Article
Robert Cemper · Aug 5, 2017 3m read

Winning in GIS competition

GIS stands for Geographic Information System.
and it's not a typical arena for Caché. But it's definitely an environment with high data volume.

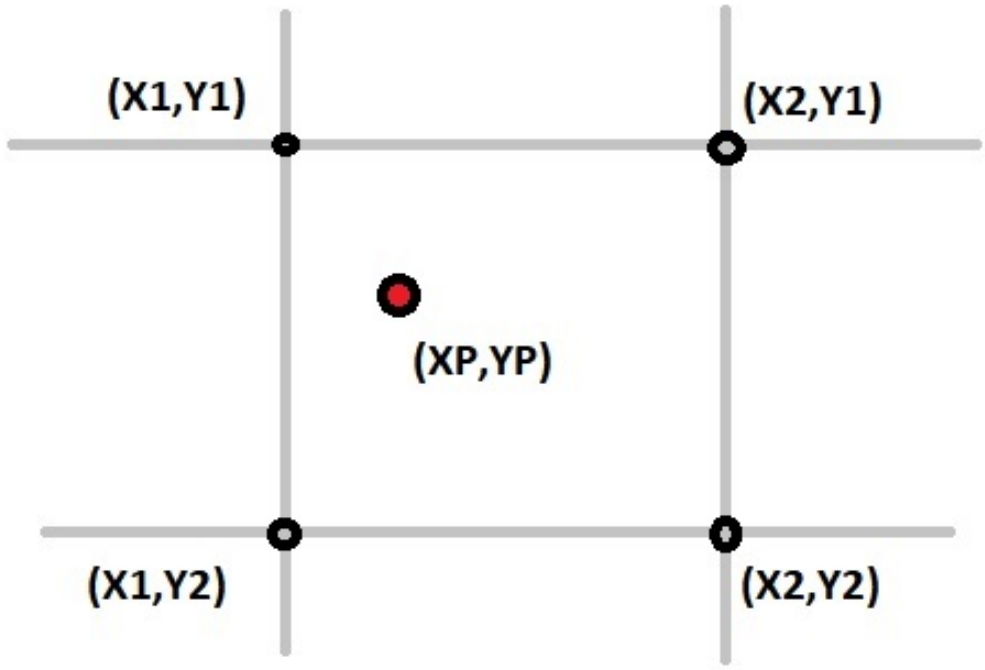
You see 3 major areas

- Visual front end:
A mature area well covered by a bunch of commercial and open source products.
No need for Caché there.

- GIS mathematics:
JTS (Java Topology Suite) is fixed standard that covers all requirements and can be linked to Caché by the Java Gateway
or the C, C++ incarnation of this standard library using Caché Call Out Gateway.
So far no added value by Caché.

- the Database that holds all the geographic objects and its attributes, especially geographic coordinates
for road maps, highways, companies, private houses, railways, demographic data, ...

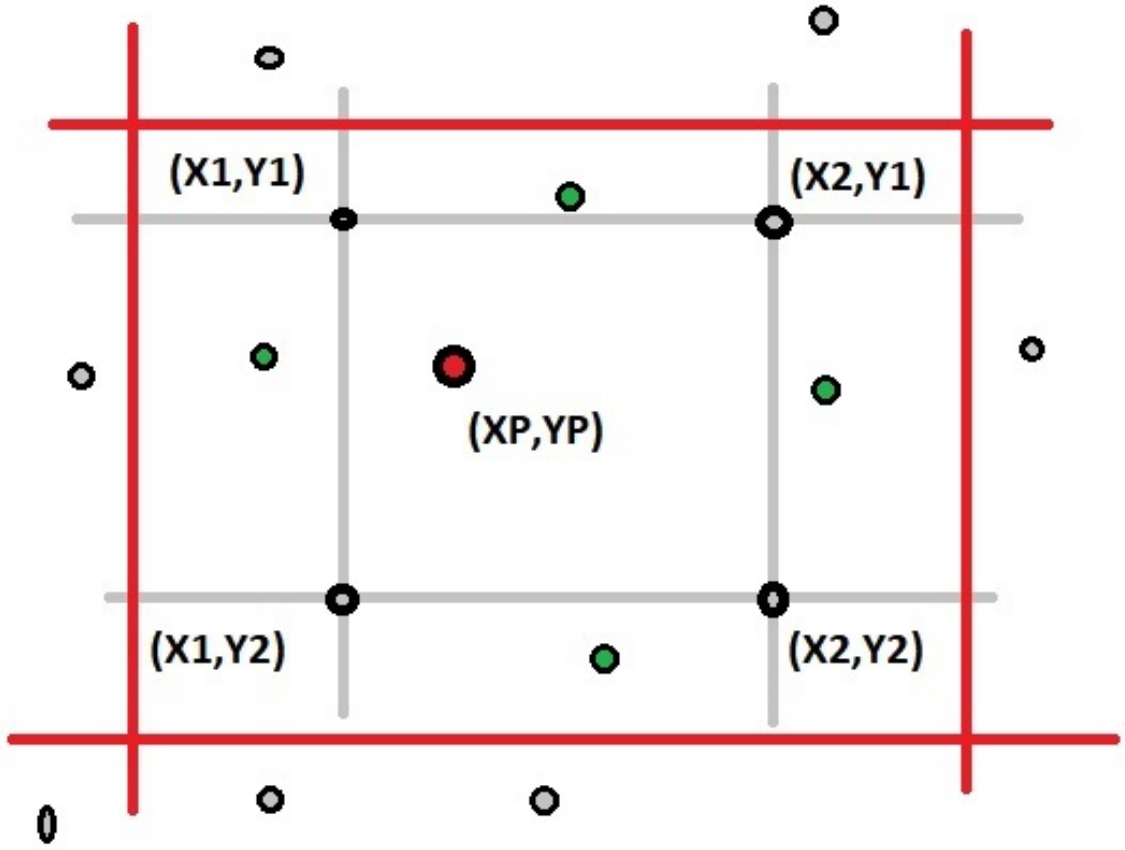
This is the area where Caché can win.
Delivering data based on classic attributes is something we understand quite well. Nothing new.
The challenge starts with comparing locations. The map you work on is represented as collection of bounding rectangles
and your task is to find if a specific element is inside that rectangle. The theory behind is known as R-Tree.
https://en.wikipedia.org/wiki/R-tree



The basic task is to sort out all points that don't fit running series of
SQL SELECT ... WHERE xp BETWEEN x1 AND x2 AND yp BETWEEN y1 AND y2 ....

How can this query be be accelerated:
- Forget about using floating point coordinates for indexing but keep them for JTS purpose
- Use SQLcomputed values instead. Preferably Integers.
  >> Integer calculation is the fastest in any machine
  >> Integers have the smallest footprint in memory, $LB(),index, ...
- Honor the required precision
>> 1° in latitude is 110....111 km
>> 1° in longitude is depending on latitude between 11km (84°N) ... 111km (0°N)
so if the target of accuracy is about 10mt 4 digits will be enough.
Typical GIS sources feed you with 6...8 decimal digits

For the process itself you iterate from a rather large grid down to your requirements like an inverse pyramid



The picture illustrates it:
You drop those that don't fit at all (gray)
In the next cycle you iterate with higher precision and much less possible targets excluding more candidates (green)

As we came from bounding rectangles the remaining result still requires a final check using JTS
but on a significant lower numer of instances.

Where comes the speed from:
- pure Integer calculation. When I saw the difference first I was beaten down and couldn't believe it.
- keep your indices slim to achieve high buffer hit rates
- filtering down results in iteration . Larger index but less elements to check
- don't care about number of inidces you generate. It's all static data, that you can compact and that change rarely
- use all the freedom and flexibility offered by Caché [%PARALLEL wasn't available then]

BTW.
I found this all during a benchmark against my favorite competitor [O. from Redwood City, CA]

The exercise:
out of 1 mio places with >1 person and income > $$$ [almost all]
find those that are closer than 300 mt to a highway.
the highway map had 5 mio elements (GY)

On an identic VM configuration Caché took depending on precision ~12...20 min
while O.Spatial was stopped with no result, no sign of progress after 3 hrs+

I think the principles of this exercise could apply to other big data situations as well.

#Caché