Generación, almacenamiento y consulta de datos espaciales masivos*

Alejandro Cortiñas, Miguel R. Luaces

Universidade da Coruña, Laboratorio de Bases de Datos, A Coruña, Spain
{alejandro.cortinas, luaces}@udc.es

Resumen En este artículo presentamos resultados preliminares para dos problemas que surgen en el ámbito del almacenamiento, consulta y visualización de conjuntos masivos de objetos móviles: una herramienta para la generación de conjuntos de datos masivos de objetos móviles en una red de carreteras y su posterior almacenamiento en diferentes sistemas de almacenamiento, y una serie de experimentos de visualización de 40 millones de datos geolocalizados en los que enfrentamos una solución tradicional con una alternativa Big Data actual.
Palabras clave: big data geográfico, sistemas de información geográfica, generación de datos

1. Introducción

La tecnología actual hace factible la recogida en tiempo real de volúmenes masivos de información geográfica. Por ejemplo, es posible recoger en tiempo real la ubicación de la flota de taxis de una ciudad (alrededor de 16,000 licencias en la comunidad de Madrid), o de un servicio de coches compartidos como *car2go* (con alrededor de 500 vehículos). En pocos años, con el avance de la tecnología de vehículos conectados, parece factible conocer en tiempo real la ubicación de todos los vehículos que circulan por las carreteras.

Seleccionar la tecnología más adecuada para dar soporte estos escenarios de uso es complicado. Uno de los problemas que se presenta es la dificultad de disponer de conjuntos de datos que permitan evaluar la respuesta de la tecnología tanto a la hora de gestionar el volumen de datos de entrada al sistema como a la hora de consultar y visualizar los datos almacenados. Por ello, en este articulo describimos la arquitectura de un sistema que permite simular la carga real sobre uno de estos sistemas mediante la simulación de un número elevado de conductores que circulan por una red de carreteras e informan de su posición al servidor de forma periódica. Además, el sistema también proporciona una solución para alimentar diferentes subsistemas de almacenamiento de forma sencilla

* Financiado por el CDTI y el Ministerio de Economía y Competitividad (PGE y Fondos FEDER) [TIN2016-78011-C4-1-R, TIN2016-77158-C4-3-R, TIN2013-46238-C4-3-R, TIN2013-46801-C4-3-R, Ref. IDI-20141259, Ref. ITC-20151305 y Ref. ITC-20151247]; y por la Xunta de Galicia (cofinanciado por FEDER) [Ref. ED431G/01]

con el mismo conjunto de datos de forma que se puedan probar bajo las mismas condiciones de carga y evaluar su rendimiento ante las mismas consultas. En el artículo también mostramos resultados preliminares en los que utilizamos el sistema propuesto para almacenar y visualizar 40 millones de ubicaciones de conductores en un sistema gestor de bases de datos espaciales (PostgreSQL + PostGIS) y una herramienta Big Data para el almacenamiento y consulta OLAP de datos (Druid.io). Aunque ya existen métodos para la generación de objetos móviles en redes (por ejemplo [1]), no hemos encontrado ningún método que se centre en la generación de datos en tiempo real y que esté preparado para probarlo contra tecnologías de almacenamiento diversas. Por otra parte, ya existen alternativas para la visualización de millones de puntos (por ejemplo [2]). Sin embargo, estas soluciones se centran en la realización de consultas o en la visualización de resultados, pero no en ambos problemas a la vez.

2. Generación de objetos móviles en redes

La figura 1 muestra los componentes del sistema de generación y almacenamiento de datos. El código fuente de los componentes puede descargarse del GitLab del grupo de investigación[1]. El componente `OSM2Network` es el responsable de convertir un conjunto de datos obtenido desde OpenStreetMaps en una base de datos basada en PostGIS y PgRouting que permita el cálculo de rutas utilizando osm2po como base del proceso. El componente `Route Simulator` se ejecuta en el lado cliente del sistema y consta de dos componentes: `Route Computation` utiliza la base de datos generada por `OSM2Network` para calcular las rutas de los conductores, y `Driver Simulator` se ejecuta en tantos *threads* como conductores simultaneos se desee simular. Cada conductor comienza en una posición aleatoria de la red, calcula una ruta hasta un destino aleatorio, y genera posiciones a lo largo de los tramos de la ruta con la periodicidad especificada suponiendo una velocidad aleatoria expresada como un porcentaje de la velocidad máxima permitida. Por ejemplo, un conductor puede generar posiciones cada segundo suponiendo que circula siempre a un 80 % de la velocidad máxima de la vía, mientras que otro conductor puede generar posiciones cada cinco segundos circulando a un 105 % de la velocidad máxima de la vía.

El componente `Storage System` se ejecuta en el lado servidor y es el responsable de recoger y almacenar los datos en tiempo real. El punto de entrada es un sistema de mensajería (Apache Kafka) que permite desacoplar la producción de los eventos de su almacenamiento. El componente `Streaming Platform` consume los eventos de Kafka y los reenvía a diferentes sistemas de almacenamiento. Su implementación es genérica ya que es muy sencillo implementar extensiones al componente que se comporten de forma diferente simplemente implementado un interfaz Java. En nuestro caso hemos realizado dos implementaciones: una que almacena los eventos en Postgres + PostGIS (el componente `PostgreSQL Writer`) y otra que los almacena en Druid.io (el componente `Druid Writer`).

[1] https://gitlab.lbd.org.es/groups/massive-geo-data

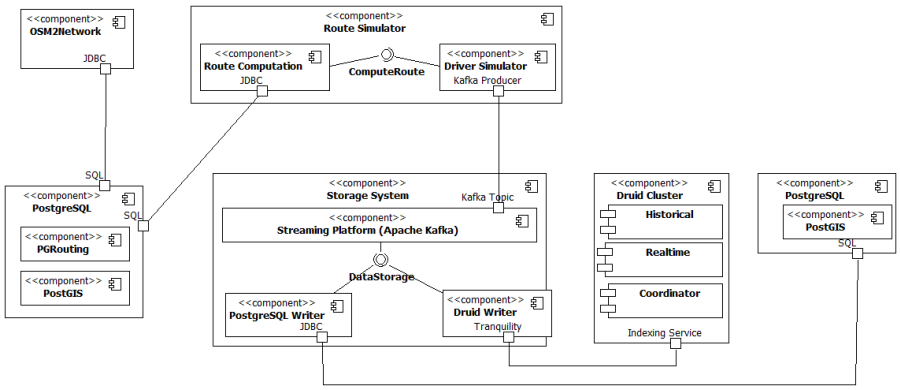

Figura 1. Diagrama de componentes del sistema

En las pruebas que hemos realizado el sistema se comporta de forma estable y eficiente. La utilización de Kafka de forma asíncrona hace que el cliente (`Route Simulator`) sea muy ligero y pueda generar eventos para 1000 conductores simultáneos en un ordenador de escritorio (Intel Core i7-3770, 4 núcleos, 3.40GHz, 8GB de RAM) sin apenas carga para el sistema. El cálculo de las rutas se ha desplegado en un servidor dedicado distinto para evitar el retraso de la simulación en tiempo real (Intel Xeon E5-2603v3, 3 núcleos, 1.60GHz, 8GB de memoria). El lado servidor se ha ejecutado en una máquina que alojaba tanto el componente de almacenamiento `Storage System` como Postgres y Druid (Intel Core i5-4440, 4 núcleos, 3.10GHz, 14 GB). El sistema se ha comportado de forma estable y ha permitido generar conjuntos de datos entre los 3,2 millones de puntos (en 3 horas) y los 40,7 millones de puntos (en unas 24 horas de ejecución continuada).

3. Visualización de conjuntos masivos de objetos móviles

La finalidad de los datos generados por el sistema es su visualización interactiva en un mapa en el que el usuario podrá establecer, mediante operaciones de zoom y desplazamiento en un mapa y un control de rango de tiempo, un intervalo espacial y temporal para el conjunto de eventos que desea visualizar. Para poder obtener una visualización fluida es necesario realizar una agregación de los puntos en el lado servidor de la aplicación y enviar hacia el lado cliente únicamente el resultado de la agregación, esto es, una colección de puntos geográficos con el número de eventos que se agregan bajo ese punto geográfico. La alternativa más simple consiste en realizar la consulta *obtener todos los puntos en el rango (xmin, ymin, tmin) - (xmax, ymax, tmax)* y aplicar un algoritmo de agregación sobre el resultado, pero es una solución computacionalmente costosa. Nuestra propuesta consiste en preprocesar cada evento cuando se inserta en el sistema añadiendo a cada punto geográfico una versión del mismo con menor precisión. Por ejemplo, si cada punto se representa en un sistema de coordenadas geográficas en el que

las coordenadas representan la longitud y la latitud en grados, un valor con una precisión de 9 decimales representa sobre la superficie de la Tierra un máximo de 1 milímetro. Almacenando 7 versiones adicionales del mismo punto geográfico con 7 precisiones diferentes (entre 2 y 8 decimales) hace que realizar el proceso de agregación sea tan simple como agrupar los eventos por valores iguales de coordenadas y contar el número de elementos.

En la siguiente tabla se muestran los tiempos de ejecución (en ms) de cuatro tipos diferentes de consultas sobre los conjuntos de datos utilizando las versiones de 2, 3, 4 y 5 decimales. La consulta *agregar todos* calcula todos los grupos sin filtrar, la consulta *Bounding box* calcula los grupos tras seleccionar los eventos dentro de un rango espacial aleatorio, y las consultas de *Distancia* calculan los grupos tras seleccionar únicamente los eventos más cerca que una distancia aleatoria pequeña (del orden de 0.1 grados) o grande (más que 0.1 grados).

	2 decimales		**3 decimales**		**4 decimales**		**5 decimales**	
	PG	**Druid**	**PG**	**Druid**	**PG**	**Druid**	**PG**	**Druid**
3.21M dataset								
Agregar todos	1 031	**398**	5 105	**948**	**6 076**	8 432	**11 436**	28 861
Bounding box	2 727	**780**	988	**727**	**2 806**	3 223	**4 587**	9 224
Distancia (pequeña)	467	**237**	852	**590**	930	**692**	**1 936**	2 399
Distancia (grande)	**3 514**	4 824	7 307	**5 346**	**7 231**	9 095	**13 803**	32 435
40.7M dataset								
Agregar todos	75 269	**3 655**	165 433	**6 140**	164 777	**27 660**	**203 098**	error
Bounding box	245 283	**6 312**	255 204	**6 436**	723 715	**25 331**	793 819	**53 091**
Distancia (pequeña)	448 786	**10 371**	236 916	**5 195**	275 405	**5 568**	218 413	**19 802**
Distancia (grande)	102 075	**64 634**	767 211	**49 623**	753 729	**64 008**	**197 401**	error

Podemos observar que Druid se comporta claramente mejor en consultas con menor precisión (menos decimales) ya que está diseñado para realizar de forma eficiente las agrupación comunes en consultas OLAP, y la menor precisión implica una mayor posibilidad de agrupación. En el caso del *dataset* pequeño PostGIS se mantiene cerca y obtiene mejores tiempos en las pruebas con una granularidad más fina (4 y 5 decimales). En el *dataset* grande PostGIS es mucho menos eficiente que Druid en las consultas con menor precisión, pero Druid tiene problemas ejecutar ciertas consultas a partir de 5 decimales, como vemos en la tabla. En cualquier caso, es necesario realizar pruebas más exhaustivas que permitan determinar los escenarios en los que una tecnología es mejor que otra.

Referencias

1. Xu, J., Güting, R.H., Qin, X.: Gmobench: Benchmarking generic moving objects. GeoInformatica **19**(2) (2015) 227–276
2. Doraiswamy, H., Vo, H.T., Silva, C.T., Freire, J.: A gpu-based index to support interactive spatio-temporal queries over historical data. In: Proc. of the IEEE Conf. on Data Engineering (ICDE). (2016) 1086–1097