Article
Pran Mukherjee · Oct 24, 2022 10m read

Open Exchange

PerfTools IO Test Suite

Purpose

This set of tools (RanRead, RanWrite, and the combined RanIO) is used to generate random read and write events within a database (or pair of databases) to test Input/Output operations per second (IOPS). They can be used either in conjunction or separately to test IO hardware capacity, validate target IOPS, and ensure acceptable disk response times are sustained. Results gathered from the IO tests will vary from configuration to configuration based on the IO sub-system. Before running these tests ensure corresponding operating system and storage level monitoring are configured to capture IO performance metrics for later analysis. The suggested method is by running the System Performance tool that comes bundled within IRIS. Please note that this is an update to a previous release, which can be found here.

Installation

Download the **PerfTools.RanRead.xml**, **PerfTools.RanWrite.xml**, and **PerfTools.RanIO.xml** tools from GitHub here.

Import tools into USER namespace.

```
USER> do $system.OBJ.Load("/tmp/PerfTools.RanRead.xml","ckf")
```

```
USER> do $system.OBJ.Load("/tmp/PerfTools.RanWrite.xml","ckf")
```

```
USER> do $system.OBJ.Load("/tmp/PerfTools.RanIO.xml","ckf")
```

Run the Help method to see all entry points. All commands are run in USER.

```
USER> do ##class(PerfTools.RanRead).Help()
```

- do ##class(PerfTools.RanRead).Setup(Directory,DatabaseName,SizeGB,LogLevel)
  Creates database and namespace with the same name. The log level must be in the range of 0 to 3, where 0 is "none" and 3 is "verbose".
- do ##class(PerfTools.RanRead).Run(Directory,Processes,RunTime in seconds)
  Run the random read IO test.
- do ##class(PerfTools.RanRead).Stop()
  Terminates all background jobs.
- do ##class(PerfTools.RanRead).Reset()
  Deletes statistics of prior runs. This is important to run between tests because otherwise statistics of prior runs are averaged into the current one.
-

do ##class(PerfTools.RanRead).Purge(Directory)
Deletes namespace and database of the same name.

- do ##class(PerfTools.RanRead).Export(Directory)
  Exports a summary of all random read test history to comma delimited text file.

```
USER> do ##class(PerfTools.RanWrite).Help()
```

- do ##class(PerfTools.RanWrite).Setup(Directory,DatabaseName)
  Creates database and namespace with the same name.
- do ##class(PerfTools.RanWrite).CreateGlobals(Directory,NumGlobals,NumSubscripts)
  Creates the requested numbered subscripts (1-N) under the globals ^RanWriteGlobal1, ^RanWriteGlobal2, etc. Default 5 globals, 1M subscripts each.
- do ##class(PerfTools.RanWrite).AddGlobals(Directory,NumGlobals,NumSubscripts)
  Creates more Globals if needed after the initial creation is done.
  NumSubscripts is optional in this case, since by default it will take the size of ^RanWriteGlobal1
- do ##class(PerfTools.RanWrite).CountSubscripts(Directory)
  Counts (via $order) the number of subscripts in ^RanWriteGlobal* in the indicated directory.
- do ##class(PerfTools.RanWrite).GetSizes(Directory)
  Outputs as CSV the number of globals and number of subscripts in ^RanWriteGlobal* in the indicated directory.
- do ##class(PerfTools.RanWrite).Run(Directory,NumProcs,RunTime (sec),TransactionLength,HangTime (ms))
  Run the random write IO test. All parameters other than the directory have defaults.
- do ##class(PerfTools.RanWrite).Stop()
  Terminates all background jobs.
- do ##class(PerfTools.RanWrite).Reset()
  Deletes statistics of prior runs.
- do ##class(PerfTools.RanWrite).Purge(Directory)
  Deletes namespace and database of the same name.
- do ##class(PerfTools.RanWrite).Export(Directory)
  Exports a summary of all random write test history to comma delimited text file.

```
USER> do ##class(PerfTools.RanIO).Help()
```

The commands here are a combination of the two above and work the same way, with the notable exception of:

- do ##class(PerfTools.RanIO).Run(Directory,NumProcs,RunTime (sec),TransactionLength,HangTime (ms),PctReads)

The final argument here, PctReads, indicates the percentage of operations that are reads, defaulting to 50%.

Setup

Create an empty (pre-expanded) database called RAN at least twice the size of the memory of the physical host to be tested. Ensure empty database is at least four times the storage controller cache size. The database needs to be larger than memory to ensure reads are not cached in file system cache. You can create manually or use the following methods to automatically create a namespace and database.

```
USER> do ##class(PerfTools.RanRead).Setup("/ISC/tests/TMP","RAN",200,1)
```

OR

```
USER> do ##class(PerfTools.RanIO).Setup("/ISC/tests/TMP","RAN",200,1)
```

Created directory /ISC/tests/TMP/
Creating 200GB database in /ISC/tests/TMP/
Database created in /ISC/tests/TMP/

NOTE: One can use the same database for all tools, or use separate ones if intending to test multiple disks at once or for specific purposes. The RanRead and RanIO code allows one to specify the size of the database, but the RanWrite code does not, so if a pre-sized database is used it's probably best to use the RanRead or RanIO Setup command even if one will use the database with RanWrite.

After the directory is created, RanRead can be run immediately, but if you wish to run RanWrite or RanIO you must first pre-create the Globals it will use. The number of Globals should equal or exceed the number of processes you wish to run with RanWrite. The rest of this will be demonstrated with RanWrite, but works the same with RanIO. Note that the names of the created Globals will be ^RanWriteGlobal1 and so on even when using RanIO; this was to make sure that the two tools are interoperable.

```
USER>do ##class(PerfTools.RanWrite).CreateGlobals("/ISC/tests/TMP",10,1000000)
```

Starting global creation
.....

```
USER>do ##class(PerfTools.RanWrite).CountSubscripts("/ISC/tests/TMP")
```

Number of Globals is: 10
Number of subscripts in ^RanWriteGlobal1 is: 1000000
Number of subscripts in ^RanWriteGlobal2 is: 1000000
Number of subscripts in ^RanWriteGlobal3 is: 1000000
Number of subscripts in ^RanWriteGlobal4 is: 1000000
Number of subscripts in ^RanWriteGlobal5 is: 1000000
Number of subscripts in ^RanWriteGlobal6 is: 1000000
Number of subscripts in ^RanWriteGlobal7 is: 1000000
Number of subscripts in ^RanWriteGlobal8 is: 1000000
Number of subscripts in ^RanWriteGlobal9 is: 1000000
Number of subscripts in ^RanWriteGlobal10 is: 1000000
Number of subscripts in all globals is: 10000000

NOTE: The more subscripts each Global has, the lower the chance that there will be double writing to the same

block during a given Write Daemon cycle, resulting in more "dirty" blocks. This is what drives the performance measurement. Also, the CountSubscripts function walks the entire tree to count subscripts directly, which can take a great deal of time if the Globals are large. If you simply want a count of what was created based upon the "Size" parameter in each created Global, you can run GetSizes, which looks like this:

```
USER>do ##class(PerfTools.RanWrite).GetSizes("/ISC/tests/TMP")
```

10,1000000,1000000,1000000,1000000,1000000,1000000,1000000,1000000,1000000,1000000

Finally, if you create X Globals initially, and later decide you want to run X+Y processes, you can add to the existing Globals without re-creating them by using the AddGlobals function:

```
USER>do ##class(PerfTools.RanWrite).AddGlobals("/ISC/tests/TMP",12)
```

Filling 2 new Globals, starting with number 11
Number of subscripts is 1000000
Starting global creation
....

Methodology

Start with a small number of RanRead processes and 30-60 second run times. Then increase the number of processes, e.g. start at 10 jobs and increase by 10, 20, 40 etc. Continue running the individual tests until response time is consistently over 10ms or calculated IOPS is no longer increasing in a linear way.

As a guide, the following response times for 8KB and 64KB Database Random Reads (non-cached) are usually acceptable for all-flash arrays:

- Average <= 2ms
- Not to exceed <= 5ms

Run

For RanRead, execute the Run method increasing the number of processes and taking note of the response time as you go. The primary driver of IOPS for RanRead is the number of processes.

```
USER> do ##class(PerfTools.RanRead).Run("/ISC/tests/TMP",5,60)
```

InterSystems Random Read IO Performance Tool

RanRead process 11742 creating 5 worker processes in the background.
  Prepped RanReadJob 11768 for parent 11742
  Prepped RanReadJob 11769 for parent 11742
  Prepped RanReadJob 11770 for parent 11742
  Prepped RanReadJob 11771 for parent 11742
  Prepped RanReadJob 11772 for parent 11742
Starting 5 processes for RanRead job number 11742 now!
To terminate run: do ##class(PerfTools.RanRead).Stop()
Waiting to finish.....
Random read background jobs finished for parent 11742
RanRead job 11742's 5 processes (62.856814 seconds) average response time = 1.23ms
Calculated IOPS for RanRead job 11742 = 4065

For RanWrite, each process attaches to a single ^RanWriteGlobal and writes multiple transactions to that Global

and only that Global. The transaction length parameter is how many writes are performed by each job during each cycle, while the Hangtime parameter is the delay between transactions in milliseconds. In other words, if you have a transaction length of 50 and a hangtime of 2, each process will write 50 times as quickly as possible into its associated ^RanWriteGlobal, and then pause for 2 milliseconds, then repeat. Execute the Run method decreasing the Hangtime parameter and/or increasing the number of jobs. Those parameters are the primary drivers of IOPS for RanWrite.

```
USER>do ##class(PerfTools.RanWrite).Run("/ISC/tests/TMP",10,60,50,2))
```

Number of Globals is: 12
Total number of subscripts in all globals is: 600000
RanWrite process 1424 creating 10 worker processes in the background.
Starting 10 processes for RanWrite job number 1424 now!
To terminate run: do ##class(PerfTools.RanWrite).Stop()
Waiting to finish.....
Random write background jobs finished for parent 1424
RanWrite job 1424's 10 processes (60 seconds) had average response time = .0545ms
Calculated IOPS for RanWrite job 1424 = 183414

In order to run RanRead and RanWrite together, instead of "do", use the "job" command for both of them to run them in the background, or use RanIO. In the case of RanIO, each "transaction" as above in RanWrite will be a batch of reads OR writes, as decided by random selection and the ReadPct parameter. So, if you have 10 processes running in 10 separate Globals, each process will run in its associated Global, and each cycle it will "roll the dice" to decide if it's a read or write cycle, and then do TransactionLength iterations followed by a wait of HangTime milliseconds.

Results

In order to acquire the simple results for each run that are saved in USER in SQL table PerfTools.RanRead, PerfTools.RanWrite, and PerfTools.RanIO (yes, this is separate from the other two) use the Export command for each tool as follows.

To export the result set to a comma delimited text file (csv) run the following:

```
USER> do ##class(PerfTools.RanRead).Export("/ISC/tests/TMP/ ")
```

Exporting summary of all random read statistics to /ISC/tests/TMP/PerfToolsRanRead20221023-1408.txt
Done.

IMPORTANT NOTE: These tools have no way to measure disk writes. Disk reads are measured on a 1:1 basis, but all writes from RanWrite and RanIO are to the database in memory. As such, the writes are handled by the Write Daemon as always, and the Write Daemon is very efficient. If multiple adjacent blocks on disk are "dirty" (i.e. have been modified since the last Write Daemon cycle), they may be written in a single IOP. Up to 16 blocks can be written together. Due to this, the write IOPS reported by RanWrite and RanIO can be significantly different than the write IOPS reported by mgstat or iostat. This is to be expected, and is part of why the performance should be measured by external tools in addition to the reporting from these tools.

Analysis

It is recommended that one use the built-in SystemPerformance tool to acquire true understanding of the system being analyzed. Commands to SystemPerformance need to be run in the %SYS namespace. To switch to that, use the ZN command:

```
USER> ZN "%SYS"
```

To find details on any bottlenecks in a system, or if one requires more details about how it runs at its target IOPS, one should create a SystemPerformance profile with high cadence data acquisition:

```
%SYS> set rc=$$addprofile^SystemPerformance("5minhighdef","A 5-minute run sampling ev
ery second",1,300)
```

Then run that profile (from %SYS) and immediately switch back to USER and start RanRead and/or RanWrite or RanIO using "job" rather than "do":

```
%SYS>set runid=$$run^SystemPerformance("5minhighdef")

%SYS> ZN "USER"

USER> job ##class(PerfTools.RanRead).Run("/ISC/tests/TMP",5,60)

USER> job ##class(PerfTools.RanWrite).Run("/ISC/tests/TMP",1,60,.001)
```

One can then wait for the SystemPerformance job to end, and analyze the resultant html file using tools such as yaspe.

Clean Up

After finished running the tests remove the history by running:

```
USER> do ##class(PerfTools.RanRead).Reset()
```

#Analytics #Caché #HealthShare #InterSystems IRIS #Open Exchange #TrakCare
Check the related application on InterSystems Open Exchange

Source URL:https://community.intersystems.com/post/perftools-io-test-suite