Desarrollo de una aplicación de *Web Mapping* con teselas vectoriales en la plataforma NODEJS

Development in NODEJS platform of a Web Mapping application with vector tiles

Alfonso Sancho Miró

REVISTA **MAPPING**
Vol. 28, 193, 6-17
enero-febrero 2019
ISSN: 1131-9100

Resumen

Desde que en 1993 el centro de investigación Xerox PARC (California, Estados Unidos) creara el primer visualizador web de cartografía, la comunidad de desarrollo web de Sistemas de Información Geográfica (SIG) ha evolucionado considerablemente. En la actualidad se ha extendido ampliamente la arquitectura de visualizadores web basada en cuatro pilares fundamentales: un sistema gestor de bases de datos espaciales como Postgis; un servidor de mapas como Geoserver, Mapserver o Deegree; el protocolo Web Map Service (WMS); y una librería de construcción de visualizadores basada en JavaScript, como OpenLayers o Leaflet. Los visualizadores así implementados permiten cargar los datos vectoriales en formato imagen, a través del estándar WMS. Esta arquitectura se suele configurar para que soporte teselado y cacheo de imágenes, lo que proporciona un alto rendimiento. No obstante, las nuevas demandas de los usuarios plantean la necesidad de implementar visualizadores web que carguen directamente datos vectoriales, con el objeto de disponer de la geometría en el lado del cliente.

En este artículo se presenta un caso de estudio realizado en la Infraestructura de Datos Espaciales de la Confederación Hidrográfica del Guadalquivir, en el que se ha buscado una solución, aún en fase de desarrollo y pruebas, basada en la sustitución del servidor de mapas y en la utilización de teselas vectoriales. Los resultados son positivos y suponen un punto de partida para nuevas aplicaciones de web mapping que se van a desarrollar en este organismo.

Palabras clave: Mapas web, teselas vectoriales, teselas vectoriales Mapbox (MVT), PostGIS, Memcached, Google Protocol Buffers, NodeJS.

Abstract

Since 1993 when the Xerox PARC research center (California, United States) created the first web mapping viewer, the web development community of Geographic Information Systems (GIS) has evolved considerably. At present, the web viewer architecture has been widely extended based on four fundamental pillars: a spatial database management system such as Postgis; a map server like Geoserver, Mapserver or Deegree; the Web Map Service (WMS) protocol; and a library of construction of viewers based on JavaScript, such as OpenLayers or Leaflet.

The viewers thus implemented allow loading vector data in image format, through the WMS standard. This architecture is usually configured to support tiling and image caching, which provides high performance. However, the new demands of users raise the need to implement web viewers that directly load vector data, in order to have the geometry on the client side.

This article presents a case study carried out in the Spatial Data Infrastructure of the Guadalquivir Hydrographic Confederation, in which a solution has been sought, still in the development and testing phase, based on the replacement of the map server and in the use of vector tiles. The results are positive and represent a starting point for new web mapping applications that will be developed in this organization.

Keywords: Web mapping, vector tiles, Mapbox Vector Tiles (MVT), PostGIS, Memcached, Google Protocol Buffers, NodeJS.

Confederación Hidrográfica del Guadalquivir.
Ministerio de Agricultura y Pesca, Alimentación y Medio Ambiente
asancho@chguadalquivir.es

Recepción 08/01/2019
Aprobación 23/01/2019

1. INTRODUCCIÓN

En la Confederación Hidrográfica del Guadalquivir – CHG en adelante- se ha venido apostando desde hace ya más de una década por los sistemas de información geográfica basados en web, tanto desde la visión del trabajo corporativo como desde el punto de vista de difusión y divulgación de la información. A mediados de la década de 2000 se puso en marcha el Sistema de Información Territorial (SIT), que incluía la aplicación web *Hidrobase* como visualizador de cartografía. A finales de esa misma década, se modernizó el SIT para transformarlo en una Infraestructura de Datos Espaciales (IDE) (IDECHG, 2018), incorporando nuevos servicios de descubrimiento, visualización y descarga; poniendo a disposición de los usuarios la información geográfica de las demarcaciones del Guadalquivir, Ceuta y Melilla.

Desde que se puso en marcha esta IDE en 2009, la tecnología de desarrollo de visualizadores web avanzó considerablemente, lo que provocó que la IDE de la CHG se quedase un tanto desfasada. Por ello, entre 2014 y 2016, se llevó a cabo una nueva actualización de esa tecnología, incorporando tecnologías como *OpenLayers* 3, cacheado de teselas ráster y diseño adaptativo para permitir el acceso desde dispositivos móviles.

Actualmente la IDE es una herramienta bien valorada tanto por los usuarios como por los propios trabajadores de la CHG. A pesar de que está concebida como una herramienta de divulgación, un importante número de trabajadores del organismo ha tomado la IDE como herramienta de trabajo diario, debido a su rapidez de funcionamiento y a su facilidad de uso.

Este hecho ha colocado a la IDE frente a un nuevo reto: cubrir las necesidades de los usuarios internos del organismo. Estos usuarios, conforme profundizan en el uso de la IDE, detectan y proponen nuevas necesidades. Una de las más solicitadas es la posibilidad de realizar algunas operaciones de análisis y procesamiento espacial. Para lograr alcanzar ese objetivo, el actual visualizador de la IDE, basado en teselas ráster, debería incorporar la posibilidad de trabajar con información vectorial.

Este es el punto de partida desde el cual se comenzó a buscar la forma de añadir información vectorial al visualizador tratando de no penalizar el rendimiento.

1.1 La arquitectura actual del visualizador web de la IDE de la CHG

Actualmente, el visualizador web de cartografía de la IDE de la CHG se basa en la siguiente arquitectura:

- Sistema gestor de bases de datos espaciales *Postgis*
- Servidor de mapas *Mapserver* y protocolo *Web Map Tile Service* (WMTS)
- Componente *Mapcache* para cacheado
- Servidor de aplicaciones *Tomcat*, donde se ejecuta la aplicación web
- *OpenLayers*, biblioteca de construcción de visualizadores basada en *Javascript*

La principal ventaja de este modelo basado en

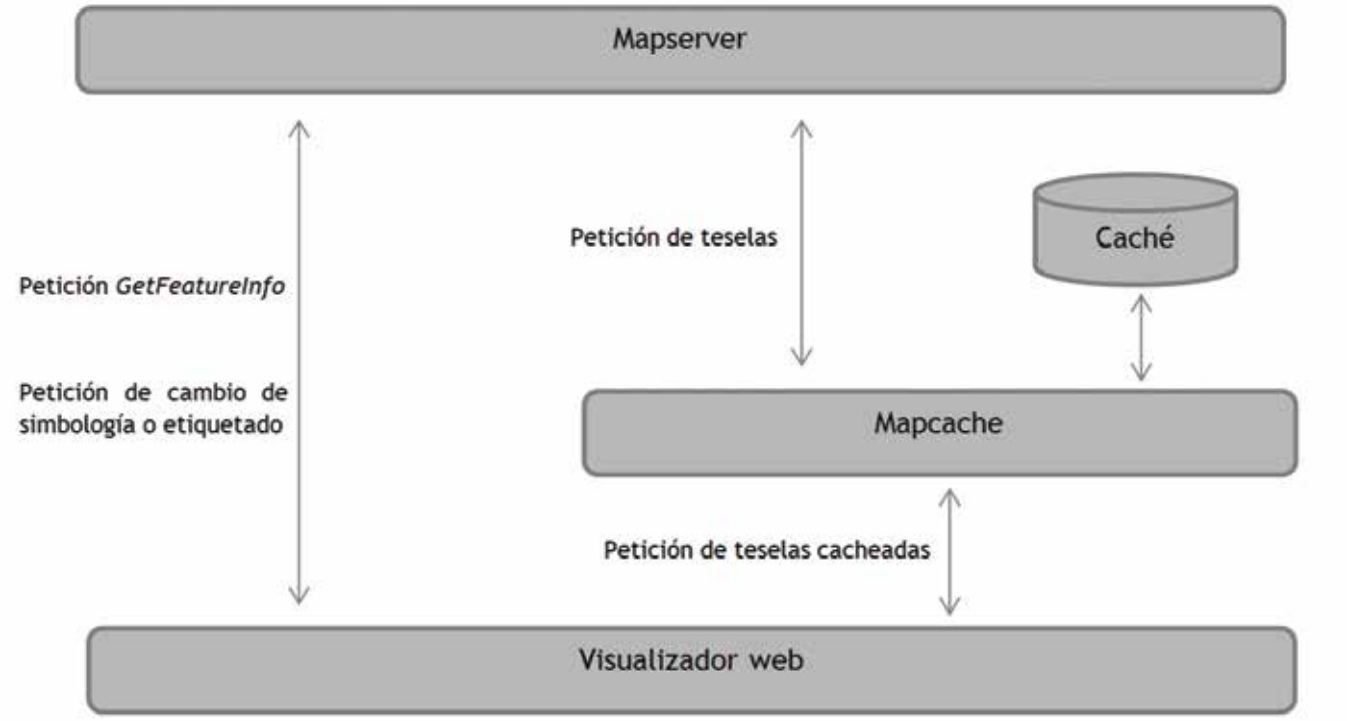

Figura 1. Arquitectura del visualizador de la IDE de la CHG

teselas cacheadas es la rapidez con la que se cargan las capas en el mapa. La mayoría de ellas tienen un tiempo de carga de 1-5 segundos, dependiendo de la capa y del nivel de zoom. El tamaño medio de las teselas cacheadas es de 5 kB.

El visualizador incorpora herramientas que permiten al usuario cambiar la simbología de una capa o etiquetar sus entidades a partir de un atributo. Para ello el visualizador web tiene configuradas varias fuentes de información. Si el usuario solicita un cambio de simbología o un etiquetado, la fuente de información de dicha capa cambia de *Mapcache* a *Mapserver*. El visualizador entonces envía a *Mapserver* los nuevos parámetros de simbología o los parámetros de etiquetado y éste sirve la capa con los parámetros solicitados. Esto se consigue gracias a la funcionalidad de sustitución de parámetros en tiempo de ejecución que ofrece *Mapserver* (2018).

Respecto a la obtención de información de las entidades, el visualizador utiliza la operación *GetFeatureInfo*. *Mapserver* permite una personalización muy alta de la salida ofrecida por esta operación, utilizando plantillas HTML, lo que permite ofrecer al usuario la información en un formato amigable y comprensible.

La principal ventaja de esta arquitectura es que los usuarios pueden disponer de herramientas avanzadas que normalmente no están disponibles en entornos basados en WMS/WMTS. El inconveniente más importante es que el visualizador de la IDE tiene un acoplamiento alto con *Mapserver*, lo que implica que actualmente no podría sustituirse *Mapserver* por otro componente como *Geoserver*, por ejemplo.

1.2 Datos vectoriales versus ráster en una aplicación de web mapping

Los datos vectoriales aportan algunas ventajas frente a los datos ráster. Permiten una mayor flexibilidad en la representación de mapas, como la posibilidad de aplicar diferentes estilos. Además, es posible almacenar y transferir no sólo las geometrías, sino también los atributos de una entidad. Una tercera ventaja es que los datos pueden utilizarse y transformarse en el lado del cliente para realizar análisis y procesamiento espacial mediante bibliotecas Javascript cono Turf (Turf, 2018).

No obstante, la utilización de datos vectoriales también presenta algunos inconvenientes. La transmisión de este tipo de datos desde el servidor al cliente puede suponer un desafío, debido al tamaño de este formato de información. La IDE de la CHG cuenta con algunas capas de información cuyo volumen es bastante elevado. Algunas de ellas cuentan cn alrededor

Tabla 1. Tiempo de descarga en función del tamaño de la información

TAMAÑO	TIEMPO MEDIO DE DESCARGA
20 MB	~ 10 segundos
50 MB	~ 26 segundos
100 MB	~ 52 segundos
500 MB	~ 4 minutos
1 GB	~ 9 minutos

** El tiempo medio de descarga se ha calculado tomando como referencia un ancho de banda de 15,5 Mbps, que es el ancho de banda medio disponible en España, según el estudio de la corporación Akamai para el primer trimestre de 2017 (Akamai, 2017).*

800 000 entidades vectoriales de tipo polígono, como por ejemplo, las parcelas de riego. Esto provoca que el tiempo de transmisión sea elevado. En la siguiente tabla se muestra el tiempo medio de descarga en función del tamaño de la información a transmitir:

Como puede observarse, para capas de información geográfica que ocupen menos de 50 MB, el tiempo de respuesta puede ser asumible. No obstante, hay que tener en cuenta que hoy en día un tiempo de respuesta superior a 20 segundos es excesivo, más aún en el contexto de una aplicación web de cartografía dinámica. Observando de nuevo la tabla anterior, cuando se supera el tamaño de los 50 MB, el tiempo de respuesta se convierte en prohibitivo.

El tamaño de la información vectorial no afecta sólo a la transmisión, sino también al renderizado del mapa en el navegador. Si hay muchas entidades vectoriales en el mapa, es posible que el navegador no disponga de recursos suficientes para el renderizado de todas, lo que puede provocar lentitud en el funcionamiento e incluso el bloqueo del navegador, más aún si estamos trabajando en dispositivos con recursos limitados de computación.

2. ESTRATEGIAS PARA EL USO DE DATOS VECTORIALES EN UNA APLICACIÓN WEB

En la actualidad, con el creciente volumen de datos geográficos disponibles a nivel global –Big Geodata-, se hace cada vez más urgente la implementación de estrategias eficientes de transmisión de información vectorial entre el servidor y el cliente.

2.1 Arquitectura basada en el protocolo Web Feature Service (WFS)

A la hora de plantear una arquitectura de transmisión de datos vectoriales, la aproximación más sencilla consiste en utilizar la arquitectura ya existente, basada en los siguientes componentes:

- Sistema gestor de bases de datos espaciales Postgis
- Servidor de mapas Mapserver y protocolo Web Map Tile Service (WMTS)
- Componente Mapcache para cacheado
- Servidor de aplicaciones Tomcat, donde se ejecuta la aplicación web
- OpenLayers, biblioteca de construcción de visualizadores basada en Javascript

Esta arquitectura se transformaría para añadir el protocolo Web Feature Service (WFS), ofrecido por el servidor de mapas y que sirve la información en formato vectorial, generalmente utilizando el estándar GML (OGC, 2018). Por lo tanto, la arquitectura de datos vectoriales quedaría así:

- Sistema gestor de bases de datos espaciales Postgis
- Servidor de mapas Mapserver y protocolo Web Feature Service (WFS)
- Servidor de aplicaciones Tomcat, donde se ejecuta la aplicación web
- OpenLayers, biblioteca de construcción de visualizadores basada en Javascript

Sin embargo, como se ha señalado anteriormente, los tiempos de respuesta para las capas de información de gran tamaño serían muy elevados, pudiendo superar el minuto, lo que hace inviable utilizar esta opción sin añadir alguna estrategia que permita mejorar el rendimiento.

2.2 Estrategia basada en transmisión progresiva

La transmisión progresiva a nivel general consiste en enviar en primer lugar una versión poco detallada del elemento a transmitir –por ejemplo, de una imagen- para a continuación, proceder al envío de versiones más refinadas del elemento. De esta manera, el usuario puede ver una versión inicial a pesar de no contar con todos los detalles. Esta estrategia ha sido ampliamente utilizada para la transmisión de imágenes por Internet.

Una de las primeras estrategias de transmisión progresiva de datos vectoriales estaba relacionada con los datos representados mediante mallas triangulares, propuesta por Hoppe (1998). Las mallas triangulares consisten en una colección de triángulos y vértices que aproximan una superficie en 3D.

Posteriormente aparecieron otras estrategias con un punto en común: la generalización. La generalización de la cartografía consiste en la selección de los rasgos que deben ser representados a una determinada escala, simplificando sus características, pero manteniendo y mejorando los rasgos cartográficos significativos (Plazanet, Affholder y Fritsch, 1995).

Utilizando la generalización, se envía un mapa vectorial menos detallado al cliente y luego se van añadiendo más detalles al mapa de forma incremental. La primera versión del mapa se suele crear utilizando un algoritmo de generalización a los datos vectoriales, como el algoritmo de Douglas-Peucker (Saalfeld, 1999). Ha habido diversas propuestas de algoritmos en esta línea.

Una de las desventajas de estos algoritmos es que suelen consumir mucho tiempo y, por lo tanto, no son prácticos para las aplicaciones web del mundo real si no se dispone de recursos de computación de altas prestaciones. Otro inconveniente es que pueden dar lugar a topologías inconsistentes, haciendo necesario contar con un mecanismo de evaluación de coherencia.

2.3 Teselas vectoriales

Las teselas vectoriales son un concepto similar a las teselas ráster, pero en lugar de contener imágenes ráster precargadas, los datos de la tesela son una representación vectorial que incluye la geometría y los atributos de las entidades que están dentro del ámbito de la tesela (Antoniou, Morley, Haklay, 2009).

Hasta hace no mucho, la generación y uso de teselas vectoriales carecía de estándares formales en la comunidad SIG. Uno de los esquemas de teselado más utilizado para datos vectoriales es el denominado Google XYZ, que tiene el siguiente patrón: /{nombre capa}/{x}/{y}/{z}.

La ventaja del empleo de teselas vectoriales es el menor tiempo de carga requerido para una tesela vectorial que para la capa completa, algo que además puede mejorarse si se emplean formatos de representación ligeros. Además, los mapas basados en teselas permiten solicitar los datos que únicamente estén dentro de la vista del usuario, evitando así la descarga de contenido innecesario en el cliente que podría afectar al rendimiento de la aplicación.

2.3.1 Formatos de representación de teselas vectoriales

En la actualidad existen tres principales tendencias respecto al formato de representación de las teselas vectoriales:

- Formatos basados en XML (eXtensible Markup Language)

XML es un formato basado en texto con el que se puede codificar información comprensible tanto por máquinas como por humanos. Dentro del ámbito de la información geográfica, el lenguaje Geograpahic Markup Language (GML) es un claro ejemplo de aplicación de XML.

El punto débil de XML es que el tamaño de los ficheros es mayor que utilizando otros formatos, debido principal-

0.0	1.0	2.0	3.0
0.1	1.1	2.1	3.1
0.2	1.2	1.3	1.4
1.5	1.6	1.7	1.8

Figura 2. Ejemplo de mapa dividido en 16 teselas

mente a que cada elemento en XML debe ir identificado con dos etiquetas, una de apertura y otra de cierre. Esta duplicidad de contenido convierte a los lenguajes derivados de XML en poco eficientes para la transmisión de información.

- Formatos basados en JSON (JavaScript Object Notation)

JSON es la notación que utiliza Javascript para representar objetos. En los últimos años ha adquirido mucha popularidad debido al gran uso de Javascript en el desarrollo web y al menor tamaño de los documentos JSON comparados con los documentos XML, puesto que JSON sólo emplea una etiqueta identificativa de cada elemento, en lugar de las dos que necesita XML.

GeoJSON es el estándar basado en JSON que se ha adoptado en el ámbito SIG. Es un formato estándar abierto para codificar colecciones de entidades vectoriales.

TopoJSON es un formato relativamente reciente de código abierto para codificar objetos espaciales. Puede verse como una extensión de GeoJSON. Una ventaja de TopoJSON es que dispone de mecanismos para reducir el tamaño de los datos. Uno de estos mecanismos consiste en la eliminación de redundancias que existan en la geometría -por ejemplo, límites compartidos por varias entidades vectoriales-. Además, TopoJSON emplea un mecanismo de compresión llamado compresión delta (Delta encoding, 2018).

- Formato binario mediante Google Protocol Buffers

Los formatos mencionados anteriormente están basados en texto. En la actualidad, dentro del desarrollo de arquitecturas web basadas en servicios, es muy común el empleo de estos formaos. Sin embargo, existe la posibilidad de emplear formatos binarios.

Google ha desarrollado el formato Google Protocol Buffers, que consiste en un mecanismo extensible, independiente de la plataforma y del lenguaje, para serializar datos estructurados, de forma rápida y sencilla; para posteriormente transmitirlos en formato binario.

2.3.2 Comparación del rendimiento de los formatos de representación

A la hora de comparar los formatos descritos en el apartado anterior, suele medirse el tiempo que se tarda en serializar un objeto representado en dicho formato. La serialización es un proceso de codificación de un objeto en un medio de almacenamiento con el fin de transmitirlo a través de una conexión en red, bien en binario o bien utilizando algún formato basado en texto como XML o JSON.

Existen diversos estudios disponibles en Internet en los cuales se ha comparado el rendimiento de la serialización de documentos en XML, JSON y Google Protocol Buffers. Los resultados apuntan a que el mejor rendimiento, con diferencia, se obtiene mediante Google Protocol Buffers, seguido de JSON y, por último, XML. De los estudios disponibles en Internet, puede concluirse que el rendimiento de Google Protocol Buffers en la serialización es hasta 6 veces mayor que en XML y JSON (Novak, 2014), (Krebs, 2016).

2.3.3 Mapbox Vector Tiles (MVT)

Mapbox, proveedor de mapas on-line, liberó en abril de 2014 la primera versión de la especificación Mapbox Vector Tiles (MVT), distribuida bajo la licencia Creative Commons Attribution 3.0 US. Se trata de un formato de teselas vectoriales codificadas en binario con Google Protocol Buffers (Mapbox Vector Tiles, 2018).

Las teselas MVT utilizan el esquema de teselado denominado Google XYZ, mediante el cual una tesela se localiza a través de tres parámetros: {X}{Y}{Z}. En este esquema, las teselas se representan con dos números: X e Y, partiendo de la esquina noroeste del mapa. Los valores de X aumentan de oeste a este y los valores de Y aumentan de norte a sur. En la figura 2 se presenta un ejemplo de mapa dividido en 16 teselas, donde cada tesela tiene unas coordenadas X, Y.

Respecto al tercer parámetro, la coordenada Z, ésta hace referencia al nivel de zoom, puesto que la división en teselas será diferente según dicho nivel de zoom. La tesela (0.1) no será la misma para un nivel 2 de zoom que para un nivel 6, por ejemplo. Por lo tanto, para especificar correctamente la tesela que queremos, necesitamos los tres parámetros X, Y y Z.

3. LA ARQUITECTURA SELECCIONADA PARA EL VISUALIZADOR DE TESELAS VECTORIALES

Uno de los principales objetivos que se plantearon en la IDE de la CHG a la hora de abordar un proyecto basado en teselas vectoriales era precisamente el implementar una arquitectura muy sencilla que incorporase el mínimo número de componentes posible.

Tras un estudio de las alternativas que se planteaban para construir un visualizador de teselas vectoriales, se

eligieron los componentes que se representan esquemáticamente en la figura 3. En los siguientes apartados se detallan los aspectos más importantes de cada uno de ellos. La arquitectura se ha implementado en los servidores de desarrollo de la IDE, encontrándose aún en fase de evaluación.

3.1 Los datos: PostgreSQL + PostGIS + Mapbox Vector Tiles (MVT)

PostgreSQL, junto con su extensión PostGIS, se está convirtiendo en una de los sistemas gestores de bases de datos geoespaciales favoritos dentro de la comunidad SIG. No es de extrañar que en un futuro pueda convertirse en líder, puesto que las últimas versiones incorporan funcionalidades muy interesantes como:

- Funcionalidades NoSQL mediante la incorporación del formato JSON-B, que cuenta con operadores nativos para trabajar con información en formato JSON al estilo del sistema gestor de bases de datos MongoDB (Marini, 2018) (Datos JSON, 2018).
- Desde PostGIS 2.4, soporte nativo al formato Mapbox Vector Tiles (MVT).

Esta segunda característica es la que determinó la elección de PostgreSQL + PostGIS como el sistema gestor de bases de datos para almacenar la información geoespacial. El hecho de que el propio PostGIS soporte el formato MVT nos permite prescindir de servicios intermedios como TileStache.

3.1.1 Manejo de teselas MVT con PostGIS

Como se ha indicado previamente, PostGIS incorpora soporte nativo a MVT desde la versión 2.4. Para ello incorpora las instrucciones ST_AsMVT y ST_AsMVTGeom. ST_AsMVT devuelve una representación en formato MVT de un conjunto de filas correspondiente a una capa. Por otro lado, ST_AsMVTGeom se puede utilizar para transformar la geometría almacenada en PostGIS en espacio de coordenadas de tesela MVT, es decir, en coordenadas tipo {X}{Y}{Z}, tal y como se detalló en el apartado 2.3.3 El resto de datos de fila, diferentes de la geometría, se codificarán como atributos.

Además, a la hora de utilizar las instrucciones anteriores para obtener teselas MVT, es posible utilizar también las instrucciones ST_Intersects y ST_MakeEnvelope de forma conjunta con las anteriores para implementar una estrategia de carga de teselas según la envolvente –bounding box-, evitando así la descarga de todo el contenido de la capa, lo que afectaría al rendimiento.

3.2 Capa intermedia de cacheado: Memcached

En un sistema de información donde se utiliza base de datos, un aspecto importante para obtener un buen rendimiento es minimizar las consultas a la base de datos. Para el caso de teselas vectoriales, una forma de lograr ese objetivo es emplear algún mecanismo de cacheado.

Memcached (2018) es un software de caché de objetos en memoria, de arquitectura distribuida, libre y de código abierto que ha sido ampliamente utilizado por diversos servicios web. Se utiliza para acelerar aplicaciones web dinámicas mediante el almacenamiento de datos y objetos en memoria, reduciendo así la cantidad de datos que la base de datos debe leer. El mecanismo de Memcached consiste en una tabla hash –es decir, pares clave/valor– que puede estar distribuida en varias máquinas y permite tanto almacenar como recuperar objetos rápidamente.

Puesto que los objetos se almacenan utilizando una clave, en nuestro caso se ha utilizado el siguiente código para almacenar una tesela:

```
<nombre de la capa><x><y><z>
```

De esta manera se tiene un identificador único para cada tesela. Por ejemplo, la tesela con identificador «embalses348», corresponde a la capa de embalses, en el nivel de zoom 8 y coordenadas de tesela X: 3, Y: 4.

En el presente caso de estudio se ha instalado un único

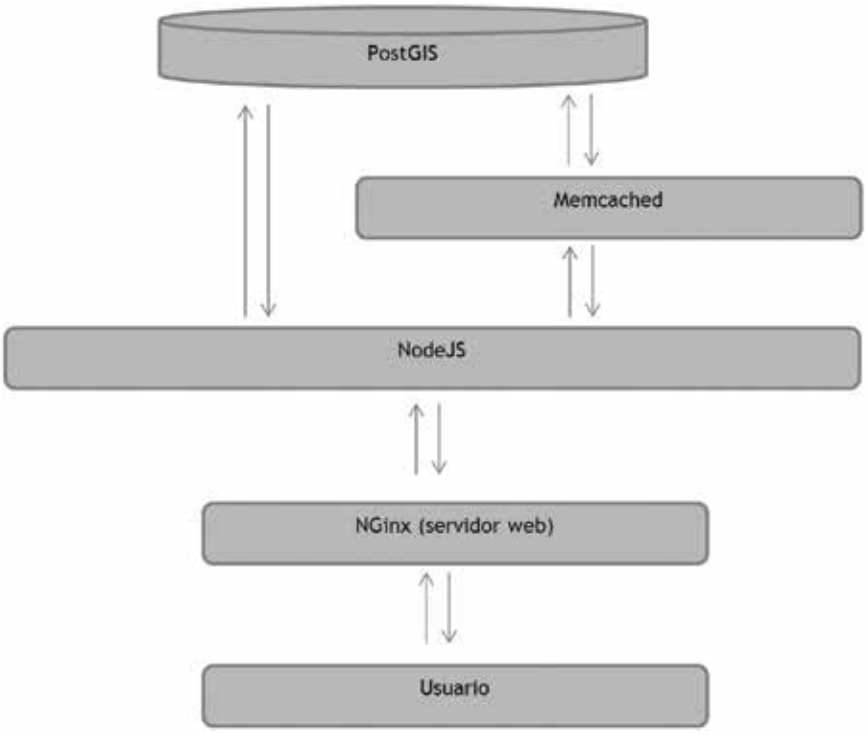

Figura 3. Esquematización de la arquitectura planteada para el visualizador de teselas vectoriales

nodo Memcached, no obstante, en un futuro podrían añadirse más nodos de cacheado si fuera necesario. A la hora de evaluar las necesidades de computación para Memcached, ha sido determinante realizar diferentes pruebas para conocer el peso real de las teselas vectoriales en memoria; pudiendo así determinar si la memoria instalada en el servidor era suficiente para el cacheado o sería necesario añadir nuevos nodos para incrementar la cantidad de memoria disponible. En el apartado 4 de este artículo se muestran los datos observados respecto a la memoria consumida por Memcached.

En el hipotético caso de que Memcached detectara que no hay memoria disponible, ante la solicitud de almacenamiento de una nueva tesela, el software expulsaría de la caché la tesela o teselas cuya última fecha de utilización sea la más antigua, hasta obtener el espacio necesario para almacenar la nueva tesela.

3.3 La aplicación: NodeJS + Express + OpenLayers

3.3.1 NodeJS: todo en uno

NodeJS es un entorno de ejecución de JavaScript orientado a eventos asíncronos diseñado para construir aplicaciones en red altamente escalables (NodeJS, 2018). Esta breve descripción resalta las principales características de este entorno:

- Utiliza JavaScript como lenguaje de programación. NodeJS incluye un motor de compilación de ECMAScript (2018) en el lado del servidor. De cara al desarrollo, esto es una ventaja importante, al poder utilizar el mismo lenguaje en la capa de presentación como en la lógica de negocio.
- Es un entorno orientado a eventos asíncronos. Esto significa que las operaciones de entrada/salida no son bloqueantes. Casi ninguna función en NodeJS realiza entrada/salida directamente, así que el proceso nunca se bloquea, lo que facilita el desarrollo de sistemas escalables.

Además, NodeJS funciona en un único hilo de ejecución, donde pueden ejecutarse hasta cientos de miles de operaciones sin incurrir en costes asociados al cambio de contexto. Este diseño atiende a necesidades de aplicaciones altamente concurrentes, en el que toda operación que realice entradas y salidas debe tener una función de retorno. No obstante, un inconveniente de este enfoque es que NodeJS requiere de módulos adicionales para escalar la aplicación según el número de núcleos de procesamiento de la máquina en la que se ejecuta.

Por lo tanto, NodeJS viene a realizar la función de servidor de aplicaciones en nuestra arquitectura. NodeJS ejecuta la aplicación web de forma completa; tanto la capa de presentación como la lógica de negocio y el acceso a datos.

La utilización de esta plataforma hace posible que eliminemos el componente Mapserver, ya que el propio NodeJS se encarga de obtener las teselas vectoriales desde PostGIS y/o Memcached y trasladarlas a la capa de presentación. Esto aporta otro aspecto muy positivo: el manejo de la seguridad se hace mucho más sencillo, puesto que la gestión de usuarios, la autenticación y la autorización serán gestionadas por NodeJS en todos los niveles de la aplicación. En la arquitectura anterior se hacía necesario implementar mecanismos de seguridad tanto en la aplicación web como en Mapserver. Con la nueva arquitectura, sólo es necesario implementar las medidas de seguridad en NodeJS.

Las extensiones de NodeJS

NodeJS es un sistema altamente extensible. Al realizar la instalación, obtenemos un sistema con las funcionalidades básicas. Mediante el Node Package Manager (NPM) podemos instalar extensiones para añadir funcionalidades a través de la instrucción «npm install <nombre_extensión>». Las principales extensiones que se han instalado en este caso de estudio han sido plugins para poder trabajar con Memcached y PostGIS, así como un plugin para trabajar con proyecciones geográficas –plugin SphericalMercator–.

3.3.2 Express, framework para el desarrollo rápido de aplicaciones web en NodeJS

Crear un proyecto de aplicación web desde cero en NodeJS puede ser una tarea relativamente sencilla si se utiliza alguno de los marcos de desarrollo –frameworks- que se han creado para la plataforma. Estos marcos proporcionan el esqueleto de una aplicación web básica, a la que únicamente es necesario añadir contenido.

Express es uno de los frameworks más utilizados para NodeJS. Según sus creadores, es un framework de desarrollo de aplicaciones robusto, rápido, flexible y muy simple. Para empezar a trabajar con Express, basta en primer lugar con instalar la extensión correspondiente en NodeJS –mediante la orden «npm install express»–; creando posteriormente un directorio para nuestro proyecto y ejecutando la instrucción «express» dentro de él. Una vez hecho esto, el esqueleto del proyecto web se ha creado dentro del directorio elegido.

Express crea la estructura de proyecto básica que se muestra en la figura 4. Existen dos ficheros fundamentales: app.js y package.json. El primero es el fichero base del proyecto, donde se indican sus componentes así como directivas de seguridad, similar al web.xml de los proyectos Java. El segundo consiste en un fichero en formato JSON que contiene los nombres y versiones de las librerías que se utilizan en el proyecto. El directorio public almacena las imágenes, ficheros de script

y hojas de estilo. Los otros dos directorios contienen las dos partes fundamentales dentro de la estructura del proyecto: los enrutadores y las vistas, cuyos directorios se denominan routes y views, respectivamente. Por simplificar, asumiremos que los enrutadores serán los encargados de implementar la lógica de negocio de nuestro sistema, así como de acceder a los componentes de datos, mientras que las vistas implementarán la capa de presentación, que será ejecutada en el navegador web del usuario.

La lógica de negocio y el acceso a datos

Como se ha explicado en el párrafo anterior, los enrutadores de Express se encargan de implementar la lógica de negocio del sistema. Estos componentes se escriben en lenguaje Javascript, el cual será compilado y ejecutado en el servidor.

En nuestro visualizador web de teselas vectoriales, la capa de lógica de negocio será la responsable de recibir las solicitudes de teselas desde la capa de presentación y obtenerlas bien llamando a Memcached o bien llamando al componente de datos en caso de que la tesela que se ha solicitado no esté aún cacheada.

En la figura 5 se muestra un fragmento de código fuente correspondiente al proceso enrutador para obtener una tesela de la capa geográfica de embalses.

Este proceso se ejecutaría cuando la capa de presentación –concretamente, OpenLayers– solicite una tesela de la capa geográfica de embalses. En primer lugar, solicita a Memcached la tesela; dicha llamada puede devolver un elemento vacío, lo que significaría que Memcached no dispone de caché para esa tesela. En ese caso, se construye la envolvente –bounding box– de la petición y se llama al componente de acceso a datos para pedir la tesela. Este componente se encargaría de comunicarse directamente con la base de datos PostGIS, utilizando las instrucciones ST_AsMVT y ST_AsMVTGeom comentadas en el apartado 3.1.1. Una vez que el componente de acceso a datos devuelve la tesela solicitada, ésta se envía a Memcached para que la almacene en la caché y finalmente se devuelve a la capa de presentación para que la dibuje en el mapa. Volviendo a la llamada inicial a Memcached, si la tesela solicitada ya estuviera cacheada, simplemente se devolvería a OpenLayers.

Obsérvese en la figura 4 que la instrucción response.setHeader indica a la capa de presentación que el contenido que se está enviando se encuentra en formato application/x-protobuf, es decir, codificado mediante Google Protocol Buffers.

La capa de presentación

Para escribir páginas HTML, NodeJS cuenta con diversos motores de plantillas, entre los que destaca Pug, antes conocido como Jade. Pug proporciona una sintaxis muy sencilla, clara y directa de escribir contenido HTML. Pug elimina muchos de los caracteres propios de HTML, como pueden ser los corchetes '<' y '>', o las etiquetas de cierre. Esto hace que tengamos código mucho más limpio y conciso, con el que resulta más sencillo trabajar. Al igual que en HTML, en Pug podemos integrar código Javascript y etiquetas de estilo CSS.

Escribir un simple párrafo de texto en HTML se haría de la siguiente manera:

```
<div>
<p>Esto es un párrafo dentro de la sección</p>
</div>
```

Mientras que en Pug bastaría con escribir:

```
div
    p Esto es un párrafo dentro de la sección
```

Cabe destacar que el anidamiento de etiquetas en Pug se realiza a través de la tabulación, por lo que es importante tabular correctamente el código, algo que no es realmente necesario en HTML.

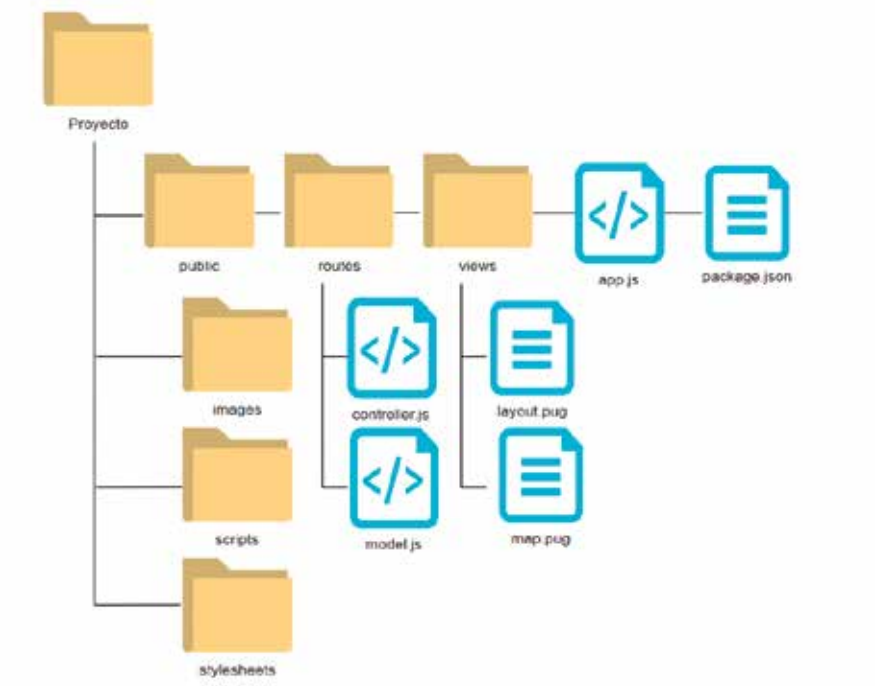

Figura 4. Estructura básica de un proyecto simple con NodeJS y Express

```
router.get("/embalses/:x/:y/:z", function(request, response) {
        let     bbox     =     mercator.bbox(request.params.x,     request.params.y,
        request.params.z);
        memcached.get('embalses'+request.params.x+''+request.params.y+''+request.
        params.z, function (error, result) {
                if ((error) || ((result === null))) {
                        if(error) {
                                console.log('Memcached: '+ error);
                        } else {
                                console.log("Memcached:
                                embalses"+request.params.x+""+request.params.y+""+req
                                uest.params.z +" no result");
                        }
                        embalsesDAO.get(request, bbox , function(err, mvt) {
                                if (err) {
                                        console.log(err);
                                } else {
                                        memcached.set('embalses'+request.params.x+''+r
                                        equest.params.y+''+request.params.z,
                                        mvt.rows[0].st_asmvt);
                                        response.setHeader('Content-Type',
                                        'application/x-protobuf');
                                       response.send(mvt.rows[0].st_asmvt);
                                }
                        })
                } else {
                        console.log("Memcached:
                        embalses"+req.params.x+""+req.params.y+""+req.params.z    +"
                        result OK");
                        response.setHeader('Content-Type',          'application/x-
                        protobuf');
                response.send(result);
                }
        });
})
```

Figura 5. Ejemplo de código fuente de enrutador para la capa geográfica «Embalses»

Por otro lado, Pug permite que las páginas hereden de otras, lo que significa que el contenido de una página se anexa al contenido de otra. Esto es útil si queremos hacer que una parte de la página sea reutilizada por varias páginas, como puede ser la cabecera.

3.3.3 OpenLayers para la representación de cartografía en el navegador

OpenLayers es una librería Javascript de alto rendimiento, repleta de funcionalidades para crear mapas interactivos en la web. Puede mostrar mapas teselados, datos vectoriales y marcadores cargados desde cualquier fuente en cualquier página web. Dentro de la estructura de nuestro proyecto NodeJS, podemos integrar los scripts de OpenLayers en una página .pug o bien crear un fichero de script, colocarlo en la carpeta public/scripts y referenciarlo desde el archivo layout.pug.

OpenLayers soporta de forma nativa el formato MVT. Para utilizarlo, es necesario crear una capa de tipo ol.layer.VectorTile, a la cual se le asocia un origen de datos de tipo ol.source.VectorTile. Dentro de dicho origen es necesario indicar que el formato es ol.format.MVT. La figura 6 muestra un ejemplo de código fuente de creación de capa MTV en OpenLayers.

```
var embalses = new ol.layer.VectorTile({
      declutter: true,
      source: new ol.source.VectorTile({
            format: new ol.format.MVT(),
            projection: 'EPSG:3857',
            url: 'embalses/{x}/{y}/{z}'
      }),
      style: new ol.style.Style({
            image: new ol.style.Icon ({
                  src: 'images/markers/
                  embalse.png'
            }))
      })
});
```

Figura 6: Ejemplo de código fuente de creación de capa MTV con OpenLayers

3.4 Resumen del proceso de solicitud de tesela

La figura 7 muestra un diagrama de actividad que representa, de forma esquemática, el flujo que sigue el proceso de solicitud de una tesela.

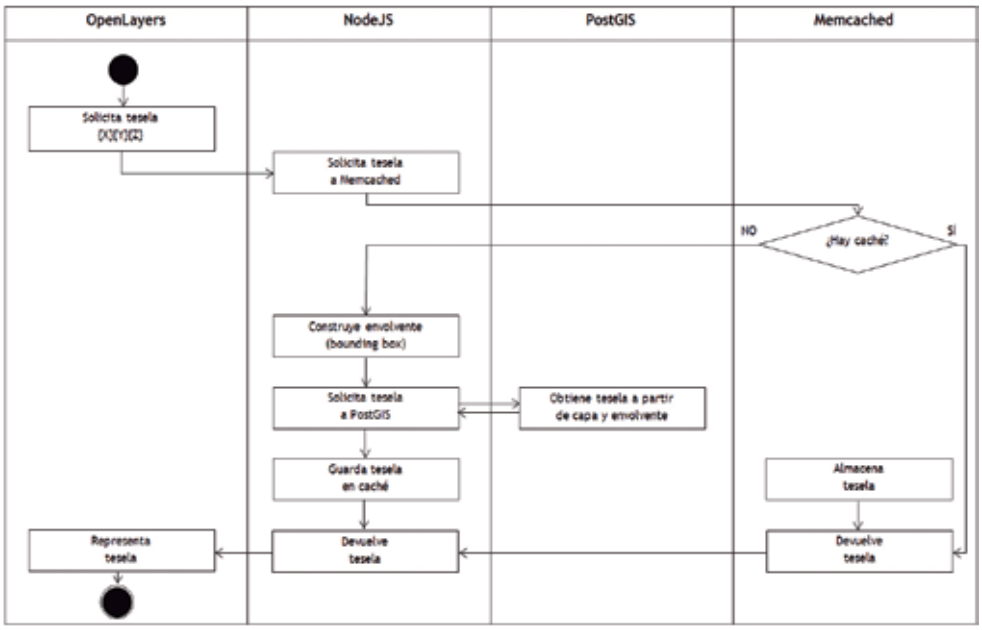

Figura 7. Diagrama de actividad del proceso de obtención de tesela vectorial

3.5 El servidor web Nginx

Nginx es un servidor web ligero de alto rendimiento, libre, de código abierto y multiplataforma. El sistema es usado por una larga lista de sitios web conocidos, como WordPress, Netflix, GitHub o SourceForge (Nginx, 2018).

Nginx fue inicialmente desarrollado con el fin explícito de superar el rendimiento ofrecido por el servidor web Apache. Según algunos estudios recientes, Nginx consigue mejorar los datos de rendimiento de Apache, proporcionando además una mejor gestión de las conexiones y de la seguridad. No obstante, Nginx adolece de una menor flexibilidad en la configuración (Walker, 2014).

4. RESULTADOS

En este apartado se muestra el prototipo desarrollado en la IDE-CHG para el visualizador de teselas vectoriales, así como los resultados agregados de las pruebas de evaluación del rendimiento, así como del tamaño de memoria empleado para el cacheo de teselas. El prototipo se ha instalado en el entorno de desarrollo de aplicaciones de la CHG y se encuentra aún en fase de evaluación y pruebas.

4.1 El prototipo

Para construir el prototipo se ha implementado la arquitectura descrita anteriormente. La interfaz desarrollada consiste en un mapa base –OpenStreetMap– y tres capas overlay de tipo MVT. Las capas utilizadas han sido las siguientes:

Las figuras 8, 9 y 10 muestran el aspecto del visualizador y de las capas utilizadas.

4.2 El tiempo de respuesta

El tiempo de respuesta es el tiempo que transcurre desde que el navegador solicita una tesela hasta que ésta se representa en el mapa, lo que incluye el tiempo de respuesta del servidor, el tiempo de respuesta de la base de datos y el tiempo de renderizado. Se han hecho pruebas a distintos niveles de zoom para las tres capas incluidas en el prototipo. El tiempo de respuesta se ha medido para las teselas sin caché necesarias para la representación de una vista del mapa para el usuario, ya que las teselas cacheadas se sirven de forma prácticamente inmediata –del orden de los milisegundos–. La figura 11 muestra una media de los tiempos de respuesta obtenidos. Obsérvese que la capa de superficies de riego es la que más tarda en procesarse; no obstante, consideramos que un tiempo de respuesta de 5,25 segundos es aceptable teniendo en cuenta que son teselas sin cachear.

Nombre	Tipo	Nº de entidades
Embalses	Punto	172
Ríos	Línea	7363
Superficies de riego	Polígono	223 801

4.3 El espacio en memoria

La utilización del sistema de cacheado de teselas permite que el visualizador ofrezca un muy alto rendimiento una vez que las teselas están cacheadas. No obstante, esto tiene un coste: el espacio utilizado en memoria.

Sin embargo, tras las pruebas realizadas, se ha comprobado que el tamaño de una tesela en formato Google Protocol Buffers es muy pequeño, lo que permite que el total de memoria utilizada no sea excesivo. La

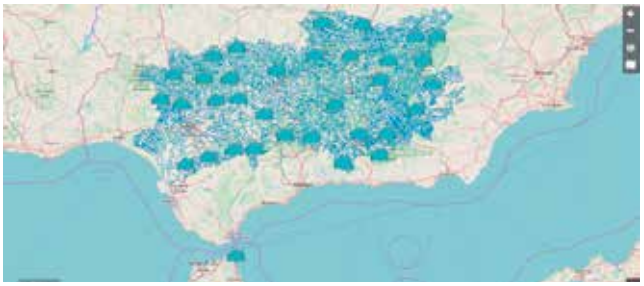

Figura 8. Prototipo del visualizador de teselas vectoriales, mostrando la capa de ríos y embalses a nivel de cuenca completa

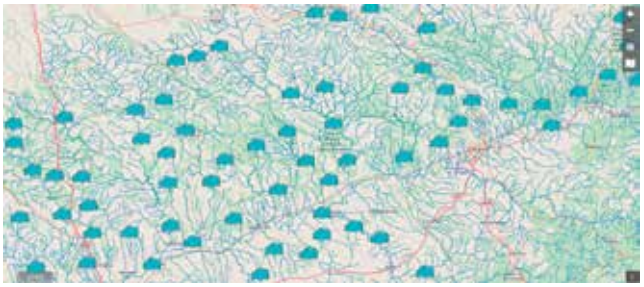

Figura 9. Prototipo del visualizador de teselas vectoriales, mostrando la capa de ríos y embalses a un determinado nivel de zoom

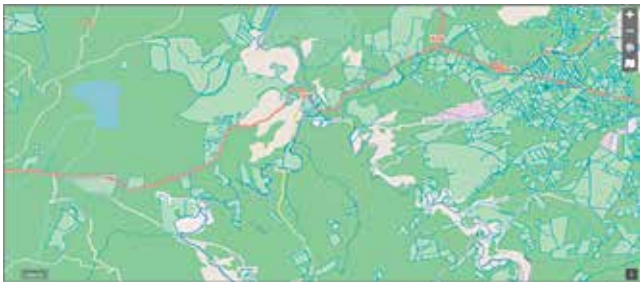

Figura 10. Prototipo del visualizador de teselas vectoriales, mostrando la capa de superficies de riego a un determinado nivel de zoom

figura 12 muestra el máximo de memoria consumido por las teselas durante las pruebas realizadas. Se han hecho pruebas a todos los niveles de zoom para las tres capas incluidas en el prototipo.

5. CONCLUSIONES Y TRABAJOS FUTUROS

En este primer prototipo de visualizador de teselas vectoriales realizado en la IDE de la CHG, se ha tratado de obtener un visualizador de arquitectura sencilla y alto rendimiento. La plataforma NodeJS nos ha ayudado a implementar todas las funcionalidades en un único entorno, característica que favorece además el control y la implementación de medidas de seguridad. Por otro lado, el formato Mapbox Vector Tiles (MVT), que se codifica en binario mediante Google Protocol Buffers, parece ser el

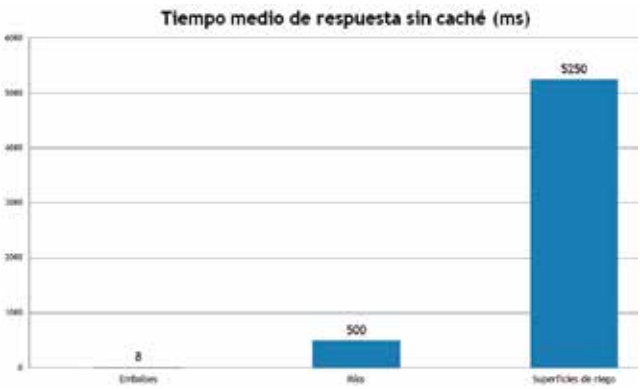

Figura 11. Tiempo medio de respuesta, en milisegundos, de las teselas sin cachear necesarias para una vista

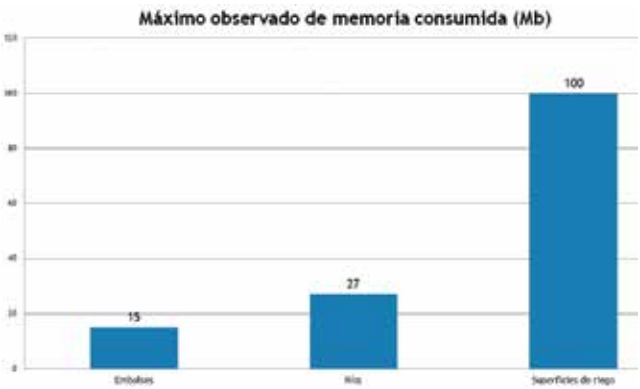

Figura 12. Espacio máximo consumido en memoria por las teselas cacheadas

formato óptimo para el empleo de teselas vectoriales.

Para capas geográficas de tamaño reducido, el tiempo de respuesta de las teselas en formato MVT es muy pequeño, del orden de milisegundos. Sin embargo, para capas con muchas entidades geográficas, este tiempo aumenta considerablemente. Para paliar este problema, se ha implementado un mecanismo de cacheado en memoria RAM, mediante Memcached. La ventaja de este sistema es que una vez cacheada, una tesela se devuelve de forma inmediata, en un tiempo del orden de los milisegundos. Un inconveniente que conlleva este mecanismo de cacheado es el consumo de memoria RAM.

Los resultados de las pruebas realizadas han sido satisfactorios, puesto que la combinación de teselado vectorial y cacheado en memoria permite obtener tiempos de respuestas muy reducidos, provocando que la navegación del usuario en el mapa sea fluida. El prototipo implementado supone un punto de partida para nuevas aplicaciones de web mapping que se van a desarrollar en este organismo.

El trabajo que se pretende desarrollar a partir de ahora es el estudio de mecanismos para lograr un uso eficiente de la memoria RAM por parte del sistema de cacheado. Algunas estrategias que se plantean para lograrlo son las siguientes:

- Cachear únicamente las capas que superen un tamaño determinado; puesto que las capas pequeñas se procesan rápidamente sin necesidad de cacheado.
- Implementar una arquitectura distribuida de cacheado, de forma que haya varios servidores realizando esta tarea, en cada uno de los cuales estaría instalado el producto Memcached.

REFERENCIAS

Akamai (2017). Informe de la corporación Akamai sobre el Estado de Internet del primer trimestre de 2017. Disponible en: https://www.akamai.com/us/en/multimedia/documents/state-of-the-internet/q1-2017-state-of-the-internet-connectivity-report.pdf

Antoniou, V., Morley, J., Haklay, M. M. (2009). Tiled vectors: A method for vector transmission over the Web. Web and Wireless Geographical Information Systems, pp. 56-71. Springer Berlin Heidelberg.

Datos JSON en PostgreSQL (2018). Referencia oficial. Disponible en: https://www.postgresql.org/docs/9.4/static/datatype-json.html

Delta encoding (2018), IETF RFC 3229. Disponible en: https://tools.ietf.org/html/rfc3229

ECMAScript (2018). Especificación del lenguaje. Disponible en: http://www.ecma-international.org/publications/standards/Ecma-262.htm

Hoppe, H. (1998). Efficient implementation of progressive meshes. Disponible en: http://hhoppe.com/efficientpm.pdf

IDECHG (2018). Infraestructura de Datos Espaciales de la Confederación Hidrográfica del Guadalquivir. Disponible en: http://idechg.chguadalquivir.es/nodo

Krebs, B. (2016). Beating JSON performance with Protobuf. Disponible en: https://auth0.com/blog/beating-json-performance-with-protobuf/

Mapbox Vector Tiles (2018). Especificación del formato. Disponible en: https://www.mapbox.com/vector-tiles/specification/

Mapserver (2018). Mapserver: Run-time Substitution. Disponible en: https://mapserver.org/cgi/runsub.html

Marini, E. (2018). Trabajando con datos JSON en PostgreSQL. Linuxito. Disponible en: https://www.linuxito.com/programacion/1060-trabajando-con-datos-json-en-postgresql

Novak, M. (2014). Serialization Performance comparison (C#/.NET) – Formats & Frameworks (XML–DataContractSerializer & XmlSerializer, BinaryFormatter, JSON–Newtonsoft & ServiceStack.Text, Protobuf, MsgPack). Recuperado de: https://maxondev.com/serialization-performance-comparison-c-net-formats-frameworks-xmldatacontractserializer-xmlserializer-binaryformatter-json-newtonsoft-servicestack-text/

OGC (2018). Open Geospatial Consortium: Protocolo WFS. Disponible en: http://www.opengeospatial.org/standards/wfs

Plazanet C., J.G. Affholder y E. Fritsch (1995). The importance of Geometric Modeling in Linear Feature Generalization. Cartography and Geographic Information Systems, vol. 22, núm. 4, pp. 291-305.

Proyecto Express (2018). Marco de desarrollo para NodeJS. Disponible en: https://expressjs.com/es/

Proyecto Memcached para cacheado en memoria (2018). Disponible en: https://memcached.org/

Proyecto Nginx (2018). Servidor web. Disponible en: https://www.nginx.com/

Proyecto NodeJS (2018). Disponible en: https://nodejs.org

Saalfeld, A. (1999). Topologically Consistent Line Simplification with the Douglas-Peucker Algorithm, Cartography and Geographic Information Science, vol. 26:1, pp. 7-18.

Turf (2018). Proyecto Turf para análisis espacial en JavaScript. Disponible en: http://turfjs.org/

Walker, R. (2014). Nginx vs Apache. Disponible en: https://anturis.com/blog/nginx-vs-apache/

Xiaohong Shang (2015). A Study on Efficient Vector Mapping With Vector Tiles Based on Cloud Server Architecture. Trabajo fin de máster, Universidad de Calgary.

Sobre el autor

Alfonso Sancho Miró

Se tituló en Ingeniería Técnica en Informática de Gestión en 2005, obteniendo posteriormente el título de Ingeniero en Informática en 2015. Desde 2009 forma parte como funcionario de la plantilla de la Oficina de Planificación Hidrológica de la Confederación Hidrográfica del Guadalquivir. Desde su incorporación comenzó a trabajar como técnico de mantenimiento de la Infraestructura de Datos Espaciales (IDE) del Organismo, asumiendo posteriormente nuevas responsabilidades hasta llegar actualmente a gestionar la IDE. Entre 2014 y 2017 llevó a cabo varios proyectos de renovación de la IDE, principalmente en lo referente a la arquitectura tecnológica, lo que conllevó una mejora considerable en el funcionamiento y una mayor aceptación por parte de los usuarios. Actualmente trabaja en la incorporación de nuevas tecnologías a la IDE así como en temas relacionados con la teledetección.