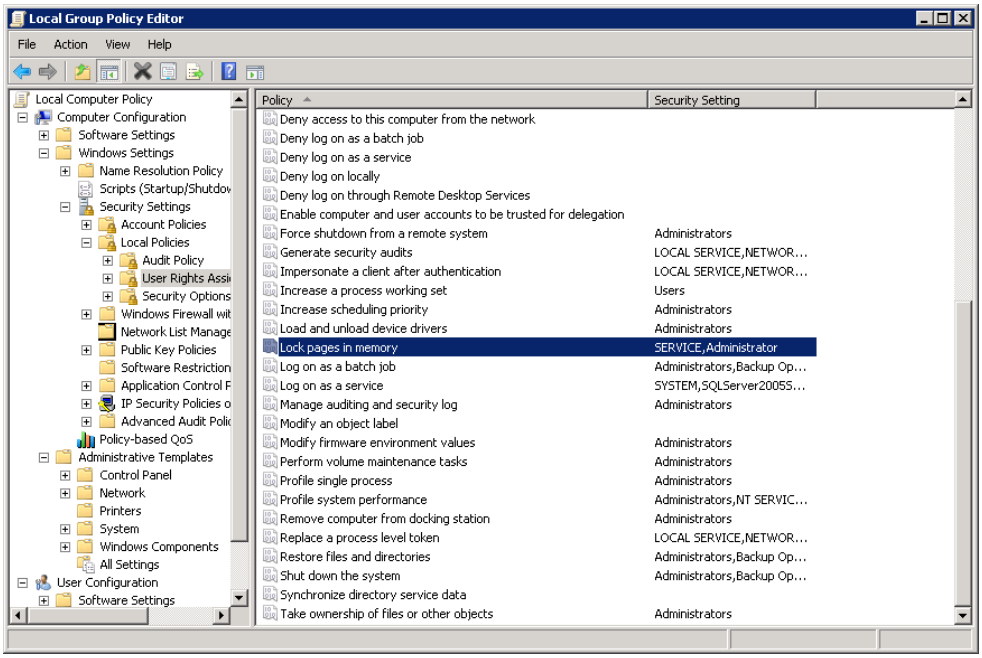
This section describes the parameters to change to optimize the performance of the Infor LN application on Microsoft Windows 2008 (R2) 64-bit.

Large pages and Lock pages

To use large pages on Windows, assign the "Lock pages in memory" privilege to any user that runs the database. This includes service accounts and administrators.

Take these steps:

1. Select Control Panel -> Administrative Tools -> Local Security Policy or start gpedit.msc
2. Select Local Policies -> User Rights Assignment



3 Double click "Lock pages in memory".

4 Add users and/or groups.

5 Reboot the machine.

Desktop Heap Size

When you run a large number of Windows programs, including bshells, "Out Of Memory" error messages may appear when you attempt to start new programs or try to use programs that are already running. In the Windows System Event viewer the following warning message from the “Win32k” source may appear: “A desktop heap allocation failed.”

This error can occur if the desktop heap in the WIN32 subsystem is depleted; even though you still have plenty of physical and page file memory available.

The following symptoms have been notified:

- Connection problems to the Infor LN server or MS SQL Server
- Popup messages on console from **sort.exe** with message “**User32.dll or Kernel32.dll Fails to Initialize”**
- Windows background services simply die for no obvious reason

To resolve this problem, there are several options, which are described in more detail in InforXtreme solution 941132. From porting set 8.6a.03 onwards running on Windows 2008, the following options can solve the problem:

1. Use combo-db drivers.
   Using combo drivers (Refer to Chapter 4) can reduce the needed Desktop Heap up to 50% and will increase performance.
2. Set "Baan Windows Desktop Size" on Logic service in Infor Manager snapin.
   This setting defines the size specific for the “Baan Windows Desktop”. 8192 is a good starting value. Increase the value if the issue still occurs.
3. Modify the non-interactive desktop heap size. This should be considered only if above option did not resolve the issue.

   Windows maintains registry settings to control the heap size for desktops associated with non-interactive window stations. To increase the non-interactive desktop heap size, take these steps:

   a. Click **Start**, type regedit in the **Search** box,
   b. Click **regedit.exe** in the **Programs** list.
   c. Type your password if you are prompted for an administrator password or for confirmation and click **Continue**.
   d. Locate and click the following registry subkey:

      ```
      HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Control\Session Manager\SubSystems
      ```

   e. Right-click the **Windows** entry and select **Modify**.
   f. In the **Value data** section of the **Edit String** dialog box, locate the **SharedSection** entry, and increase the third value for this entry. The value is in kilobytes (KB). You must increase this value into steps of 512, 1024, 2048, 3072 or 4096, where 2048 is a good starting point. Example:

      ```
      SharedSection=1024,20480,2048
      ```

   g. Restart the system.

Notes:

1. The first SharedSection value (1024) is the shared heap size common to all desktops. This includes the global handle table, which holds handles to windows, menus, icons, cursors, and so forth, and shared system settings. It is unlikely that you would ever need to change this value.
2. The second value of the SharedSection registry entry is the size of the desktop heap for each desktop that is associated with an interactive window station. The heap is required for each desktop that is created in the interactive window station (WinSta0). User objects like hooks, menus, strings, and windows consume memory in this desktop heap. It is unlikely that you would ever need to change this second SharedSection value. 64-bit Windows has a default interactive desktop heap size of 20MB. We do not recommend that you set a value that is over 20480 KB.
3. The third SharedSection value is the size of the desktop heap for each desktop that is associated with a "non-interactive" window station. If this value is not present, the size of the desktop heap for noninteractive window stations will be same as the size specified for interactive window stations (the second SharedSection value).
   Because the non-interactive desktop heap is mapped into the address space of each and every process', this value should not be set to an arbitrarily high value, but should only be increased

sufficiently to allow all the desired applications to run. Note that many server side applications will spawn multiple processes for each user request.

For more information and solutions when running on an older Windows version or porting set see:

- InforXtreme solution 941132
- http://blogs.technet.com/b/askperf/archive/2007/07/24/sessions-desktops-and-windows-stations.aspx
- http://support.microsoft.com/support/kb/articles/Q142/6/76.ASP
- http://support.microsoft.com/support/kb/articles/Q126/9/62.ASP

Chapter 4 Tuning Infor LN

4

This chapter describes the most important settings and parameters for performance in Infor LN, the place to store them, and the effect of various values.
The database dependent settings are described in the database specific Performance, Tracing and Tuning Guide.

Combo bshell

Formerly, the bshell and database driver ran as separate processes. Nowadays, when using the combo bshell the database driver is linked as a library in the bshell. The combo bshell has been developed for performance reasons as it saves an additional process and extra communication between these processes. In the former situation expensive context switches between bshell and database driver a significant amount of CPU was spent on system activity. Running in combo mode will gain more than 10 percent CPU for OLTP and even more than 20% for batches.

To enable the combo mode, an extra line must be added in the lib/ipc_info file. The ipc_info file contains:

- The driver name (first argument)
- Reference to combo (second argument)
- The library name (third argument)

To enable combo mode for Oracle on UNIX the ipc_info file must contain:

```
oracle8         d  ora8_srv6.2
oracle8         s  ${BSE}/bin/ora8_srv6.2
```

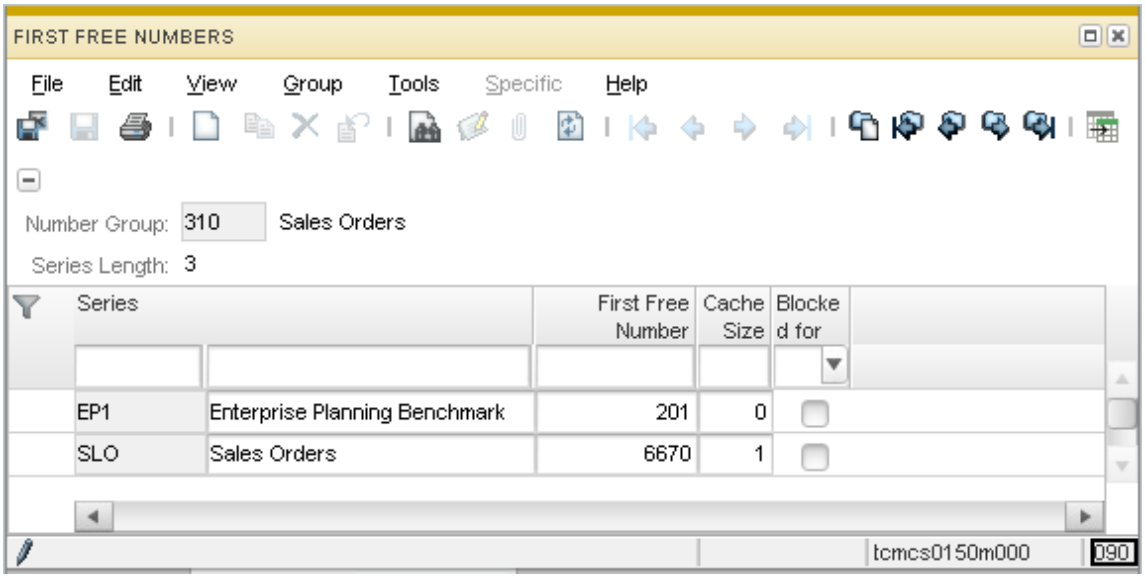
To enable combo mode for Oracle on Windows the ipc_info file must contain:

```
oracle8         d  ora8_srv
oracle8         s  ${BSE}\bin\ora8_srv.exe
```

It is important to set the line containing the “d” option before the line with the “s” options. The first Oracle database driver makes sure that the bshell starts in combo mode. If for some reason another database driver needs to be started, it will be started in non-combo mode as only one database driver can be started in combo mode. In normal situations only one database driver will be started, but when the table definitions are modified to support for example multiple databases this could be the case.

First Free Numbers

First Free Numbers (FFN) is used for all kind of sequences in tables such as order numbers, invoice numbers, contract numbers etc. These numbers are setup per Number Group and series Length. The example below shows the series EP1 and SLO for Number Group 310 and series Length 3. The last used number for the EP1 series is 201 and the First Free Number is 201. The EP1 series do not use caching; the SLO series uses caching.



Cache for First Free Numbers

When a program needs a new first free number (FFN), it reads the current value, increases this with 1, updates the record in the database and when the transaction is committed, the new number can be used by the next program that needs a FFN. The time between the update of the FFN and the commit, this record is locked for other users/programs. This can become a bottleneck as some programs need some time (read do some work) before the transaction can be committed.

From Baan 5 and higher, the session has been enhanced with a cache. Setting the cache to 1 or higher, enhances the program that a FFN is read, updated and directly committed in a separate process. This shortens the lock on the FFN table and prevents locking issues on this table. The size of the cache determines how many numbers will be cached for this particular process.

Cache settings

Setting the cache size to 0 (zero) the application behaves as normal. The lock on the FFN table stays open until the commit. No gaps will occur in the sequence numbers, and the sequence is ordered on time.

Setting the cache size to a value of 1 or higher, will increase the FFN record with the cache size and directly commit this value in a separate process. The calling program receives the number(s) that can be used as FFN. When the calling program uses less numbers than the cache size or a transaction is aborted, gaps in the sequences can occur.

Cache size	Description
Cache size = 0	No caching is applied; user request a new number, the number is committed after the entire transaction is completed. Advantage: no gaps in numbers. Disadvantage: number series is locked during the entire transaction, other users cannot request a new number from the same series until the entire transaction is completed. Usage: use this option in case there are no number gaps allowed. Request a new number as close as possible to the end of the transaction to reduce locking time. This setting may introduce performance and locking issues in high volume implementations.
Cache size = 1	User requests a single new number and the number is committed immediately. This happens independently of the completion of the entire transaction. Advantage: number series is locked as short as possible, resulting in less locking and improved performance. Disadvantage: possibility of gaps in the number series in case the entire transaction does not finish. Usage: this is the default value for number series to avoid performance and locking issues in high volume implementations.
Cache size > 1	User requests multiple new numbers and all the numbers are committed immediately. This happens independently of the completion of the entire transaction(s). Advantage: number series is locked as short as possible and the number of updates to the number series is reduced, resulting in less locking and improved performance. Disadvantage: possibility of large gaps in the number series in case the entire transaction does not finish. Usage: this is only recommended for batches in case the cache size =1 setting does not solve the locking issues sufficiently.

Conclusion

When gaps in FFN sequences are not allowed, the cache in FFN must not be used. When gaps in FFN sequences are allowed, it is recommended to set the cache to 1 when multiple users/ process will use that same range. It is only recommended to set the FFN cache size higher than 1 when the cache size to 1 still shows locking on the FFN table. For more information about First Free Numbers, see *Infor LN - Performance Guidelines (U9502 US)*.

Environment variables

To optimize and fine-tune the Infor LN application, some environment variables can be set. For example:

- Set the environment string of the user interface.
- Specify the “-set” option in the user environment.

If certain variables are often used by a user, they can also be specified on a more common place. Depending on the operating system, they can be placed in:

- The $BSE/lib/bse_vars file on UNIX systems.
- The $BSE/lib/defaults/db_resource file.
- The $BSE/lib/defaults/env.shell file on UNIX systems for processes started with parallel processing by the ASCII interface.
- The $BSE/lib/tabledef6.2 file
- The Infor Manager on Windows systems.
- In the Windows or UNIX environment (shell).

The bse_vars file

The bse_vars file can be placed on several places on a system. The search order for reading of this file is as follows:

4. The current directory; for BW interfaces, this will be the home directory.
5. The $BSE/lib directory.
6. The /usr/bse directory. This is an absolute PATH and independent from the place of the $BSE variable.

If a variable is found in one of these files, it will be set with the particular value. The order of searching also specifies the priority for setting the variable. We recommend to set variables for all users in the $BSE/lib/bse_vars file and for specific users in their Worktop/BW command field.

The variables can be set in the $BSE/lib/bse_vars file as a normal environment variable:

```
<Variable>=<Value>
```

Example:

```
SLMHOME=/usr/slm
```

Note: On UNIX from porting set 8.8a.01 onwards, variables specified in the $BSE/lib/bse_vars file, and already existent in the environment, are now overwritten by the value specified in the bse_vars when connecting via a client using rexec or blogin protocol. The value of the variable in the bse_vars file has precedence over the (previous) value of the environment variable. With older porting sets, the value in bse_vars was silently ignored if a variable was already specified in the environment (of the rexecd or blogind process). Variables in the bse_vars file, which are not already set as environment variable, are always added to the environment, similar to previous porting set versions.

With this new behavior, it is possible to override the PATH variable. Note that values need to be specified as fully qualified values, i.e. no expansion of its own variable is done. For example, the following construct is invalid: PATH=$PATH:/home/user/mybin, but PATH=/home/user/mybin is valid.

The db_resource file

Several variables can be specified in the db_resource file. If your operating system is UNIX, the db_resource file is located in the $BSE/lib/defaults directory.

On Windows platforms, the db_resource file is located in %BSE%\lib\defaults. The variable name for the db_resource is identical to the environment variable but specified in lowercase.

To separate variable and value, a colon ':' must be used, as follows:

```
dbslog:1000
```

The tabledef6.2 file

The $BSE/lib/tabledef6.2 file can contain environment variables used for every bshell as well. We do not recommend storing environment variables in this file, but you have to be aware that environment variables set in this file will be used as well.

Infor Manager

If Infor LN is installed on a Windows system, the Infor Manager can be used to set environment variables. The Infor Manager will write the variable in the registry path HKLM\Software\Baan\<BSE>\Environment.

Shell or environment

It is also possible to set environment variables in the shell or the windows environment as well. Normally, these parameters will not be read by the bshell. However, be aware that these parameters will be read by for example the login deamon. Make sure that the environment is clean when starting the login daemon. If the **blogind** is started with certain environment variables, this will result in unexpected behavior, such as failing connections or starting in a wrong company number. All environment variables that are set when blogind is started are inherited by the started programs (bshell, fs and database drivers).

For example, if the BSE_COMPNR variable is set, the bshell starts in the specified company, not in the users default company.

Some precarious variables are:

- BSE_COMPNR
- BSE_USER, USER
- PACKAGE_COMB
- ORA_USE_VARCHAR

Environment parameters

The following environment variables are important for performance:

- BAAN_SQL_TRACE
  See the paragraph about tracing with BAAN_SQL_TRACE. This variable can also be specified in the db_resource file.
- BDB_ALWAYS_FLUSH
  See the paragraph about Call Graph Profiling.
- BDB_DRIVER
  When a database driver other than the driver specified in the table definitions is needed, you can specify the new one with the BDB_DRIVER variable. By specifying this variable, the table definitions will not be read from file. The tables that need auditing will not be found anymore. We recommend using this variable only for test and tracing purposes.

  Use this example to test a new database driver other than the one installed and specified in the ipc_info file:

  ```
  BDB_DRIVER="oracle8_optim(SQL_TRACE=TRUE)"
  ```

  Another example is when a certain user wants to start the driver on a remote system:

  ```
  BDB_DRIVER="db-remote"
  ```
- BSE_TMP
  Some temporary Infor LN files can become large. It is important that performance critical sessions have their BSE_TMP set to a file system that is not too slow, such as the following:

  ```
  BSE_TMP=$BSE/tmp
  ```
- BSE_SORT
  The sorting files are always large and on a disk which is not too slow, for example:

  ```
  BSE_SORT=/u2/sort
  ```
- DBSLOG
  See the paragraph about tracing with DBSLOG. This variable can also be specified in the db_resource file.
- DBSLOG_LOCK_PROF
  The DBSLOG_LOCK_PROF parameter specifies the minimum duration of a lock that must be logged. Any locks of shorter duration will not be logged. This variable specifies the minimum number of seconds, to a precision of milliseconds, which must elapse before a lock is logged. Lock time is calculated as the time from when the first record in a transaction is locked to the

time of the commit or abort. This is the longest time a record remains locked during a transaction. Note that the appropriate dbslog categories must be set, for example:

```
DBSLOG_LOCK_PROF=2.0
```

This variable can also be specified in the db_resource file.

- DBSLOG_NAME
  The DBSLOG_NAME parameter specifies the output of the DBSLOG file, such as the following:

  ```
  DBSLOG_NAME=/bigfiles/locking.log
  ```

  This variable can also be specified in the db_resource file.

- DISABLE_DEBUGGER
  This parameter prevents 4GL debug screens and continues when a session is compiled in debug mode. It has the same meaning as the '-nodebug' option that can be added in the BW configuration screen, such as the following:

  ```
  DISABLE_DEBUGGER=1
  ```

- MAX_RETRY
  The MAX_RETRY parameter sets the maximum number of retries for a transaction. Each time the transaction fails and jumps back to the db.retry point, the number of retries is increased with 1. This will go on until the value of MAX_RETRY is reached or the transaction succeeds. If the number of MAX_RETRY is reached, the session will stop with an error message. The default value is 10 and must be sufficient. If an error shows that 10 retries are insufficient, the variable MAX_RETRY can be increased. However, it is more important to check the applications and settings as LOCK_RETRY and TEST_RETRY. For example:

  ```
  MAX_RETRY=20
  ```

  From porting set 8.4a onwards this parameter can also be used as db_resource setting.

- <DB>PROF
  See the paragraph about tracing with <DB>PROF. This variable can also be specified in the db_resource file.
- PROFILE_ALL
  See the paragraph about Call Graph Profiling.
- PROF_CLIENT
  See the paragraph about Call Graph Profiling.
- PROF_DIR
  See the paragraph about Call Graph Profiling.
- PROF_RTIME
  See the paragraph about Call Graph Profiling.
- SQL_TRACE
  See the paragraph about SQL_TRACE. This variable can also be specified in the db_resource file.
- TEST_RETRY
  The TEST_RETRY variable specifies how often the system must go back to a retry point at the moment of committing. This variable must not be enabled in a live environment to avoid performance and locking problems, such as the following:

  ```
  TEST_RETRY=3
  ```

In this example, at the moment the commit is hit, the driver jumps back to the retry point without committing the data. When the commit is hit again, it goes back to the retry point. The third time the commit is reached the actual commit will be done.

- USR_DBC_RES / USR_DBS_RES
  The USR_DBS_RES variable specifies the place of a db_resource file. It only contains a relative path starting from the $BSE environment. An example of this variable can be the following:

  ```
  USR_DBS_RES=lib/defaults/batch_resource
  ```

  The use of this variable is to specify another file other than the default db_resource file for specific sessions/jobs. The difference between the variables is that USR_DBC_RES variable sets the location for the client, and USR_DBS_RES sets it for the server.

Dynamic Scrollbar

Infor LN has a dynamic scrollbar. When a multi-occurrence session is started, a query is run to see how the scrollbar must look like. That query can take a while and can hamper the performance. To determine if this is a problem, a Call Graph Profile or any other query tracing tool can be used. The application query used for determining the dynamic scrollbar can be found by searching for “SELECT count(*):1”. If this query takes a long time, the dynamic scrollbar can be disabled. From Infor Enterprise Server 8.6 onwards a specific session is available containing a list of sessions for which the dynamic scrollbar is disabled. This session can be started via “Maintain parameters” (ttaad0100m000). By default, the dynamic scrollbar is disabled when a session counts 10,000 records or more.

When running the Enterprise Server 8.5 or lower, the environment variable SCROLL_LIMIT=-1 must be used, to disabled the dynamic scrollbar. This environment variable can be set in the bse_vars file and will disable the dynamic scrollbar for all users and all sessions.

Database driver parameters in the db_resource

The list will discuss the database independent performance-related db_resource variables. The database dependent variables are discussed in the database specific performance, tracing and tuning guides.

- rds_full
  The resource rds_full can only be used when running in non-combo mode and sets the maximum number of rows transferred in one block from driver to bshell. A lot of batch sessions use set-oriented queries, but most of the OLTP sessions still fetch 1 row at a time; therefore, the best starting value is somewhere around 2. In most situations you will not see much difference when setting it to 1 or 3, but with many more users on the system the database can become loaded too much by SQL statements or reading too much data by not setting this value correctly. For porting sets older than 8.4b the default value for this resource is 5; from porting set 8.4b and

onwards the default is 2. In the application the value of rds_full can be overwritten to e.g. 100 for a certain query by adding the following code:

```
hint array fetching and array size 100
```

When setting rds_full to a certain value, also look for the value of array fetch; these must be the same for optimal data transport from database to database driver.

For some batch sessions and for the download of data, the value of rds_full can be increased to a higher value. For some batch sessions, values up to 10 can give up to 10 percent performance improvement. For downloading the data, the value can be increased to 100 or higher for best performance.

- dbsinit
  The resource dbsinit specifies database server optimization; the value of this variable must be read octal. Currently, the only valid value is 01, which is the default value from portingset 8.6a onwards, that specifies that an optimistic approach must be used when checking for references in parent tables. The referenced row in the parent table is not locked, which improves the overall concurrency.
- retry_delay
  If the bshell had to retry a transaction, a random time between 1 and 3 seconds of sleep was forced. From porting set 8.4a.02 onwards, the resource retry_delay is introduced as an alternative for the random number. The new behavior is that, per retry, a defined delay time is used. The formula for this delay is as follows:

  Delay time = # of retries * retry_delay.

  The default value for retry_delay is 200 (ms). This default can be overwritten by specifying the retry_delay in the db_resource file. The value should contain a valid number in milliseconds.
- bdb_max_sessions
  The resource bdb_max_sessions defines the number of sessions per driver. If any driver has reached this threshold, a new driver will be started to handle any new sessions. We recommend keeping this variable to the default value (0) to prevent additional memory consumption.
- bdb_max_session_schedule
  The resource bdb_max_session_schedule indicates the number of sessions that will be kept open as a minimum in the database-driver. Every time a session is closed, the driver will look to the value of this parameter and decide if a session needs to be kept open. Specifying this value too low will cause additional overhead during closing and starting of a session, but specifying it too high will cost too much memory. Depending on the configuration, use, and requirements, this variable can be set to a certain value. Usually, the value of 3 is sufficient, but for batch sessions that switch between jobs, it is advised to increase this value. An example is as follows:

  ```
  bdb_max_session_schedule:250
  ```
- mle_join_type
  The resource mle_join_type determines the type of join used in case of a Multi Language Enabled environment between the data table and the corresponding shadow table with translations. The default join type is INNER. It has been discovered that with more than 5 languages SQL Server 2005 generates not optimal execution plans, despite the FIRST ROW hint. This was corrected by using an LEFT join type between the tables. This resource is implemented for MLE tables for all databases; however it should be applied after extensive

testing. The default join type (INNER) has been found the most efficient under normal circumstances. The following values are supported:

- 0: INNER (default)
- 1: LEFT
- 2: RIGHT
- 3: FULL

Driver parameters in the database definition

You can set all types of parameters in the session “Database Definitions” (ttaad4510m000). Most of the variables specified there can also be set in the db_resource.

We recommend you checking if variables can be placed in the db_resource to keep maintenance easy and have a fast overview about the various used parameters.

After making changes in the database definition, you must convert the data to runtime.

Driver parameters in storage file

There are two parameter files that influence the behavior of the database driver:

- The storage parameter file.
- The driver parameter file.

Database storage and driver files

Database storage files contain parameters that influence the creation of tables and indexes. The contents of these files differ per database, therefore the content is explained in the database specific performance, tracing and tuning guide. Only performance topics on this are discussed. For more information about these files, consult the driver manuals.

Auditing

Infor LN specifies some tables to be audited, which are mostly parameter tables; however, in most situations other tables also need to be audited. Auditing always costs performance, but in certain situations it can become a performance bottleneck. It is therefore recommended to minimize the number of tables to be audited. Auditing can be specified in session “Tables by Database” (ttaad4111m000).

Besides Infor LN auditing, it is possible to use database auditing. For performance reasons, the number of tables audited by the database must be kept to a minimum.

Shared tables

It is possible to share tables across companies, which can be specified in session “Logical Tables” (ttaad4120m000). During the conversion of runtime of this session, the data is stored in the $BSE/lib/compnr6.2 file. When starting a new database connection, the driver reads the content of this file to see where all tables are stored. The larger this file, the longer the startup of the driver will be and the longer the preparation of some queries will become. For administrative and performance usage, it is advised to load the shared tables as much as possible in modules instead of using specific tables. For example:

```
tcemm:500:501,502,503,504,505
tcfin:500:501,502,503,504,505
```

Database authorizations

In Infor LN, it is possible to use the following types of database authorizations:

- Company authorizations (ttams3144m000)
- Table authorizations (ttams3142m000)
- Field authorizations (ttams3143m000)
- Table data authorizations (ttams3145m000)

The company, table and field authorizations are checked in the bshell/driver, but table data authorizations will be checked in the database. Table data authorizations will be added to the query and will therefore have an impact on the performance; usually, the performance will not be optimal. Table data authorizations can be split into 2 categories:

- Table data authorizations being part of an index. The performance can be acceptable if the table data authorizations are only on the first field of the table and the table is most often searched via a certain index.
- Table data authorizations not being part of an index. These authorizations are potential performance bottlenecks. If the table is small (less than 100 rows), it usually does not affect the performance, but if the table is large the impact can be huge. Note that tables are relatively small when created, but will grow over time. Therefore, it is important to prevent table data authorizations as much as possible, even when tables are small.

The general rule for authorizations is that table data authorizations must be kept to a minimum to prevent performance problems.

Shared memory usage

Introduction

Infor LN uses shared memory for sharing frequently used data definitions, objects, and reports. It saves performance and memory because all shared objects do not need to be loaded in private memory of each user again.

Shared memory block size

It is advised to make one block for all shared memory data. All data will be loaded in shared memory after the rc.start command (by the srdd_init –i command) so it is easy to verify how much memory is needed. The amount of shared memory in use will be shown with this command.

```
$ shmmanager6.2 -s
SEMID 945818141
Bshell common ptrs :
Reserved                0x0
Objects                 0xa0000000001e088 version 0x08070104 (caller is using dynamic
shm directory service)
not used                0x0
not used                0x0

not used                0x0
Security Files          0xa00000000000930
Compnrs                 0xa00000003b1ecb8
Db driver DD            0xa00000003b2ef70

Shm Flags               0xa0000000001b258 version 0x08070104 flags 0x00000001
Date Currency           0xa000000000019d0 mtime: Tue Apr 20 11:03:05 2010
not used                0x0
Bshell Cmd              0x0

Audit SeqNr             0xa00000003b56f00
Options                 0xa000000000004b8
not used                0x0
Shm Timer               0xa000000000004a8

      DESCRIPTION OF SEGMENT TABLE
USED BYTES 62222120 FREE BYTES 4886744 SHMID 31460072 NO ATTCH 2
NODE a00000000000000 SEG [0] ATCH_ADDR a00000000000000 FREE_ADDR a00000003b56f28
```

This example shows 1 segment of 64Mb, which is the recommended and default value. The segment size can be set by specifying shm_segment_size in the $BSE/lib/shm_config file. This file can be used to override programmed defaults for shared memory. We recommend using the defaults. Setting values here is only needed when attaching to shared memory fails. The size of a block is equal to the sum of USED BYTES and FREEBYTES.

Package combinations in shared memory

The most important use for shared memory is to store the database definitions of a package combination. The package combinations that must be loaded in shared memory are specified in session “Package Combinations” (ttaad1520m000). It is recommended to load all package combinations linked to a user and company in shared memory.

Objects in shared memory

To reduce memory usage and improve performance, it is possible to load some objects in shared memory. Infor LN offers the possibility to load frequently used objects into shared memory by measuring the used objects during a period of time. See the *Infor LN Administration Guide (U8854 US)* for more details.

Reports in shared memory

For reports, the same rule applies as for other objects: when opened and closed often, it is advised to place these objects in shared memory. Reports can be tracked and filled in the same way as normal objects are. At least the top 25 of reports must be loaded in shared memory.

Porting set

The porting set is the set of executables located in the $BSE/bin directory. Besides bug fixes, performance enhancements are made in the porting set. Therefore, it is recommended to use a recent porting set.

Varchar

With older porting sets all strings where stored with fixed length in the database. Nowadays it is possible to use also the varchar datatype in the database. This reduces data size and data growth significant. It also increases performance, as more data can be kept in the database buffer cache.

For IBM DB2 the varchar data types were already introduced in the past.

From 8.4b porting set release, the Infor LN Microsoft SQL Server database driver will use VARHCAR and NVARCHAR data types as standard.

Since porting set 8.6a you can also use VARCHAR2 and NVARCHAR2 for Oracle. However, data structures need to be changed: all tables containing strings must be recreated. The usage of varchar in Oracle needs to be specified with the resource ora_use_varchar:1

Standard Program

The standard program takes care of a lot of functionality for standard functions; these functions are used in all Infor LN sessions. Performance improvements in the standard program have an impact on all sessions. Because the standard program is updated frequently and performance improvements are also done regularly, it is recommended to use a recent version of the Enterprise Server, which includes the standard program.

Log files

The log files indicate the environment healthiness. The more problems that occur during a day, the more likely it will be that performance can be improved by reducing these errors. The most important files are the bshell and database driver log files. We recommend checking the log files and event viewer on Windows systems on a regular basis.

Chapter 5 Batch Performance Optimization

5

This chapter describes what settings can be changed to increase the speed of Infor LN batch processes. OLTP users must not experience performance degradation by settings needed for batch processes. Therefore, the system can be optimized for OLTP users and for batch sessions, but the special settings must be made separately.

This chapter also describes how batch performance can be increased using the so-called performance boosters and how they can be used and tuned. Currently, the following performance boosters exist:

- Parallel processing
- Table boosters
- Prevent progress indicator

Besides these boosters, other configuration options exist to improve the batch performance.

Performance Boosters

Parallel processing

Parallel processing is a mechanism in the application to speed up a batch session by spreading the load over multiple bshells.

Parallel processing is programmed in some batch sessions and is based on the application logic of tccomdll0200 for older applications and in the standard tools set for current versions. The new version is introduced in Infor LN FP7 and is back ported to Infor LN FP3. In the new version, a controlling bshell is no longer needed.

Parallel processing has the advantage that the duration of a batch process can be lowered drastically and the throughput can be increased. The disadvantages of parallel processing are:

- Tracing and debugging is difficult.
- Contention on resources increase.
- Increased possibility for failures.
- Additional consumption of resources, such as CPU, memory.
- Additional Infor LN licenses.

We recommended that you implement and use parallel processing with care. The following sections describe several aspects of implementing and tuning parallel processing.

Preparations

Before testing parallel processing, we strongly recommended that you test the batch sessions without parallel processing and check whether there are bottlenecks. For example, if disk I/O is already a bottleneck, parallel processing will not improve your system's performance.

Configuration

Parallel processing is implemented in only a few sessions. To find out if a session has parallel processing capabilities, the session must be run once. If the session has parallel processing capabilities, running the session will generate a record in table ttaad720 in the application company. The settings for parallel processing are in the **Parallel Processing Configuration** (ttaad7520m000) session. This session shows all the sessions containing performance boosters. Search this session for the session you want to use for parallel processing. The **Mode** field is set to **Not Active**, which is the default value of the performance configuration. The session has the following options:

- **User:** If a user is specified in this field, the values are valid only for that particular user. If this field is left blank, the values are valid for all users. When parallel processing searches for settings, the user-dependent settings are read before the common settings. To specify settings for a user, you must add a record. This is the only valid reason to add a record to this session; any other reason can lead to confusions for the user, because they will not be recognized by the sessions.
- **Servers:** In all situations, the session starts in parallel only when the value in the **Value** field is **2** or more. This value specifies the number of bshells that will be started to run the session in parallel.
- Mode: Parallel processing can be activated by specifying one of the following values:
  - **Not Active**: Parallel processing is not active for this session.
  - **Manual**: If a session is not started from a job, the session will start in parallel.
  - **Job**: If a session is started from a job, the session will start in parallel.
  - **Manual & Job**: The session will always be started in parallel.

After you specify a value, you can run the session in parallel.

Tuning

In general, due to the single threaded architecture of Infor LN, each batch session consumes a processor core. Therefore, if a system contains 8 cores and at least 2 cores are needed for OLTP users, you can set the number of parallel **Servers** to **6**. When running in a 3-tier configuration, you may want to set a higher value because the load will be distributed over the application server and database server. To find the optimal number of servers, we recommend that you run the same batch several times with several values for **Servers**. As a rule of thumb, 50% of the batch processing is

done by the bshell (including database driver) and the other 50% will be processed by the database and network.

After using parallel processing for a session for over a year, we recommend that you check whether the number of servers is still optimal for that configuration, because the dataset and other settings might have changed. If you change the system, for example, decrease or increase the number of cores, reconsider the settings for parallel processing.

Troubleshooting

A batch session with parallel processing can fail. Generally, an error message is displayed. Sometimes more information is required. In that case, you can use the log.parbshell file. Alternatively, the environment variable TRACEPARBSHELL can be used. This variable provides additional information in the log.parbshell file depending on its value:

- 0: no tracing
- 1: trace API calls
- 2: trace communication
- 3: trace API calls and communication

The construction of a trace line is:

```
[date time] <user>:<session>(pid), <component> (bshellpid): <message>
```

A <component> is one of the following:

- ClientAPI
- ServerAPI
- SupportAPI
- ServerCOM
- ClientCOM

Debugging

When even more in-depth analysis is required and the sources must be debugged, the following steps must be completed:

7 Compile all the required sources in debug mode.

8 Start a maximum of two servers to prevent too much work on all debug screens.

9 Set the following option in the startup process of the default BW:

```
-set APP_DS=bw
```

With this option, the BW interface and a debug screen are displayed for every new connection.

10 Start the client bshell and start debugging.

Documentation

Further documentation about parallel processing is available in the Infor LN programmer's guide, version 7.6.0 3751 or later.

Table boosters

Table boosters are programmed to cache table data. Reading from bshell memory is much faster than reading from database or disk. In many sessions, lots of redundant data is read. For example, when items with their units must be read, in many cases the unit will be the same and does not need to be read again. With the help of tccomdll0201, a whole tools set is prepared to cache table data.

Configuration

After running a session, its configurable table boosters are shown in session tcmcs0598m000 "Table Boosters" (tcmcs0158m000 for Baan IV). This section describes the configuration options.

User

If a user is specified in this field, the values are only valid for that particular user. When this field is cleared, the values are for all users. When looking for settings, the user-dependent settings will be read first, then the common settings. To specify settings for a user, a record must be added. This is the only valid reason to add a record in this session; any other reason can lead to confusions for the end user because they will not be recognized by the sessions.

Load option

The load option specifies how records will be read:

- **Incremental**: A record will first be read from memory; if not available there, it will be read from the database and stored in memory. No index is generated to quickly find a record, and each record will be sequentially scanned in memory.
- **Full**: The table will be read completely in memory when the first record is needed. An index is also generated to find a certain record quickly.

Max No of Rows

The max no of rows option specifies the maximum number of rows to be read in memory. The bshell has a maximum size of 30Mb per table (20 Mb for Baan IV). The maximum memory that is default used is 5Mb per table booster. To increase this, the environment variable TCCOMDLL0201_MAX_ALLOC can be used. The value of this environment variable has to be given

in bytes. Default value will be 5.120.000 bytes per table booster. An example will set the maximum to 6 Mb:

```
-set TCCOMDLL0201_MAX_ALLOC=6144000
```

Booster valid

Parallel processing can be activated by the changing the “Booster Valid” option into one of the following:

- **Manual**: If a session is not started from a job, it will use table boosters.
- **Job**: If a session is started from a job, it will use table boosters.
- **Manual and Job**: The session will always use table boosters.

Tuning

For tuning the table boosters, the size of the table and the usage of that table must be known; this is to find out if incremental or full load must be used. The advantage of incremental load is that only the needed records are placed in memory. The disadvantage of incremental load is that no index can be generated. This can harm if more than approximately 400 records are read. With this knowledge, the following rule of thumb can be used to identify the settings for a table booster.

Only a few records from a table are read

If only a few records (less than 50% with a total of less than 400) are read, the **Incremental** option must be used.

More than 400 records need to be read

Do not use the incremental option because the average access time from memory on this amount of records without index is slower than database access.

If more than 400 records must be read, and the table is large, it is advised to calculate the average number of times this table will be read compared to the number of records the table contains.

If the number of times the table will be read is much higher than the number of records in the table, it is recommended to load the table complete in memory by specifying the full option.

Most records from a large table are read multiple times

If all records (or more than 50%) are read multiple times and the table is larger than 400 rows, the “ Full” load option must be used. Ensure a table booster has a limited memory capacity and not shared between users, but claimed for each user that starts this booster.

If none of the above options apply, it is better to keep the table booster disabled because it will not give performance improvements.

The **Max No of Rows** option allows a maximization of the number of rows. If more rows are needed, these will be read from the database. It is better to disable a table booster than reducing the “Max No of Rows” value to save memory.

Troubleshooting

In order to trace DLL tccomdll0201, use these environment variables:

-set LOG_COMDLL0201=1	Logs memory allocation and other technical matter.
-set LOG_COMDLL0201=2	Logs the “idb” calls including the arguments. So, when turning on one or more table boosters, this trace option tells you what functions have been used and what the arguments and table are.

Prevent progress indicator

Most of the batch sessions within Infor LN are equipped with a progress indictor showing the part of the workload that must be processed is already completed. For automatically started batches, and batches without interface, this progress indicator does not make sense. Therefore, it is advised to disable the progress indicator because the progress indicator also consumes CPU power. The progress indicator consumes additional CPU because of the following:

- Before a progress indicator can be started, the number of records to be processed is counted.
- The interaction between bshell and UI takes time for every update of the progress indicator.

Disabling of the progress indicator can be done in tcmcs0597m000 “Performance Boosters”. If a session has the option **Prevent Progress Indic**, it can enable and disable the progress indicator. The following fields can be set:

- User. If a user is specified in this field, the values are only valid for that particular user. When this field is left blank, the values are for all users. When looking for settings, the user-dependent settings will first be read, then the common settings. To specify settings for a user, a record must be added. This is the only valid reason to add a record in this session, because any other reason can lead to confusions for the end user because they will not be recognized by the sessions.
- Value. This field has no meaning for the progress indicators.
- Booster valid. Parallel processing can be activated by the changing the “Booster Valid” option into the following:
- Manual. If a session is not started from a job, it will prevent the progress indicator.
- Job. If a session is started from a job, it will prevent the progress indicator.
- Manual and Job. The session will always prevent the progress indicator.

Environment and driver parameters

The following parameters may need to be checked for optimal batch performance:

- bdb_max_session_schedule
- cursors
- array size (fetch and insert)

Consult the database specific performance, tracing and tuning guide for more information about these parameters and the value to be used for batch sessions.

Batches can be started easily with different values by specifying the variables in the command line, or redirecting to a separate db_resource file with the USR_DBC_RES and USR_DBS_RES environment variables.

Network

When running batches in a 3-tier configuration, the network is an important factor, which may delay batches by a factor 2 to 3 compared to running the batch in hostmode.

To improve the batch performance, consider the following:

- Run the batch on the database server. A disadvantage of running a batch on the database server is that a (partial) BSE environment on the database server also must be created and maintained.
- Disable interrupt moderation on the network card. To reduce the number of interrupts, many NICs use interrupt moderation. With interrupt moderation, the NIC hardware will not generate an interrupt immediately after it receives a packet. Instead, the hardware waits for more packets or a time-out to expire before generating an interrupt. The hardware vendor determines the specific algorithm for packet size and time-outs.

The measured roundtrip time for a packet is one of the most commonly used techniques to determine the network bandwidth between two endpoints. However, when interrupt moderation is enabled, receiving a packet does not generate an immediate interrupt and therefore the perceived roundtrip time for a particular packet becomes larger than the average time. The Infor LN batches suffer from interrupt moderation, while OLTP users mostly take advantage of it. Consider to disable interrupt moderation when batch duration is a problem. Benchmarks showed that the batch duration can be increased up to 50% when disabling interrupt moderation on the network card. Consult the network card vendor documentation for information about how to disable interrupt moderation. Note that the CPU usage might increase. We strongly recommend verifying the proposed changes in a test environment before moving to production.

Chapter 6 Tracing Infor LN

6

When performance problems occur, it is sometimes necessary to find the exact cause of the problem. If the performance problem is inside an Infor LN session, tracing can be used to identify what happens. Tracing can be performed on several levels.

The following list contains the most frequently used tracing methods for performance purposes:

- Call Graph Profiling

  The Call Graph Profiler (CGP) can be used to find functions and application queries that take a long time. The CGP has call graph functionality that shows relations between parent and child functions.

- BAAN_SQL_TRACE

  The output of this trace shows the Infor LN queries; queries from the standard program are also traced. It is difficult to find the most consuming queries without additional tools. Not all DML and DDL queries are shown in this output.

- ORAPROF, MSQLPROF, INFPROF and DB2PROF

  Depending of the sued database any of these variables can be used. When setting this variable, the output can be large, but shows all queries from the driver to the database, including timing information. Because each query is placed on one line, a formatting tool is required for analyzing.

- bsql

  You can use a tool to trace and improve application queries. Using bsql prevents you from running a large session to see if a query has been optimized.

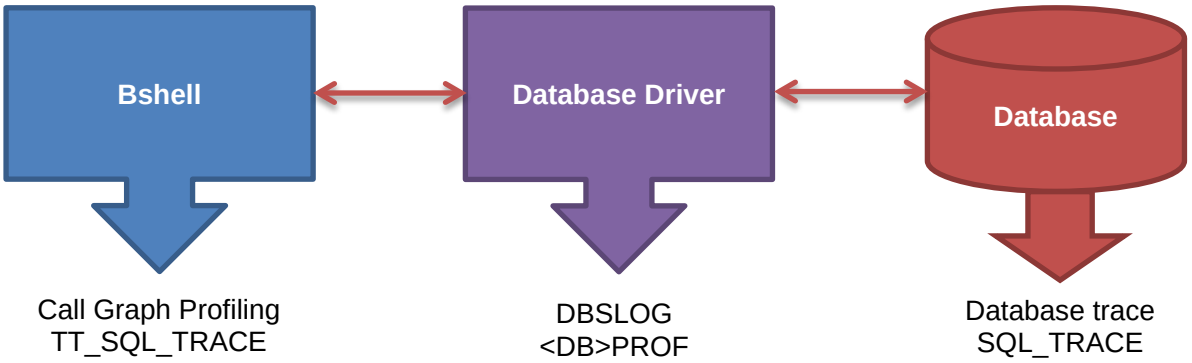
- DBSLOG

  This tracing method has many possibilities. In this document, this method is only used to find input variables for a query that must be reproduced by bsql.

- SQL_TRACE

  This option is an Oracle method to trace all Oracle queries and is described in the Performance, Tracing and Tuning Guide for Oracle (B0078 US). The SQL_TRACE option has advantages above ORAPROF, because the output is shorter and the standard format possibilities are much better. This option shows the Oracle execution plan and consumed resources in Oracle.

Each tracing method is integrated in part of the application. The following figure shows a high-level overview of the components with trace possibilities. The trace possibilities have an effect on each of these components.



Call Graph Profiling

From a bshell perspective, this tool helps to understand where the time is spent.

Call Graph Profiler (CGP) shows the 3GL and 4GL functions and application queries within these functions of the session. CGP also shows the relationship between the functions: how many times functions are called from other functions and how many times other functions are called from a function. For all of this information, timing is added.

Compiling objects in profile mode is not required when using CGP, although compiling objects can help to get function names instead of function numbers when using a 6.1x or 7.1x porting set.

The output of CGP is in HTML format, which makes it easier to navigate and find the performance problems.

Using CGP or any other tools consumes additional CPU power, spent on timing and other resource functions. Usually, a 5 to 10 percent additional CPU will be taken.

The profile output is generated when a session ends normally.

Terminology

To understand the next sections, you must understand this terminology:

- **Utime**: The user time; this is the CPU time that the bshell spent in its own code.
- **Stime**: The system time; this is the CPU time that the operating system is spending on behalf of the bshell.
- **CPU time**: The amount of processing time spent on execution the code in the bshell process. CPU time is the utime + stime. This calculation does not include the time spent waiting for the database or the disk, nor does it include the time spent waiting for user interaction. When running in combo mode, the CPU time also includes the CPU time used by the database driver.
- **rtime**: The amount of run time, sometimes referred to as wall clock time or elapsed time. The rtime is the amount of time that the process requires to execute the code, including the waiting time for databases, disks, user interaction, or other processes.

- **children/parents***:* The term “child,” or sometimes “descendants,” is used for the functions that are called from a particular function. “Parent” is also used for the callers of a particular function.

Features

All sessions are logged, except for some tools sessions that provide no performance information such as the following:

- ttstpmsg. Popup message.
- ttstpdisplay. Print output for device “D.”
- ttstppollmess. Display system message.
- ttstpsplopen. Select print device.
- ttstpsplclose. Print spooler close.
- ttstpoledaemon. OLE Daemon.
- All ttdsk* programs. Menu browser and so on.

The profile output is in HTML format. The output file format is as follows:

```
profile.<pid>.<session name>.<bshell_pid>.html
```

Explanation:

- pid is the operating system process id.
- session_name is the code of the session, such as tccom0501m000.
- bshell_pid is the process id of the session within the bshell. So, when starting the session twice, two trace files are generated, each of which has another bshell_pid.

  By default, the file is placed in the BSE_TMP directory.

Usage

To enable CGP, these parameters can be used:

PROFILE_ALL

This variable activates CGP. If you set to 1, all sessions are traced. If you set this to “tccom” all session codes that contain this string are traced. Multiple strings can be given, such as “tccom,ti.” In this way, all session codes that contain tccom or ti are traced. Normally you like to trace all sessions.

Example:

```
PROFILE_ALL=1
```

PROF_RTIME

If you set this variable to 1, the output contains a section based on runtime next to the CPU ordered section. As of porting set 8.4, this functionality has been enabled by default.

PROF_DIR

This variable can be set to redirect the output to another directory instead of the default BSE_TMP. The supplied directory must exist.

Example:

```
PROF_DIR=/home/bsp/prof
```

PROF_CLIENT

This parameter must be set to an action and a directory.

The action can be “start” or “save” to store the output on a client’s PC and to start the default internet browser to open the profile.

The directory must be a fully qualified directory on the local PC.

Example:

```
PROF_CLIENT=start:C:\temp
```

This option can only be used when running porting set 7.3a or later.

BDB_ALWAYS_FLUSH

This option is introduced in porting set 6.1c.08, 7.1d.09, 7.6a.02, and 7.6b.

In the Infor LN environment, a Data Manipulation Language (DML) such as an insert, update, or delete action is normally delayed until the next query is going to be performed. Just before the next query, the DML action is performed; this keeps the length of the lock caused by the DML operation to a minimum amount of time. When debugging, testing, or tracing with Call Graph Profiler, for example, this behavior can give unexpected results. Unexpected results in a normal environment can be a result of the following:

- When a DML action hits a locked record, the action runs the actual statements on the next query. When that next query is a normal query, it seems illogical if the debugger returns to a retry point on that point.
- Time on a query looks extremely long because a delete operation was done before the query; this delete operation was forced to check a reference in another table. The time for this check of the reference and the delete will be added to the time of the query.
- To force an immediate action on a DML action, the DML action can be extended with a third argument (eflag). Extending the DML action in this way can influence the behavior of the session and asks for additional coding.

  That is why the variable BDB_ALWAYS_FLUSH has been introduced. When setting the variable to a non-zero value, such as BDB_ALWAYS_FLUSH=1, all DML actions are immediately flushed instead of being buffered.

  From porting set 8.4 onwards, this variable has been set by default to the value 1 when CGP is enabled.

Note that the BDB_ALWAYS_FLUSH variable is only intended for test, trace and debugs environments.

bshcmd

You can activate the profiler with the bshcmd command. After triggering bshcmd, only newly started sessions are profiled.

Example:

```
bshcmd6.2 -T "PROFILE_ALL=tdpur" <bshell-pid>
```

Output

The output of the CGP contains the following parts first:

- General trace information
- CPU, wait and run time
- Hyperlink to the query summary
- Hyperlinks to sections sorted on CPU time (user time plus system time)
- Hyperlinks to sections sorted on real time (if PROF_RTIME has been set or porting set 8.4 or later is used)

General trace information

This section, contains information about the session, used system, BSE, company, package combination, 4 GL set, bshell name, porting set, trace date, and a link to some system information.

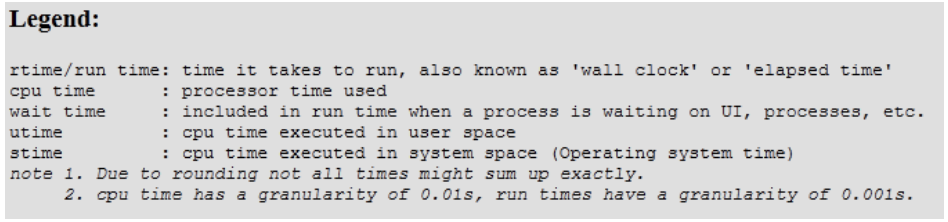
See the general information on the Call Graph Profile screenshot.

When you open the system information file, these settings of various variables are displayed:

The system information is useful for situations in which the behavior of the bshell or database driver is dependent on any of these parameters.

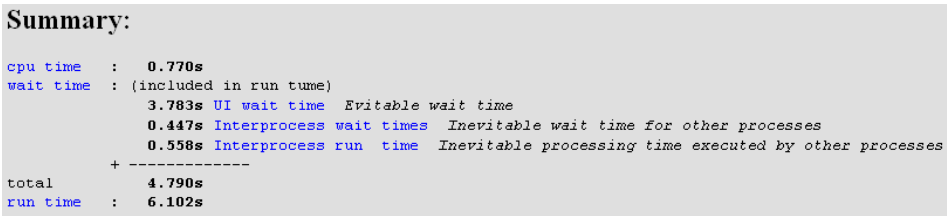
Legend

This section shows the meaning of some parameters, see this figure.



CPU, wait and run time

This figure shows how much CPU time has been spent on the CPU and how long the session ran. The wait time is split into several sections.



The various wait times include the following:

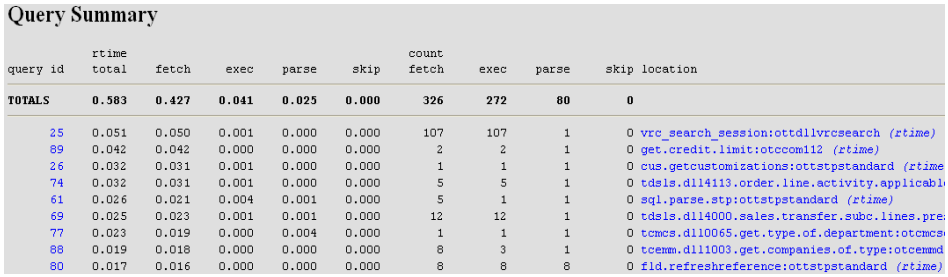
- Bshell waiting for UI events
- Database flush operations such as insert, update, and commit actions
- 3GL inter-process communication such as bms commands
- 3GL process zoom actions such as zoom.to$ commands
- 3GL process sleeps such as brp commands

The 3GL sections also contain hyperlinks to the CGP of the related sessions.

Query summary

The query summary, as shown in the following figure, contains a link to the queries. These queries are sorted in descending order on run time. Here, the output is sorted on run time because the most time-consuming part here is usually in the database driver and database.

Because the total time of the queries is included, a quick investigation can be made to see if it makes sense to look for queries that need to be improved.



Query Summary

query id	rtime total	fetch	exec	parse	skip	count fetch	exec	parse	skip	location
TOTALS	**0.583**	**0.427**	**0.041**	**0.025**	**0.000**	**326**	**272**	**80**	**0**	
25	0.051	0.050	0.001	0.000	0.000	107	107	1	0	vrc_search_session:ottdllvrcsearch *(rtime)*
89	0.042	0.042	0.000	0.000	0.000	2	2	1	0	get.credit.limit:otccom112 *(rtime)*
26	0.032	0.031	0.001	0.000	0.000	1	1	1	0	cus.getcustomizations:ottstpstandard *(rtime*
74	0.032	0.031	0.001	0.000	0.000	5	5	1	0	tdsls.dll4113.order.line.activity.applicabl
61	0.026	0.021	0.004	0.001	0.000	5	1	1	0	sql.parse.stp:ottstpstandard *(rtime)*
69	0.025	0.023	0.001	0.001	0.000	12	12	1	0	tdsls.dll4000.sales.transfer.subc.lines.pre
77	0.023	0.019	0.000	0.004	0.000	1	1	1	0	tcmcs.dll0065.get.type.of.department:otcmcs
88	0.019	0.018	0.000	0.000	0.000	8	3	1	0	tcemm.dll1003.get.companies.of.type:otcemmd
80	0.017	0.016	0.000	0.000	0.000	8	8	8	0	fld.refreshreference:ottstpstandard *(rtime)*

Following the link on the query id shows where the query is parsed in section sorted on utime + stime. On the right is a link to the rtime section; this link is shown as rtime in blue at the end of the line. In this case, following the link of query id 89 shows the query text, as shown here:

```
select   tccom112.crlr:o.credit.limit
from     tccom112
where    tccom112._index1 = {:i.itbp}
and      tccom112.cofc = ""
as set with 1 rows
```

If you search for query id 89, a sql.exec and a sql.fetch section are displayed. The parse shows the application query; in the other sections, only timing is shown for these areas. The sql.parse, sql.exec, and sql.fetch sections are not 3GL or 4GL sections, but are the states of a query on the bshell. During the parse, the query is parsed. During the exec phase, the query execution plan is generated in the database; during the fetch, the result of a select query is retrieved. It is possible that the query is opened in function ”A,” and records are fetched in function ”B.” For example, this can be done by using dynamic queries. To find the place of dynamic queries, you can use the Find option in your Internet browser.

In porting set 7.2 and later, a query cache is available in the bshell. This query cache results in a different way of query representation: In the new way, a query is reused when hard closed, but this is only possible with dynamic queries.

Object summary

The object summary, shown in the following figure, shows all used objects and the amount of time spent in this object. The object summary is sorted on CPU time in the utime + stime section, and sorted on run time in the rtime section.

run time per object

object	run time	cpu time	perc	perc_t
ottstpstandard	**4.308**	0.320	70.6	70.6
ottstp_stddll	**1.038**	0.040	17.0	87.6
otcmcsdll0005	**0.264**	0.240	4.3	91.9
ottdllvrcsearch	**0.056**	0.010	0.9	92.9
otccom112	**0.043**	0.000	0.7	93.6
otdslsdll4113	**0.032**	0.000	0.5	94.1
otccomdll0201	**0.029**	0.000	0.5	94.6
otdsls4100	**0.029**	0.020	0.5	95.0
otdslsdll4000	**0.027**	0.010	0.4	95.5
ottstpamdll	**0.025**	0.030	0.4	95.9
otcmcsdll0065	**0.023**	0.000	0.4	96.3

The “perc” column gives the percentage of time spent for this object; the “perc_t” column gives the percentage used by this object, including all objects above. This helps to find objects that can contribute to a performance improvement.

Object function summary

Clicking an object’s link displays the functions in the object as shown in this figure:

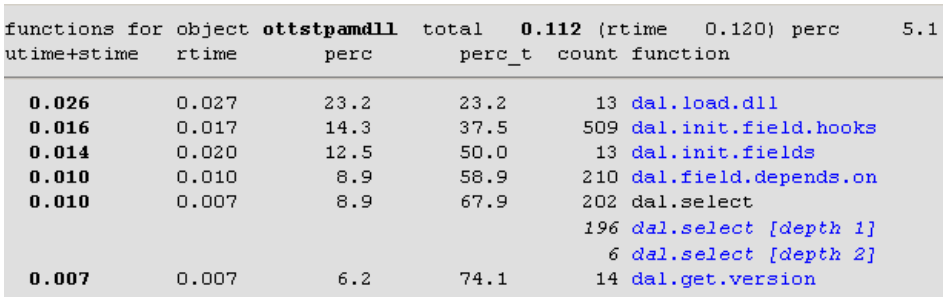
functions for object **otccomdll0201** total **0.029** (cpu time 0.000) perc 0.5

run time	cpu time	perc	perc_t	count	function
0.022	0.000	73.4	73.4	19	idb.eq
0.003	0.000	10.1	83.4	24	define.real.leng
0.001	0.000	2.3	85.7	13	init.buf.main.index
0.001	0.000	1.8	87.6	36	idb.alloc.mem
0.000	0.000	1.7	89.3	18	idb.init.buf
0.000	0.000	1.5	90.8	64	search.table.name

In the object function summary, the list stops when perc_t has a value of 90 percent or more.

If a function directly or indirectly calls the same function, this is shown by the depth layer.

For example, when a DAL calls another DAL, the dependency is found by the depth, as shown in this figure:



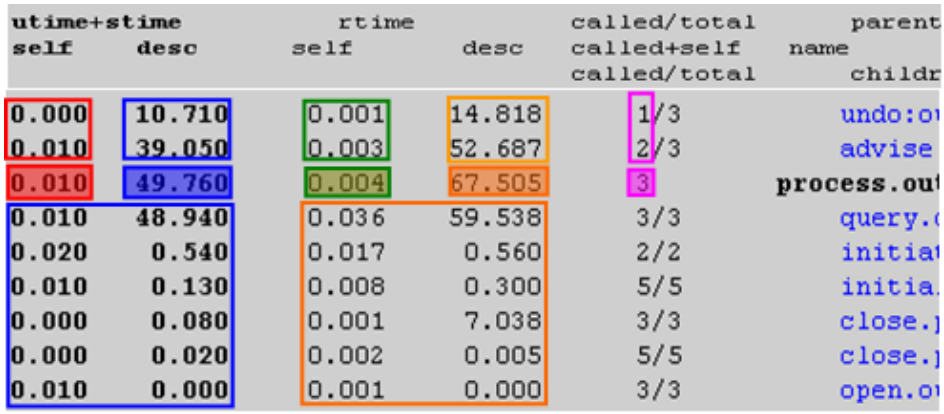
```
functions for object ottstpamdll  total   0.112 (rtime   0.120) perc     5.1
utime+stime   rtime       perc       perc_t  count function

  0.026       0.027       23.2       23.2       13 dal.load.dll
  0.016       0.017       14.3       37.5      509 dal.init.field.hooks
  0.014       0.020       12.5       50.0       13 dal.init.fields
  0.010       0.010        8.9       58.9      210 dal.field.depends.on
  0.010       0.007        8.9       67.9      202 dal.select
                                                196 dal.select [depth 1]
                                                  6 dal.select [depth 2]
  0.007       0.007        6.2       74.1       14 dal.get.version
```

If the “dal.select [depth 1]” function has no direct function call to depth 2, the function call is indirect. This is also tracked by the profiler.

Call Graph

This section contains all functions, the parent of the functions, and the child functions. The following figure is an example of part of the output:



```
utime+stime          rtime          called/total       parent
self     desc        self     desc  called+self    name
                                    called/total       childr

0.000    10.710      0.001    14.818      1/3          undo:o
0.010    39.050      0.003    52.687      2/3          advise
0.010    49.760      0.004    67.505      3          process.ou
0.010    48.940      0.036    59.538      3/3          query.
0.020     0.540      0.017     0.560      2/2          initia
0.010     0.130      0.008     0.300      5/5          initia.
0.000     0.080      0.001     7.038      3/3          close.
0.000     0.020      0.002     0.005      5/5          close.
0.010     0.000      0.001     0.000      3/3          open.o
```

In the above Call Graph profile, the process.outbound function was called three times, (seen in the purple shaded box). The function was called once from the “undo” function and twice from “advise” function, (seen in the purple square).

The shaded red box shows how much CPU time each function uses (0.010 sec). The red square indicates how this time was generated from the calling functions: The call from the “advise” function caused all CPU time required for the “process.outbound” function.

The large blue square shows the timing information of all functions called from the “ process.outbound” functions. The left side shows information about the functions, and the right side shows information about the children of the “child” functions. The functions are ordered on total CPU time, where the most consuming function is the closest to the “process.outbound” function.

The shaded blue box is the summary of the shaded red box (i.e. CPU time) and the large blue square (i.e. time consumed by functions). This is the total CPU time of all functions started from the “ process.outbound” function including the time consumed by the “process.outbound” function.

The upper blue square is the distribution of the time of the parent functions (shaded blue box). In the above illustration, the “undo” function is responsible for 10.710 seconds in the “process.outbound” function and its children.

The function that contributed the most to the process.outbound function is placed closest to the “ process.outbound” function. This applies for its parents and children.

The same process is repeated on the runtime columns, except for the ordering of data. To sort data on runtime, select the section based on “rtime”.

On the left side of the columns (not in the illustration) a percentage column is added. The information in this column helps to calculate how much time has been spent on both functions and called functions.

Flat Profile

This section contains the functions ordered by CPU time or run time, and also contains the number of times a function was called. This section does not show a relationship between functions or the queries.

Analyzing the output

You can use the profiler to look for improvements. However, some of the improvements require application insight while the other improvements require help from database specialists. Here are the guidelines to analyze a profile:

Run time vs. CPU time

If the bshell, driver, and database are located on the same system, the run time is usually twice the CPU time when running in non-combo. If the run time is larger, you must check the queries. If the CPU time is almost the same as the runtime, we recommend that you look for CPU intensive tasks.

Queries

Bad performing queries are listed in the queries section. Always check to see how much time can be gained by improving a query. For example, a query that takes 20 seconds may be high at first glance. However, you might also not notice that query, if a session has run for more than an hour.

Expensive functions

The flat profile directly shows the most expensive functions. The functions that consume most of the CPU time are displayed first in the ‘utime + stime section’. However, if the rtime section is also included, functions ordered on run time, which can help to find performance critical code, are also displayed.

Note that functions can be costly when they are called too often, which then increases the expense per call.

Function tree

To find performance overheads in the application, use the function tree next to the call graph. When using the function tree, it is useful to have application knowledge. When checking for performance overhead, check for the following functions:

- Functions that are called too often, such as functions to read parameters or other static data.
- Functions that are called too often because of a badly written application. For example, if 10 orders (each with five lines) must be updated and certain functions are called 500 times, check to find out why these functions are being called so often.
- **Note:** Every function that is not called improves performance because its child functions are not called.
- In the “percentage” column, it helps to determine how much can be gained by not calling a function at all. We recommend that you look at the areas where performance can be most improved.

BAAN_SQL_TRACE

This trace variable gathers information from the bshell. All application queries sent to the database can be logged, except DML and DLL queries. Queries that are generated by the standard program can be recognized because those SQL statements are printed in uppercase. Queries that are built partly by the standard program will be printed in both uppercase and lowercase; the uppercase parts are usually generated by the standard program/query extensions. The lowercase parts in those queries have been added by the developer itself.

If BAAN_SQL_TRACE is used without specifying a file, the output is sent to the bshell i.e. the <pid> file in the BSE_TMP directory. This file will be removed when the bshell is closed. Therefore, we recommend writing the trace in a separate file by specifying a file name in the BW command line with the following option:

```
-keeplog -logfile <FileName> -set BAAN_SQL_TRACE=<value>
```

The “keeplog” option prevents the log file from being removed from the system when the bshell is closed.

Apart from being able to set BAAN_SQL_TRACE as an environment variable, you can also set it from the BW option dialog. With this option, you can switch on tracing just before the real tracing is required; this can be stopped just after the trace. The settings can be configured in the “Debug Bshell” section of the option dialog on the **BDB/SQL Tracing** tab.

The following BAAN_SQL_TRACE options are currently valid:

- 0000001 Show parse tree
- 0000002 Show exec tree
- 0000004 Show filter details of exec tree
- 0002000 Show calls of internal SQL functions and bind variables

Combinations of these options are allowed by adding them in an octal way. For example:

```
-set BAAN_SQL_TRACE=2007
```

When you make a first trace, the value 2000 probably already provides sufficient information. Because the output of BAAN_SQL_TRACE is barely readable, we recommend that you use the Call Graph Profiler instead. An example of the output with BAAN_SQL_TRACE=2000 is shown in this figure:

```
=============================================
[1] SQLStatement::Constructor
[1] SQLStatement::Prepare( SessionId=1 Mode=4 )
StatementText:
select  ttaad001.*
        from    ttaad001
        where   ttaad001.seqn = 1
        and     ttaad001._compnr = 0
        as set with 1 rows

[1] SQLStatement::Prepare() returns 0
[1] SQLStatement::Exec( Compnr=0 )
[1] SQLStatement::Exec() returns 0 nrows 0
[1] SQLStatement::Fetch()
[1] SQLStatement::Fetch() returns 100
[1] SQLStatement::Break()
[1] SQLStatement::SoftClose()
[1] SQLStatement::SoftClose() returns 0
[1] SQLStatement::Break() returns 0
)=============================================
```

BAAN_SQL_TRACE with -dbgflow

If it is not possible (for example the object is closed with a kill command) or too difficult to use profiling, it becomes more difficult to find the location of application queries that perform badly. One possible option is the usage of BAAN_SQL_TRACE=2000 in combination with the bshell trace command -dbgflow. The “dbgflow” option writes every change of an active object or DLL in the trace file. Therefore, when a badly performing query is found in this trace, such as with BAAN_SQL_TRACE=2000, you can find the object that contains this query.

We recommend that you log the changes in a separate file; do this by specifying a file name in the BW command line with this option:

```
 -dbgflow -keeplog -logfile <Filename> -set BAAN_SQL_TRACE=2000
```

Example of the output of BAAN_SQL_TRACE:

```
B:0000022:::(00001):Flow: -->>  (depth 01):    main() (object ottstpstdlib)
B:0000023:::(00001):Flow: -->>  (depth 02):     get.multibyte.factor() (object
ottstpstdlib)
B:0000024:::(00001):Flow: <<--  (depth 02):     get.multibyte.factor() (object
ottstpstdlib)
B:0000025:::(00001):Flow: -->>  (depth 02):     init.logfin.comp() (object
ottstpstdlib)
[1] SQLStatement::Constructor
[1] SQLStatement::Prepare( SessionId=1 Mode=4 )
StatementText:
```

```
select  ttaad001.*
        from    ttaad001
        where   ttaad001.seqn = 1
        and     ttaad001._compnr = 0
        as set with 1 rows

[1] SQLStatement::Prepare() returns 0
[1] SQLStatement::Exec( Compnr=0 )
[1] SQLStatement::Exec() returns 0 nrows 0
[1] SQLStatement::Fetch()
[1] SQLStatement::Fetch() returns 100
[1] SQLStatement::Break()
[1] SQLStatement::SoftClose()
[1] SQLStatement::SoftClose() returns 0
[1] SQLStatement::Break() returns 0
```

When a new function is called, it is written in the trace file as follows:

```
B:0000023:::(00001):Flow: -->>  (depth 02):     get.multibyte.factor() (object
ottstpstdlib)
```

Hereby get.multibyte.factor() is the new function to be called in object ottstpstdlib.

When a function is left, the following is written to the trace file:

```
B:0000024:::(00001):Flow: <<--  (depth 02):     get.multibyte.factor() (object
ottstpstdlib)
```

All trace information that is performed in-between these pieces is implemented in the DAL object or DLL object. In the above example, the query that is mentioned is located in the code of ottstpstdlib.

<DB>PROF

When using the <DB>PROF variable, the queries from the driver that is sent to the database and their timestamps can be traced. The string “<DB>” must be replaced with the used abbreviation for the used database.

Depending on the used database driver, the following traces are available: **DB2PROF, INFPROF**, **MSQLPROF, ORAPROF**.

The value given to the <DB>PROF variable is the threshold in seconds. Each query that takes more time, or equals this value, will be reported, such as when setting the following:

```
ORAPROF=4.0
```

All queries that take longer than four seconds are reported. To obtain all the queries, use the following setting:

```
ORAPROF=0.0
```

The <DB>PROF variable can be set like all other environment variables in the Infor LN Web UI or BW command field, such as:

```
-set ORAPROF=0.0
```

The <DB>PROF variable can also be used in db_resource. The variable name to use is the same, but in lowercase:

```
oraprof:0.0
```

The output of <DB>PROF is the same as the used variable, but the output file is in lowercase. Using ORAPROF gives the file ‘oraprof.’ The output file is created in the directory where the bshell is started. New data is added to an existing file.

DB2PROF

The DB2 UDB driver writes timing information for each application action (parse/exec/fetch) that takes longer than the time (in seconds) defined by the DB2PROF.

The following db2prof output is obtained by running the following commands:

```
export DB2PROF=0.0
bsql6.2 –c090 –q "select item,dsca from tcibd001 where seak='COST ITEM'"
```

The output shows the various steps in the execution of the query and the time each step took.

No performance problems are detected in the execution of this query:

```
2011-06-10[12:53:24]: Profiling value = 0.0000 sec
------ Profiling value exceeded -----
<root><bsql_00>:2011-06-10[12:53:25.819]:
Time (parse) : 0.000022 seconds
SELECT a0.t_item,a0.t_dsca FROM FP6BKIT.ttcibd001090 a0  WHERE a0.t_seak = CAST(? AS
VARCHAR(9))   OPTIMIZE FOR 5 ROWS
-------------------------------------
```

There are more sophisticated ways to obtain tracing and to explain information from DB2 UDB: CLI trace, Dynamic SQL Snapshot, and Explain Output. Usually, these tracing facilities produce more detailed information about the running Infor LN sessions, including the time spent in the application and the processing time on the DB2 UDB server. Overall, these DB2 tracing tools require a higher level of analysis and, consequently, they deliver a more detailed and accurate picture of the running system.

ORAPROF

The Oracle driver writes timing information for every application action (parse/exec/fetch).
The differences between tracing with ORAPROF and tracing with SQL_TRACE are as follows:

- SQL_TRACE output is much smaller.
- SQL_TRACE output can be easily formatted with the included tools.
- Oracle tracing works with a precision of 1/100 sec, and ORAPROF is typically much more precise.

- Difficult to trace parallel processing with ORAPROF because all output is directed to the same file.
- Timestamps with ORAPROF include the round trip time between the driver and database. In case of expected network problems, a comparison between the two output files can help.

Example of the ORAPROF output is as follows:

```
2011-06-10[12:53:24]: Profiling value = 0.0000 sec
------ Profiling value exceeded -----
<root><bsql_00>:2011-06-10[12:59:10.342]:
Time (parse) : 0.000000 seconds
SQL statement:
SELECT /*+ FIRST_ROWS index(a0 ttcibd001603$idx2) */ a0.t$item,a0.t$dsca FROM
baan.ttcibd001603 a0 WHERE a0.t$seak = :1
-------------------------------------
```

This output is generated by the following command:

```
ORAPROF=0.0 bsql6.2 -q "select item, dsca from tcibd001 where seak =   'COST ITEM'"
```

The output shows the various steps of the query: parse, execute, and fetch. For each step, the time is measured.

MSQLPROF

The Microsoft SQL Server driver writes timing information for each application action (parse/exec/fetch) that takes longer than the time (in seconds) defined by the MSQLPROF.

The following MSQLPROF output is obtained by running these commands:

```
set MSQLPROF=0.0
   bsql.exe –q "select item,dsca from tcibd001 where seak='COST ITEM'"
```

The output shows the various steps in the execution of the query and the time each step took.

```
2011-06-10[12:53:24]: Profiling value = 0.0000 sec
------ Profiling value exceeded -----
<baan><?>:2011-06-10[12:53:25.819]:
Time (parse) : 0.000000 seconds
SQL statement:
SELECT a0.t_item,a0.t_dsca FROM dbo.ttcibd001090 a0 WITH ( INDEX ( Ittcibd001090_2a ))
WHERE a0.t_seak = ?  OPTION (FAST 10)
-------------------------------------
```

There are more sophisticated ways to obtain tracing and to explain information from Microsoft SQL Server: MSQL Query Analyzer (SQL Server Management Studio in Microsoft SQL Server), MSQL Profiler, Server-Side Traces, Trace Flags and others. Usually, these tracing facilities produce more detailed information about the running Infor LN sessions, including the time spent in the application and the processing time on the Microsoft SQL Server Database. Typically, these Microsoft SQL Server tracing tools require a higher level of analysis and, consequently, they deliver a more detailed and accurate picture of the running system.

DBSLOG

The DBSLOG variable provides detailed debugging information about the online processing of the driver. The information is logged in the dbs.log file in the driver's current directory.

This trace option is usually not used for performance tracing, but has two important features for tracing the Infor LN application for performance:

- It can provide the value of used variables. To obtain the queries and their used variables, you can use DBSLOG=500.
- It can log retries and measure the duration of a lock with the 100000 and 200000 value. Tracing on locks and retries can be useful in environments in which processes scale badly or other locking issues are expected to be found.

The output of DBSLOG is generated in the current directory; for BW users, this is the home directory of the users. The default file name is dbs.log unless otherwise specified by the DBSLOG_NAME variable.

Other values of DBSLOG are as follows:

- 0000001 Data Dictionary information of tables within the driver.
- 0000002 Query info (SQL Level 1).
- 0000004 Query plan info (SQL Level 2).
- 0000010 Row action information.
- 0000020 Table action information.
- 0000040 Transaction action information.
- 0000100 DBMS input/output data (SQL Level 2).
- 0000200 Administration file info (SQL drivers).
- 0000400 DBMS SQL statements.
- 0001000 General debug statements.
- 0002000 Query processing info (for BAAN_SQL_TRACE info).
- 0004000 Data buffering info (communication).
- 0100000 Lock retries logged (includes session name).
- 0200000 Logs duration and tablename of the first lock set in a transaction. Use DBSLOG_LOCK_PROF=n to set a threshold.

To define multiple categories, add the octal values. These values are checked bit wise to determine if a given category must be logged.

To trace all locks longer than a certain time, specify the DBSLOG_LOCK_PROF variable. To prevent too much logging, we recommend that you set this variable to 0.5 seconds. Additionally, to redirect the output of the trace, specify the DBSLOG_NAME variable.

You can set the DBSLOG variable in the BW command field, similar to how you set all other environment variables. For example, to log all SQL statements and DBMS input variables and output data, use:

```
-set DBSLOG=500
```

This figure shows an example output of DBSLOG=500:

```
<63439014> 2011-06-28[11:06:53]: Logging started mode 0500
        ---- LOG DBMS  INFO [0000100] ----
        ---- LOG SQL   INFO [0000400] ----

SQL> SET CURRENT QUERY OPTIMIZATION = 0

SQL> SELECT a.t_pacc,a.t_keyr,a.t_desc,a.t_Refcntd,a.t_Refcntu FROM
FP6BKIT.tttadv999000 a WHERE a.t_pacc=? AND a.t
_keyr=?  OPTIMIZE FOR 5 ROWS   [1105734f0]  t_pacc [1105734f8]  t_keyr [110229c9a]
[110229c90]  [110229c89]  [1102
29c82]  [110229c7b]
----- DBMS Where Input ----
Bind nr 1 : pacc       : [1105734f0] string  : db_ind   8 : db_size   8 : '        '
Bind nr 2 : keyr       : [1105734f8] string  : db_ind  22 : db_size  22 : 'moduttdsk
'
Fetch : end of set.
----- DBMS Where Input ----
Bind nr 1 : pacc       : [1105734f0] string  : db_ind   8 : db_size   8 : 'b61au   '
Bind nr 2 : keyr       : [1105734f8] string  : db_ind  22 : db_size  22 :
'progttdskbbrowser   01'
----- DBMS Output Row ----
Bind nr 1 : pacc       : [110573520] string  : 'b61au   '
Bind nr 2 : keyr       : [110573528] string  : 'progttdskbbrowser   01'
Bind nr 3 : desc       : [11057353e] string  : '0ottdskbmbrowser|a1|||1||||||
             '
Bind nr 4 : Refcntd    : [110573590] long    : <0>
Bind nr 5 : Refcntu    : [110573598] long    : <0>
```

Caution: Although possible, it is not recommended to set trace variables like DBSLOG in the db_resource file, as it will generate a large portion of data for every user, which will have a huge performance impact.

Tracing with bsql

A tool in the porting set called bsql, can be used to start queries. When a badly performing Infor LN query is found, it can be tested with this tool. This tool can be started only from the command prompt.

The most useful options for performance tracing are as follows:

- -f <QueryFile>. bsql will process the query from the specified file.
- -o <OutputFile>. Redirects the output to the specified file.
- -c <Company>. Performs the query in the specified company.

If a problem query is detected with the Call Graph Profiler, the query can be copied to a file. The variables must be replaced with real values; if guessing the values is too difficult, a DBSLOG trace can provide the values of the variables for a query. The value DBSLOG=500 can be used to provide queries and variable values. BAAN_SQL_TRACE=2000 can also be used to obtain information about the bind variables.

After completing the file with the query, the query can be started with bsql. If many rows are expected as output, usually a dummy file must be used. For Windows systems, the NUL file has no

disk bottlenecks; for UNIX, the /dev/null device has no disk bottlenecks. If the output is required, we recommend that you use a fast disk to place the output file.

Below is an example of how bsql is used:

```
bsql –f bad_query –o NUL –c 100
```

The bsql command can be used with environment variables or system tools, as shown in the example:

```
$ time ORAPROF=0.0 SQL_TRACE=true bsql6.2 –f bad_query –o /dev/null –c 100
```

In the last example, bsql processes the query from the bad_query file and puts the output in /dev/null. The query is started in company 100. The UNIX ‘time’ command is used to calculate the CPU usage and wall clock time for the bsql command. To find the Oracle SQL statements, the ORAPROF and SQL_TRACE variables are added.

Choosing the correct tracing method

Because the Call Graph Profiler is easy to use, it should be used for all performance traces. This tool can analyze most of the performance problems.

If the Call Graph Profiler is not sufficient, and it is not clear why a query takes a certain amount of time, database tracing can be added. SQL_TRACE shows the output for the Oracle Database; the Explain option of Dynamic SQL snapshot shows the output of DB2 UDB.

If there is a locking issue, DBSLOG can help.

<DB>PROF and BAAN_SQL_TRACE output is usually too large and difficult to interpret; therefore, they must only be used when required. For example, if there is a problem in the database driver or network.

Before you run traces for all tests, we recommend that you isolate the problem as much as possible. If a session that calculates projects performs badly during one project, we recommend that you trace only that project instead of all projects. Isolating the problem will help you get to the real bottleneck faster.

Chapter 7 Performance Wizard

7

Usually when a problem appears it needs to be resolved quickly. This chapter helps you approach and solve these problems quicker.

Session related

Performance problems can be session related. In this section, we describe some situations that can occur.

Session takes too long to start the first time

Normally, the first time a session starts, the objects need to be loaded in memory and the database cache needs to be filled. This is the reason why it takes a little longer to start a session for the first time, when compared to starting the session the second time. However, if it takes more than 20 seconds to start a session the first time, it could be because of the following reasons:

- It takes a lot of time to start the database connection.
  - Check if login using database specific tools like Oracle sqlplus is fast.
  - If login using database specific tools is fast, check if the first Infor LN query sent to the database takes more time than expected; the easiest way to do this is by tracing with a Call Graph Profile in combination with a database trace. Check out the first application and database query generated in the starting session.
- Objects/data definitions are not loaded in shared memory. Particularly when stored on a slow disk, it is advised to store as much as possible in shared memory.

If this does not help, check the following:

- Check the CPU usage of the system.
- Check the size of tabledef. If this file is large, try to reduce the size. Also check that no multiple drivers are started, by specifying environment variables in some lines.
- Check the size of the <db>_driver_param file. During the startup of a new database connection, this file will be read. The smaller the files, the faster the database driver can start.
- Check the amount of memory available for the database.

All sessions are slow even when used by few users

If all sessions are slow even when used by a few users, check the following:

- Check if the system is not too heavily loaded. It is possible that the system suffers from so-called “run away” processes or a backup. To see what processes run on the system and how much CPU they consume, use system command tools. Also check the disk utilization. If any of these problems are found, it is important to solve it before analyzing further.
- Check if the connection to the database is slow. Check the Call Graph Profiler output for where the time is being spent. If the output also contains simple queries, it is recommended to check the database connection and see if the database is tuned well. To see if it is a connection or database problem, it is advised to run (besides the CGP) the database trace such as SQL_TRACE for Oracle. If the timing between the application and the database tracing differs a lot, it is probably the connection; if the database tracing shows large times for easy queries, database tuning is recommended.

Some sessions are slow

Slow sessions can sometimes be caused by locking, bad performance of the session, wrong expectations of the session, or not the optimal configured application. If this is the case, check the following:

- Check if a session is running slow when running alone on the system. If so, the session must be traced as described in the section **Particular (batch) session** is slow.
- If the session is faster when running it alone on the system, it can indicate that locking problems are causing performance degradation. It is advised to check the database for queries waiting for a releasing lock.

Session is slow when a few users run the same session

If a session is slow when a few users run the same session, check if locking problems are causing the bad performance. To check if locking occurs, use database tools or **Log for locking**.

Particular (batch) session is slow

Depending on the possibilities, if a particular session is slow, the following traces can be made:

- Call Graph Profiling
- Database trace

For more information about tracing, refer to chapter 6.

Note: Application settings and the data that must be processed have an important effect on performance.

System related

Performance problems can be system related. In this section, some situations that can occur are described.

System uses 100 percent CPU

If the system uses 100 percent CPU, it is not necessarily a problem. It's just an optimal use of the available resources. However, it can become a problem when the load on the system is so high, that the response time to the user increases to an unacceptable level. System utilities such as perfmon on Windows and sar or vmstat on UNIX can give more insight into what is happening on the system.

System is not 100 percent CPU bound but sessions are slow

If the system is not 100 percent CPU bound but sessions are slow, it is important to check which sessions are slow and under what circumstances. Also check what is happening on the system with regards to memory and disk I/O.

Note: High disk I/O on a certain disk might be caused by memory pressure, as the system will start paging out memory pages to disk.

Also check the "Some sessions are slow" section in this chapter.

Process related

Performance problems can also be process related. In this section, some situations that can occur are described.

Bshell is consuming most of the CPU time

If an OLTP bshell is consuming > 70% CPU of a processor core for more than 20 seconds, we might call Houston with the message saying "we have a problem".

Possible reasons for a bshell process consuming most of the CPU time are:

- Table boosters are used. Table boosters are tables loaded in bshell memory. Reading from these tables can lead to more CPU time on the bshell, particularly when table boosters are activated with the "Incremental Load" feature. Check if table boosters are used to the optimum as described in the section "About table boosters".

- A lot of sorting is performed. Some functions need to sort data in one way or another. With profiled objects, check which functions consume most of the time and why sorting is done. Usually, sorting is very CPU/disk intensive or both.
- Sorting is performed in the database driver. In certain situations, it is possible that data must be sorted in the driver. If sorting needs more than a certain amount of memory, disk space will be allocated to do the sort. Check if huge/many sort files, which start with qp, exist in the BSE_SORT directory. If sorting causes performance degradation, the problem function/query can be found with profiled objects. Try to eliminate driver sorting as much as possible.
- Re-parsing of queries.

Database is consuming most of the CPU time

Possible reasons for database consuming most of the time are:

- Database is not tuned optimal. Refer to the database specific Performance, Tracing and Tuning Guide.
- Parse times - Check if parsing is not taking too much time in the database. If parsing is done often, see if it can be reduced by database tuning or changing driver settings.
- Sorting can lead to high database times. Check if queries force the database to do lot of sorting.

Bad queries - This can be checked by running Call Graph Profiling or database tracing.

Appendix A Performance Checklist

A

The following checklist contains most of the topics that should be checked on every Infor LN configuration before analyzing a performance problem in more detail:

11 Environment variables

- Which environment variables are set and do they have the correct value?

12 Driver parameters

- Which variables are set in the db_resource file and do they have the correct value?

13 Table definitions

- Do all tables use the same driver?
- Do all tables use the same environment variables?

14 Auditing

- Are all tables specified for auditing required?

15 Shared tables

- Are the shared tables grouped optimal?

16 Authorizations

- Have Infor LN authorizations been used efficiently?
- Have database authorizations been used efficiently? Are field authorizations used?

17 Shared memory usage

- Are the correct package combinations, objects and reports loaded in shared memory?

18 Porting set

- Is a recent porting set used?

19 Standard program

- Is a recent Enterprise Server standard program used?

20 Log files

- Do the (system) (error) log files grow rapidly? Any messages in the Windows event viewer?
- Do the Infor LN database driver or bshell log files show (performance) problems?

21 Configuration

- Is the Infor LN environment configured optimal for the usage of OLTP vs. Batches?

Appendix B Format Trace Output

B

With the scripts in this appendix, the output of a BAAN_SQL_TRACE=0200 trace can be formatted in a more readable way. The output of this script gives the following columns:

- **QID**: Query ID, this number corresponds to a query later in the output.
- **Count**: The number of fetches done on this query.
- **Time**: The time needed for the total amount of fetches.
- **Avg**: Average Time per Fetch (= Time/Count).
- **First**: Shows the time needed for the first fetch.

An example of the output:

```
QID  Count Records    Time  First
  19      3      18    3.02   0.02
  28      2       4    2.01   0.01
  33      1      13    1.86   0.86
  26      1       6    1.03   0.03
  27      1       3    1.02   0.02
```

After the QIDs and their times, the queries show up exactly as in the original output:

```
Query : 33
SELECT tcibd001.*, tcmcs001.tccu, tcmcs003.nwrh, tcmcs048.cref,tcmcs052.plnk
FROM tcibd001,  tcmcs001, tcmcs003, tcmcs048, tcmcs052
WHERE  {tcibd001.item} >=  {:tcibd001.item} AND (tcibd001.cwun refers to tcmcs001
and tcibd001.cwar refers to tcmcs003
and tcibd001.cpcp refers to tcmcs048
and tcibd001.cprj refers to tcmcs052)
ORDER BY tcibd001._index1

Query : 34
SELECT ttadv999.*
WHERE ttadv999._index1 > { :1, :2 }
ORDER BY  ttadv999.pacc ASC,  ttadv999.keyr ASC
AS SET WITH 3 ROWS
```

The script:

```
if [ $# -ne 2 ];then
   echo "Usage $0: <inputfile> <outputfile>"
   exit
fi

Awk=awk
[ -x /usr/xpg4/bin/awk ] && Awk=/usr/xpg4/bin/awk
[ -x /usr/bin/nawk ] && Awk=/usr/bin/nawk

$Awk '
```

```
/^Fetch times of Query/ {
   getline
   getline
   i=0
   while (substr($1,1,5)!="-----") {
      if ($1!="")
         InSQL[++i]=$0
      getline
   }
   for (j=MaxQID;j>0;j--) {
      for (k=i;k>0;k--)
         if (InSQL[k]!=SQL[j,k])
            k=-1
      if (k==0)
         j=-j
   }
   if (j==0) {
      MaxQID++
      QID=MaxQID
      for (j=1;j<=i;j++)
         SQL[QID,j]=InSQL[j]
      Lines[QID]=i
   }
   else
      QID=-j-1
   getline
}
/^Nr Rows Fetched/ {
   LastFetch=$5
   Fetch[QID]+=$5
   Count[QID]++
   getline
}
/^Fetch Time for 1st Row/ {
   First[QID]+=$7
   getline
}
/^Max Fetch Time/ {
   Max=$5
   getline
}
/^Average Fetch Time/ {
   AvgTot=LastFetch*$5
   getline
}
/^Average Fetch Time/ {
   if ($7+Max>AvgTot) {
      Tot[QID]+=$7+Max
   }
   else
      Tot[QID]+=AvgTot
   Tot[QID]+=1
   getline
}
END {
   for (i=1;i<=MaxQID;i++) {
      Q[i]=i
```

```
      for(j=i;j>1 && Tot[j-1]<Tot[j];j--) {
         swap(Count, j)
         swap(Fetch, j)
         swap(First, j)
         swap(Tot, j)
         swap(Q, j)
      }
   }
   print " QIDCount Records    Time  First"
   for (i=1;i<=MaxQID;i++)
      printf("%4d %6d %7d %7.2f %6.2f\n",
         Q[i], Count[i], Fetch[i], Tot[i], First[i])
   print
   for (i=1;i<=MaxQID;i++) {
      print "Query : " i
      for (j=1;j<=Lines[i];j++)
         print SQL[i,j]
      print
   }
}
function swap(A, i) {
   t=A[i-1]
   A[i-1]=A[i]
   A[i]=t
}' $1 > $2
```

Appendix C Application Response Time Measurement

C

This appendix describes how transactions of sessions can be measured by Application Response Time Measurement (ARTM). Infor LN ARTM is an implementation of the ARM interface.

Examples of ARTM are:

- Response times when processing an order.
- Response times when searching for an account.

An example of when ARTM is useful is when Service Level Agreements around response times of critical Infor LN sessions have been defined. The ARTM data can be sent to an external tool that supports standard ARM calls such as HP Glance or IBM Tivoli. These tools are capable of presenting the data in several ways to the end-user. Both products have agents for several platforms, so customers also running on other platforms than HP and IBM can be supported.

ARTM is designed for system administrators to monitor the overall user response time of the whole Baan system and the specific functionality within a specific application. The data can be analyzed later.

ARTM helps the IT manager to meet service level agreements and helps the system administrator to detect slow responses to users and find the bottlenecks in an installed Infor LN system.

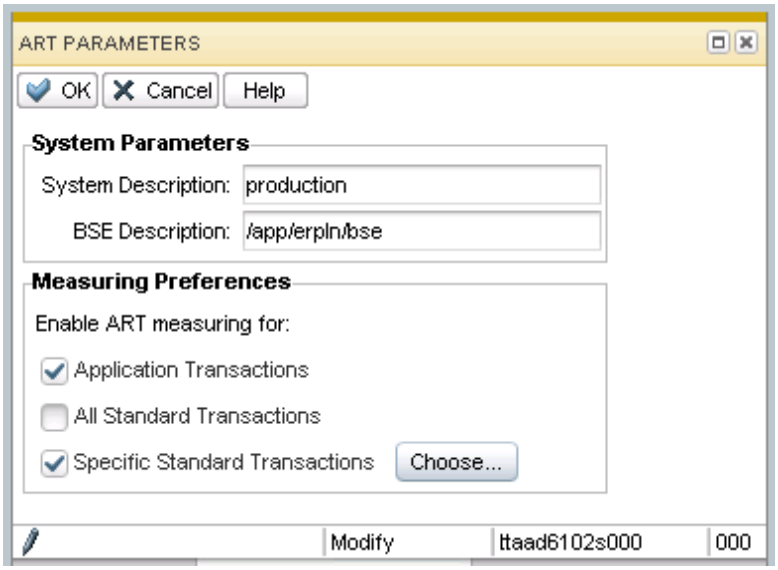
ARTM is not a tracing system that generates huge amounts of data. The information provided by ARTM must be kept to a minimum, to give the system administrator a quick overview of the responsiveness of the system. The overhead is minimal; tests showed an additional overhead less than 0.5%.

This implies that transactions must be implemented on a high level, starting when a user action begins, and ending when the user action is finished. Besides the default transactions that can be logged, it is also possible to create own logging by 4GL code.

More information about ARM can be found in the Infor LN online help.

Setup

In the ttaad6102s000 session, the parameters for ARTM can be specified:



System parameters are:

- System description: Name of the system, such as Production.
- BSE description: BSE, such as /app/inforln/bse.

Additionally, it is possible to measure this:

- Application transactions: These are the custom programmed transactions. You must select this option.
- All Standard Transaction: Traces all standard transactions in all sessions. It generates a lot of data and this option is usually cleared.
- Specific Standard Transactions: This option can be used instead off “All Standard Transactions”. It allows the user to specify transactions that need to be measured. To select a specification click **Choose**.

By default, the ARTM calls are not sent to the ARM agent. Only after setting the art_enable parameter is ARTM activated. The following parameters can be used to enable ARTM:

BAAN_ART_ENABLE / art_enable

Valid values for BAAN_ART_ENABLE/ art_enable are 1 (enable) or 0 (disable = default). Specifying 'art_enable:1' in $BSE/lib/defaults/all will enable ARTM calls for all users. Specifying this resource in the $BSE/lib/users/u<user> file enables ARTM calls for only this user. The environment variable can also be used and is useful for testing.

BAAN_ART_USER / art_user

The BAAN_ART_USER/ art_user variable specifies the name of the user under which transactions will be registered. By default, this name will be 'BAAN'; for example, all transactions will be registered under the same name. If it is required that specific users be monitored separately, this can be configured by specifying this variable in the BW configuration file or the $BSE/lib/user/u<user> file.

Programming own transactions

Sometimes, more than the default information is needed. For example, by default, a processing session gives the amount of processed orders to ARM. This extra information is also needed when the SLA says that each order might take 10 seconds to be processed. Additionally, an order line with 100 order lines will probably take more time than an order with 10 lines. This functionality can be included by the following functions:

- artm.define.transaction.class
- artm.redefine.transaction.class
- artm.begin.transaction
- artm.update.transaction
- artm.end.transactionThe following code is an example of how these functions can be used:

```
long   nr.orders, nr.lines
long   defined.tra.class.id, tra.id, i, ret
       |* Redefine the transaction belonging to
       |* form command "Process orders"
       defined.tra.class.id = artm.define.transaction.class(
               "Process Orders", "Order Processing Session",
               ART.COUNTER, "Order Count",
               ART.COUNTER, "Order Line Count")
        |* User has started order processing.
        |* Start the transaction
        tra.id = artm.begin.transaction(
                defined.tra.class.id,
                ART.COUNTER, 0,
                ART.COUNTER, 0)
        for i = 1 to 10
                |* process.order sets nr.orders and nr.lines
                process.order()
                |* Give feedback about the order processed
                ret = artm.update.transaction(
                        tra.id,
                        ART.COUNTER, nr.orders,
                        ART.COUNTER, nr.lines)
        endfor
        |* All orders are processed. End transaction.
        ret = artm.end.transaction(
                tra.id,
                ART.TRANSACTION.SUCCESS,
                ART.COUNTER, nr.orders,
```

```
                ART.COUNTER, nr.lines)
}
```

See the Programmers Manual for more information about programming on performance monitoring.

Tracing and debugging

Tracing and debugging can be done by setting the BAAN_ART_TRACE / art_trace variable.

BAAN_ART_TRACE / art_trace

This variable can be used to trace ARM calls. By default, tracing is off (0) and can be turned on by specifying a value. Valid values are as follows:

22 Basic tracing of each call to the ARM library.

23 Basic tracing and the binary blocks sent to the ARM library.

24 Does not bind the ARM library, but uses internal stub functions. Application developers who do not have the ARM library installed can use this to do basic tracing of the ARM calls they programmed.

25 Same as 3, but now includes the binary blocks.

By default, the output of the trace is sent to the $BSE/tmp/bshell.<pid> file. The trace made by BAAN_ART_TRACE=5 is useful to test ARTM. An example of the output:

```
B:0000000:::(00001):arm loaded: Baan internal ARM tracing stubs binded
B:0000001:::(00001):arm_init called (time stamp 0)
B:0000002:::(00001):    application name   = BaanERP
B:0000003:::(00001):    user name          = baan
B:0000004:::(00001):    flags              = 00000000
B:0000005:::(00001):    data pointer       = 0
B:0000006:::(00001):    data size          = 0
B:0000007:::(00001):    returned global id = 10000000

B:0000016:::(00007):arm_getid called (time stamp 13990)
B:0000017:::(00007):    application id     = 10000000 (268435456)
B:0000018:::(00007):    transaction name   = cprrp0520m000/Read
B:0000019:::(00007):    transaction detail = Read record(s)
B:0000020:::(00007):    flags              = 00000000
B:0000021:::(00007):    data pointer       = 8044A1C
B:0000022:::(00007):    data size          = 344
B:0000023:::(00007):    returned class id  = 10020000

B:0000030:::(00007):arm_start called (time stamp 55476)
B:0000031:::(00007):    class id           = 10020000 (268566528)
B:0000032:::(00007):    flags              = 00000000
B:0000033:::(00007):    data pointer       = 8044A80
B:0000034:::(00007):    data size          = 256
```

```
B:0000035:::(00007):    returned trans id  = 10020002

B:0000036:::(00007):arm_stop called (time stamp 55486)
B:0000037:::(00007):    application id     = 10020002 (268566530)
B:0000038:::(00007):    flags              = 00000000
B:0000039:::(00007):    data pointer       = 8044A80
B:0000040:::(00007):    data size          = 256
B:0000041:::(00007):    returned           = 10020002
```

The output shows:

- Block 1: Initialization of ARM.
- Block 2: Session cprrp0520m000 is started and the Read call is initialized.
- Block 3: In session the Read option is activated (start time = 55476). Time is in milliseconds.
- Block 4: The Read option is finished (end time = 55486). Therefore, the total duration of the find was 55486 – 55476 = 10 ms.

The relation between the different calls can be monitored by following the application, transaction, and class ids.