Tuning Informix Engine Parameters

by: William D. Burton

Introduction

Tuning engines is one of my all-time favorite activities, be they internal combustion or database engines. More often than not, a well-organized tuning session will provide *several orders of magnitude improvement* in performance. Obviously, this makes for some very happy users! Being able to improve the performance of a database engine is not due to any mysterious talents of mine, but rather to the fact that systems go out of tune rather quickly—or were never in tune in the first place.

As a rule, important dynamic systems should be tuned every three months or so. Having said that, the single most important point in this entire article is to test every critical system, and never believe too strongly what you read, hear, or *even what you demonstrated to be fact yesterday!*

Test, test, test.

I have collected more than six books and numerous articles that discuss tuning Informix® databases. It has been my observation that they conflict more often than they agree. I see several reasons for this:

- Frequent changes to underlying database architecture obsolete recommendations.

  With the release of Informix Dynamic Server™ (IDS) version 7.1, the Informix engine disk I/O subsystem changed so markedly that all previous NUMAIOVPS recommendations were incorrect.

- UNIX implementations differ widely at granularities important to the tuning of the system.

  A series of tests I performed at Informix's benchmark labs in Menlo Park demonstrated that configuring twice as many CPUVPS as hardware CPUs was optimal for HP-UX while the version of Solaris that we were testing preferred a 1:1 ratio.

- The rate of hardware advances quickly overtakes information learned.

  In a matter of just a couple years, disk size went from 2GB to 28GB. Huge solid-state (RAM) disk arrays are now prevalent. We are even seeing self-configuring disk storage units that reconfigure themselves on the fly as their on-board algorithms determine optimal performance (theoretically).

Note that these examples discuss wide variability in what are arguably some of the most significant tuning parameters. This should leave you with three important conclusions:

1. It is silly to speak of such things in absolute terms.
2. Each important end user environment must be measured and tuned individually (i.e., "cookie cutter" tuning will cause trouble for the DBA).
3. Most important, due to the fluid nature of this fundamental knowledge set, DBAs should continue to be well compensated.

That being said, what follows is my collection of notes for tuning most Informix **onconfig** parameters. These are slanted to the UNIX user. I keep a copy of these in my organizer for easy field reference. Most are directly out of a book. The comments I make are from my own experience. Where I have found more than one suggestion, I have listed them in chronological order, oldest first. Not surprising, you will note many conflicting pieces of information here. This entire document should be thought of as a guideline or starting point; again, nothing is gospel.

This article will cover the following topics:

- **onconfig** parameters
- Miscellaneous tuning notes
- Database tuning the Informix way

onconfig Parameters

For the most part, the recommendations below assume that the "system under test" is a dedicated database server. If it isn't, it should be. Certainly any mission-critical or high-performance database should be on a dedicated machine. The Informix engine is basically an operating system unto itself. All it really wants from the operating system are slices of CPU time. If it must share its sandbox with applications, other database engines, or even other Informix instances it is operating out of the bounds of its target design. Acknowledgement that this level of isolation is not always possible in the real world has been addressed recently with new **onconfig** parameters.

Since DBAs generally do not have the luxury of an extended tuning session, and must make quick changes to a production system to get it back within acceptable parameters, I have noted recommendations that allow for a "big bang for the buck" with a (*). Important adjustments for tuning in a "shared environment" (where applications and/or other database engines are running on the same machine) are marked by a (§).

Main

ROOTSIZE

- Logical Logs + [Phys logs] + [Temp Tables] + [Data] + [On-Arc.catalogs] + crtl info (res. pages)

This is a basic size calculation formula. For systems of any importance, most of these logical units should be on separate spindles.

SERVERNUM

- Unique for each database engine on CPU

CPU Usage

RESIDENT (§)

On a dedicated server turn this on (1). The parameter is ignored if it is not supported. In a mixed application environment this may cause Informix to consume more resources than desired.

NUMCPUVPS (*) (§)

- If there are more than 3 Hardware CPUs (HW CPUs) then set NUMCPUVPS to one less than the number of CPUs
- The number of scan threads (fragmentation issue) should be a multiple or factor of NUMCPUVPS. Determine this with **onstat -g glo**, **ath** or **rea**
- For data loading set this equal to the number of HW CPUs
- If HPL is used, then set this to equal the number of HW CPUs - CONVERTVPS (**onpload** parameter)

I have seen systems that prefer a 1:1 VCPUS to HW CPUs relationship, and I have seen some that prefer 2:1. If the number of HW CPUs is four or less, it is my recommendation that NUMCPUVPS should be set to 1 or 2. Overall, though, I have found no hard and fast rules for this important parameter and encourage experimentation for each system.

MULTIPROCESSOR and SINGLECPU

Both of these turn on different housekeeping mechanics. If NUMCPUVPS is set to one, it is **very important** to turn both of these off (0 and 1, respectively). You will see no benefit configuring a single CPUVP unless these are both off. This is more often than not overlooked! Both should really be considered as one parameter, if one is adjusted the other should be adjusted likewise.

NOAGE (§)

Set this to 1. If supported, this parameter turns off UNIX nicing (this is good). Nicing is the UNIX mechanism that lowers a process' priority over time to ensure equality in a mixed environment. A dedicated server should have NOAGE enabled. Tuning for a mixed environment is not so black and white. It may be best to have this turned off initially and then turned on if it is later desired to give Informix more CPU time.

AFF_SPROC (§)

- CPU number to start binding to

The **mpstat** command will provide the number of CPUs and the number that is to be used for the AFF_SPROC parameter. This numbering system seems whimsical.

AFF_NPROCS (§)

- Number of hardware CPUs to bind to
- NUMCPUVPS = HW CPUs - AFF_SPROC

This can be very beneficial for systems that are not dedicated database servers. Other non-Informix processes may be allowed to run on the HW CPUs identified here, but the CPUVPS will be restricted to those identified.

USEOSTIME

- Set to 0 as internal timer is faster

I have never seen OS timing used.

CPU Notes:
The command **onstat -g glo** should show a 10:1 ratio between time spent in UNIX user versus system states for CPUVPS (this has become perhaps 3:1 with KAIO). If system states is too high on a system with just a few HW CPUs, try using just 1 CPUVPS.

If **onstat -g rea** consistently shows multiple threads waiting, adding CPUVPS can often increase performance.

UNIX Commands:

```
mpstat
sar
uptime
```

A very marginal tool, `load average`, is a combination of system resource measurements. I have seen slow systems with a `load average` measurement of 2, and I have seen systems that seemed find in double digits.

`uptime` is only useful relative to an earlier measurement on the same machine, and then only as a broad indication of further measurements required.

Disk I/O

BUFFERS (pages) (*)

- **Key OLTP tuning field**
- Start with 20% RAM, maybe up to 50% RAM (on a dedicated machine why not start at 50%?)
- Increase this setting until the increase in cache hits is insignificant or excess system paging occurs. Use **sar** or **vmstat** to determine excess paging
- The target for OLTP environments is 95% read, 85% write cache hits
- Maximize for data loading (50% or more) (except HPL express mode)
- More buffers can mean longer checkpoints

NUMAIOVPS (*) - *Suggestions for this seem to change with the weather*

- 1 per database disk plus 1 for each chunk accessed frequently
- If KAIO is used, allocate 1 plus 2 for each cooked chunk (get rid of any cooked chunks)
- For KAIO systems, set to 2 for the OnLine engine plus 1 per controller containing cooked chunks
- For systems without KAIO, set to 2 for the OnLine engine plus 1 for each controller, then add as indicated
- 1 per dbspace
- 1 per disk
- 1 per mirrored pair
- 1 per chunk
- Note that DSA (Informix Dynamic Scalable Architecture™) spawns one read thread per dbspace (AIO or KAIO)

Use **onstat -g ioq** to monitor I/O queues and add as required. My suggestion is to get a system that supports KAIO and then set NUMAIOVPS to 2.

RA_PAGES

- Most machines limit the number of read ahead pages to 30 (all will limit at 32)
- For systems that perform light scans, do not set RA_PAGES higher than 32
- If this is set too high, it will lower %cached reads
- If **bufwaits** is unusually high, RA_PAGES may be set too high or the difference between RA_PAGES and RA_THRESHOLD may be too small

Set higher for typically sequential DSS databases.

RA_THRESHOLD

- Set this close to RA_PAGES (i.e., RA_PAGES 32 and RA_THRESHOLD 30), if **bufwaits** increase reduces RA_THRESHOLD
- Ideally RA-pgsused = (ixda-RA + idx-RA + da-RA)

DBSPACETEMP (*)

Configure at least two, each on a different spindle. Configure more if you are building large indices. DSS environments should hardware stripe a small number of TEMPDBS across multiple disks.

FILLFACTOR (indices)

- 90 is typical, 100 for tables where only DELETE or SELECT operations are performed
- This forces very compact indices & efficient caching initially
- 50-70% for tables with high INSERTS to delay the need for node splitting

MIRROR

Always mirror. A few years ago, the person that tests this at Informix posted on USENET his suggestion to **use hardware mirroring instead of Informix every time**. This makes sense. Hardware solutions are always faster than software. In order of preference, I would suggest hardware, OS, and then Informix mirroring. For machines where availability is paramount, you can mirror across controllers and even arrays.

IOSTATS

- When set to 1, this undocumented parameter will generate read and write timings in the **syschktab** SMI table (see Appendix C of the DSA Performance Tuning Training Manual)
- To reduce overhead, turn this off (0) when not taking measurements

TBLSPACE_STATS

- When set to 1, this undocumented parameter will generate **tblspace** profile statistics that can be viewed with **onstat -g ppf**
- To reduce overhead, turn this off (0) when not taking measurements

Disk I/O Notes:
Increasing the UNIX priority of AIO processes can improve the performance of data returned from disk.

Monitor I/O with **onstat -g ioq** (**iof** and **iov** are also worthy). When AIOs are used, `gfd len` should be less than 10 and `maxlen` less than 25. `maxlen` often breaks 25 during engine initialization (when it is unimportant), so make this distinction. **onstat -D** will show hotspots at disk level and **onstat -g ppf** at the partition level.

When building an important data warehouse for my previous employer, the Sun hotshot suggested placing all data in only the middle 2GB sectors of each 4GB disk, leaving the remaining unused. The highly paid Informix representative felt strongly that using only the leading 2GB of sectors would perform better. I suggested that we test, and if they were within 5% that the configuration be decided by ease of maintenance. **In this case**, the leading sectors proved 2% faster than the middle. I do not recall which I ended up implementing. The moral of the story is test, test, test. Take nothing at face value.

If your system is I/O bound, verify if it be controller or disk bound. The solutions are different.

Throughput = (pg_size * num_pgs_requested/max_transfer_rate) + latency

The maximum space required for an index build on non-fragmented tables is:
(key_size+4) * num_recs *2

The maximum space required for an index build on fragmented tables is:
(key_size+8) * num_recs *2

The use of clustered indices can greatly increase the performance of sequential reads. As Informix does not maintain the "cluster," dynamic tables will soon lose any advantage.

Informix recommends using fragmentation over hardware striping unless the table is a poor candidate for fragmentation.

UNIX Commands:
```
iostat
sar
vmstat
```

Logging

CLEANERS (*)

- 1 per disk if < 20 disks
- 1 per 2 disks if 20 to 100
- 4 per disk if > 100
- Configure at least one per LRU queue pair (this ratio has recently worked best for me)

If **-f** indicates that all cleaners are active, allocate more.

PHYSDBS

Place on a separate spindle

PHYSFILE

- = user threads * 5 (or size of most freq blob) * 4

PHYSBUFF

- = user threads * 5 * 4
- UNLESS tblspace blobs in database w/o logging, then: user threads * size of freq blob pg * 4

LOGFILES

- > 3

In all my manuals, I can find no good reference for performance tuning this parameter. I have no experience adjusting this except for eliminating anomalous behavior.

LOGSIZE

- > 200kb
- Keep this small for greatest recovery if your tape device is slow, or your blob-pages are volatile
- (user connects * maxrows (in one trans)) * 512

LOGBUFF

- Size of 3 logical log buffers in RAM
- Determines frequency of flushing to disk

LOGSMAX

- LOGFILES + 3

CKPTINTVL

- This adjusts the interval of checkpoints, see LRU
- For data loading (except HPL express mode) and parallel index builds set at 3000
- A large interval will allow the size of the physical log to determine when checkpoints occur (i.e., amount of work accomplished)
- **onstat -F** will show if writes are LRU-driven or CHKPTINTVL-driven
- **onstat -l** and **-m** will determine if checkpoint interval is driven by physlog = 75% full or this parameter

LRUS (*)

- Max (4, NUMCPUVPS)
- If **onstat -R** shows #dirty pages > LRU_MAX, add LRU Qs. If no change, increase CLEANERS.
- 4 per CPUVP (this ratio has recently worked best for me)
- 1 per 500-750 buffers, up to 128 (max)
- More LRUs better support a large number of users by reducing **buffwaits**

Monitoring **onstat -g spi** shows contention for individual LRU queues.

LRU_MAX[MIN]_DIRTY - *%buffers assigned to modified queue*

- Lower this setting to decrease checkpoint duration
- For data loading (except HPL express mode) and parallel index builds 70 and 80% will allow to almost fill before flushing

Monitor **onstat -R** queue length and **-F** writes forced by this parameter by CLEANER thread

LTAPEDEV

Set to /dev/null, Informix will just mark the LL as backed up, not even going through the motions. MUCH faster. This should be done only on systems where logical log recovery is not required, and the user has been made aware of the implications.

LTXHWM

- 50

LTXEHWM

- 60

Logging Notes:

If the physical log frequently fills, decrease CKPINTVL.

Physical logging - buffsize/pages I/O should be greater than 75%. If this is near 100%, increase PHYSBUFF size. Physical and logical log buffers should be about 75% full when flushed.

Huge bang for the buck makes checkpoints another early thing to look at.

Network

NETTYPE

- 1 if CPU, assign additional poll threads to NETVPs
- 300 for a single HW CPU, 350 if more
- For data loading, one per CPUVP. Each poll thread should be on a CPU class VPS (running a poll thread *in-line*)
- Do not increase user connects as this will increase work for the poll thread

Network Notes:
Be careful of network congestion masquerading as a slow database. If measurements show no bottlenecks on the "system under test" on a system with large connectivity, check the comms link. 60% or greater line utilization indicates saturation levels sufficient enough to cause network-induced delays.

UNIX Commands:
`netstat`

Session

LOCKS

- Maximum number consumed by any query multiplied by the number of concurrent users

Memory

SHMVIRTSIZE (kilobytes) (*)

- This is a key tuning parameter for decision support systems (DSS)
- DSS may be up to 75% RAM if paging is not induced

For OLTP environments, create a large resident (buffers) and a small Virtual (SHMVIRTSIZE). For DSS environments, this is reversed.

SHMADD (kilobytes)
10 - 20% of SHMVIRTSIZE. Too many added shared memory segments used to bring a system to its knees, although this has been mitigated with releases 7.2 and later.

SHMTOTAL (§)

- Set to 0 for unlimited

This can be used to make Informix more polite, reserving resources for other applications. I have also had to use this when a failing **malloc** panics **oninit**. Since consolidating shared memory segments can bring a huge bang for the buck, one of the first things I check is **onstat -g seg**.

Note that **onmode -F** (used to free memory segments) can cause system failures on an active system and should be avoided.

Memory Notes:
To calculate shared memory segments corresponding to a database instance:

```
shmid - 52564801*.0001 = SERVERNUM
```

This is very useful when cleaning up after an engine crash on a system that is shared with other Informix instances.

UNIX Commands:
`Sar -g 5 5` (will show paging activity)
`Vmstat`

DSS Parameters

PDQPRIORITY
- If > 0, this will enhance parallelism of index builds (which, after 7.2 are always parallel to some degree)

MAX_PDQPRIORITY
- 100

DS_TOTAL_MEMORY
- DSS should be 90% of SHMVIRTSIZE

DS_MAX_SCANS
- 1048576

DSS Notes:
A quantum can be calculated with

(PDQPRIORITY/100)*(DS_TOTAL_MEMORY/DS_MAX_QUERIES).

Each sort thread gets quantum/#sort threads memory.

A users' effective priority is (pdqpriority/100) * (MAX_PDQPRIORITY/100) where pdqpriority is set by the environment variable or the SET PDQPRIORITY statement.

Debugging/Recovery

DUMPSHMEM

Zero. As many systems that I have seen fail due to Informix filling /tmp, the "out of the box" default should be off.

Data Replication

DRINTERVAL

- Delays on the primary database caused by synchronizing to the secondary database can be reduced by adjusting this parameter down
- Smaller intervals equal more frequent synchronizing, while larger intervals will reduce frequency (at the cost of duration)

DRTIMEOUT

- May need to increase this in a WAN (Wide Area Network) environment

A datascope or OS comms trace can prove invaluable in tuning these two parameters.

Miscellaneous Notes

oncheck -pr can be used to replace lost **onconfig** files.

Scans and Sorts for index builds are always parallel with 7.2. Index builds on fragmented tables add parallel B-(sub)tree builders.

UPDATE STATISTICS - the single most important SQL statement for query performance. Automate this to run nightly via cron.

Use HIGH for:

- Lead columns in each index
- All columns queried with equality filters (=)
- All join columns
- First column to uniquely distinguish a composite index from another on the same table and all columns preceding

Use MEDIUM for:

- All other columns

Use LOW for:

- All index columns not run through on HIGH

To speed up UPDATE STATISTICS, set PSORT_NPROCS to 2, use DBSPACETEMP, and do NOT set DBUPSPACE (limits RAM for UPDATE STATISTICS).

Database Tuning the Informix Way

The following came from a class handout. I have made significant changes. As an aside, this exercise is one of the best learning opportunities a DBA will get.

1. Set user expectations.
2. Build your test environment, ideally using the target system. If this is not possible, build an **exact** duplicate.
3. Establish performance objectives.
4. Measure database activity and use of resources, enlisting the aid of a seasoned System Administrator.
5. Adjust user expectations as required.
6. Review application programs to make sure appropriate access methods are used to retrieve data and algorithms are efficient.
7. Identify performance problems such as excessive use of CPU, memory, or disks.
8. Have the System Administrator tune the operating system.
9. Measure database activity and use of resources, enlisting the aid of a seasoned System Administrator.
10. Tune Informix Dynamic Server.
11. Measure database activity and use of resources, enlisting the aid of a seasoned System Administrator.

12. Optimize the placement of logs, sort space, and temporary space.
13. Measure database activity and use of resources, enlisting the aid of a seasoned System Administrator.
14. Optimize table placement, sizes of extents, and fragmentation.
15. Measure database activity and use of resources, enlisting the aid of a seasoned System Administrator.
16. Make sure the indices are appropriate.
17. Measure database activity and use of resources, enlisting the aid of a seasoned System Administrator.
18. Configure an UPDATE STATISTICS command and schedule it for off-peak hours.
19. Measure database activity and use of resources, enlisting the aid of a seasoned System Administrator.
20. Optimize background activities such as logging, checkpoints, and page cleaning.
21. Measure database activity and use of resources, enlisting the aid of a seasoned System Administrator.
22. Schedule backups and batch jobs for off-peak hours.
23. Measure database activity and use of resources, enlisting the aid of a seasoned System Administrator.
24. Repeat steps 3 through 23.

It is important to set aside as much time as possible for this task. Three long days is a minimum. In a development environment, ensure that two weeks are dedicated to this at the end of each project. This will invariably get squeezed to three days, which, as noted above, is the absolute minimum to perform a decent job.

Summary

I have compiled these Informix configuration parameters and suggested settings over the years. I keep a copy in my organizer for easy field reference.

Remember: test, test, test and have fun!

This document is maintained at www.bamph.com/informix.htm. A version suitable for downloading into a PDA or notebook can also be found there.

About the Author

Bill Burton is a founder of bamph. He has been working with databases since the early 1980's. Bill is an Informix Certified Professional whose previous clients include PeopleSoft, Chase Manhattan Bank, The Money Store, and DHL.

Having recently sold his hot rod Harley, Bill's current toy is a 1966 Jeep CJ5 with a fuel-injected Chevy 327, 4:88 gears, and a 4-speed Muncie. His bicycle has disc brakes. Bill can be reached at www.bamph.com and bill@bamph.com.

About bamph

bamph is a Web design company based in Sacramento, CA. Whether it be Web site development, database design, graphic art, cold fusion programming, hosting, or off-site support, bamph serves all your Internet needs. bamph can be found at www.bamph.com. Good enough isn't.

Bibliography

INFORMIX-OnLine Dynamic Server Performance Tuning Training Manual, Liz Suto, Course Designer (Informix writes the best technical manuals I have seen in my 23 professional years. This is my favorite. Thanks Liz!)

Tuning Informix Dynamic Server and Your System for Optimum Performance, Art S. Kagel, *Tech Notes*, Volume 8, Issue 3, 1998 (Art is a valued and frequent contributor to the comp.databases.Informix newsgroup, a wonderful resource)

Informix Performance Tuning, Second Edition, Liz Suto

Performance Guide for Informix Dynamic Server V 7.3

My dear friend James Larned contributed to this document. I would also like to thank George Clark for being such an inspiration so many moons ago.