Artículo
Andrey Shcheglov · 13 dic, 2018 Lectura de 6 min

How are stars counted? How InterSystems Caché eXTreme is used in Gaia

Astronomers ' tools

5 years ago, on December 19, 2013, the ESA launched an orbital telescope called Gaia. Learn more about the Gaia mission on the official website of the European Space Agency or in the article by Vitaly Egorov (Billion pixels for a billion stars).

However, few people know what technology the agency chose for storing and processing the data collected by Gaia. Two years before the launch, in 2011, the developers were considering a number of candidates (see "Astrostatistics and Data Mining" by Luis Manuel Sarro, Laurent Eyer, William O ' Mullane, Joris De Ridder, pp. 111-112):

- *IBM DB2*,
- *PostgreSQL*,
- Apache Hadoop,
- Apache Cassandra and
- *InterSystems Caché* (to be more precise, the Caché eXTreme Event Persistence technology).

Comparing the technologies side-by-side produced the following results (source):

Technology	Time
DB2	13min55s
PostgreSQL 8	14min50s
PostgreSQL 9	6min50s
Hadoop	3min37s

Technology	Time
Cassandra	3min37s
Caché	2min25s

The first four will probably sound familiar even to schoolchildren. But what is *Caché XEP*?

Java technologies in Caché

If you look at the Java API stack provided by *InterSystems*, you will see the following:

- The Caché Object Binding technology that transparently projects data in Java. In *Caché* terms, the generated Java proxy classes are called exactly like that - projections. This approach is the simplest, since it saves the "natural" relations between classes in the object model, but doesn't guarantee great performance: a lot of service metadata describing the object model is transferred "over the wires".
- JDBC and various add-ons (*Hibernate*, *JPA*). I guess I won't tell you anything new here apart from the fact that Caché supports two types of transaction isolation: READUNCOMMITTED and READCOMMITTED – and works in the READUNCOMMITTED mode by default.
- The *Caché eXTreme* family (also available in .NET and Node.js editions). This approach is characterized by the direct access to the low-level data representation (so-called "globals" – quanta of data in the *Caché* world) ensuring high performance. The Caché XEP library simultaneously provides object and quasi-relational access to data.
    - Object – the API client no longer needs to care about object-relational representation: following the Java object model (even in cases of complex multi-layer inheritance), the system automatically creates an object model on the *Caché* class level (or a DB schema if we want to use the terms of the relational representation).
    - Quasi-relational – in the sense that you can run SQL queries against multiple "events" stored in a database (to be exact, requests using the SQL subset) directly from the context of an *eXTreme*-connection. Indices and transactions are fully supported as well. Of course, all loaded data become immediately accessible via JDBC and a relational representation (supporting all the powerful features of ANSI SQL and SQL extensions specific to the *Caché* dialect), but the access speed will be completely different.

Summing up, here's what we have:

- "schema" import (*Caché* classes are created automatically), including
- import of the Java class hierarchy;
- instant relational access to data – you can work with *Caché* classes the way you work with tables;
- support of indices and transactions via *Caché eXTreme*;
- support of simple SQL queries via *Caché eXTreme*;
- support of arbitrary SQL queries via the underlying JDBC over TCP connection (*Caché* uses the standard Type 4 (Direct-to-Database Pure Java) driver).

This approach offers some advantages in comparison with comparable relational (higher access speed) and various NoSQL solutions (instant access to data in the relational style).

The "nuance" of configuring *Caché eXTreme* prior to connecting is the environment set-up:

- the GLOBALSHOME variable has to point to the *Caché* installation folder and
- LDLIBRARYPATH (DYLDLIBRARYPATH for *Mac OS X* or PATH for *Windows*) has to contain ${GLOBALSHOME}/bin.

Additionally, you may need to increase the stack and heap size of the *JVM* (-Xss2m -Xmx768m).

Some practice

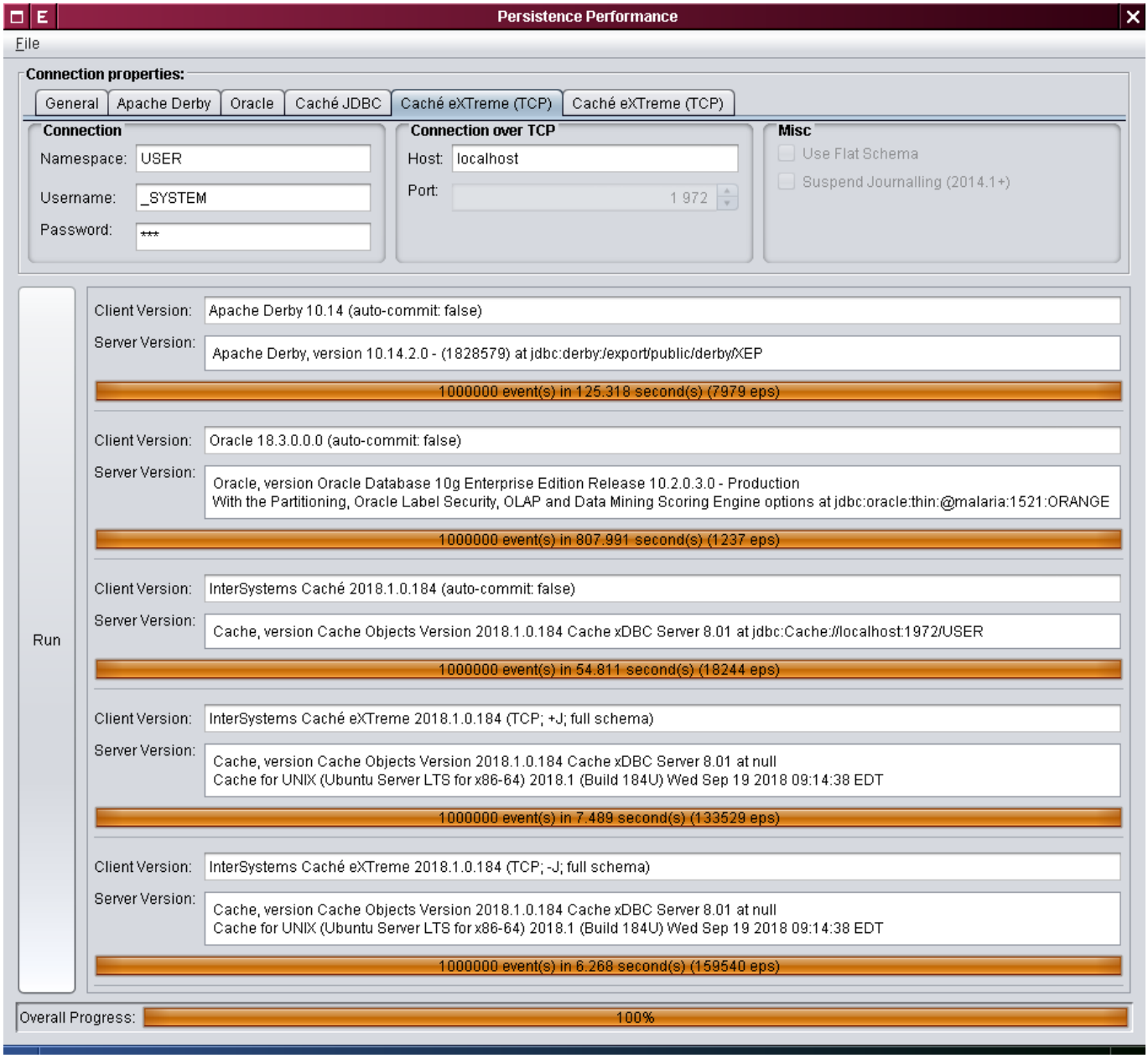
The authors were interested in how *Caché eXTreme* would behave while writing an uninterrupted stream of data in comparison with other data processing technologies. We used historical stock price data in the CSV format from the website of the "Finam" holding. Sample data file:

```
<TICKER>,<PER>,<DATE>,<TIME>,<LAST>,<VOL>
NASDAQ100,0,20130802,09:31:07,3 125.300000000,0
NASDAQ100,0,20130802,09:32:08,3 122.860000000,806 906
NASDAQ100,0,20130802,09:33:09,3 123.920000000,637 360
NASDAQ100,0,20130802,09:34:10,3 124.090000000,421 928
NASDAQ100,0,20130802,09:35:11,3 125.180000000,681 585
```

The code of the *Caché* class modeling the above structure might look like this:

```
Class com.intersystems.persistence.objbinding.Event Extends %Persistent [ ClassType =
 persistent, DdlAllowed, Final, SqlTableName = Event ]
{
Property Ticker As %String(MAXLEN = 32);

Property Per As %Integer(MAXVAL = 2147483647, MINVAL = -2147483648);

Property TimeStamp As %TimeStamp;

Property Last As %Double;

Property Vol As %Integer(MAXVAL = 9223372036854775807, MINVAL = -9223372036854775810)
;
}
```

We also wrote some basic and naive test code. This "naive" approach can be justified by the fact that we are not really measuring the speed of the code generated by *JIT*, but the speed at which the code that is completely unrelated to *JVM* (with the exception of *Apache Derby*) can write to the disk. Here's how the test program window looks like:



Our contenders:

- *Apache Derby* 10.14.2.0
- *Oracle* 10.2.0.3.0
- *InterSystems Caché* 2018.1 (JDBC)
- *InterSystems Caché* 2018.1 (eXTreme)

Note that since tests are somewhat approximate, we saw no practical purpose in providing exact numbers: the margin of error is fairly high, while the goal of the article is to demonstrate the general tendency. For the same reasons, we are not specifying the exact version of JDK and the settings of the garbage collector: the server-side JVM 8u191 with -Xmx2048m -Xss128m reached a very similar level of performance on *Linux* and *Mac OS X*. One million events were saved in each test; several warm-up runs (up to 10) were performed before each test of a particular database. As for *Caché* settings, the routine cache was increased to 256 MB and the 8kb database cache was expanded to 1024 MB.

Our testing yielded the following results (the write speed values are expressed in events per second (eps)):

Technology	Time, s (less is better)	Write speed, eps (more is better)
Apache Derby	140±30	7100±1300
Oracle	780±50	1290±80

Technology	Time, s (less is better)	Write speed, eps (more is better)
Caché JDBC	61±8	17000±2000
Caché eXTreme	6.7±0.8	152000±17000
Caché eXTreme, transaction journaling disabled	6.3±0.6	162000±14000

1. *Derby* offers speeds varying from 6200 to 8000 eps.
2. *Oracle* turned out to be as fast as 1290 eps.
3. *Caché* in the JDBC mode gives you a higher speed (from 15000 to 18000 eps), but there is a trade-off: the default transaction isolation level, as mentioned above, is READUNCOMMITTED.
4. The next option, *Caché eXTreme*, gives us 127000 to 167000 eps.
5. Finally, we took some risk and disabled the transaction log (for a given client process), and managed to achieve the write speed of 172000 eps on a test system.

Those who are interested in more accurate numbers can view the source code. You will need the following to build and run:

- *JDK* 1.8+,
- *Git*,
- *Maven*
- *Oracle* JDBC driver (available from Oracle Maven Repository)
- Maven Install Plugin for for creating local *Caché JDBC* and *Caché eXTreme* artifacts:

```
$ mvn install:install-file -Dfile=cache-db-2.0.0.jar
$ mvn install:install-file -Dfile=cache-extreme-2.0.0.jar
$ mvn install:install-file -Dfile=cache-gateway-2.0.0.jar
$ mvn install:install-file -Dfile=cache-jdbc-2.0.0.jar
```

and, finally,

- *Caché* 2018.1+.

#Bases de datos #Java #JDBC #Prueba #Caché

URL de fuente:https://es.community.intersystems.com/node/455641