**[Increasing Buffers and Reducing I/O with Informix](http://www.informix-dba.com/2010/07/increasing-buffers-and-reducing-io-with.html)**

You can improve performance dramatically by making some simple changes to the ONCONFIG and bouncing the engine. The first and most important change to make is to increase Buffers, this is where Informix will be caching all of the data you are reading in from disk. More data cached means less I/O and less I/O has to mean better performance and since I/O will most likely be your current bottleneck, reducing I/O should result in the most performance gain.  
  
**How Informix Caches Data**  
  
  
When you execute SQL Informix determines what pages on disk it needs to read or write to satisfy this SQL request.  Informix will first check to see if a page it needs is in memory and if it is not it will read the page from disk and put it into memory.  This memory cache is made up of many buffers called a buffer pool.  Each buffer is the size of a page on disk.  So far we've only used the default page size of 2 KB for our dbspaces, so our buffer pool is made up of many 2 KB buffers.  
  
**LRU Queues**  
  
The size of the buffer pool is static and created at engine start up.  After some time the buffer pool will become full of pages read in from disk and new reads will need to kick some pages out of memory to make room.  Informix doesn't just arbitrarily pick pages at random to kick out of the pool, that would be stupid.  Informix assumes that the pages in memory that have not been used in a while aren't likely to be used again anytime soon and chooses these pages to be replaced by the new pages you need in the buffer pool.  
  
The bufffers in the buffer pool are divided evenly among multiple Least Recently Used Queues (LRU), when a buffer is used by a SQL request it is moved to the end of the queue putting the most recently used pages at the tail of the queue.  When a page needs to to be kicked out of the buffers a page at the head of a LRU queue is picked.  For the record, I am oversimplifying the LRU queues, there is more going on here like FIFOs, buffer priority and Free/Modified LRU queues that make up a LRU queue, but for now this is all you need to know about the anatomy of a LRU queue.   
  
**Dirty Pages in LRU Queues**  
  
When Informix needs to kick a page out of memory to make room for a new page it prefers to pick a page that has not been modified.  When Informix does this it can simply replace the data in the buffer with the new data.  Since this page has not been modified it doesn't need to do anything else, if the page had been modified Informix would need to write the changes to disk.  When there are no unmodified pages in the LRU queue Informix performs a Foreground Write, a single write of a modified page to disk, before it replaces the data in the buffer.  These are bad for performance and you don't want to see them.  Monitor Foreground Writes by executing onstat -F, the number of Foreground Writes will appear at the head of this output.

informix> onstat -F

IBM Informix Dynamic Server Version 12.10.UC7IE -- On-Line -- Up 2 days 18:22:18 -- 144148 Kbytes

Fg Writes     LRU Writes    Chunk Writes

7             146           3169

address  flusher  state    data     # LRU    Chunk    Wakeups  Idle Tim

4aefb620 0        I        0        0        347      239020   238928.516

4aefbc28 1        I        0        0        1        238674   238927.649

**Avoiding Foreground Writes**  
  
You can prevent Foreground Writes (FG) by configuring LRU writes.  A LRU write is more efficient than a FG write because it is performed by a page cleaner thread in the background and not by your session thread as in a FG write.  LRU writes are triggered when a LRU queue reaches a maximum percent dirty that you configure and continue writing modified pages to disk until the LRU queue reaches a minimum percent dirty that you configure.  
  
LRU writes also minimizes the number of pages written to disk at Checkpoint time.  During a checkpoint all modified pages are written to disk to provide a consistent point of recovery in the event of a failure.  Keeping the number of pages written during a checkpoint low is not as important as it used to be before non blocking checkpoints came around.  During a blocking checkpoint no pages can be modified so all sessions attempting to insert, update or delete data block until the checkpoint completes.  This behavior resulted in many OLTP engines getting configured with LRU max dirty of less than 1% to keep the checkpoint times short.  With the introduction of non blocking checkpoints checkpoint time isn't as much of a concern and LRU max dirty of 20% to 70% is more common.  
  
**How Many Buffers Do I Need?**  
  
The more the merrier, until you reach the point of dimishining returns.  If your data size is only 1 GB then a buffer pool of 2 GB is just going to be 1 GB of wasted memory.  Also, if you only routinely access 1 GB of a 100 GB data set then 1 GB of your 2 GB buffer pool will be holding cached data that isn't accessed very often.  Figuring out the optimal buffer pool size for your data and your available memory resources is a balancing act.  
  
If you are using the free for production release of Informix called Innovator-C you are limited to 2 GB of shared memory for the engine.  This shared memory limitation applies to all of the memory Informix uses, not just buffers.  Typically the memory Informix uses is made up of 2 areas, the Resident portion which includes the buffer pool and the Virtual portion which includes memory required for session threads, sorting, grouping and other things.  An optional 3rd and much smaller portion is created if you use shared memory connections (we have not configured any of these yet).  For starters we will configure 50% of our available 2 GB for buffers.  At 2 KB per buffer this is 524288 buffers.  
  
**How Many LRU Queues Do I Need?**  
  
There are many schools of thought on this one.  The manuals say at least 4 LRU queues and no more than CPUVPs configured.  For Innovator-C that is limited to a max of 4 CPUVPs  this is a recommended setting of 4.  The default ONCONFIG setting contradicts the manuals and suggests a value of 8 for the 1 CPUVP it has configured.  I think these are too low because the penalty for under configuring LRUs is greater than the penalty for over configuring LRUs when you have a lot of concurrent sessions.  An older suggestion is to configure 2000 buffers per LRU, but I think this results in too many LRUs.  The real answer is to pick something in the middle and monitor the Bufwaits Ratio (BR), a formula devised by the infamous Art Kagel that can be calculated from the onstat -p output.

informix> onstat -p

IBM Informix Dynamic Server Version 12.10.UC7IE -- On-Line -- Up 2 days 20:00:34 -- 144148 Kbytes

Profile

dskreads   pagreads   bufreads   %cached dskwrits   pagwrits   bufwrits   %cached

1220       1295       219268     99.45   5449       9476       44802      87.84

isamtot    open       start      read       write      rewrite    delete     commit     rollbk

143006     11032      14820      56477      12433      845        337        2643       3

gp\_read    gp\_write   gp\_rewrt   gp\_del     gp\_alloc   gp\_free    gp\_curs

0          0          0          0          0          0          0

ovlock     ovuserthread ovbuff     usercpu  syscpu   numckpts   flushes

0          0            0          6.03     32.56    704        356

bufwaits   lokwaits   lockreqs   deadlks    dltouts    ckpwaits   compress   seqscans

44         0          62182      0          0          6          1449       3724

ixda-RA    idx-RA     da-RA      RA-pgsused lchwaits

51         0          13         64         11123

Bufwaits Ratio (BR) = (bufwaits / (pagreads + bufwrits)) \* 100 or (44 / (1295 + 44802)) \* 100 = 0.095%  
  
Shoot for a BR less than 7% (lower the better) adding LRUs as needed.  We'll start with 63 LRUs.  
  
**How Dirty Should My LRU Queues Get?**  
  
With the addition of non blocking checkpoints super dirty LRU queues that are flushed at checkpoint time are not as big of a deal as they used to be.  Checkpoint writes are more efficient than LRU writes and LRU writes are more efficient than Foreground writes, so the goal is to encourage Checkpoint writes while limiting Foreground writes via LRU writes in between checkpoints.  There is no formula for figuring this out, you will have to monitor how buffers are being flushed to disk via onstat -F and adjust LRU min/max dirty accordingly.  Flushing more aggressively when we have Foreground writes and less agressively when we don't.  We'll start with a LRU max dirty of 60% and a LRU min dirty of 50% (which happen to be the default values).  
  
**Making the Configuration Changes**  
  
Now for the easy part.  These configuration changes are made by modifying the BUFFERPOOL ONCONFIG parameter and bouncing the engine.

informix> vi $INFORMIXDIR/etc/$ONCONFIG

BUFFERPOOL      size=2K,buffers=524288,lrus=63,lru\_min\_dirty=50.000000,lru\_max\_dirty=60.000000

informix> onmode -ky

informix> oninit -v

**Do I have the Right Number of Buffers Configured?**  
  
Scenario 1:  Too many buffers configured  
  
If you have too many buffers then you are needlessly wasting memory that could be used for other things (Informix sorting, grouping and session memory or the OS and other applications).  
  
The first place to check is the onstat -P output after your engine has been running under normal load for a while.  If you have more buffers configured than you have data then there will be some buffers that are marked as Other, these are the unused buffers.

informix> onstat -P | tail -5

Percentages:

Data  32.04

Btree 18.02

Other 49.94

In this example 49.94% of my buffer pool is unused, I can probably safely reduce the size of my buffer pool by 50% and not suffer any loss in performance.  
  
If all of your buffers are in use and you need to reduce the memory footprint of your Informix engine you can use the output of onstat -p to monitor your Read Cache Hit % (this will be the first output labeled %cached and is the number of times you needed to read a page and it was in memory, the second %cached is your Write Cache Hit %).  If this number is very high (above 98%) you might be able to reduce the number of buffers without suffering much performance loss.  However, I would only reduce buffers if I couldn't reduce shared memory any where else.  
  
Scenario 2: Not enough buffers configured  
  
This is the most likely scenario as your working set of data will typically be larger than your buffer pool.  Use the same onstat -p command to monitor the Read Cache Hit % and continue to add buffers until you reach an acceptable percentage.  95% or better is great, above 90% is good and anything less than 90% is probably a sign of needing more buffers.  
  
That's it for basic buffer tuning, not much to it really.