Article
Ray Fucillo · Sep 2, 2020 7m read

Integrity Check: Speeding it Up or Slowing it Down

While the integrity of Caché and InterSystems IRIS databases is completely protected from the consequences of system failure, physical storage devices do fail in ways that corrupt the data they store. For that reason, many sites choose to run regular database integrity checks, particularly in coordination with backups to validate that a given backup could be relied upon in a disaster. Integrity check may also be acutely needed by the system administrator in response to a disaster involving storage corruption. Integrity check must read every block of the globals being checked (if not already in buffers), and in an order dictated by the global structure. This takes substantial time, **but integrity check is capable of reading as fast as the storage subsystem can sustain**. In some situations, it is desirable to run it in that manner to get results as quickly as possible. In other situations, integrity check needs to be more conservative to avoid consuming too much of the storage subsystem ' s bandwidth.

Plan of Attack

This following outline caters for most situations. The detailed discussion in the remainder of this article provides the necessary information to act on any of these, or to derive other courses of action.

1. If using Linux and integrity check is slow, see the information below on enabling Asynchronous I/O.
2. If integrity check must complete as fast as possible - running in an isolated environment, or because results are needed urgently - use Multi-Process Integrity Check to check multiple globals or databases in parallel. The number of processes times the number of concurrent asynchronous reads that each process will perform (8 by default, or 1 if using Linux with asynchronous I/O disabled) is the limit on the number of concurrent reads in flight. Consider that the average may be half that and then compare to the capabilities of the storage subsystem. For example, with storage striped across 20 drives and the default 8 concurrent reads per process, five or more processes may be needed to capture the full capacity of the storage subsystem (5*8/2=20).
3. When balancing integrity check speed against its impact on production, first adjust the number of processes in the Multi-Process Integrity Check, then if needed, see the SetAsyncReadBuffers tunable. See Isolating Integrity Check below for a longer-term solution (and for eliminating false positives).
4. If already confined to a single process (e.g. there ' s one extremely large global or other external constraints) and the speed of integrity check needs adjustment up or down, see the SetAsyncReadBuffers tunable below.

Multi-Process Integrity Check

The general solution to get an integrity check to complete faster (using system resources at a higher rate) is to divide the work among multiple parallel processes. Some of the integrity check user interfaces and APIs do so, while others use a single process. Assignment to processes is on a per-global basis, so checking a single global is always done by just one process (versions prior to Caché 2018.1 divided the work by database instead of by global).

The principal API for multi-process integrity check is **CheckLIst^Integrity** (see documentation for details). It collects the results in a temporary global to be displayed by Display^Integrity. The following is an example checking three databases using five processes. Omitting the database list parameter here checks all databases.

```
set dblist=$listbuild("/data/db1/","/data/db2/","/data/db3/")
set sc=$$CheckList^Integrity(,dblist,,,5)
do Display^Integrity()
```

```
kill ^IRIS.TempIntegrityOutput(+$job)

/* Note: evaluating 'sc' above isn't needed just to display the results, but...
   $system.Status.IsOK(sc) - ran successfully and found no errors
   $system.Status.GetErrorCodes(sc)=$$$ERRORCODE($$$IntegrityCheckErrors) // 267
                          - ran successfully, but found errors.
   Else - a problem may have prevented some portion from running, 'sc' may have
          multiple error codes, one of which may be $$$IntegrityCheckErrors. */
```

Using CheckLIst^Integrity like this is the most straight-forward way to achieve the level of control that is of interest to us. The Management Portal interface and the Integrity Check Task (built-in but not scheduled) use multiple processes, but may not offer sufficient control for our purposes.*

Other integrity check interfaces, notably the terminal user interface, ^INTEGRIT or ^Integrity, as well as Silent^Integrity, perform integrity check in a single process. These interfaces, therefore, do not complete the check as fast as it's possible to achieve, and they use fewer resources. An advantage, though, is that their results are visible, logged to a file or output to the terminal, as each global is checked, and in a well-defined order.

Asynchronous I/O

An integrity check process walks through each pointer block of a global, one at a time, validating each against the contents of the data blocks it points to. The data blocks are read with asynchronous I/O to keep a number of read requests in flight for the storage subsystem to process, and the validation is performed as each read completes.

On Linux only, async I/O is effective only in combination with direct I/O, which is not enabled by default until InterSystems IRIS 2020.3. This accounts for a large number of cases where integrity check takes too long on Linux. Fortunately, it can be enabled on Cache 2018.1, IRIS 2019.1 and later, by setting **wduseasyncio=1** in the [config] section of the .cpf file and restarting. This parameter is recommended in general for I/O scalability on busy systems and is the default on non-Linux platforms since Caché 2015.2. Before enabling it, make sure that you ' ve configured sufficient memory for database cache (global buffers) because with Direct I/O, the databases will no longer be (redundantly) cached by Linux. When not enabled, reads done by integrity check complete synchronously and it cannot utilize the storage efficiently.

On all platforms, the number of reads that an integrity check process will put in flight at one time is set to 8 by default. If you must alter the rate at which a single integrity check process reads from disk this parameter can be tuned – up to get a single process to complete faster, down to use less storage bandwidth. Bear in mind that:

- This parameter applies to each integrity check process. When multiple processes are used, the number of processes multiplies this number of in-flight reads Changing the number of parallel integrity check processes has a much larger impact and therefore is usually the first thing to do. Each process is also limited by computational time (among other things) so there increasing the value of this parameter is limited in its benefit.
- This only works within the storage subsystem ' s capacity to process concurrent reads. Higher values have no benefit if databases are stored on a single local drive, whereas a storage array with striping across dozens of drives can process dozens of reads concurrently.

To adjust this parameter from the %SYS namespace, **do SetAsyncReadBuffers^Integrity(**value**)**. To see the current value, **write $$GetAsyncReadBuffers^Integrity()**. The change takes effect when the next global is checked. The setting currently does not persist through a restart of the system, though it can be added to SYSTEM^%ZSTART.

There is a similar parameter to control the maximum size of each read when blocks are contiguous on disk (or nearly so). This parameter is less often needed, though systems with high storage latency or databases with larger block sizes could possibly benefit from fine tuning. The value has units of 64KB, so a value of 1 is 64KB, 4 is 256KB, etc. 0 (the default) lets the system to select and it currently selects 1 (64KB). The ^Integrity function for this parameter, parallel to those mentioned above, are **SetAsyncReadBufferSize** and **GetAsyncReadBufferSize**.

Isolating Integrity Check

Many sites run regular integrity checks directly on the production system. This is certainly the simplest to configure, but it ' s not ideal. In addition to concerns about integrity check ' s impact on storage bandwidth, concurrent database update activity can sometimes lead to false positive errors (despite mitigations built into the checking algorithm). As a result, errors reported from an integrity check run on production need to be evaluated and/or rechecked by an administrator.

Often times, a better option exists. A storage snapshot or backup image can be mounted on another host, where an isolated Caché or IRIS instance runs the integrity check. Not only does this prevent any possibility of false positives, but if the storage is also isolated from production, integrity check can be run to fully utilize the storage bandwidth and complete much more quickly. This approach fits well into the model where integrity check is used to validate backups; a validated backup effectively validates production as of the time the backup was made. Cloud and virtualization platforms can also make it easier to establish a usable isolated environment from a snapshot.

---

* The Management Portal interface, the *Integrity Check* Task and the IntegrityCheck method of SYS.Database select a rather large number of processes (equal to the number of CPU cores), lacking the control that ' s needed in many situations. The management portal and the task also perform a complete recheck of any global that reported error in effort to identify false positives that may have occurred due to concurrent updates. This recheck occurs above and beyond the false positive mitigation built into the integrity check algorithms, and that may be unwanted in some situations due to the additional time it takes (the recheck runs in a single process and checks the entire global). This behavior may be changed in the future.

#Best Practices #System Administration #Caché #InterSystems IRIS
