An IBM White Paper
Informix Dynamic Server

Automatic ReadAhead

Scott Lashley, STSM, Informix Kernel Architect

12/2/2011

Contents

Introduction

The Informix server continues its dedication to making it the industry standard for ease of use. With the introduction of Automatic ReadAhead in Informix 11.70.xC4, we've eliminated another tuning knob while also improving performance. In this white paper we'll discuss what ReadAhead is, when it occurs and how you the DBA can monitor ReadAhead to help better tune your system.

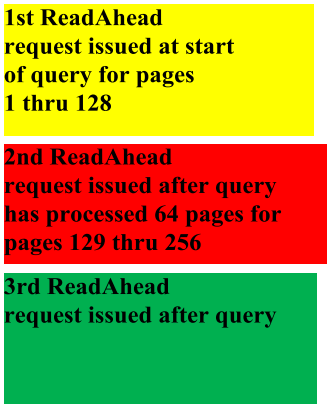
What is ReadAhead?

ReadAhead is a function of the Informix server that anticipates the data needed to satisfy a query and asynchronously issues I/O read requests to bring the data into the bufferpool. By issuing the I/O requests asynchronously, the processing necessary for reading the requests into the bufferpool overlaps with the processing being done by the query.

The Informix server supports several different types of ReadAhead.

Data ReadAhead

Data ReadAhead is used when a query is doing a sequential scan on a partition. The query is going to scan all the data pages starting at the beginning of the partition and scanning to the end of the partition. ReadAhead will submit an asynchronous request for a whole bunch of pages, starting with page 1. As the query is processing the data and consuming the pages, the query will continue to submit subsequent asynchronous requests for more pages when the pages left to process from the previous request are sufficiently depleted.



Firgure 1

Figure 1 is an example of data ReadAhead using 128 page I/O requests.

Index ReadAhead

Index ReadAhead is used when a query is doing a key only range scan. A key only range scan is a scan that requires only the data columns contained within the index and does not require any additional columns from the table. The Index ReadAhead scan uses the TWIG level of the btree to issue asynchronous I/O requests for the LEAF pages of the btree. ReadAhead requests are bounded by the range of the scan.

Figure 2 is a drawing of a typical Informix btree. Consider the following key only scan:

```
select c1 where c1 < 'Q'
```

We start the scan by traversing down the left side of the btree. When we come to the TWIG level[1] we initiate ReadAhead. In Node 4, we see that it points to Nodes 2 and 3 so we issue asynchronous I/O calls for those 2 nodes. We then slide to the right from Node 4 and read Node 7 and issue more asynchronous I/O calls for LEAF pages 5, 6 and 8. Asynchronous requests will continue to be generated until we reach TWIG node 11 and LEAF node 9 because LEAF node 9 contains our stop key 'Q'.

Index/Data ReadAhead

Index/Data ReadAhead is used when a query is using an index to access a table and the query requires data from the table that isn't contained within the index. This type of scan uses Index ReadAhead to traverse through the Index just like a key only range scan. Whenever the range scan encounters a new LEAF node, the scan will also submit asynchronous I/O requests for all the data pages represented by all the rowids that the keys point to.

What's new with Automatic ReadAhead?

Automatic ReadAhead provides several new features that make queries run faster without requiring the DBA to tune any knobs.

Automatic ReadAhead

Prior to 11.70, the DBA could turn ReadAhead on for all queries by setting the ONCONFIG parameters RA_PAGES and RA_THRESHOLD. All table and range

[1] The LEAF level is the bottom of the btree. The TWIG level is the level just above the LEAF level.

This is a global setting impacting all users and queries.
SET ENVIRONMENT AUTO_READAHEAD
This is a session specific setting.

The simplest way to configure ReadAhead is to set it in the ONCONFIG file.

Ex) AUTO_READAHEAD 1

For those who are upgrading from a previous version of Informix and are using ONCONFIG parameters RA_PAGES & RA_THRESHOLD, RA_PAGES will be honored and internally converted to using Automatic ReadAhead.

We've decided to simplify the configuration by no longer honoring RA_THRESHOLD (it is deprecated). In our testing we found that in most cases, performance remained the same when the threshold for starting the next set of asynchronous I/O requests is ½ of the total number of pages to be read (in other words, ½ of RA_PAGES).

For more information on all the possible settings, click the links above to take you to the Informix 11.70 documentation.

How do I monitor ReadAhead?

We've added a new onstat option to monitor ReadAhead:

onstat -g rah

What does it mean to have effective read ahead?

In the onstat –g rah output, there is a stat column called ***eff***. The range of values for this column is 0 to 100. The higher the value, the more effective ReadAhead is. A higher value means that the asynchronous I/O requests are not finding the pages in the bufferpool and therefore ReadAhead is being effective in helping improve performance.

My onstat –p statistics are different, why?

Prior to Automatic ReadAhead, when the Informix instance was configured with ReadAhead on, every table and range scan employed ReadAhead regardless if the data was cached in the bufferpool or not. With Automatic ReadAhead, ReadAhead won't be employed by a query until the query encounters I/O from disk. That means that some queries may never do ReadAhead where as before, they always did ReadAhead. Because of this, onstat –p output can yield different results from prior Informix versions when running a similar workload.

How can I use ReadAhead to help tune my Informix server for optimal query performance?

Tuning ReadAhead falls into two areas:

1. Tuning specific queries
2. Tuning the overall system

Turning specific queries

In most cases, the default setting of 128 pages is sufficient. But, there can be situations where an additional performance gain could be achieved by increasing the number of pages to be submitted. The most likely situation where a larger request would be beneficial is when the I/O subsystem is somewhat slow or busy and the processing of the data is very fast. Let's use the example of…

```
select count(*) from t1 where c1 !=
NULL
```

The predicate processing is a very simple expression (!= NULL) and the number of rows returned is 1 (count *). The overall amount of processing of this query is very small making the processing of the data demanding. If t1 is a large table, this query will be dominated by I/O. One would expect the CPU to be able to process the data much faster than the data can be read from disk. In most cases, requesting 128 pages per ReadAhead request should drive the query at disk speed. But, if the I/O subsystem is slow or very busy, sometimes it is more efficient to submit more pages. To test this on your system, use the SET ENVIRONMENT statement to set the ReadAhead settings for a specific session to see if the performance of the query improves.

When doing FIRST N queries, the query won't require all the possible data. In order to save on unneeded I/O, it may be beneficial to scale down the number of pages to be read ahead since they won't be required to satisfy the query.

Tuning the overall system

In the vast majority of cases, setting AUTO_READAHEAD to 1 is enough. For existing systems that had ReadAhead turned off[2], those systems should now be able to have ReadAhead turned on and utilize the ReadAhead functionality without paying any penalty to queries where the data is already cached.

When it comes to tuning, changing the ReadAhead configuration will most likely have little or no impact to the overall performance of most queries. But, the ReadAhead statistics can be very useful in determining what's happening as queries are executing. When the working set[3] is large and most of the queries access data within the working set, then you should expect a high cache hit rate and the ReadAhead efficiency to be low because very few pages will need to be read from disk. In this case, having a low ReadAhead efficiency isn't a bad thing, it's a good thing because there is little to no disk I/O activity. If this scenario describes your system, there are 2 things you can do to further improve performance:

1. Increase the bufferpool size to increase the size of the working set to make sure queries that are doing disk I/O will find the data they need in the bufferpool.
2. If queries are doing occasional disk I/O, those individual sessions that are executing those queries may change the AUTO_READAHEAD setting from Passive (1) to Aggressive (2) to make sure the occasional disk I/O has an asynchronous I/O request submitted. Note that the Aggressive setting causes queries to always issue ReadAhead requests regardless if the data is cached so that some queries which are highly cached might encounter more CPU overhead to perform the ReadAhead requests. Aggressive mode causes ReadAhead to behave similar to previous versions of Informix that used ReadAhead.

If you have queries that don't access data in the working set, then you should see a high efficiency value in onstat –g rah. To improve performance for these types of

[2] ONCONFIG parameter RA_PAGES = 0 will turn off ReadAhead.

[3] Working Set Definition

queries, the best solution is to change the query to a light scan[4]. This will make sure the working set remains intact while providing good query performance. If that's not possible, then ReadAhead is an excellent alternative.

Further questions can be directed to the author.

[4] Light Scan