04 April 2012

|  |  |  |
| --- | --- | --- |
| **Applies To** | InterSystems Caché 2010 | |
| **Printed** | 17/04/2012 | |
| **Changes** | 04/04/2012 | Initial release |
|  |  |
|  |  |

# Summary

## Tools

TrakCare and TrakCare Lab systems require the necessary Input/Output (IO) performance characteristics acceptable user experience. This document is a guide to InterSystems tools that can be used to exercise system IO and understand IO performance and throughput characteristics. using Caché:

* Perftools.RANREAD - random read
* Perftools.JRNSYNC – (ECP only) ECP induced JRNSYNC requests
* GMash – database write cycle

While the tools use Caché to drive IO they are not a TrakCare application test.

This document details how to install and run the tools, and gives general guidelines on interpreting results but it is important to remember results gathered from the IO tests will vary from configuration to configuration based on the IO sub-system put in place. Be sure to discuss the expected results with the primary Technology Architect responsible for the configuration, as the system is designed for specific performance and throughput.

Any abnormalities or performance issues discovered should also be referred to the system vendors or system integrators for validation using their tools and remedy.

This document assumes that it was distributed with the following tools to be imported as detailed in the install instructions in following sections:

* ZPERFTOOLS.rsa
* suites.rsa

## pButtons

InterSystems pButtons is a performance monitoring tool that has been designed to be a non-invasive utility to measure database usage and operating systems statistics. pButtons is the tool used for capturing Caché database performance statistics and is configured on all servers where Ensemble is installed, i.e. DB, Application, Report and BI servers. The pButtons tool is included in the Ensemble 2010 release.

For more information about pButtons please see Caché DocBook:

<http://docs.intersystems.com/cache20101/csp/docbook/DocBook.UI.Page.cls?KEY=GCM_pbuttons>

To download the latest pButtons utility and documentation go to:

<ftp://ftp.intersystems.com/pub/performance/>

NOTE: Versions of pButtons after 16-Nov-2011 include Windows Performance Monitor (perfmon) data. Before configuring Windows perfmon check to see if pButtons will automatically run perfmon. Older versions of pButtons will not collect operating system statistics on Windows servers so perfmon.exe needs to be configured manually for the same duration and time interval as pButtons.

# Database Random Reads

^Perftools.RANREAD utility is used to generate random reads on one or more databases at the same time.

## Installation and configuration

Create an empty (pre-expanded) database called ZREADTEST approximately twice the size of physical memory in the disk group to be tested, i.e. where there TrakCare production databases are located.

Import **ZPERFTOOLS.rsa** which contains Perftools routines into **%SYS** namespace

## Methodology

The Technology Architect will have suggested disk group, database size, and number of jobs based on storage details (Controller cache size, number of disks in disk group, RAID type). An acceptable response time will be stated. For example:

|  |  |  |
| --- | --- | --- |
|  | Value | Example |
| Disk Group |  | TRAKPROD (8 Disk RAID 10) |
| Database size |  | 40 GB |
| Number of jobs (start) |  | 9 |
| Target IOPS Y5 |  | 400 |
| Peak IOPS disk group |  | 1400 |
| Target service time (ms) |  | 6 |

As a guide the following response times are usually acceptable;

* Systems with ECP: 4-6 millisecond (0.004-0.006) average random read service response times
* Systems without ECP: up to 10 millisecond (0.010) average random read service response times

The goal of this utility is to drive as many jobs as possible to achieve target IOPS and ensure acceptable disk response times are also sustained.

* Start operating system monitoring (windows PERFMON, nmon, iostat, etc) to monitor IOPS
* Starting at a suggested number of jobs (above) step up/down jobs until. For example, Recommend to start with 1 process and 10,000 iterations and then increase the processes to 2, 4, 8, 16 and so on.
* Service times start to increase beyond acceptable range/number of aggregate IOs too low, its good to know where the breaking point of a particular storage is.
* Maximum IOs at the acceptable response time range is the maximum supported random reads for the configuration.
* Compare that to the expected results from the Technology Architect.

The utility keeps response times in the **^%ZPERFTOOLS** global.

IOPS can be monitored using operating system utilities such as **iostat, nmon** or Windows **PERFMON.EXE.**

**IMPORTANT: Between runs the ^%PERFTOOLS global should be killed.**

## Example

An example: If this is a 16-disk RAID10 with 15K drives we would expect to get optimal response and throughput of ~2500-3000 random reads per second with 16-20 jobs and 3-5ms average response time.

Below is an example of bad results.

%SYS>**D ^****Perftools.RANREAD**

Database directory? ?

1) \mgr\

:

:

7) \mgr\user\

Database directory? 7 \mgr\user\

How many processes to start? 100

How many iterations? 1000

Starting 1 jobs in the background.

Set global node ^%ZPERFTOOLS("RANDREAD")=0 to terminate jobs.

%SYS>**d ^%G**

For help on global specifications DO HELP^%G

Global ^%ZPERFTOOLS("RANREAD",

^%ZPERFTOOLS("RANREAD","BATCH","^^\mgr\user\",2,1,"Response Time")="104.7 (ms)"

^%ZPERFTOOLS("RANREAD","BATCH","^^\mgr\user\",2,2,"Response Time")="107.07 (ms)"

^%ZPERFTOOLS("RANREAD","BATCH","^^\mgr\user\",2,3,"Response Time")="108.73 (ms)"

^%ZPERFTOOLS("RANREAD","BATCH","^^\mgr\user\",2,4,"Response Time")="110.25 (ms)"

^%ZPERFTOOLS("RANREAD","BATCH","^^\mgr\user\",2,5,"Response Time")="110.89 (ms)"

^%ZPERFTOOLS("RANREAD","BATCH","^^\mgr\user\",2,6,"Response Time")="111.71 (ms)"

The example above started 100 processes each trying to do 1000 random reads each on the “user” database, and as the results show, the average response time achieved as ~110ms (very poor results).

In this case process count was lowered until the system is able to sustain an average of 6ms response time for each process.

Once average 6ms response is reached the number of IOPS capable can be seen.

## Database Writes

^GMash is used to produce a large number of write updates. ^mgstat is used to monitor the physical write rate.

### Installation and configuration

* Create an empty (pre-expanded) database and namespace of 20 MB in the disk group to be tested (eg. TrakCare production disk group). For example ZWRITETEST. The database is not used but you want to run GMash in the namespace in the disk group.
* Import **suites.rsa** Which contains GMash routine into newly created namespace

### Methodology

For a single database or write daemon, expect to sustain 1000+ physical writes per second.

* Before starting ^GMash, be sure to start a ^mgstat collection with 1-second intervals.
* Start operating system level monitoring eg. nmon, iostat, etc
* Then run ^GMash, and only start 10 processes because it is intensive, eg **Start^Gmash(10,1000)**.
* Start - Start^GMash(jobs,limit)

jobs = number of jobs to start

limit = maximum total global storage for all jobs (Mb)

* Monitor - w $$Status^GMash
* Stop – Stop^GMash
* After running for 5-10 minutes stop ^GMash and collect the mgstat data

NOTE: Although the database created during GMash will not be very large a very large amount of journal files will be created, expect several GBs per minute.

^mgstat data can be collected by output to a csv file,

eg: **d ^mgstat(1,600,/outputpath/outputfile.csv)**

### Example

When looking at the ^mgstat data, pay close attention to the PhyWrs and WIJwri columns. The PhyWrs column should report a large spike in activity (thousands of writes/second) and drop back to 0 in a few seconds. It should NOT report a “trickle” effect of writes – meaning a few hundred or less and takes a long time to complete. Below is an example of a well setup configuration:

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Date | Time | Glorefs | GRratio | PhyRds | Rdratio | Gloupds | Rourefs | RouLaS | **PhyWrs** | WDQsz | WDphase | **WIJwri** |
| 2/10/2009 | 12:52:41 | 124264 | 0 | 0 | 0 | 44690 | 0 | 0 | 0 | 6639 | 0 | 0 |
| 2/10/2009 | 12:52:42 | 161159 | 0 | 0 | 0 | 48315 | 0 | 0 | 0 | 2349 | 5 | **292** |
| 2/10/2009 | 12:52:43 | 115573 | 0 | 0 | 0 | 42248 | 1 | 0 | 0 | 7833 | 7 | 0 |
| 2/10/2009 | 12:52:44 | 67132 | 0 | 0 | 0 | 29906 | 0 | 0 | **9030** | 2962 | 7 | **294** |
| 2/10/2009 | 12:52:45 | 107312 | 0 | 0 | 0 | 33073 | 0 | 0 | 0 | 5204 | 7 | 0 |
| 2/10/2009 | 12:52:46 | 24262 | 0 | 0 | 0 | 10395 | 632 | 5 | **9051** | 6035 | 0 | 2 |
| 2/10/2009 | 12:52:47 | 95716 | 0 | 0 | 0 | 28850 | 0 | 0 | 0 | 4442 | 0 | 0 |

In the example above the system is efficiently flushing ~9,000 blocks (70MB) in less than 1 second. This is very good throughput. Note that the WDPhase column for timestamp 12:52:44 reports a value of 5. This is significant in indicating the throughput capability. Not only did the WRTDMN write the 9030 blocks during that second, the system also wrote 294 WIJ blocks for the NEXT cycle in that same second.

It should also be noted that the buffer pools used during this test was smaller than normally configured in order to trigger the write daemon cycle more often and exercise the writes more frequently.

## ECP Journal Sync Throughput

Load ^Perftools.JRNSYNC utility onto one or more application servers (ECP clients) and start up 10 to 100 jobs on each. This will generate a significant JRNDMN write rate on the database server. Measure the physical write rate using OS tools (iostat, sar, or nmon) and the write rate should be in the 2000-3000 range. Anything less than 1000 is a problem.

Below are some examples of both good and bad results.

[GOOD RESULTS]

**REMOTE>d ^Perftools.JRNSYNC**

**How many processes to start? 50**

**How many interations? 1000**

**Starting 50 jobs in the background.**

**Set global node ^%ZPERFTOOLS("JRNSYNC")=0 to terminate jobs.**

**REMOTE>D ^%G**

**For help on global specifications DO HELP^%G**

**Global ^%ZPERFTOOLS("JRNSYNC" -- NOTE: translation in effect**

**^%ZPERFTOOLS("JRNSYNC","BATCH",2,1,"Response Time")=".081 (ms)"**

**^%ZPERFTOOLS("JRNSYNC","BATCH",2,2,"Response Time")=".114 (ms)"**

**^%ZPERFTOOLS("JRNSYNC","BATCH",2,3,"Response Time")=".218 (ms)"**

**^%ZPERFTOOLS("JRNSYNC","BATCH",2,4,"Response Time")=".241 (ms)"**

**Note: There will be 50 entries (one for each process), but I only listed the first 10 for display purposes in this document.**

**^%ZPERFTOOLS("JRNSYNC","BATCH",2,5,"Response Time")=".243 (ms)"**

**^%ZPERFTOOLS("JRNSYNC","BATCH",2,6,"Response Time")=".233 (ms)"**

**^%ZPERFTOOLS("JRNSYNC","BATCH",2,7,"Response Time")=".22 (ms)"**

**^%ZPERFTOOLS("JRNSYNC","BATCH",2,8,"Response Time")=".209 (ms)"**

**^%ZPERFTOOLS("JRNSYNC","BATCH",2,9,"Response Time")=".193 (ms)"**

**^%ZPERFTOOLS("JRNSYNC","BATCH",2,10,"Response Time")=".172 (ms)"**

[BAD RESULTS]

**REMOTE>d ^Perftools.JRNSYNC**

**How many processes to start? 50**

**How many interations? 1000**

**Starting 50 jobs in the background.**

**Set global node ^%ZPERFTOOLS("JRNSYNC")=0 to terminate jobs.**

**REMOTE>D ^%G**

**For help on global specifications DO HELP^%G**

**Global ^%ZPERFTOOLS("JRNSYNC" -- NOTE: translation in effect**

**^%ZPERFTOOLS("JRNSYNC","BATCH",4,1,"Response Time")="6.042 (ms)"**

**^%ZPERFTOOLS("JRNSYNC","BATCH",4,2,"Response Time")="5.392 (ms)"**

**^%ZPERFTOOLS("JRNSYNC","BATCH",4,3,"Response Time")="5.634 (ms)"**

**^%ZPERFTOOLS("JRNSYNC","BATCH",4,4,"Response Time")="5.371 (ms)"**

**Note: There will be 50 entries (one for each process), but I only listed the first 10 for display purposes in this document.**

**^%ZPERFTOOLS("JRNSYNC","BATCH",4,5,"Response Time")="5.75 (ms)"**

**^%ZPERFTOOLS("JRNSYNC","BATCH",4,6,"Response Time")="5.386 (ms)"**

**^%ZPERFTOOLS("JRNSYNC","BATCH",4,7,"Response Time")="6.177 (ms)"**

**^%ZPERFTOOLS("JRNSYNC","BATCH",4,8,"Response Time")="5.887 (ms)"**

**^%ZPERFTOOLS("JRNSYNC","BATCH",4,9,"Response Time")="6.267 (ms)"**

**^%ZPERFTOOLS("JRNSYNC","BATCH",4,1,"Response Time")="5.802 (ms)"**