Artículo
Jose Tomas Salvador · Sep 3, 2019 Lectura de 4 min

Evaluación de Sharding #2

Hola a todos,

como prometí en mi post previo Evaluación de Sharding #1, continué investigando el impacto del número de *shards*.

Para completar la revisión, he añadido también las instancias en Windows (Server 2012 R2) 8 cores:

- Cache for Windows (x86-64) 2016.2.2 - 12 GB global buffers
- IRIS for Windows (x86-64) 2018.1.1 - 400 MB global buffers, no sharding

En LINUX (Ubuntu 16.04 LTS) 2 cores:

- IRIS for UNIX (Ubuntu Server LTS for x86-64) 2018.1.1 400MB global buffe

Pruebas: no shards, 2 shards, 3 shards, 4 shards.

La tabla que utilicé tiene 26,5 millones de registros y es idéntica en todas las instancias con idénticos índices.

Las consultas:

```
SELECT $LISTLENGTH($LISTFROMSTRING(LIST(<property>))) FROM ... WHERE ... %STARTSWITH  <value>
```

Esta fue utilizada con 2 valores diferentes, resultando en 47000 y 19000 accesos. Me refiero a ella como consultas simples S47 y S19.

```
SELECT$LISTLENGTH ($LISTFROMSTRING(LIST(<property1>))) FROM....
   WHERE ... %STARTSWITH <value1>
    AND <attribute> %INLIST ( SELECT $LISTFROMSTRING(LIST(<property2>)) FROM ...
                      WHERE. .. %STARTSWITH  <value2>
                    )
```

resultando en ~3000 accesos. Me refiero a ella como consulta compleja CX3.

```
SELECT $LISTLENGTH($LISTFROMSTRING(LIST(>property1>))) FROM ... As A JOIN ...
As B ON ...
   WHERE ... %STARTSWITH  <value1> and ... .. %STARTSWITH  <value2>
```

resultando en ~5500 accesos. Me refiero a ella como consulta compleja CJ5.

Esta selección era la más cercana a la aplicación real. Otras también eran interesantes en cuanto a comportamiento, pero no tan relevantes de cara a la realidad.

Como era de esperar y requerido, todas las consultas de test generaron los mismos resultados en las distintas configuraciones.

La referencia (*baseline*) a la que me refiero es Caché en WIN. Los resultados se representan por tiempo de ejecución como porcentaje sobre esta referencia.

Para todos los valores, tomé una media de las 3 últimas ejecuciones de al menos 5 totales, para evitar posibles desviaciones por *buffering*:

- `IRIS on WIN no shard vs. Caché on WIN`
  `S47: 92%   S19: 98%   CX3: 87%   CJ5:  67%`
- `IRIS on LNX no shard vs. Caché on WIN`
  `S47: 59%   S19: 56%   CX3: 50%   CJ5:  57%`
- `IRIS on LNX 2 shards vs. Caché on WIN`
  `S47: 21%   S19: 23%   CX3: 38%   CJ5:  52%`
- `IRIS on LNX 3 shards vs. Caché on WIN`
  `S47: 10%   S19: 14%   CX3: 27%   CJ5:  48%`
- `IRIS on LNX 4 shards vs. Caché on WIN`
  `S47:  7%   S19: 12%   CX3: 24%   CJ5:  44%`

Creo que los resultados no necesitan mucha explicación:

- Cuantos más *shards* tienes, más rápido se ejecutan tus consultas
- Cuanto más simple es la consulta, mayor es la mejora de rendimiento
- Las "obras de arte" en SQL pueden ser contraproducentes y podrían requerir un diseño más sofisticado de la *shard key*

Algunas observaciones con respecto a la carga inicial:

- No medí los tiempos de carga con precisión ya que no era mi objetivo al principio
- En cualquier caso, cuantos más *shards* tengas, más tiempo llevará la carga
- Mi sensación *grosso modo* es que la carga con 4 *shards* llevó al menos el doble de tiempo que la carga en

2 *shards;* y en 3 *shards,* algo entre medias de ambos
- No me sorprendió, ya que mover tal cantidad de datos vía ECP no puede ser impresionantemente rápido. Pero como ves, al final compensa

Basado en este aprendizaje, veo 2 opciones para mantener la tabla actualizada:

- INSERT, UPDATE, DELETE directos sobre SQL
- utilizar el parámetro de clase DSTIME para ejecutar una actualización en *batch* dependiendo de tu aplicación

Espero que esto te ayude a decidir sobre TU implementación del *sharding*.

#Sharding #InterSystems IRIS

URL de fuente: https://es.community.intersystems.com/post/evaluaci%C3%B3n-de-sharding-2