Article
Robert Cemper · Sep 22, 2018 3m read

Sharding evaluation #1

IRIS brought us a new **WOW** feature - **SHARDING** !
Definitely a great thing!
But how can I find out if it suits my actual applications?
Is there a practical advantage to go for it with my well cooked transactional application?
Or is it just for new still to be designed applications?

I did and still do a series of investigations that I like to share with you.

First:
It means to move from an (almost) single server environment to an environment with multiple narrow bound and synchronized servers.
This means more and especially more qualified control over your instances.
Not necessarily a big deal but definitely something to consider by resource planning.
And of course also for all kind of availability management starting with backup up to failover with mirrors.

Second:
Sharding is only useful if your target tables have a significant size to benefit from parallel processing to compensate for the overhead in cross-server communications over ECP.
^GSIZE tells you where your research should start.
Next, you should find out if your typical result sets are large enough to visualize improvements.
Search for a few records is rather an exercise to apply a suitable index. Of course, you can do it with Sharding as well.
It just could be hard to identify and present the gain to your management.

Based on this ideas a table with 26.5 mio records was selected that typically had resultsets of 1000 to 10000 hits.
With this table, a set of significant queries was selected to compare the performance on a single server against a Sharded setup with 4 Sharding servers around a master.

Differently, to the production environment running on Windows, I decided to use for the 5 test servers Ubuntu for its smaller footprint.
As no isolated LAN for the ECP connections was available I ran most tests during out of business hours to avoid interference with other network traffic.

Surprise #1:
Moving from Win (12 GB buffer 8 cores) to Linux (400MB buffer 2 cores) resulted in a reduction of runtime to 50% and even below even as this was a significantly smaller box. [CORRECTION 75% see comment]
I expected Linux to be faster. But the dimension I saw was incredible (and thus verified it several times).
And not being an expert on Linux at all this was just a simple server instance out of the box.

Next phase was to distribute data on 4 shards. I decided to use the default shard key.
Loading took some time but finished overnight.

Surprise #2:
Sharding was clear faster than the single server. (as expected)
Query run time for simple queries went down to **45 ... 80%** from the single server.
Less attractive: a complex join took on Sharding at least double the time as on the single server. (the surprise)

So I went back to documentation Choose a Shard Key + Evaluate Unique Constraints

My lazy join worked exactly on a UNIQUE Constraint and was unrelated to Shard key !!

So I rebuild my Sharded table again using the main part of the UNIQUE constraint as Shard key.

Surprise #3:
- Load time reduced to ~**60%**
- Simple queries went down to **15...25%** of the single server
- The cumbersome complex join now finished with ~ **50%** runtime of the single server.

My personal learning:
- There are situations where reading AND understanding documentation first creates faster success.
- Defaults are not always the best solutions.

Summary:

- be aware of the effort to maintain additional servers
- select carefully if your table fits Sharding in size and structure
- think twice if you select your Shard key
- run serious tests to align expectations with reality

My next steps will be evaluating the impact of the number of shards.
I started with the maximum available. Next, I will check to see how it works with 2 and 3 shards in use.

I'll keep you posted once I have collected meaningful figures.

Sharding evaluation #2

#AI #Sharding #InterSystems IRIS