UNIVERSIDAD TECNICA FEDERICO SANTA MARIA

Repositorio Digital USM **https://repositorio.usm.cl**

Tesis USM TESIS de Pregrado de acceso ABIERTO

2017

NIVELES DE CACHÉ WEB SOBRE INFORMACIÓN GEORREFERENCIADA UTILIZANDO GEOHASH

SUAZO MATUS, JAVIER ANDRÉS

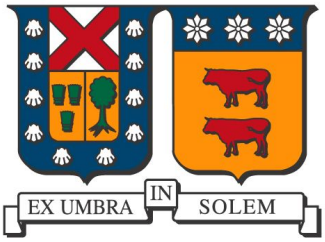
http://hdl.handle.net/11673/41242

Repositorio Digital USM, UNIVERSIDAD TECNICA FEDERICO SANTA MARIA

Universidad Técnica Federico Santa María

Departamento de Informática

Santiago - Chile



Niveles de Caché Web sobre Información Georreferenciada utilizando *Geohashes*

Javier Andrés Suazo Matus

Memoria para optar al título de Ingeniero Civil Informático

Profesor Guía: José Luis Martí Lara

Junio 2017

DEDICATORIA

Dedicado a mi madre y hermano que han sido mis pilares en cada etapa de mi vida.

A todos aquellos compañeros que me apoyaron en algún momento de la carrera.

A mi gran amigo y compañero de batallas Jorge Navarro, gracias por apoyarme tanto durante todo este camino.

Resumen

Resumen*—**El presente documento trata sobre la modificación de un servicio web de datos georreferenciados para que acepte filtrado geoespacial mediante ***geohashes, a fin de mejorar su rendimiento y disminuir su carga de trabajo. Antes de esta intervención, sólo era posible filtrar los datos mediante coordenadas y radios, razón por la cual no se podían utilizar niveles de caché o usar correctamente los existentes producto de la alta variabilidad de los parámetros utilizados en las peticiones. Para poder llevar a cabo esta mejora, se intervino el motor de búsqueda, se incorporó un tipo de caché de aplicación y se hizo uso de un Proxy Caché Inverso. En situaciones de alta demanda, mientras el cliente se beneficia por tener que gatillar menos peticiones y obtener datos con mayor rapidez, el servicio almacena respuestas reutilizables en varios niveles de caché, lo que provoca una disminución en la cantidad de consultas y en la cantidad de trabajo pesado. La solución genera beneficios para el sistema donde fue implementada y sirve de inspiración para el uso de ***geohashes*** en otros sistemas web.

***Palabras Claves*—**geohashes; coordenadas; caché; servicios; web.

Abstract

Abstract This document deals with the modification of a georeferenced data web service so that it accepts geospatial filtering through geohashes, in order to improve its performance and reduce its workload. Before this intervention, it was only possible to filter the data using coordinates and darius, which is why it was not possible to use cache levels or correctly use the existing ones, due to the high variability of the parameters used in the requests. In order to carry out this improvement, the search engine was intervened, a type of application cache was incorporated and an Inverse Cache Proxy was used. In situations of high demand, while the client benefits from having to trigger fewer requests and obtain data more quickly, the service stores reusable responses at various levels of cache, which causes a decrease in the number of queries and in the amount of hard work. The solution generates benefits for the system where it was implemented and serves as inspiration for the use of geohashes in other web systems.

***Keywords*—**geohashes; coordinates; cache; services; web.

INTRODUCCIÓN

Cuando se construyen servicios web de consulta de datos, normalmente se diseñan considerando aspectos del sistema que tienen que ver con la lógica del negocio, dejando en

segunda instancia otros aspectos igual de importantes, como el manejo de errores o el rendimiento. En muchos casos se suele poner toda dicha lógica en el ***back-end*** para simplificar la del lado del cliente que se encarga de exponer los datos a los usuarios. La situación típica es la de una aplicación web cliente que requiere listar cierta información filtrada y paginada; lo normal es que la aplicación realice la petición a cierto ***endpoint*** de la API, agregando los parámetros de que corresponden al caso. El servicio procesa la consulta, teniendo que recurrir a las correspondientes fuentes de datos y luego arma la respuesta que después es entrega al solicitante. Cuando el solicitante recibe la respuesta, simplemente imprime la información.

Si bien, muchas veces no se existe ningún tipo problema con construir los servicios de esta manera, cuando la demanda del servicio es alta, puede ocurrir que este enfoque traiga problemas de sobrecarga del servicio y en consecuencia se produzca un aumento en los tiempos de respuesta, que finalmente se traduce en una disminución en el grado de usabilidad de las aplicaciones, perjudicando la experiencia del usuario. Generalmente, si los datos son de carácter público, es decir, que no están relacionados con un usuario en particular, y si además tales datos tienen un nivel de variabilidad bajo, los arquitectos suelen incorporar niveles de caché que intercepten las consultas y respondan en lugar del servicio. La estrategia es efectiva y por lo general disminuye los tiempos de respuesta, ya que típicamente el caché está disponible en memoria. La solución es buena siempre y cuando las condiciones lo permiten. Para que un nivel de caché sea útil es necesario que sus respuestas almacenadas tengan un alto nivel de concurrencia. Si una respuesta almacenada en caché es consultada solo unas cuantas veces o nunca, el aporte que hacen estos intermediarios en el sistema es pobre.

¿Cuándo no es posible contar con niveles de caché útiles?. Si los valores de los parámetros suelen ser siempre diferentes entre una consulta y otra, no sirve de mucho almacenar respuestas de consultas que raramente y probablemente nunca vuelvan a ser consultadas. Es este el escenario al que se enfrentan quienes deben diseñar y desarrollar servicios cuyos datos necesitan ser filtrados geoespacialmente, ya que los valores de las coordenadas son muy variables. Hay que tener en cuenta que la posición geográfica de cada uno de los usuarios que utilizan cierta aplicación cliente es distinta una de otra y que generalmente requerirán de datos que están geográficamente cerca de ellos.

En el caso específico de este trabajo, un cliente de ***Modyo Spa***, en adelante ***Modyo***, requiere exponer datos georreferenciados en su sitio web a través de un mapa. Como su sitio web tiene diario una gran cantidad de visitas y el enfoque de filtrado de las consultas impedía usar niveles de caché, se optó por una actualización de datos del mapa de forma manual por medio de un botón. Esta solución es bastante burda y poco amigable para el usuario final. Encontrar una opción mejor que no fuera en desmedro de la experiencia del usuario se volvió una necesidad imperante. Fue así cómo se optó por un enfoque de filtrado diferente basado en ***geohashes*** en lugar de coordenadas.

Agregar este nuevo tipo de filtrado en la API requirió de intervenir el motor de búsqueda, incorporar un caché en la aplicación y hacer uso de un servidor caché. Por otro lado, el autor se vio en la necesidad de desarrollar una ***Single Page Application*** demostrativa que estuviera construida con la lógica necesaria para filtrar datos usando ***geohashes***.

CAPÍTULO 1: DEFINICIÓN DEL PROBLEMA

1.1 Contexto

Modyo Spa es una empresa dedicada a construir soluciones TI mediante ***Modyo DX*** (***Digital Experience***)***,*** una plataforma web desarrollada por la misma compañía. Quienes contratan la licencia de uso de esta plataforma, la pueden utilizar para construir sitios web, cuyo diseño estético, estructura y contenido dinámico es administrable por medio de un set de aplicaciones propias de la plataforma. ***Modyo DX*** puede ser utilizado en conjunto con otras tecnologías cuando se necesita cubrir las necesidades de clientes que requieren soluciones más sofisticadas, con diseño y desarrollo personalizado de experiencia de usuario, lógica de negocio e integración con otros sistemas externos.

Algunos clientes de ***Modyo*** utilizan el ***Marketing de Fusión***, también llamado ***Marketing Colaborativo***. En resumen, esta estrategia de marketing consiste en generar alianzas comerciales con otras empresas, de tal manera que sus clientes tengan la posibilidad de gozar de beneficios y promociones de otros comercios. Por ejemplo: gracias a una alianza comercial entre el ***Banco X*** y el ***Cine Y***, un cliente del ***Banco X*** tiene un 30% de descuento en entradas en el ***Cine Y*** todos los miércoles. Esta alianza busca aumentar la captura y fidelización de clientes tanto del ***Banco X*** como del ***Cine Y***. Ahora, imaginando que el ***Banco X*** requiere construir una sección en su sitio web donde se expongan todas las promociones y beneficios que tiene para para sus clientes, producto de las alianzas comerciales que ha formado tanto con el ***Cine Y*** como con otros comercios, podría contratar los servicios de ***Modyo*** para desarrollar su sitio web.

Promotions es una de las aplicaciones con las que cuenta ***Modyo DX,*** diseñada para administrar a dicho tipo de publicaciones y proveer de servicios de consulta de las mismas por medio de una ***API REST***. Cada promoción se puede describir, categorizar, relacionar con palabras claves, añadir imágenes, configurar flujos de revisión, opciones de publicación y targetización. También es posible incluir y georreferenciar todas las sucursales o puntos donde una determinada promoción está disponible (en adelante ***ubicaciones***), así el usuario del sitio puede conocer la localización geográfica de las sucursales donde puede hacer uso de la promoción o beneficio publicado. Además, cuenta con dos ***endpoints***, uno para consultar ***promociones*** y otro para consultar ***ubicaciones***, ambos con diversas opciones de filtro, orden y paginación.

La georreferenciación de estos lugares, en conjunto con otras herramientas y servicios, permite implementar distintas funcionalidades en un sitio web, tales como ordenar el contenido en base a la cercanía con una coordenada geográfica de referencia o visualizar las ***ubicaciones*** como ***marcadores*** en un mapa. Todo esto suena bien, pero sin mecanismos de optimización, el rendimiento del sistema puede verse afectado si la demanda de información es muy grande.

1.2 Situación Actual

La aplicación ***Promotions*** es un administrador de contenidos que está bien diseñada para cubrir las necesidades típicas referentes a publicaciones web sobre promociones, pero sus

endpoints actualmente no tienen incorporado mecanismos de optimización para filtrado de ***ubicaciones*** de promociones mediante parámetros de ubicación geográfica. Hace pocos meses, un ***Cliente*** de ***Modyo*** pidió la reconstrucción completa de su catálogo de beneficios, proyecto que involucró básicamente un rediseño completo y desarrollo de una ***Aplicación Web*** dentro de su sitio, cuya vista principal exponga las promociones por medio de una grilla o listado en conjunto con las sucursales asociadas (***ubicaciones***) mediante ***marcadores*** sobre un mapa. La aplicación soporta todos los requisitos de contenido requeridos por el ***Cliente***, pero considerando que el sitio web público del ***Cliente*** es uno de los más visitados en el país y tiene disponible cientos de promociones para sus clientes, cada una con al menos una sucursal y en algunos casos decenas de ellas, se predijo que el rendimiento del sistema no sería muy bueno con la versión de la aplicación que se tenía a la fecha. A continuación se explica la causa.

Figura 1: Explicación gráfica de los conceptos ***sección visible*** y ***marcador***

Como se puede observar en la **figura 1**, los ***marcadores*** son aquellos puntos dentro del mapa de un sitio web cualquiera, que representan un lugar referente a un contenido. En el contexto de la aplicación ***Promotions***, un ***marcador*** representa la ***ubicación*** de una sucursal de una promoción o beneficio (de un ***contenido***). Todo mapa requiere mantener actualizados los ***marcadores*** que se encuentran geográficamente dentro de la ***sección visible*** del mundo que está observando el usuario. Si éste interactúa con el mapa alejándose, acercándose o moviéndose dentro de él, esta sección cambia, y por tanto los marcadores dentro del mapa deben ser actualizados (así como también el listado de contenidos asociados, si es que existe uno), ya que esta interacción revela nuevas zonas del mapa mundi mientras que otras desaparecen, y en consecuencia deben aparecer y desaparecer los ***marcadores*** correspondientes a dichas zonas. Un mapa implementado de forma básica gatilla una consulta por cada alteración de la ***sección visible.*** A su vez, un servicio

implementado de igual forma procesará todas las consultas por “fuerza bruta” y sin mecanismos que impidan toda esa carga de trabajo.

Ya que en la fecha en que se desarrolló la ***Aplicación Web*** para el ***Cliente,*** el servicio de ***Promotions*** era simple y no tenía incorporado mecanismos que amortigüen la carga de procesamiento que significa tener a muchos usuarios interactuando con el mapa a la vez, se optó por solucionar el problema de una manera simple pero poco elegante, definiendo como regla lo siguiente: “Cada vez que la ***sección visible*** sea alterada, la ***Aplicación Web Cliente*** no debe gatillar una consulta nueva que refresque los ***marcadores***; en lugar de ello debe aparecer un botón de actualización de datos dentro del mapa que el usuario debe decidir cuándo presionar”. Con esta regla se evita realizar consultas demasiado seguido y en consecuencia disminuye la carga del servicio de manera abismante, pero se hace un enorme sacrificio de usabilidad al impedir que los ***marcadores*** se actualicen automáticamente.

Sacrificar la experiencia del usuario al interactuar con los elementos de un sitio web para evitar sobrecargas en el servicio no es lo ideal. Por ello, es necesario implementar una solución que no empobrezca la usabilidad del ***Sitio Web*** y que a la vez no sobrecargue el ***Servidor de Aplicaciones*** con una gran número de consultas costosas de procesar.

Luego de lo ocurrido con este ***Cliente***, para el equipo de ***Modyo DX*** se volvió una necesidad importante el poder ofrecer una versión mejorada de ***Promotions*** que fuese más sofisticada en cuanto a eficiencia y usabilidad.

1.3 Identificación del Problema

Cuando se habla de minimizar la sobrecarga de los servicios que deben procesar consultas que requieren mucho procesamiento de filtrado de datos, la alternativa a usar por excelencia es la incorporación y/o el buen uso de niveles de caché [Venka14], pues son conocidos por su gran efectividad cuando las condiciones lo permiten, e incluso son capaces de mejorar los tiempos de respuesta de los servicios web considerablemente. Pero, ¿por qué no se ha optado por hacer esto para mejorar el desempeño del servicio de obtención de ***ubicaciones***?. Es aquí donde surgen las causas que lo impiden:

Filtrado de información con parámetros extremadamente dinámicos

Hoy en día, un sitio web que consuma datos desde el ***endpoint*** de ***ubicaciones*** de la API de ***Promotions***, puede filtrar las consultas de sucursales mediante los siguientes parámetros: categoría de la promoción asociada, coordenadas del punto central de la ***sección visible***, radio de cercanía con dicho punto y coordenadas de las esquinas de la ***sección visible***. El primero se puede considerar un parámetro de filtrado con un dominio reducido de opciones, ya que el dominio de las categorías no debiese superar el orden de magnitud de las decenas de posibilidades. Distinto es el caso de los demás parámetros de filtrado; tanto el punto central como los de las esquinas del rectángulo de observación se representan con un par latitud - longitud, siendo ambos números reales con un grado de precisión de hasta 14 decimales, cuyos rangos van de -85 a +85 y de -180 a +180 respectivamente. Algo similar ocurre si utiliza el radio como parámetro, ya que su valor representa la cantidad de metros o

kilómetros máxima a la que deben estar las ***ubicaciones*** a filtrar respecto de la coordenada del punto centro.

Alto nivel de variabilidad en los valores de los filtros de consulta

Debido a la primera causa, es muy improbable dos clientes consulten por exactamente la misma combinación de valores en los parámetros de filtrado, ya que estos dependen de la sección de mapa que observa el usuario. La excepción a la regla son un par de casos, donde la carga surge de valores por defecto o bien de valores que son resultado de la búsqueda de un lugar en específico mediante un buscador que relaciona lugares y zonas con puntos geográficos.

Ambas causas impiden implementar niveles de caché en el servicio o hacer buen uso de los existentes en ***Modyo DX*** para consultas de este tipo, ya que los niveles de caché que pueden aportar al sistema necesitan que la variabilidad de los valores de los filtros sea menor. Cabe señalar que este problema no sólo se presenta en el contexto descrito en este documento; es un problema que debe enfrentar y resolver cualquier organización que desarrolla soluciones digitales y que requiere filtrar y exponer en aplicaciones y sitios web información georreferenciada usando mapas.

Existe una alternativa que permite mitigar las causas que originan el problema. Consiste en reemplazar el enfoque de filtrado actual basado en coordenadas y distancia máxima por uno nuevo basado en la división del mapa del mundo mediante ***geohashes.***

1.4 Objetivos de la Solución

Objetivo General

Modificar un servicio web de datos georreferenciados incorporando una opción de filtrado por ***geohashes*** e integrando niveles de caché para disminuir su carga de trabajo y mejorar su rendimiento.

Objetivos Específicos

- Modificar el servicio web para dar soporte al filtrado de datos mediante ***geohashes***.
- Implementar y utilizar niveles de caché que permitan reducir la carga del servicio y mejorar su rendimiento.
- Comparar consultas que usan filtro por coordenadas con consultas con filtrado mediante ***geohashes*** por medio de pruebas de carga y estrés, para verificar si el rendimiento del sistema mejora y si la carga de trabajo de su servicio disminuye bajo escenarios de alta demanda.

1.4 Alcances

El presente trabajo se limita a mejorar puntualmente el rendimiento del servicio mediante técnicas que reducen la cantidad de consultas con alto costo de procesamiento de filtrado geoespacial de datos. Sin embargo, disminuir la carga de procesamiento del servicio puede ser contraproducente al rendimiento general de otras partes del sistema, ya que se pueden generar problemas cuando el volumen de los datos a almacenar en caché es muy grande. Aunque este trabajo tiene en cuenta el problema y se toman algunas medidas para mitigarlo, no se enfoca en encontrar una solución óptima este ámbito.

La solución presentada está desarrollada bajo las limitaciones del ambiente en el que se trabajó; esto quiere decir que las técnicas y las tecnologías usadas son las que provee o soporta ***Modyo DX***. Por ejemplo, para incorporar un nivel de caché en específico pueden existir varias alternativas, pero se utiliza la que provee la plataforma (***Modyo DX***), ya que el objetivo de este trabajo no está enfocado en comparar herramientas de propósitos similares.

CAPÍTULO 2: MARCO CONCEPTUAL

Para dar paso a la explicación y los detalles de la solución, primero es necesario definir y explicar algunos conceptos..

2.1 Geohashes

2.1 Descripción

Es un sistema de geocodificación cuyo autor es Gustavo Niemeyer, quien ha puesto su invento bajo dominio público [Niemeyer08]. Consiste en una estructura jerárquica de datos espaciales que subdivide el mapa del mundo en cuadrillas rectangulares. Los ***Geohashes*** ofrecen propiedades como precisión arbitraria, prefijos similares para posiciones cercanas y la posibilidad de eliminar gradualmente caracteres del final del código para reducir su tamaño, y de esta manera perder gradualmente precisión.

Las coordenadas y los ***geohashes*** pueden ser utilizados con propósitos similares, pero no representan lo mismo. Mientras que en un mapa las ***coordenadas*** representan puntos, los ***geohashes*** representan áreas rectangulares. Como ventaja, los ***geohashes*** ofrecen una manera muy conveniente de expresar zonas geográficas delimitadas rectangularmente, ya que en lugar de dos coordenadas, cada una con un par de números decimales, utiliza una elegante secuencia de caracteres alfanuméricos. Esta es la propiedad que hace posible la solución, pues es la nomenclatura de este sistema de geocodificación la que brinda la posibilidad de usar niveles de caché sobre un sistema web. Sin embargo, esta técnica tiene sus limitaciones; no es posible representar una zona rectangular específica, por tanto, si se usan como parámetro de filtrado en una consulta de datos, el cliente necesariamente requerirá de ejecutar un segundo filtrado luego de recibir la respuesta.

2.2 ¿Cómo funcionan?

Como se mencionaba anteriormente, este sistema divide al mundo en cuadrillas, disminuyendo el tamaño de éstas a medida que la precisión aumenta. La precisión de un ***geohash*** está determinada por el número de caracteres que lo componen. En el nivel de precisión 1 (el más bajo), el mundo queda dividido como se muestra en la **figura 2a**, donde cada cuadrilla está denotada por un ***geohash*** de largo 1 (ejemplo: "D"). Ahora, para particionar una cuadrilla, es necesario utilizar la grilla de la **figura 2b.** Luego, si se desea subdividir una de precisión 2 (ejemplo: "DR") para obtener un recuadro con precisión 3 (ejemplo: "DR4"), se debe usar la grilla de la **figura 2a** y en la siguiente subdivisión la grilla de la **figura 2b** y así sucesivamente. A medida que se aumenta la precisión, se van intercalando estas grillas para subdividir las cuadrillas del siguiente nivel.

Figura 2a: División del mundo usando geohashes. Se destaca la cuadrilla D correspondiente al geohash D.

Figura 2b: División de la Cuadrilla D utilizando geohashes. Se destaca la cuadrilla R correspondiente al geohash DR.

Conforme se avanza en la precisión, naturalmente el tamaño del área que abarcan las cuadrillas disminuye desde los $5000\,Km \times 5000\,Km$ en un nivel de precisión 1 hasta los $37.2\,mm\,x\,18.6\,mm$ en el nivel 12.

La distribución de los caracteres de las grillas no siguen un orden alfabético. Su particular ordenamiento tiene que ver con el diseño del sistema y la manera en que matemáticamente es posible codificar un par latitud - longitud en un ***geohash*** y viceversa. La explicación matemática se encuentra explicada por el mismo autor de los ***geohashes*** en ***Wikipedia***.

2.2 Caché Web

2.2.1 Definición

Es toda memoria que almacena imágenes de recursos dentro de un sistema web para que sean accedidos más rápidamente que los recursos originales, y a la vez sean reutilizados en consultas posteriores similares características. Su existencia tiene como fin: disminuir los tiempos de respuesta, ahorrar consumo de tráfico de datos, mejorar el rendimiento del sistema en general y disminuir la carga de trabajo de los servicios web. A grandes rasgos funciona de la siguiente forma:

1. Se requiere un determinado recurso.
2. Se revisa si existe un caché valido para responder en lugar del recurso original.
3. Si existe, se responde con la imagen guardada en memoria caché.
4. Si no existe, se busca el recurso de forma normal y se crea una imagen de éste para consultas posteriores similares.

2.2.2 Niveles de Caché

Un sistema puede incorporar varios niveles de caché. Cada nivel puede ser visto como capa que es incorporada en algún punto del sistema, siendo la capa más externa a la más cercana al navegador (cliente) y la más interna aquella que se encuentra más cerca de el o los servicios web. Si la capa más externa no responde, se revisa la capa más interna y así sucesivamente, dejando como última alternativa el procesamiento puro de la consulta al ***Servidor de Aplicaciones***, teniendo que éste recurrir a las fuentes de datos originales para poder responder. Dependiendo de la complejidad del sistema, los niveles pueden ser muchos. A continuación se hace una breve descripción de los niveles más importantes.

- **Browser Caché o (Caché de Navegador):** si un usuario visita frecuentemente un sitio web y su navegador descarga todos los elementos que lo componen en cada visita, la obtención de los recursos necesarios sería muy ineficiente, pues obtener cada elemento en la red es lento y costoso para el sistema. Los navegadores web almacenan algunos de los elementos en la memoria local del usuario con el objetivo de cargar recursos sin la necesidad de consultarlos nuevamente a servicio del que provienen, de esta manera el sistema completo se ve beneficiado. Mientras que el usuario tendrá una percepción de tiempo de carga del sitio mucho más rápida y un ahorro en consumo de datos de navegación, el o los servicios web tendrán menos consultas que procesar. Su comportamiento está regido totalmente por el protocolo HTTP [HTTP14]. Si un recurso se puede almacenar o no localmente depende de las directivas de los encabezados de las respuestas de los servicios, los que además determinan el tiempo en que el recurso a guardar es válido y no es necesario de descargar nuevamente.

- **CDN Caché:** si existe una CDN (***Content Delivery Network***), también puede almacenar caché. Muy útil debido a los beneficios que otorga este tipo de redes [CDN17].

- **Proxy Caché:** caché generado por un servidor proxy ubicado entre el cliente y la infraestructura del ***Servidor de Aplicaciones***. Existen dos tipos, dependiendo del tipo de servidor: Proxy Caché Directo y Proxy Caché Inverso [JohnV12]. Pueden estar ambos presentes. Son la última capa antes de que el servicio sea quien tenga que responder.

- **Caché de Aplicación:** cuando ninguna de las capas más externas puede responder en lugar del ***Servidor de Aplicaciones***, es éste quien debe responder. No obstante, esto no significa que no pueda construir su respuesta con memorias caché en lugar de procesar la respuesta. Dependiendo de la tecnología con que esté construido el ***back-end***, el servicio puede almacenar distintos tipos de caché que le permitan responder de manera más eficiente.

Es importante mencionar que el comportamiento de los niveles que van desde el Browser Caché hasta el Proxy Caché están regidos total o parcialmente por las normas que dicta el protocolo HTTP [HTTP14].

2.2.3 Rails Caché

Modyo DX está construido principalmente con el framework ***Ruby on Rails***. Parte del Caché de Aplicación se encuentra desarrollado con los tipos de caché que disponibiliza el framework [RGuides]. A continuación se describen los más importantes:

- **Action Caché:** Ruby on Rails utiliza el patrón de arquitectura MVC (modelo - vista - controlador) [Puglisi16], donde los controladores son aquellos que contienen la lógica del negocio y además son los encargados de procesar las consultas y responder. Este tipo de caché se antepone al controlador y responde en lugar de él siempre y cuando contenga una respuesta caché válida para la consulta y se hayan hecho todas validaciones previas de filtros y permisos.

- **Fragment Caché:** Es utilizado para almacenar fragmentos de una página HTML que son comunes entre varias vistas.

- **Entity Caché:** Este tipo de caché almacena objetos que son frecuentemente requeridos para evitar que sean consultados constantemente a la base de datos.

2.2.4 Query Caché

Por último, otro Caché de Aplicación es usado para agilizar las consultas SQL en el motor de base de datos, que en este caso es MySQL [MySQL12]. Evita que dos consultas idénticas sean ejecutadas siempre y cuando la tabla o tablas involucradas en las consultas no hayan sufrido modificaciones en sus datos, de lo contrario el caché expira.

2.3 Pruebas de Rendimiento

Básicamente, todo producto informático que vaya a ser altamente demandado en ambiente productivo debiese contar con una evaluación previa de rendimiento mediante simulaciones. Por lo general, el propósito de este tipo de pruebas se realizan sobre ambientes de similares características al de producción y con ellas se busca examinar por medio de indicadores la capacidad de respuesta, la estabilidad, la escalabilidad, la confiabilidad, la velocidad y el uso de los recursos del sistema [Blazemeter16]. Gracias a la ejecución de estas pruebas, se pueden detectar cuellos de botella o errores, algo muy beneficioso si se tiene en cuenta que prevenirlos antes de lanzar los servicios a su uso real puede significar un ahorro considerable de recursos, tanto para la organización cliente como para la la prestadora de soluciones informáticas.

Si bien en este trabajo es necesario validar la solución por medio de este tipo de pruebas, también es necesario realizar un análisis comparativo con el enfoque preexistente de

filtrado geoespacial para determinar qué tanto mejoran el rendimiento del servicio y disminuye la carga de trabajo.

Las instrucciones de una prueba son generalmente configuradas por un software o un ***script*** de pruebas. Dichas instrucciones puede contener diversos escenarios. Cada escenario puede estar compuesto por una consulta simple (como por ejemplo una petición GET) o una serie de consultas interdependientes (ejemplo: un flujo de compra ***online*** completo). Además, cada escenario es configurado con ciertos parámetros:

- **Concurrencia:** cantidad máxima de usuarios virtuales que utilizan el servicio al mismo tiempo. Cada usuario virtual corre sobre un proceso o hilo independiente y ejecuta un determinado escenario. Cuando el escenario es ejecutado por completo y recibe la última respuesta, inmediatamente repite el proceso.

- **Ramp Up:** periodo de tiempo en que la prueba aumenta progresivamente la cantidad de usuarios virtuales hasta llegar al valor máximo posible, la concurrencia.

- **Hold For:** periodo de tiempo en que la prueba mantiene constante el valor establecido en el parámetro de la concurrencia.

Las pruebas de rendimiento se clasifican principalmente en dos tipos. La configuración de estos tres parámetros determina si una prueba es de carga o estrés.

2.3.1 Pruebas de Carga

Una prueba de carga consiste en la simulación de una gran cantidad de usuarios concurrentes por un periodo determinado de tiempo. Se caracteriza por un corto o nulo ***Ramp Up*** y un largo ***Hold For***. La idea es determinar con qué nivel de carga el servicio responde con un rendimiento aceptable. De esta manera, quien realiza las pruebas puede tener un mayor grado de certeza que al momento de pasar el software desarrollado a producción, el sistema soportará una determinada cantidad de usuarios simultáneos. Si se realizan varias pruebas modificando los parámetros de carga, se puede determinar qué tan escalable es la solución.

2.3.2 Pruebas de Estrés

Este tipo de pruebas tiene por propósito determinar los límites del sistema, donde el servicio ya no es capaz de responder. Su configuración es opuesta a las pruebas de carga, pues se caracterizan por tener un largo ***Ramp Up*** y un corto o nulo ***Hold For***, además de una concurrencia extremadamente alta en comparación con los niveles normales de esperados en su uso real. Con estas pruebas, se pueden identificar fugas de memoria, lentitud, problemas de seguridad y corrupción de datos. Es beneficioso realizarlas, ya que con ellas es posible determinar el comportamiento del sistema ante aumentos inesperados en la cantidad de clientes simultáneos.

CAPÍTULO 3: SOLUCIÓN PROPUESTA

En esta sección se detalla todo el desarrollo llevado a cabo para conseguir filtrar geoespacialmente las ***ubicaciones*** de la promociones, se definen las pruebas a que se realizaron para validar la solución y finalmente se analizan los resultados obtenidos.

3.1 Descripción de la solución

La primera parte de la solución consiste en una modificación del ***endpoint*** de ***ubicaciones*** de la API de ***Promotions*** para que soporte filtrado de los datos georreferenciados mediante ***geohashes***. De esta manera es posible hacer buen uso del ***Servidor Caché*** con el que ya cuenta el sistema e incorporar otros más, por lo que esta parte de la solución también contempla la incorporación de otro nivel de caché sobre el código fuente del ***endpoint***.

Lo descrito anteriormente es implementable, pero el hecho de que sus cambios contribuyan con mejoras al sistema es tan sólo un supuesto. Es importante demostrar con datos que el sistema ha sido beneficiado con las modificaciones hechas. Por este motivo, la segunda parte del trabajo consiste en desarrollar un script de pruebas de carga y estrés, ejecutarlo con diferentes configuraciones sobre el ***endpoint*** modificado en un ambiente de similares características a uno de producción y obtener datos sobre el comportamiento del sistema.

Cuando se utilizan ***geohashes*** en lugar de coordenadas, parte de la lógica es traspasada al lado ***front-end*** del sistema, (aplicaciones que consumen datos del servicio), por tanto es necesario realizar una tercera parte del trabajo que consiste en el desarrollo de una ***Single Page Application*** que sirva de aplicación demostrativa.

La **figura 3** muestra en términos generales la arquitectura de la solución y sus componentes principales.

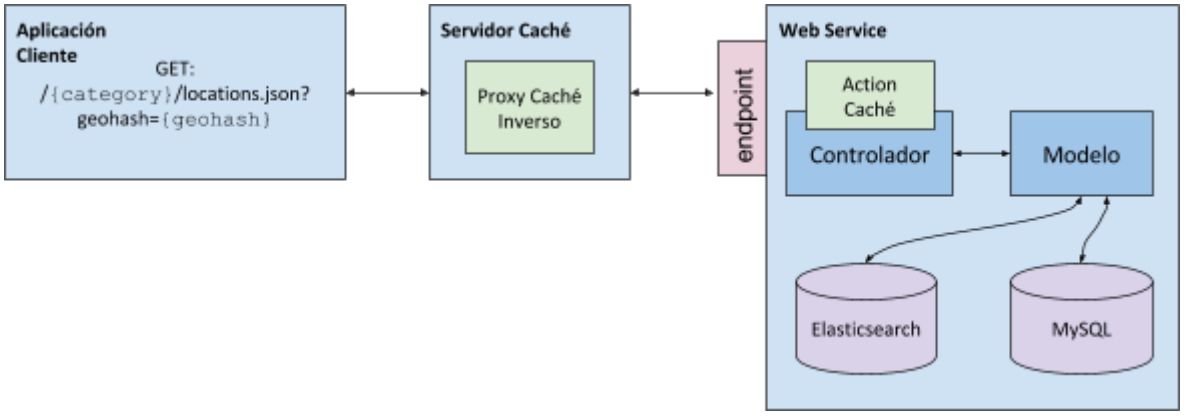


Figura 3: Arquitectura de la Solución

3.1.1 Incorporación de *Geogashes* como una Nueva Opción de Parámetro de Filtrado

Los ***geohashes*** contribuyen a resolver el problema descrito en la **sección 1.3** ya que permiten implementar niveles de caché e incluso y utilizar los pre-existentes. Para su correcto uso, es necesario desarrollar parte de la lógica en las aplicaciones clientes, obligandolas a una reducción considerable de la cantidad de consultas gatilladas durante la interacción usuario-mapa. Para entender mejor cuales son las ventajas de utilizar los ***geohashe***s, es necesario compararlos con las otras dos alternativas que ya se encontraban presentes en el sistema.

Opciones de Filtrado en el Endpoint de Ubicaciones.

La **figura 4** representa una carga inicial de puntos sobre un mapa más la simulación de un desplazamiento de la ***sección visible***, usando las tres opciones de filtrado con las que ahora dispone el ***endpoint*** y considerando que el comportamiento descrito es el de una aplicación cliente de lógica sencilla. Las zonas delimitadas de color negro representan la carga inicial de la zona del mapamundi que está siendo expuesta (***sección visible***), mientras que las zonas de color azul representan un desplazamiento generado por el usuario del mapa a una nueva zona de observación. Las zonas demarcadas de color rojo discontinuo representan las zonas cubiertas por las respuestas del servicio.

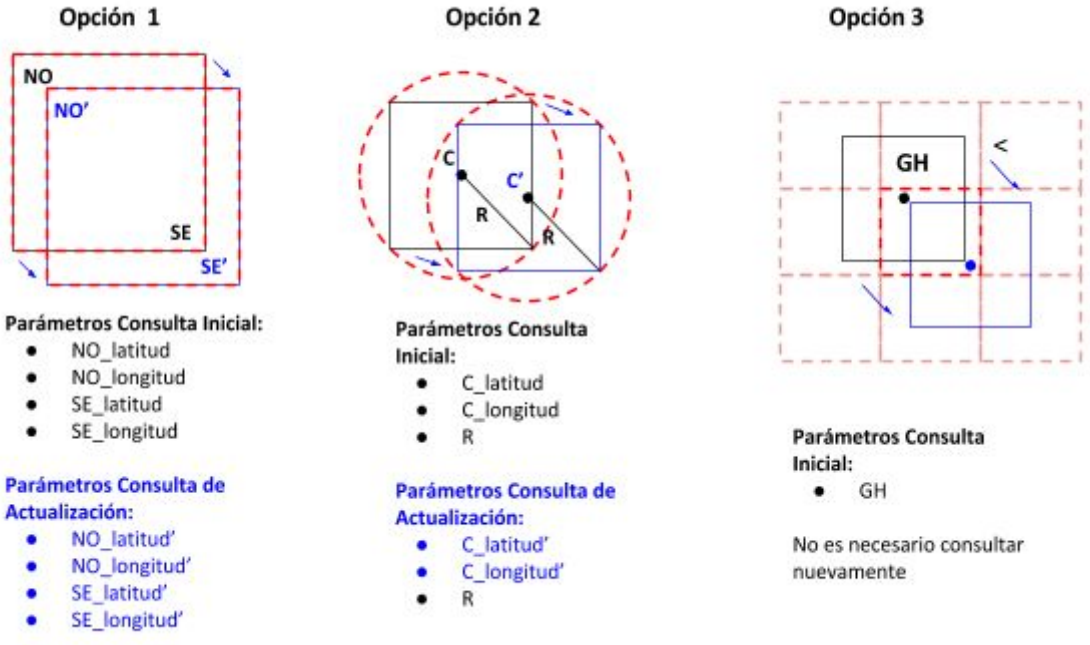


Figura 4. Descripción gráfica de opciones de filtrado de datos georreferenciados y la modificación de los parámetros al realizar un desplazamiento.

A continuación, se describen las tres opciones posibles:

- **Opción 1:** los parámetros representan una ***sección visible*** delimitada por una coordenada al nor-oeste (NO) y otra al sur-este (SE), donde cualquier pequeño

desplazamiento o alejamiento sobre el mapa, modifica los valores, teniendo que consultar nuevamente al servicio para traer los puntos que probablemente existan en las áreas no cubiertas por la primera respuesta. Una acción de acercamiento no debería gatillar la segunda consulta debido a que su respuesta está contenida dentro de la ***sección visible*** inicial, en cuyo caso basta con filtrar los datos consultados la última vez.

- **Opción 2:** en este caso, los parámetros representan una circunferencia limítrofe. Al realizar un desplazamiento se modifica la coordenada del punto central (C) mientras que el valor del Radio (R) no sufre modificaciones. Distinto es si se realiza una acción de alejamiento, ya que en este caso es el valor del Radio que varía, mientras el del valor del punto central permanece constante. En ambos casos, la combinación de valores de los parámetros varía y es necesario realizar una segunda consulta para cubrir zonas nuevas. Al igual que en la ***opción 1***, si se realiza una acción de acercamiento, no es necesario gatillar una consulta, solo se requiere filtrar la consulta anterior.

- **Opción 3:** el único parámetro que se utiliza es el ***geohash*** (GH) calculado en base a las coordenadas del punto central de la ***sección visible*** inicial (recuadro negro) y en base a base al ancho y al alto de dicho recuadro.

- El ***geohash*** a consultar en la ***Opción 3*** debe cumplir con las siguientes reglas:

    - El punto central de la ***sección visible*** debe estar contenido dentro del ***geohash***
    - El alto y ancho del ***geohash*** debe ser mayor o igual a la mitad del ancho y la mitad del alto respectivo de la ***sección visible***.

Si se cumplen estas reglas, existe certeza de que la respuesta cubre por completo el área de la ***sección visible***.

Como se puede observar en la **figura 4**, no basta con devolver la zona que cubre el ***geohash*** (recuadro rojo central), ya que en muchos de los casos el ***geohash*** no cubre toda el área de la ***sección visible***. Consultar por el mismo ***geohash*** pero con un grado menor de precisión tampoco cubre todos los casos, ya que el ***geohash*** calculado inicialmente podría ser justo una orilla o una esquina del ***geohash*** con un nivel menos en precisión. Consultar por el valor con un nivel aún menor de precisión tampoco soluciona el problema porque puede ocurrir nuevamente el mismo fenómeno. Por estas razones, es necesario que el servicio responda los datos que pertenecen tanto al ***geohash*** como a los ***neighbours*** de éste. El movimiento de desplazamiento representado en la ***Opción 3*** de **figura 4** no necesita realizar una consulta nueva, pues la respuesta de la consulta inicial cubre el espacio nuevo que se está observando. Lo mismo ocurre con una acción de alejamiento, siempre y cuando la ***sección visible*** resultante no sobrepase los límites del recuadro constituido por ***geohash*** y sus ***neighbors*** cargados en la primera consulta.

Consideraciones

Existen algunas consideraciones importantes a mencionar al momento de analizar las tres alternativas:

1. Si una aplicación cliente que consume el servicio está bien optimizada, ninguna de las tres opciones debiera consultar cuando el usuario realiza una acción de acercamiento. Cuando la aplicación cliente es básica, se suelen gatillar las consultas en todas las ocasiones. Para una buena implementación en el lado ***front-end***, la regla a seguir es la siguiente:

   "Gatillar una consulta sólo si los parámetros limítrofes se salen del área cubierta por la consulta previa, independiente de que sus valores pertenezcan o no a la ***sección visible*** anterior".

2. Es posible aminorar la cantidad de consultas gatilladas por la aplicación cliente al usar opciones 1 y 2, siempre y cuando:
   - La consulta inicial se realice por un área mayor al área que se está observando.
   - Se tenga en cuenta la ***Consideración 1***.
   - Se consulte por un recuadro o un radio mayor para realizar precargas de datos de zonas vecinas.

3. Si bien en todas las opciones es posible precargar datos, la diferencia fundamental que existe al usar la ***Opción 3*** está en que la acción de precargar datos no es una opción, es necesario para su funcionamiento. El desarrollador ***front-end*** debe necesariamente filtrar datos, pues la zona que cubre el ***geohash*** y sus ***neighbours*** siempre tendrá contenida a la ***sección visibile***.

4. Si se consulta por un área mayor a la observada es probable que se disminuya la cantidad de consultas producto de movimientos sobre el mapa, sin embargo existen consecuencias. La primera de ellas es un potencial aumento de tamaño de la respuesta que retorna el servicio. Si bien vendrían datos extras que evitarían eventualmente una o muchas consultas posteriores, es posible que dichos datos nunca lleguen a ser útiles si el usuario no se desplaza por el mapa. Entonces, es importante conocer y estudiar el volumen de datos a transferir dependiendo de la zona. Si una zona tiene una densidad de ***ubicaciones*** demasiado grande, es necesario restringir el nivel de ***zoom*** a un nivel que no provoque respuestas muy pesadas, las que pueden causar sobrecarga de trabajo para el navegador y un consumo excesivo de datos de navegación.

Ventaja de usar la Alternativa 3

Una infinidad de puntos geográficos pueden ser codificados bajo un solo ***geohash***, aumentando el grado de concurrencia con el que se puede consultar por el mismo valor puede ser muy alto, sobre todo en zonas geográficas con alta densidad de usuarios, permitiendo que los niveles de caché sean aplicables. Si un usuario hipotético realiza una carga inicial del mapa en base a su ubicación, es posible que el servicio responda con una respuesta almacenada en caché que pudo ser creada en base a otro usuario que estaba geográficamente cerca y que consultó primero por el mismo ***geohash***. Mientras más usuarios exploren zonas similares, más probable es que el servidor responda con respuestas almacenadas en algún nivel caché.

Intervención del Endpoint de Ubicaciones y el Algoritmo de Búsqueda de Ubicaciones

Para que el filtrado de datos sea compatible, es necesario intervenir el código del ***endpoint*** y el algoritmo encargado de realizar la búsqueda de los datos. Este último realiza búsquedas sobre un servidor de búsquedas implementado con ***Elasticsearch***[1]. Este servicio juega un rol fundamental, debido a que soporta geolocalización de datos y permite realizar búsquedas mediante coordenadas, radios y ***geohashes***. Gracias a una gema de ***Ruby on Rails*** es posible manejar este motor de búsqueda desde una aplicación ***Rails***. Para incorporar a los ***geohashes*** como parámetro de búsqueda sobre ***ubicaciones,*** hubo que intervenir tanto la estructura de los índices como el algoritmo de búsqueda. Para esta labor, se utilizó la documentación oficial del motor [Elastic]. Si bien este motor de búsqueda puede obtener las ***ubicaciones*** de un ***geohash***, curiosamente utiliza como entradas la coordenada del punto centro y el nivel de precisión en lugar de su representación alfanumérica. Gracias a la incorporación y adaptación de un decodificador basado en ruby [prGeohash08], fue posible decodificar el geohash en los valores requeridos.

Una vez procesada la petición realizada por el controlador del ***endpoint*** al servicio ***Elasticsearch*** responde con un arreglo de ids de ***locations***. Cuando se recibe la respuesta, se gatilla una consulta a la Base de Datos utilizando los ids obtenidos, como parámetro de filtrado.

3.1.2 Tipos de Caché a Utilizar e Incorporar en el Sistema

En la **sección 2.3** se ha explicado resumidamente cómo funciona el almacenamiento en caché y sus diferentes niveles. Para esta solución, se han considerado dos tipos: ***Proxy Caché Inverso*** y ***Action Caché***. Mientras que el primero es un nivel de caché externo al código del ***Servidor de Aplicaciones*** y solo se requiere de su correcto uso y configuración, el segundo es parte del ***Caché de Aplicación*** y requiere realizar más intervenciones sobre código del ***endpoint*** de ***ubicaciones***.

Proxy Caché Inverso

Este nivel de caché es básicamente un servicio que corre aparte, en adelante denotado como ***Servidor Caché***, que procesa las peticiones primero que el ***Servidor de Aplicaciones***. Para este caso, no se requiere de una configuración compleja, sólo se necesita que analice la ruta de la petición.

La **figura 3** ilustra principalmente el comportamiento del ***Servidor Caché***. Si este servicio intermediario tiene una respuesta en su memoria caché válida para dicha ruta, entonces responde al cliente, de lo contrario solicita al ***Servidor de Aplicaciones*** que procese y responda la consulta. Cuando ocurre el segundo caso, el ***Servidor Caché*** almacena la respuesta proveniente del ***Servidor de Aplicaciones*** en su memoria caché para consultas futuras que usen exactamente la misma ruta. Además, cada respuesta almacenada expira luego de vencido su tiempo de validez.

[1] https://www.elastic.co/

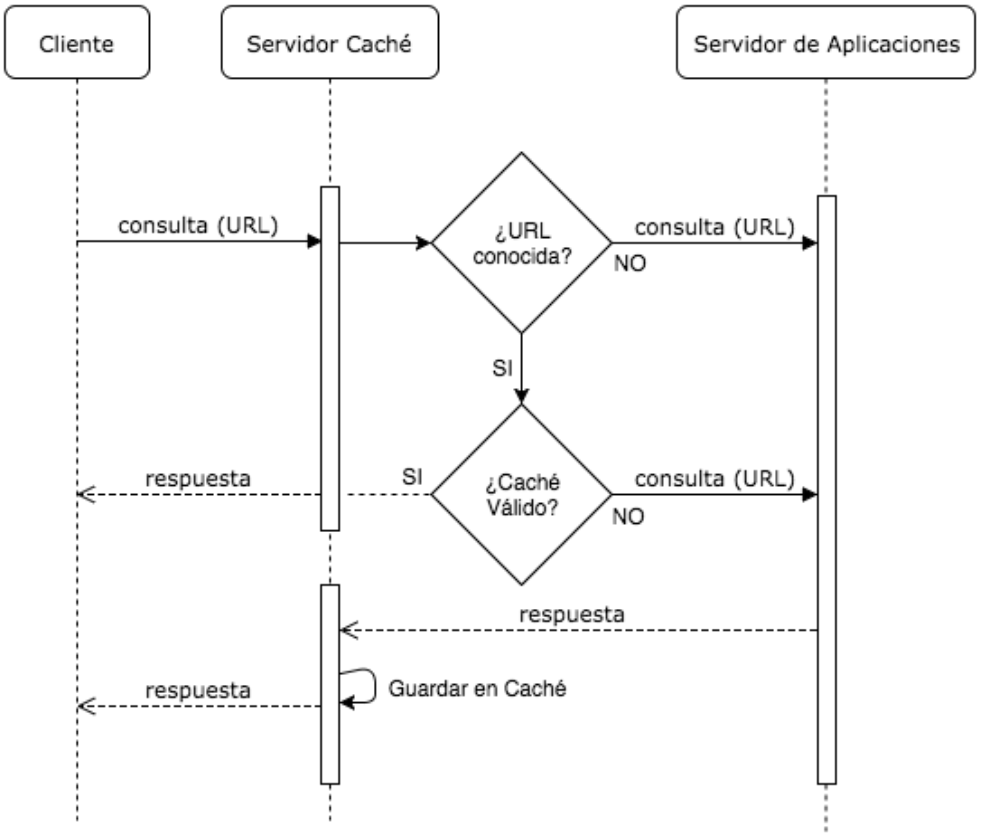


Figura 3: Diagrama de Secuencia de consulta sobre ***endpoint*** de ***ubicaciones***.

La herramienta oficial utilizada por la plataforma para montar este tipo de servicio es ***Varnish Caché***[2] ampliamente usada en sistemas web de renombre mundial.

La configuración descrita sobre este ***endpoint*** y los demás ***endpoints*** de ***Modyo DX*** de tipo "público" es la misma antes y después de este trabajo, no hubo necesidad de realizar ningún tipo de modificación sobre este servicio. La diferencia está en que que antes de esta solución, el ***Servidor Caché*** no significaba un gran aporte al servicio de ***ubicaciones***, lo que cambió radicalmente luego de incorporar el filtrado mediante ***geohashes***. Mientras más veces se realizan consultas con la misma ruta (mayor cantidad de ***hits***), más veces son reutilizadas las respuestas almacenadas en memoria caché por el ***Servidor Caché***, y por lo tanto, menos consultas pasan al ***Servidor de Aplicaciones***. En definitiva, a mayor recurrencia de las rutas, más beneficioso es contar con un ***Servidor Caché***, pero cuando se realizan las peticiones utilizando coordenadas y radios, es muy improbable que una ruta se repita más de una vez. Como consecuencia, es muy poco probable que el ***Servidor Caché*** sea el que responda, dejando toda la carga de trabajo al ***Servidor de Aplicaciones***. Al usar ***geohashes***, la probabilidad de que una consulta se repita con los mismos parámetros aumenta drásticamente. Si una aplicación cliente se modifica para utilizar este nuevo parámetro de filtrado, las respuestas que ya se estaban almacenando en el ***Servidor Caché*** ahora serán mucho más reutilizables.

[2] https://varnish-cache.org

Action Caché

Cuando el nivel de ***Caché Proxy Inverso*** (en este caso, el ***Servidor Caché***) no tiene una respuesta almacenada válida para cierta petición, es el ***Servidor de Aplicaciones*** quien debe responder. Si se habla en términos del lenguaje ***Ruby on Rails y su Arquitectura MVC***, las consultas que llegan a un ***endpoint*** son generalmente procesadas por un ***método*** definido en un controlador de la aplicación destinado para tal labor. Es éste quien normalmente analiza los parámetros que contiene la consulta, (términos de búsqueda, filtros, ordenamiento y paginación) y en base a sus valores consulta a las fuentes de datos (servidores de búsqueda, base de datos, archivos, otras APIs, etc.). Una vez que las fuentes responden los datos solicitados, el ***método*** arma la respuesta y responde lo solicitado en el formato requerido por la consulta (HTML, XML, JSON, JS, etc.).

En un controlador se pueden interponer acciones antes de los ***métodos***. Gracias a ello se pueden pre-procesar las consultas antes que el ***método***. Pero, con qué motivo hacer esto? pues en realidad existen muchos motivos, dentro de los que se encuentran el poder cargar datos necesarios o validar parámetros de la consulta. Estas acciones incluso pueden impedir que se ejecute el ***método*** cuando no es necesario. Uno de los tipos de acciones está diseñado específicamente para evitar que se ejecute el ***método*** cuando previamente ha consultado por lo mismo. Esta acción se llama ***Action Caché*** y su funcionamiento es similar al de un ***Proxy Caché Inverso***, pues valida los parámetros e impide que se ejecute el método si tiene una imagen válida de la respuesta almacena en su memoria caché. A diferencia del ***Servidor Caché***, la invalidación de las respuestas almacenadas ocurre cuando los datos cambian. Para este caso, si existe una operación CUD[3] sobre las promociones o sus sucursales, inmediatamente todas las respuestas almacenadas por el ***Action Caché*** para este ***endpoint*** dejan de ser válidas. Para explicar más en detalle cómo funciona esta invalidación, primero es necesario conocer parte del modelo de datos de la aplicación ***Promotions***.

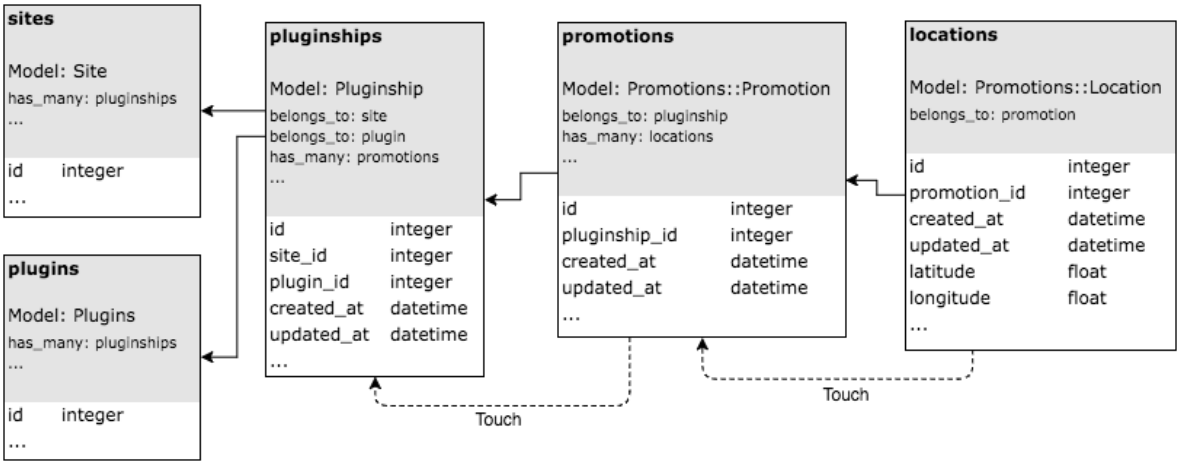


Figura 5: Diagrama Parcial de Modelos de Aplicación Promotions.

Modyo DX es una plataforma que permite crear y administrar varios sitios web sobre un mismo servicio. En la **figura 5** ***Sites*** y ***Plugins*** son entidades transversales de la plataforma. El

[3] CUD: Operaciones CRUD (***Create***, ***Read***, ***Update*** y ***Delete***) sin la acción de lectura (***Read***).

primero representa a los sitios web que existen, mientras que la segunda representa las aplicaciones que son instalables en dichos sitios, una de ellas es ***Promotions***. Una instancia del modelo ***Pluginship*** indica que una aplicación (***Plugin***) está instalada en un sitio (***Site***). De un ***pluginship*** se "sostienen" todos los datos asociados a una aplicación en un determinado sitio. Es necesario tener en cuenta que en la aplicación ***Promotions*** una promoción (***Promotions::Promotion***) puede tener muchas sucursales (***Promotions::Location***).

Cuando se ejecuta una operación CUD sobre una ***ubicación*** (***Promotions::Location***), se genera una reacción en cascada sobre las entidades dependientes superiores (***promotions*** y ***pluginships***). ***Touch*** es una acción que ha sido usada para tal efecto [Rails4]. Su actuar consiste en la modificación de un atributo de tiempo (por defecto ***updated_at***) sobre una entidad relacionada. Para este caso, el modelo ***Promotions::Locations*** se ha configurado para que, cada vez que se ejecute una operación CUD sobre una ***location*** (ubicación o sucursal), se gatille una actualización del atributo ***updated_at*** (se realice ***un touch***) sobre su respectiva ***promotion*** (promoción o beneficio). A su vez, si se realiza una operación CUD sobre una ***promotion***, se gatilla un ***touch*** sobre su respectivo ***pluginship***. En definitiva, cualquier alteración de los datos de las promociones o sus sucursales, termina alterando el atributo ***updated_at*** del ***pluginship*** que los relaciona con un sitio.

Pero, ¿para qué sirve todo esto?. En este nivel de caché se busca invalidar las respuestas almacenadas en memoria caché sólo cuando sea estrictamente necesario, esto es, cuando los datos de promociones y sucursales de la aplicación ***Promotions*** son modificados en un determinado sitio. La estrategia es hacer que el nombre que se le da a cada respuesta almacenada en caché tenga una dependencia directa con el estado estado de los datos. En Ruby on Rails, es posible tomar una instancia de un modelo y obtener un valor por medio de un método perteneciente al ***core*** del ***framework*** llamado ***cache_key***, que tiene como valores de entrada los atributos de dicha instancia, los que procesa matemáticamente utilizando ecuaciones unidireccionales que finalmente retornan como resultado un valor alfanumérico. Por ejemplo, se tiene una instancia `p` del modelo ***Pluginship*** y se ejecuta su método `p.cache_key`, cuyo resultado es un valor x. Luego se realiza una modificación de los parámetros ***latitude*** y ***longitude*** de una ***location*** que pertenece a una ***promotion*** y que a su vez pertenece al ***pluginship*** `p`. Debido a las acciones ***touchs*** gatilladas, se termina modificando el atributo ***updated_at*** de `p` producto de la reacción en cadena. Luego de esto, se vuelve a ejecutar el método `p.cache_key`, pero esta vez se tiene como resultado un un valor x' distinto de x.

Se ha conseguido que el ***pluginship*** cambie cada vez que se modifican sus datos dependientes. Ahora solo falta que el nombre de las respuestas a almacenar en caché para una determinada consulta contenga al resultado ***cache_key***. El pseudocódigo de la función utilizada para nombrar una respuesta a almacenar en caché es el siguiente:

```
concatenar(url, p.cache_key)
```

Así, cada vez que se almacena una nueva respuesta, se registra con un nombre cuya conformación depende de la URL (que contiene entre otros, los parámetros de filtrado categoría y ***geohash***) y el resultado del método ***cache_key***, que representa el estado de los datos que contiene la respuesta en el momento en que fue almacenada. De esta manera, cuando se recibe una petición en el ***endpoint*** que debe ser procesada por el ***Servidor de***

Aplicaciones, ***Action Caché*** busca en base a los parámetros de filtrado al correspondiente ***pluginship*** y utiliza la función anterior para determinar el nombre de la potencial respuesta almacenada. Si encuentra una respuesta almacenada en caché cuyo nombre es igual al resultado de la función, entonces responde con ésta, de lo contrario pide al método correspondiente dentro del controlador que procese esta consulta, luego almacena el resultado procesado en memoria caché y responde al cliente.

Existen varias alternativas para montar este nivel de caché sobre aplicaciones ***Ruby on Rails***. En este caso, la opción escogida por ***Modyo DX*** es ***Redis***[4], un motor de bases de datos que funciona en memoria.

3.1.4 Desarrollo de una *Single Page Application Demostrativa*

En secciones anteriores se ha determinado que no es posible incorporar a este método de búsqueda georreferenciada a una aplicación cliente sin que la lógica de su código se torne un poco más compleja de lo que sería una aplicación web sencilla que sólo se limita a consultar cada vez que el usuario altera la ***sección visible*** del mapa. Es necesario implementar una lógica que determine cuándo es necesario solicitar más datos al servicio.

3.1.4.1 Lógica de Consulta de Datos Mediante Geohashes

Considerando a la ***categoría*** como un parámetro opcional dentro de la consultas, se desarrolló una ***Single Page Application*** demostrativa, cuyos componentes principales están basados en el ***layout*** de la **figura 1** y cuyo algoritmo de consulta de ubicaciones sigue las siguientes instrucciones:

1. En la carga inicial de la aplicación, calcular ***geohash*** en base al punto central y tamaño de la ***sección visible***.

2. Consultar al ***endpoint*** de ***ubicaciones*** por los puntos que pertenecen a este geohash.

3. Obtener el listado las ***ubicaciones*** que coinciden con la ***sección visible***.

4. Por cada elemento del listado, agregar un item al listado de contenidos y un marcador en el mapa en la coordenada correspondiente.

5. Por cada movimiento que hace el usuario dentro del mapa (desplazarse, alejarse o acercarse), determinar si los límites de la nueva ***sección visible*** están o no contenidos dentro de la última consulta. Si lo están, repetir los pasos 3 y 4, de lo contrario repetir los pasos del 1 al 4.

6. Por cada cambio de categoría, determinar si la última consulta se realizó con o sin filtro por ***categoría***. So ocurre lo primero, repetir pasos 1 al 4, sino repetir los pasos del 3 al 4.

[4] https://redis.io

Para poder codificar una coordenada y un nivel de precisión en un ***geohash*** desde esta aplicación web, se utilizó una biblioteca ***Javascript*** desarrollada por Chris Veness [Veness16] para codificar y decodificar geohashes.

3.2 Pruebas de Carga y Estrés

En la teoría, llevar a cabo toda esta implementación significa un aporte al sistema. Pero para poder demostrarlo, es necesario someter los cambios a pruebas que determinen con números si la velocidad de respuesta del servicio en general aumenta y si la carga del Servidor de Aplicaciones es menor si se compara con el filtrado de ubicaciones mediante coordenadas cuando los ambientes y las configuraciones son iguales.

3.2.1 Script de Pruebas

Este script fue esarrollado utilizando ***Blazemeter Taurus***[5]***.*** Contiene dos escenarios, uno que realiza las consultas de prueba mediante coordenadas (***Opción 1***) y otro que las realiza mediante geohashes (***Opción 3***). Ambos escenarios gatillan peticiones GET al ***endpoint*** de ***ubicaciones*** contra el servicio de pruebas, utilizando como fuente de datos para la generación de parámetros de filtrado aleatorios a dos archivos CSV:

- **categorias.csv:** contiene 6 filas y una columna con sólo 5 categorías. Una de las filas está vacía para realizar consultas que no consideren este filtro.

- **areas_delimitadas.csv:** archivo con 40 filas, cada una con un ***geohash*** en la primera columna, cuyo valor representa a algún sector poblado de alguna ciudad dentro de Chile. Otras columnas contienen los valores mínimos y máximos de latitud y longitud del ***geohash***. Con estos valores más las reglas descritas en la Opción 3, es posible decodificar rectángulos aleatorios representados con las coordenadas de sus esquinas NO y SE, que de ser codificadas nuevamente, se obtendría el mismo geohash.

Cada vez que uno de los escenarios construye la ruta con la que hará la consulta, obtiene una fila aleatoria de cada uno de los dos archivos, y con los datos de dicha fila genera los valores de los parámetros de la petición a realizar.

Las rutas que consultan los escenarios, tienen la siguiente estructura:

- **Escenario 1:**

    - **ruta:** `{url_base}/{categoría}`/promotions/locations.json?
      bbox[]=`{NO_lat}`&bbox[]=`{NO_long}`&
      bbox[]=`{SE_lat}`&bbox[]=`{SE_long}`

    - **url_base**: indica la ruta base del servicio al que se consulta, en este caso su dominio. Ej: http://pruebas.cliente.com

[5] https://gettaurus.org

- **categoría**: categoría obtenida aleatoriamente desde categorias.csv

- **NO_lat**: latitud de esquina noroeste de un rectángulo aleatorio calculado con los datos de una fila aleatoria de areas_delimitadas.csv. Los demás valores de las coordenadas (NO_long, SE_lat, SE_long) se obtienen de la misma forma y usando la misma fila.

- **Escenario 2:**

  - **ruta:** `{url_base}/{categoría}`/promotions/locations.json?geohash=`{geohash}`

  - **geohash:** valor obtenido de una fila aleatoria de areas_delimitadas.csv.

3.2.2 Variables de Configuración

- **Concurrencia:** cantidad máxima de usuarios virtuales realizando consultas al mismo tiempo.

- **Ramp Up**: ver **sección 2.3**.

- **Hold For:** ver **sección 2.3**.

- **Servidor Caché:** mediante una cookie de sesión, es posible que una consulta pueda realizar un bypass al servidor caché y consultar directamente al ***Servidor de Aplicaciones***, ya que el servicio está configurado para no almacenar en este nivel de caché aquellas consultas provenientes de un usuario con sesión activa. De esta manera se puede "activar o desactivar" el uso del servidor caché por medio del envío u omisión de esta cookie en la consulta.

- **Escenario:** opción de filtrado geoespacial a utilizar en la prueba. Para este caso se consideran solo las opciones 1 y 3 (ver **sección 3.1.2.1**), correspondiente a los escenarios 1 y 2 descritos en la **sección 3.2.1**. Se descarta la realización de pruebas utilizando puntos centrales y circunferencias.

Pero, ¿Por qué se descarta la opción 2?, principalmente porque es una alternativa con muy pocos beneficios en comparación a las otras dos. A menos que la ***sección visible*** tenga forma de círculo y no un rectángulo (como ocurre la gran mayoría de las veces).

3.2.3 Listado de Pruebas

Al configurar las variables con diversos valores, es posible obtener un sin fin de pruebas diferentes. En esta ocasión, se realizaron 16 pruebas distintas utilizando la configuración de la **tabla 1**.

Grupos	**1**				**2**				**3**			
Nº Prueba	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**
Ramp Up [min]	0				0				10			
Hold For [min]	5				20				10			
Concurrencia	5				20				1500			
Servidor Caché	NO	SI	NO	SI	NO	SI	NO	SI	NO	SI	NO	SI
Escenario	1		2		1		2		1		2	

Tabla 1: Configuración de pruebas de carga y estrés llevadas a cabo.

Todas las pruebas están agrupadas en tres grupos de cuatro pruebas cada uno. Las pruebas que conforman un grupo tienen en común el valor de las tres primeras variables: ***Ramp Up***, ***Hold For*** y Concurrencia. Estas variables determinan el comportamiento de los usuarios virtuales. La idea es comparar las pruebas de un mismo grupo para observar y analizar el rendimiento y carga del sistema cuando estas variables se mantienen constantes, mientras se varían tanto el escenario a usar como la presencia o no del ***Servidor Caché***.

Los grupos 1 y 2 son pruebas de carga. Si se comparan los resultados de ambos grupos, es posible analizar cómo afecta el aumento en los valores de carga (tres primeras variables) para una mismo ***Escenario*** y configuración de ***Servidor Caché***.

Debido a su configuración de ***Ramp Up*** y ***Hold For***, el grupo 3 es una prueba de estrés que se busca identificar los límites que soporta el servicio cuando se ve enfrentado a un nivel abismante y poco realista de usuarios concurrentes, donde, al igual que en los dos primeros grupos, las variables que cambian son ***Servidor Caché*** y ***Escenario***.

3.2.4 Ambiente

Cliente

Blazemeter[6] es un servicio en la nube que permite emular usuarios virtuales utilizando hebras de procesamiento y distribución de trabajo en varias máquinas virtuales, permitiendo emular una concurrencia de usuario mucho mayor a la posible utilizando una herramienta de PC convencional, como por ejemplo ***JMeter***[7].

Para llevar a cabo las pruebas, se ha subido el script junto con las fuentes de datos CSV al servicio ***Blazemeter*** y se han gatillado las pruebas usando las 12 configuraciones desde un servicio AWS[8] ubicado en Virginia.

[6] https://www.blazemeter.com
[7] http://jmeter.apache.org
[8] https://aws.amazon.com/es

Servicio

Modyo tiene ambientes de prueba para sus clientes, los que son gestionados por ***Modyo*** como tal o en caso contrario por el mismo cliente.

El servicio está montado en la nube sobre un EC2[9] de AWS. Las características de la máquina son las siguientes:

- 2 CPU virtuales
- 7,5 GB de memoria
- 32 GB de almacenamiento en SSD

Como es un ambiente de pruebas, tanto el ***Servidor de Aplicaciones*** como el ***Servidor Caché*** corren sobre la misma máquina. En ambientes de producción suelen estar separados.

En esta ocasión, el servicio es gestionado por el cliente, por tanto Modyo no tiene acceso a todo el comportamiento de la máquina, solamente puede analizar con ***Newrelic*** al ***Servidor de Aplicaciones***, no pudiendo obtener indicadores del comportamiento del ***Servidor Caché***. Para efecto de análisis, esto podría considerarse una limitante en este trabajo, sin embargo es posible deducir lo que sucede con el ***Servidor Caché*** de manera indirecta mediante los resultados obtenidos del lado del cliente de pruebas.

3.2.5 Indicadores

Obtenidos en el Cliente

Corresponden a los datos que describen la calidad y velocidad de las respuestas recibidas por ***Blazemeter***. Permiten comparar entre las configuraciones y determinar cuál de ellas tiene un rendimiento mayor. Los indicadores considerados son los siguientes:

- **Media de Respuesta:** tiempo promedio en milisegundos en que se demora una petición en ser respondida.

- **Número total de Muestras:** cantidad total de respuestas recibidas durante la prueba.

- **Porcentaje de Error:** porcentaje de respuestas no exitosas.

- **Rendimiento en hits/s:** cantidad media de la cantidad de respuestas recibidas en un segundo.

- **Rendimiento en kb/s:** cantidad media de ***kilobytes*** que se recepcionan en un segundo.

- **Media de Bytes:** tamaño medio de las respuestas recibidas por la prueba en ***bytes*** .

[9] https://aws.amazon.com/es/ec2

Obtenidos en el Servicio

Datos que describen el nivel de carga, calidad y velocidad con que el servicio procesa las peticiones. Todos estos han sido obtenidos gracias a la herramienta ***Newrelic***[10] que monitorea los servicios. Los indicadores escogidos son las siguientes:

- **Media de Procesamiento:** tiempo promedio en milisegundos en que se demora el servicio en procesar una consulta.

- **Apdex:** cada petición que procesa el servicio es evaluada de acuerdo al tiempo que tomó su procesamiento. Se le otorga nota 1 si la respuesta demora menos de 5 segundos, en caso contrario se evalúa con un 0. El apdex es el promedio de las notas de las peticiones procesadas en un determinado periodo de tiempo.

- ***Throughput*:** cantidad promedio de peticiones recibidas por el servicio en una determinada cantidad de tiempo.

- **Memoria:** cantidad de memoria utilizada por el servicio en ***Megabytes***.

- **CPU:** porcentaje de uso promedio que hace el servicio del procesador. Su valor puede superar el 100% cuando se cuenta con varios ***cores,*** ya que dicho valor está relacionado con la potencia máxima de un solo ***core***.

3.2.5 Resultados y Análisis de los Datos Obtenidos

Grupos	Nº Prueba	Media de Respuestas	Muestras	% de Error	hits/s	kb/s	Media de Bytes
1	1	248	6.019	0	20,06	80,55	4.111,8
	2	242	6.198	0	20,66	80,22	3.976,1
	3	81	18.497	0	61,66	474,27	7,876,3
	4	2	534.173	0	1780	13.645,09	7.849,8
2	5	906	26.378	0	22,08	86,97	4.033,4
	6	904	26.543	0	22,12	86,4	3999,7
	7	311	77.137	0	64,28	494,38	7.875,6
	8	5	26.220.012	0	2183	16.732,54	7.848,8
3	9	250	1.258.219	99,51	2486	750,69	309,2
	10	217	1.181.466	99,51	2585	778,79	308,5
	11	412	1.016.802	0,05	1569	12023,16	7846,9
	12	52	1.346.902	0	4044	30988,49	7846,7

Tabla 2: Resultados Obtenidos desde el lado del cliente.

[10] https://newrelic.com

Grupos	Nº Prueba	Media de Procesamiento	Apdex	Throughput	Memoria	CPU
1	1	233	0,97	1130	3789	151
	2	225	0,97	1180	3789	151
	3	67,2	1	3230	3789	146
	4	25,3	1	63	3789	16
2	5	857	0,59	1310	3789	177
	6	850	0,61	1280	3789	173
	7	279	1	3690	3789	177
	8	46,8	1	61	3789	6
3	9	7340	0,07	729	3789	111
	10	7170	0,07	704	3789	105
	11	1680	0,4	2660	3789	187
	12	42,7	1	55,8	3789	2

Tabla 3: Resultados Obtenidos desde el lado del servicio.

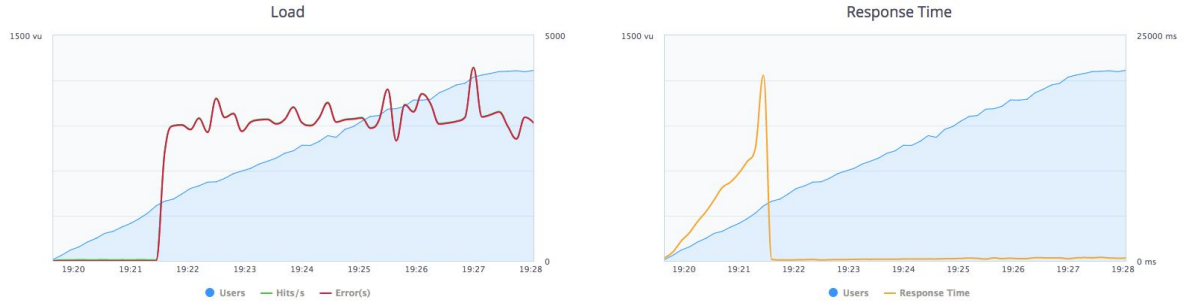


Figura 6: Gráficos del comportamiento de carga y tiempos de respuesta durante la prueba 9.

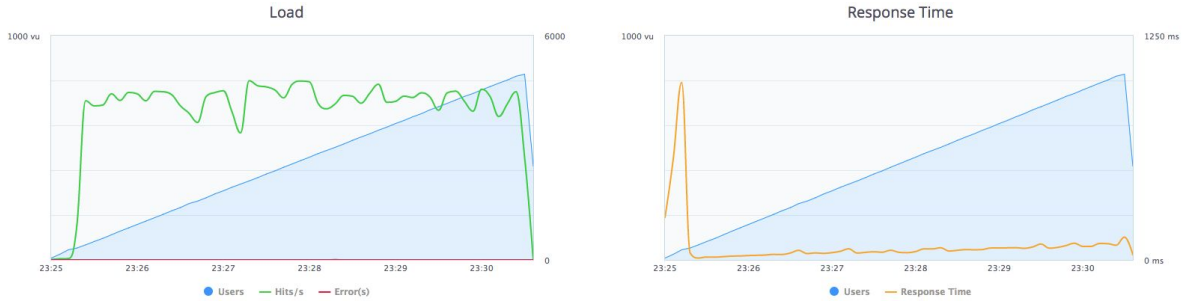


Figura 7: Gráficos del comportamiento de carga y tiempos de respuesta durante la prueba 12.

Existe un indicador que resultó ser constante en todas las pruebas. Se trata del uso de memoria. Su explicación radica en que la API es un servicio desarrollado con el framework Ruby on Rails, pero en lugar de usar por debajo Ruby puro utiliza JRuby, que en el fondo es código java interpretado de un código Ruby. Dicho código corre sobre una JVM (***Java Virtual Machine***), la cual reserva una cantidad fija de memoria al iniciarse, por tanto ***Newrelic*** no es capaz de monitorear el comportamiento de la memoria y sólo puede observar la memoria utilizada por la máquina virtual.

Otro de los comportamientos más notorios que aparece al observar los datos obtenidos de las pruebas, es la gran similitud que tienen los resultados de las dos primeras pruebas de cada grupo. En todos los grupos, la única configuración diferente entre ambas es la presencia o no del ***Servidor Caché***. Por lo tanto, los resultados demuestran que cuando se utilizan coordenadas como parámetro de filtrado, contar con un ***Proxy Caché Inverso*** no resulta ser un aporte al sistema. Esto debido a la variabilidad de los valores de los parámetros de consulta cuando se utiliza el ***Escenario 1***.

Entre las pruebas del grupo 1 y 2 lo que se hace es cuadruplicar tanto la ***Concurrencia*** como el ***Hold For***. Si se comparan los resultados las pruebas de los dos primeros grupos con sus homólogas del grupo 2, el rendimiento del sistema se vió afectado en todos los casos. Esto es normal, pues independiente de quien responda, un servicio tenderá a responder más lento cuando su demanda aumenta.

Al analizar el comportamiento de los tiempos de respuesta de los grupos 1 y 2, la diferencia entre los tiempos de respuesta entre las consultas que usan el ***Escenario 1*** versus la que usan el 2 es abismante, sobre todo cuando se hace uso del ***Servidor Caché*** (pruebas 4 y 8), donde la media de las respuestas llega a ser en promedio **cientos** de veces más rápida. Es realmente notoria la enorme diferencia que existe entre responder desde este nivel de caché a responder con una respuesta procesada desde cero. Si bien, las pruebas que usaron ***geohashes*** sin ***Servidor Caché*** obtuvieron tiempos de respuesta bastante mejores que las pruebas que usan coordenadas (entre 3 y 4 veces más rápidas), el ***Throughput*** obtenido es el mayor registrado en ambos grupos, cerca de 3 veces más. Esto indica que el caché de aplicación realiza su labor, pero a cambio de una enorme carga de trabajo. Como el servicio responde más rápido, los usuarios virtuales vuelven a gatillar las consultas con una frecuencia mayor. De aquí se puede deducir que para el caso de las pruebas 4 y 8, el servidor caché debe haber experimentado un alto nivel de carga de trabajo, ya que los valores de ***Throughput*** y porcentaje de CPU registrados para en estas pruebas son muy bajos.

Si hay algo que no varía entre las pruebas de los grupos 1 y 2 son la media en bytes de las respuestas, siendo para las pruebas que usan el ***Escenario 1*** alrededor de 4000 bytes y para el 2, 7000. Esto es totalmente normal, ya que el tamaño de las respuestas tiene que ver con los filtros de las consultas y no con las variables. En esta ocasión, en promedio el tamaño de las consultas que utilizan geohashes es un poco más del doble de sus homólogas que utilizan coordenadas.

Al observar los valores obtenidos en el grupo 3, resalta el ***Porcentaje de Error*** y el ***Apdex*** obtenido en las pruebas 9 y 10. Para estos casos, un nivel de carga tan grande sobrepasa las capacidades del servicio con creces. Al buscar los tipos de errores detectados por el cliente,

se registraron dos: 503 (Service Unavailable) y 504 (Gateway Timeout), errores que se gatillan cuando el servicio no se encuentra en condiciones de responder, lo que explica por qué la ***Media de Respuesta*** de ambas pruebas es menor que en la prueba 11, pues si se observan los datos del el lado del servidor, el valor del Apdex y la media de procesamiento de aquellas respuestas que logran ser procesadas en ambas pruebas es absolutamente inaceptable para un servicio. Distinto es el caso de las pruebas 11 y 12 que respondieron con una tasa de error muy baja, aunque con tales niveles de demanda, el rendimiento se vio afectado considerablemente en ambos casos.

En las pruebas del grupo 3 son de estrés se busca determinar con qué cantidad de usuarios concurrentes el servicio es capaz de responder con niveles aceptables. Al observar los gráficos de la **figura 6**, cuando la prueba 9 ha alcanzado cerca de los 140 usuarios concurrentes, los tiempos promedio de respuesta comienzan a superar los 5 segundos. Más adelante, alrededor de los 360 usuarios, el servicio ya no es capaz de responder exitosamente en casi todas las peticiones, en consecuencia, los tiempos de respuesta se desploman a causa de los errores. Distinto es el comportamiento de la prueba 12, ya que al observar la **figura 7**, el rendimiento se mantiene cercano al orden de los 4000 kb/s durante todo el ***Ramp Up***. Es interesante notar el comportamiento del tiempo de respuesta, el cual llega a un nivel máximo cercano al segundo (alrededor de los 45 usuarios) para luego disminuir drásticamente a niveles próximos a los 40 milisegundos. ¿A qué se debe esto?, pues al hecho de que en un comienzo las respuestas deben comenzar a almacenar en memoria caché tanto a nivel del ***Servidor de Aplicaciones*** como a nivel del ***Servidor Caché***. Una vez almacenada las respuestas de los 40 geohashes del listado CSV, ya no es necesario que el ***Servidor de Aplicaciones*** responda, dejando todo el trabajo al ***Servidor Caché***.

CONCLUSIONES

Primero que todo, gracias a la realización de este trabajo, hoy en día los clientes clientes de ***Modyo*** cuentan una alternativa de consulta de ubicaciones de promociones mucho más eficiente y escalable. Los resultados obtenidos en las pruebas de rendimiento demuestran que la alternativa de filtrado utilizando ***geohashes*** permite el uso de niveles de caché con alto grado de efectividad y sin errores, incluso con niveles de concurrencia muy por sobre lo normal.

Con esta solución, no sólo se mejora el desempeño del servicio, también disminuye considerablemente la cantidad de peticiones si se compara con una aplicación cliente básica que se limita a consultar en base a coordenadas cada vez que el usuario interactúa con el mapa. Implementar este enfoque de filtrado geoespacial requiere de un desarrollo del lado ***front-end*** del sistema más sofisticado, obligando a que la aplicación cliente realice precarga de datos vecinos que eviten gatillar nuevas consultas cuando el usuario altera un poco la ***sección visible***.

Es importante tener en cuenta que las pruebas realizadas sólo validan este trabajo en específico. Si se desea implementar esta solución en aplicaciones fuera de ***Modyo DX,*** es necesario replicar pruebas similares a las hechas en este trabajo, ya que existen muchos factores que pueden influir en el comportamiento de un sistema web, tales como: la arquitectura, las características de los servicios, la densidad de los datos y el nivel de demanda. Por ejemplo, si la cantidad de datos georreferenciados es demasiado pequeña,

quizás es mejor responder los datos sin ningún tipo de filtrado, o si la demanda del servicio es muy baja, incorporar niveles de caché puede no ser efectivo debido a la baja concurrencia. Hay que tener en cuenta que se necesita un alto nivel de demanda para poder reutilizar las respuestas almacenadas en un memoria caché independiente del nivel donde resida.

La solución en sí es muy ingeniosa pero tiene sus limitaciones, varias de las cuales pueden ser mitigadas en cierta medida por la inclusión de más lógica en el lado del cliente. En el caso hipotético de que se esté explorando una zona con una alta densidad de puntos, la aplicación cliente puede evitar respuestas pesadas incorporando limitantes al nivel de ***zoom*** del mapa para dicha zona. De esta manera, cuando el usuario explore dicha zona, no tendrá permitido alejarse más de cierto nivel.

Cuando se trata de servicios que deben soportar grandes cantidades de peticiones, es necesario tomar medidas en cuanto a infraestructura tanto en el ***Servidor de Aplicaciones*** como en el ***Servidor Caché***. Como se observó en las pruebas realizadas, en casos extremos el Servidor Caché logró responder sin errores, pero su rendimiento, aunque aceptable, se vio notoriamente afectado probablemente por el alto ***Throughput***. Por lo tanto, contar con buenos recursos para la infraestructura del ***Servidor Caché*** puede asegurar un buen desempeño en situaciones de estrés.

Para efectos de esta solución, el tamaño de las respuestas es todo un tema. Respuestas demasiado pesadas pueden provocar problemas tales como el consumo excesivo de datos o la sobrecarga de trabajo para el ***browser*** o aplicación nativa. Una decisión de diseño importante es la estructura de las respuestas. En este caso no se hicieron modificaciones al respecto, sin embargo la estructura de los datos de las respuestas es absolutamente mejorable. Actualmente, cada sucursal que forma parte de cierta respuesta, incluye tanto sus datos como la información en detalle de la promoción a la que pertenece. Si se considera que una promoción tiene muchas sucursales y luego se tiene que cierta respuesta contiene más de una sucursal asociada a una promoción, la respuesta tendrá replicada más de una vez la información de una promoción en particular. La estructura actual de los datos no es la mejor. Trabajos futuros pueden mejorar enormemente este aspecto.

Sólo se consideraron dos de los cuatro niveles de caché descritos. Ambos niveles son complementarios producto de sus diferentes tipos de invalidación. Mientras que el ***Servidor Caché*** invalida sus respuestas después de un determinado tiempo, ***Action Caché*** lo hace cuando los datos cambian. Pero, los potenciales beneficios que puede traer consigo la incorporación o buen uso de otros niveles de caché no ha sido llevado a la práctica. Sería interesante poder realizar una investigación e implementación al respecto.

Hoy en día, no existe otro sistema sistema de geocodificación como los ***geohashes*** y su invento es relativamente nuevo. Si las necesidades de los sistemas que utilizan información georreferenciada lo requieren, es probable que en un futuro alguien desarrolle una alternativa con características diferentes. Para efectos de la solución implementada, utilizar este enfoque ha sido sumamente útil.

REFERENCIAS BIBLIOGRÁFICAS

[Blazemeter16] Blazemeter Blog, **Performance Testing vs. Load Testing vs. Stress Testing**, 2016.
https://www.blazemeter.com/blog/performance-testing-vs-load-testing-vs-stress-testing

[CDN17] CDN Reviews, **What is a Content Delivery Network?**, 2017.
http://www.cdnreviews.com/what-is-cdn

[Elastic] Elasticsearch: The Definitive Guide [2.x]: **Geolocation » Geohashes**
https://www.elastic.co/guide/en/elasticsearch/guide/current/geohashes.html

[HTTP14] Hypertext Transfer Protocol, **Caching**, 2014.
https://tools.ietf.org/html/rfc7234

[JohnV12] John V., **Forward Proxy vs Reverse Proxy**, 2012.
http://www.jscape.com/blog/bid/87783/Forward-Proxy-vs-Reverse-Proxy

[Niemeyer08] Labix Blog, Gustavo Niemeyer, **geohash.org is public!**, 2008.
https://blog.labix.org/2008/02/26/geohashorg-is-public

[MySQL12] Schwartz, Zaitsev, Tkachenko, **High Performance MySQL: Optimization, Backups, and Replication**. Pág 315, 2012.

[prGeohash08] Yuichiro Masui, **pr_geohash**, 2008.
https://github.com/masuidrive/pr_geohash

[Puglisi16] Silvia Puglisi, **RESTful Rails Development: Building Open Applications and Services**. Pág. 20, 2017.

[Rails4] Obie Fernandez, Kevin Faustino, **The Rails 4 Way**, Pág. 149, 2014.

[RGuides] Rails Guides, **Caching with Rails: An overview**.
http://guides.rubyonrails.org/v4.1/caching_with_rails.html

[Veness16] Chris Veness, **latlon_geohash**, 2016.
https://github.com/chrisveness/latlon-geohash

[Venka14] Venkatesh CM, **Caching To Scale Web Applications**, 2014.
http://venkateshcm.com/2014/05/Caching-To-Scale-Web-Applications