Integración de un buscador con indexación en tiempo real en Bodas.net

Àlex Martínez Rubio

Resumen—En la web y apps de Bodas.net se utiliza el motor de búsqueda Sphinx para calcular resultados de búsqueda, listados de información y contenido relacionado. En Bodas.net se quiere explorar la posibilidad de hacer que la actualización de los índices sea en tiempo real. Sphinx tiene el problema de que la indexación en tiempo real es limitada y que la comunidad de desarrolladores es pequeña. En este proyecto, se ha realizado una comparativa entre los dos buscadores y se ha desarrollado un prototipo de integración de ElasticSearch en Bodas.net. Se ha montado un cluster de ElasticSearch virtualizado, se ha configurado y se han cargado datos de la web. Se ha adaptado el código para la actualización de los índices de ElasticSearch en tiempo real y para las búsquedas en el cluster, por lo que se ha buscado una biblioteca para ese propósito. Posteriormente se han ejecutado unas pruebas de estrés para medir tiempos de respuesta y degradación del servicio con el incremento de carga, comparándolo con la solución actual basada en Sphinx. Se han analizado los resultados y se ha demostrado que, tanto a nivel cualitativo como cuantitativo, ElasticSearch es mejor que Sphinx y se ha recomendado su implantación en Bodas.net.

Paraules clau—ElasticSearch, SphinxSearch, Motor de búsqueda, Docker, Cluster, Indexación, JMeter, Prueba de rendimiento

Abstract— On the web and apps of Bodas.net, the Sphinx search engine is used to calculate search results, lists of information and related content. The company Bodas.net wants to explore the possibility of updating the indexes in real time. Sphinx has the problem that real-time indexing has limitations and that the developer community is small. This project has done a qualitative comparison between the two search engines and has developed a prototype of ElasticSearch integration to Bodas.net. A virtualized ElasticSearch cluster has been assembled and configured, and data from the web has been loaded. The PHP code has been adapted to integrate ElasticSearch, testing the functionalities of real-time indexing and searching in the cluster, so the existence of a library for that purpose has been investigated. Subsequently, stress tests have been designed and executed to measure response times and degradation of the service with the load increase, comparing it with the current solution based on Sphinx.The results have been analyzed and it has been proven that, both quantitatively and quantitatively, ElasticSearch is better than Sphinx and an implantation in Bodas.net has been advised.

Index Terms— ElasticSearch, SphinxSearch, Search engine, Docker, Cluster, Indexing, PHP, JMeter, Benchmark

1. Introducción

ESTE Trabajo de Final de Grado ha sido propuesto por el CTO de Bodas.net, y como su título indica, consiste en implementar un prototipo de un nuevo buscador para la web, basado en una solución real-time.

Uno de los problemas que tienen con el buscador actual, SphinxSearch [1], es que no indexa los datos en real-time, y al realizar búsquedas en la web no se pueden visualizar algunos contenidos que están ya insertados en la base de datos, pero que aún no han sido indexados para su búsqueda.

El nuevo motor de búsqueda, propuesto también por la empresa, es ElasticSearch [2], uno de los más modernos y utilizados en la actualidad. El objetivo final es ver si es viable hacer el cambio del buscador actual, estudiando cómo impacta en el código y en el hardware.

Como se ha dicho, el proyecto ha sido realizado en colaboración con la empresa Bodas.net. A grandes rasgos, es una plataforma web cuyo modelo de negocio consiste en poner en contacto a novios con proveedores de servicios nupciales, como por ejemplo banquetes, vestidos o decoraciones.

Empezó como start-up en 2008 y no ha parado de crecer en los últimos años, convirtiéndose en uno de los líderes del sector. Tienen versiones de la web en más de 10 países, y en 2015 fueron adquiridos por la empresa americana WeddingWire, ampliando así aún más su mercado.

- *E-mail de contacte: alex.martinezr@e-campus.uab.cat*

- *Menció realitzada: Enginyeria de Tecnologies de la Informació.*
- *Treball tutoritzat per: Andrew Koster (dEIC)*
- *Curs 2017/18*

La metodología a seguir para cumplir con los objetivos de este TFG ha sido:

1- Estudio de literatura y comparativa cualitativa de los dos motores de búsqueda.
2- Implementación de ElasticSearch en un entorno virtualizado. Consta de las siguientes partes:
   a) Creación del cluster
   b) Indexación de datos de test
   c) Integración en PHP
3- Comparativa cuantitativa de ElasticSearch y Sphinx: benchmark.
4- Recomendación del mejor buscador para la empresa

2. ESTADO DEL ARTE:

A. Posibles alternativas

Ante la posibilidad de implementar otro buscador en vez de ElasticSearch, la respuesta de la empresa ha sido clara. ElasticSearch es de los buscadores más utilizados y que a nivel técnico puede ofrecer más lo que desean. Se ha puesto sobre la mesa una alternativa, Apache Solr [3], ya que tiene bastantes similitudes con ElasticSearch.
Solr es un motor de búsqueda publicado en 2004 basado en la biblioteca de búsqueda Lucene [4], al igual que ElasticSearch. Ambos, por tanto, están programados en Java, y los dos tienen métodos de acceso con una API RESTful y soporte de documentos JSON.
Sin embargo, al ser más antiguo y menos utilizado en la actualidad, se ha optado por ElasticSearch. Otro motivo para la elección es que Solr soporta real-time indexing, pero al igual que Sphinx debe ser especificado con una serie de atributos, mientras que Elastic está optimizado para indexar siempre de esta manera.

B. Trabajos relacionados

Al ser una solución a problemas específicos de ingeniería, las comparativas de motores de búsqueda suelen depender de muchos detalles técnicos de cada entorno de desarrollo de software. Las comparativas más sistemáticas [5] son o bien cualitativas, o bien más generales. Cada proyecto de software, por tanto, debe evaluar por sí mismo qué buscador se adapta mejor a sus necesidades.

Sin embargo, para poder hacer el tipo de comparativa a la que se enfrenta este proyecto, sí que se encuentra una amplia escala de herramientas. Estas analizan sistemas a nivel de uso de CPU, RAM, velocidad de respuesta, etc…, y la que se ha usado en este proyecto, JMeter, se especializa en pruebas de estrés y en medir la velocidad de respuesta. [6].

En cuanto a pruebas de rendimiento de ambos buscadores, este no es el primer proyecto que presenta un ejemplo, claro está. A pesar de ser moderno, ElasticSearch ya ha sido objeto de estudio de otras publicaciones [7].
Respecto a Sphinx, apenas se han encontrado referencias científicas, y las pocas que existen son de poca relevancia.

3. COMPARATIVA CUALITATIVA

Se han comparado cualitativamente los dos buscadores antes de pasar a la implementación de ElasticSearch en el entorno virtualizado. Para ello, se ha leído la documentación necesaria, y se han encontrado varios puntos en los que ElasticSearch y Sphinx difieren:

Comunidad: SphinxSearch fue creado en 2001, y la comunidad que hay detrás es muy reducida, habiendo solo una persona conocida que se encargue de su mantenimiento. Así pues, es una solución bastante desactualizada.
En cambio, ElasticSearch fue publicado en 2010, está distribuido bajo la Apache Software License, y además es uno de los motores de búsqueda más usados en la actualidad. Esto implica que hay una comunidad muy potente detrás desarrollando paquetes para ElasticSearch, añadiendo nueva funcionalidad y arreglando problemas. Esto puede ayudar a la empresa a debuggar el código con más facilidad, ahorrando así tiempo, y también, al ser uno de los más usados, es más fácil contratar a técnicos que sepan trabajar con él.

Real-time: Uno de los motivos más importantes para implementar ElasticSearch, es que la empresa desea trabajar con indexación real-time.
Actualmente, en las partes de la web que listan elementos se usa directamente una consulta a la base de datos, que contiene la información actualizada al momento. Sin embargo, para el caso de búsqueda, existe el problema de que puede no mostrar la información de ese preciso momento.
Esto se debe a que la indexación de Sphinx actualiza todos los documentos de un índice en una sola vez. Concretamente, en Bodas.net se hace mediante un proceso que se ejecuta una vez al día, o dos, dependiendo del índice. El proceso vuelve a indexar los documentos que ya existían, y los nuevos que se han creado desde la última ejecución son añadidos.

Estas actualizaciones implican tener que informar a los daemons, que deben interrumpir las conexiones y esto se traduce en degradación del servicio. También implica un mayor consumo de recursos, y un lapso de tiempo en el que no es posible buscar los elementos recientes, que es el principal problema.
En cambio, con la solución real-time de ElasticSearch, hay un feed de datos continuo que se van indexando a medida que se crean, y en un tiempo mínimo ya se pueden buscar.
Hay que decir que Sphinx, a pesar de seguir el modelo clásico de buscador, en el que se indexa todo de golpe, también soporta índices real-time. Sin embargo, se tiene que hacer a mano, mientras que Elastic lo hace automáticamente y ya está pensado para ello. La idea de la empresa es separarse lo máximo de cómo funciona el buscador internamente, y por eso en vez de usarlos, prefiere cambiar a ElasticSearch.

Escalabilidad: Según la documentación, Elastic permite escalar muy bien horizontalmente, que significa que se pueden añadir nodos para incrementar la performance, en vez de dotar de más recursos a un nodo. De esta manera, se particionan automáticamente los índices en "trozos", repartidos entre distintos nodos, y después se juntan en el momento de la búsqueda. En SphinxSearch, el particionamiento se tiene que hacer de forma manual, así que se suele optar por dar más recursos a los nodos, lo que no permite un crecimiento del cluster tan estructurado.

Monitorización: El control del cluster está más automatizado, mejorando el control de fallos y el balanceo de carga, mientras que en Sphinx, de nuevo, se hace de manera más manual.

Schema-free: ElasticSearch es de los pocos buscadores que es flexible en cuanto al tipo de datos, lo que permite tener documentos heterogéneos en un mismo índice. De momento la empresa no lo utiliza, pero hay que investigar si en un futuro les puede interesar.

Base de programación: Sphinx, al estar desarrollado en C++, seguramente comporte un consumo de memoria y recursos más austero, y que a nivel de hardware hagan falta servidores más pequeños para ponerlo en funcionamiento. Por el hecho de estar hecho en Java y tener bastante más funcionalidad, es posible que ElasticSearch vaya más lento.
Teniendo en cuenta las características de Sphinx y ElasticSearch, parece que hay motivos tanto técnicos como socioeconómicos para el cambio de SphinxSearch a ElasticSearch.

Se cree que a nivel de funcionalidad Elastic conseguirá resolver los problemas actuales de la empresa, pero como se ha dicho, puede que a nivel de performance sea peor, por la tecnología en la que está basado.

A continuación, se ha investigado si estas ventajas teóricas de ElasticSearch se aplican también en la práctica. Para eso se ha implementado ElasticSearch en un entorno de prototipo para la empresa. Después de esa fase, se ha diseñado una comparativa cuantitiva.

4. IMPLEMENTACIÓN DE LA SOLUCIÓN

Hay que implementar elasticsearch para comprobar en un entorno prototipo su funcionamiento y que una integración en el código de la empresa es posible.
Como se ha dicho en la introducción, las fases han sido:
- Creación cluster
- Indexación datos test
- Integración PHP

A. Creación del cluster con Docker

Al principio del proyecto, se han mantenido conversaciones con el CTO de la empresa y con el responsable de sistemas de la plataforma. Se exploraron varias opciones y se les propuso la utilización de Docker.

Docker [8] es la empresa líder en gestión de contenedores, un tipo de máquinas virtuales menos pesadas, ya que estas contienen todo un sistema operativo y los contenedores solo están formados por la aplicación y los requisitos para ejecutarse a sí misma.

La propuesta les pareció correcta, por la portabilidad y la rapidez de desarrollo que ofrece. Además, permite tener un control de versiones mejor que una VM, ya que simplemente hay que ejecutar la imagen de Docker actualizada.
También expresaron que Docker se usaría para un entorno de prueba, y queda pendiente de ver, en el caso de aplicar Elastic en producción, si se usaría también Docker o máquinas físicas.
Así pues, se llevó a cabo su instalación y la lectura de la documentación correspondiente. Se acordó con la empresa que para simular un entorno de producción lo mejor es tener un número impar de nodos y como mínimo 3. De

esta manera siempre se puede formar una mayoría para, por ejemplo, votar sobre un dato distribuido entre varios nodos, para saber cuál es el correcto.

Así pues, el objetivo pasó a ser montar un cluster de 3 nodos de ElasticSearch con Docker, asegurarse primero que funcionaran bien y se vieran entre ellos.

Se vio que para definir y ejecutar una aplicación Docker multi-contenedor, la mejor opción era la herramienta Compose [9]. En Docker-Compose, se utiliza un archivo YAML para configurar los servicios de la aplicación, que equivalen a cada nodo del cluster.
Después, con un simple comando se crean y arrancan todos los servicios descritos en la configuración.

Básicamente, se define el entorno de la aplicación en un Dockerfile para que se pueda reproducir en cualquier sitio. Después, se definen los servicios que la componen en el archivo Docker-compose.yml, para que puedan ser ejecutados en conjunto y de forma aislada.
Se ha seguido este desarrollo, y el grueso del trabajo ha sido crear al archivo Docker-compose.yml. En este caso el Dockerfile ha sido mínimo, y únicamente ha contenido la imagen de ElasticSearch para crear los nodos.
Para la configuración [10] de los nodos, se ha editado el archivo elasticsearch.yml que provee ElasticSearch.
Se puede proceder a arrancar el cluster y ver que tenemos 3 nodos levantados, con un estado de cluster Green.

B. Indexación de datos de prueba con Kibana

Una vez configurado el cluster de ElasticSearch, se ha experimentado con el lenguaje que usa para realizar las operaciones con datos. Existe una plataforma de análisis y visualización llamada Kibana [11], y nos proporciona una interfaz cómoda para indexar y buscar contenido.

El siguiente paso que se ha seguido es la indexación de contenido de prueba. Para el propósito de esta fase se ha diseñado un pequeño ejemplo, con el que poder probar la creación, búsqueda, actualización y borrado de datos.
Se ha indexado de dos formas: 1 documento a la vez y con la función bulk, que indexa varios documentos a la vez por batches.

Al tener datos en el cluster, con Kibana se puede ver que se han creado 10 shards. Los shards son subdivisiones de un índice, que en sí mismos son índices totalmente funcionales y que pueden alojarse en cualquier nodo del cluster. Elastic hace sharding para evitar que un índice demasiado grande consuma todos los recursos hardware de un nodo, y por tanto permite escalar horizontalmente el volumen de contenido y también distribuir en varios nodos las operaciones de búsqueda, para aumentar el rendimiento. La mecánica de cómo se distribuye un shard y también cómo Elasticsearch reconstruye los documentos al buscarlos, es completamente transparente para el usuario.

También se ha probado la búsqueda, la actualización de datos y borrado de datos, que funcionan correctamente para este set de datos de test.

Hay búsquedas mucho más avanzadas que se pueden hacer en ElasticSearch, pero en el alcance de este proyecto no se han investigado en detalle.

C. Integración parcial en PHP

En esta parte se ha añadido el buscador al código PHP de Bodas.net.

En primer lugar, se ha estudiado la estructura del código para una mayor comprensión del problema.
A continuación, se ha comprobado la existencia de un conector ElasticSearch-PHP [12], y hay uno en la documentación oficial de Elastic. Así que simplemente se ha incluido el paquete en el código y se ha empezado a usar, gracias al entorno de desarrollo de Bodas.net, es decir, el servidor apache que se proporciona para ejecutar el PHP en local, y funciones ya implementadas que pueden ser de utilidad.
Se ha creado una clase con métodos para realizar las operaciones principales de ElasticSearch, y se ha comprobado su correcto funcionamiento con ejemplos sencillos de prueba, poniendo énfasis en la inserción y la búsqueda, ya que son las más importantes.

Como primera fase, es necesaria la indexación de más contenido en el cluster de Elastic, para poder tratar con un volumen de datos más parecido al de la web.
El objetivo ha sido replicar la información de la sección de debates que está indexada actualmente en Sphinx, que corresponde al resultado de una query SQL que se encuentra en el archivo sphinx.conf. Así pues, se ha hecho la inserción de debates reales en ElasticSearch.

El proceso, por tanto, ha sido el siguiente:
Aprovechando funciones propias del código de la web que permiten hacer consultas a su base de datos, se ha ejecutado la query, debidamente limitada para que no tarde demasiado en indexar.

Después, se ha usado la función bulk, mencionada anteriormente, que sirve para indexar varios documentos de golpe. Pero, llegados a esta parte, se ha visto que no es posible realizar todo el bulk de golpe, ya que es rechazado.
Esto ocurre porque la petición de bulk es recibida por un nodo del cluster, que es el encargado de coordinar la indexación. ElasticSearch pone los todos los documentos en cola, y si el tamaño del batch es demasiado grande la memoria del nodo se ve desbordada.

Para solucionarlo, ha habido que reducir el tamaño del batch del bulk. Se ha dividido la query en dos partes, limitando los debates por su id, y se ha indexado cada conjunto de datos en un bulk distinto. De esta manera, se ha conseguido tener unos 9000 documentos en ElasticSearch, listos para su búsqueda.

Se cree que este problema no afectará mucho en el futuro, ya que solo se manifiesta cuando el volumen de datos que hay que insertar es muy elevado. Por tanto, solo habría que aplicar esta solución las primeras veces que se indexa todo el contenido, en la fase de migración de Sphinx a Elastic. Al tratarse de un buscador real-time, el contenido se indexa entero solo una vez, y después cada vez que un nuevo elemento es creado se inserta en el buscador, y no supondría un volumen de datos suficiente para colapsar la red.

Sin embargo, antes de hacer el bulk, hay otro aspecto de la inserción que hay que tener en cuenta, que es el mapping del índice. La idea es intentar replicar el de Sphinx con Elastic. El mapping ha permitido definir 3 cosas:
Por una parte, el tipo de dato que se indexa. Cada campo del índice tiene que tener el adecuado, para poder detectar bien las búsquedas de los usuarios o para poder filtrar por el campo. Por ejemplo, si se quiere filtrar por el campo ID del debate, este tiene que ser definido como entero en el mapping, ya que, si se hace directamente el bulk del resultado de la query SQL, es detectado como texto.

Hay que definir también si el campo debe ser indexado, es decir, si se puede buscar un debate a través de ese campo. Según la configuración actual de Sphinx, solo se puede buscar por unos pocos campos, como por ejemplo el título o el mensaje del debate.

Y, por último, especificar si se guardan los valores de cada campo. A diferencia de la indexación del dato, que nos permite buscar por él (buscar un debate por título, por ejemplo, recupera el debate), guardar el dato ofrece la posibilidad de recuperar su valor original. Esto puede servir para consultar un dato en el buscador en vez de en la base de datos, al tratarse de una operación más rápida.
Por defecto, en ElasticSearch todos los datos originales son guardados en el campo _source del documento, así que si no se desea guardar un campo del debate solo hay que excluirlo de _source en el mapping.

Se puede decir que el cliente PHP de ElasticSearch ha resultado muy fácil de usar, la conexión con Elastic es transparente al usuario, y viene provisto de las operaciones básicas del paradigma CRUD (Create, Read, Update, Delete), y solo hace falta ajustarlas a las necesidades del proyecto. Además, su sintaxis se basa en arrays asociativos que después son serializados automáticamente y transformados en JSON, que es con lo que trabaja Elastic, así que el manejo de datos es sencillo.

La integración en el código de la web, claro está, no ha sido completa, sino que se ha hecho a nivel de la indexación de contenido específico y para las búsquedas realizadas en el apartado de benchmark. De todos modos, se han ido poniendo los archivos en las carpetas correspondientes y usando las funciones propias de la web, pensando en cuando se tenga que implementar en producción.

5. BENCHMARK: PRUEBA DE RENDIMIENTO

A. Descripción

El test consiste es hacer una serie de búsquedas de texto completo sobre debates reales de la web, la misma información que se ha indexado en el apartado anterior.
Dichas búsquedas serán por título de debate, que al fin y al cabo es para lo que más se usa un buscador de estas características. Estas búsquedas son nombres de poblaciones, y para simular un caso más cercano a la realidad se ha hecho que las poblaciones vayan variando. De esta manera el set de datos es más rico.

Se ha usado la solución de integración de Elastic en PHP, y la solución de Sphinx ya implementada en el código, para simular un entorno de producción.
Además, las búsquedas han ido dirigidas a una URL de Bodas.net, para que se parezca más a una petición real a la web, y no peticiones directamente al puerto donde escuchan los buscadores.
Se han estudiado dos herramientas para hacer las pruebas de stress. Una de ellas es Ab [13], una herramienta para medir performance de Apache fácil de usar y mostrar los

datos en gráficas. Y JMeter [14], también de Apache.
Se ha visto que Jmeter tiene muchas más funcionalidades, como por ejemplo ejecutar peticiones simultáneas a varios sitios web, aumentar el número de threads, una interfaz muy útil, y entre ellas la capacidad de realizar peticiones variables, que para nuestro test de las poblaciones va muy bien. Así pues, se ha escogido JMeter.

Básicamente, el funcionamiento del test es el siguiente:
Jmeter funciona como un cliente que hace una petición a una url de la web, en este caso preprod.bodas.net, la web de desarrollo de Bodas.net. En la URL, va en forma de query string el nombre de la población en concreto.
El servidor apache capta la petición y ejecuta el código de la web, donde está definida la ruta de la URL. Capta el parámetro de la población y monta la búsqueda de ElasticSearch o de Sphinx.

La búsqueda se realiza, y Jmeter recibe los datos de la respuesta, que son guardados en un archivo .csv. Los datos más importantes de este archivo son: código y mensaje de respuesta (por ejemplo 200 OK), tiempo de respuesta, número de threads activos en ese momento, número de bytes enviados y recibidos, tipo de datos y mensaje de error.

B. Implementación del entorno de la comparativa

Preparación de SphinxSearch

El caso de Sphinx, para los propósitos de la comparativa se ha implementado con un contenedor de Docker. De esta manera, se facilita el desarrollo y hay consistencia con el de ElasticSearch.
En primer lugar, se necesita un archivo de configuración de Sphinx, llamado sphinx.conf. Se ha extraído una parte del archivo de Bodas.net, la correspondiente a los índices que nos interesan, los de debates. Algo que nos puede mostrar este archivo, son los campos de debates que nos interesa indexar, ya que en ElasticSearch tienen que ser los mismos.

Se ha usado un contenedor de Docker para Sphinx. Este, al iniciarse, indexa el contenido usando el archivo de configuración, donde se encuentra la query a base de datos. En este caso no ha existido el problema de tener que partir la query en dos. Puede haber dos motivos, y es que la memoria dedicada a la indexación se más elevada que en ElasticSearch, o los datos no se envían todos de golpe, sino poco a poco, para que no se colapse la RAM.
A continuación, se queda esperando peticiones de búsqueda en un puerto. Nos conectamos él a través de MySQL y ya podemos empezar a realizar peticiones de prueba. Las queries se hacen con una variante de SQL llamada SphinxQL.

Preparación de ElasticSearch:

Se ha querido hacer el test de dos formas, con 1 solo nodo de ElasticSearch y con 3 nodos. Para el primer caso, se ha transformado el cluster en un nodo, que ha consistido en cambiar unos parámetros de configuración de Elastic y en tratar un poco con Docker de nuevo.

Después se ha indexado el contenido a través de PHP, con el proceso de bulk explicado en el apartado anterior.

Desarrollo:

El primer paso para preparar el test ha sido crear las rutas a las que haremos las peticiones. Se han creado dos: /elasticTest y /sphinxTest. Una parte importante de la web tiene Symfony [15] integrado, así que se ha tenido que trabajar con él. Se trata de un framework diseñado para optimizar el desarrollo de las aplicaciones web basado en el patrón Modelo Vista Controlador. Básicamente, cada nueva ruta va asociada a una función de Action, que forma parte de un controlador, y esta función es la encargada de renderizar la vista HTML, que se encuentra en un archivo separado.
Lo segundo que hay que hacer antes del test es comprobar que las queries devuelven los mismos resultados tanto en Elastic como en Sphinx, para que sean comparables. Ha habido que diseñarlas para tal propósito, para cumplir requerimientos como, por ejemplo:

- Que la query devuelva todos los hits, así que hay que quitar las limitaciones al resultado, si las hay.
- Que los campos devueltos de cada debate sean los mismos en los dos. Tomando como referencia los campos que devuelve Sphinx en las búsquedas, se han replicado los mismos en Elastic.
- Que si hay un espacio en la población no la detecte como dos palabras distintas.

Una vez igualados los resultados, hay que hacer una última comprobación, y es que los recursos de los containers de Docker sean los mismos.

C. Ejecución del test

A continuación, se procedió a empezar el test con Jmeter.
Como hemos especificado en este apartado, hay que llamar a las URLs definidas desde el programa, /elasticTest y /sphinxTest.
Se han creado dos HTTP samplers o "muestreadores", y en ellos se especifica la URL que sigue el test, y también se indica el parámetro población que se debe enviar como query string. Este parámetro es leído de un archivo .csv, que contiene todos los nombres de las poblaciones, alrededor de unas 50, y cada petición va iterando sobre él y haciendo la búsqueda sobre una población distinta.
Como se ha dicho, el test consiste en hacer estas búsquedas, y además añade el elemento de ir aumentando el número de threads, es decir, el número de usuarios concurrentes. Así se ve como responde cada uno al aumento progresivo de carga. Y cuando el test haya terminado, generamos con Excel los gráficos para comparar los resultados, mostrados en las figuras 1-5.
Ahora se explican los casos de test que se han probado.
Las variables que han intervenido en las pruebas son las siguientes, casi todas configurables desde JMeter.

- Número máximo de threads concurrentes: El máximo de threads que estarán activos durante el test.
- Ramp-up: Es el tiempo que tardan en estar activos todos los threads, que van aumentando de número a una velocidad constante, hasta que se alcanza el número máximo de threads.
- Duración del test: Marca el final de la ejecución de JMeter.
- Heap size de la máquina virtual de Java: Es una limitación importante de memoria de ElasticSearch. Se han probado varias ejecuciones modificando este parámetro, para ver si el rendimiento del buscador se veía afectado.

Se han probado distintas combinaciones de estas variables, y se han ejecutado 3 escenarios distintos. Los valores se han escogido según los siguientes criterios:
- Primero se ha decidido la duración del test. Los valores han sido 250, 500 y 900, en segundos.
- Después el número máximo de threads. Para cada escenario, se ha escogido 25, 40 y 70, aumentando cada vez por un factor de 1,5 aproximadamente.
- A partir de estos dos valores, se ha intentado elegir un ramp-up que le permitiera al buscador adaptarse al aumento de carga, de forma progresiva y no brusca. En cada caso se añade un nuevo thread cada 1,5 segundos aproximadamente.

A su vez, cada escenario se ha ejecutado para varios valores de heap size: 256mb, 512mb, 1gb, 2gb.
También se ha tenido en cuenta que la manera de ejecutar los tests tuviera sentido.
Por ejemplo, al ser en una misma máquina, ejecutar el test con los dos buscadores a la vez puede que no sea muy realista, ya que normalmente solo se usa un buscador en un mismo hardware. Además, se depende más del scheduler del ordenador, que puede no ser justo. Por esta última razón, tampoco se han tenido los dos contenidores activos en Docker al mismo tiempo durante la ejecución.

6. RESULTADOS

A. Uso de los datos obtenidos

Para comparar los resultados, se han extraído los datos de los archivos .csv que genera JMeter al finalizar cada test, cuyos campos son descritos en el apartado anterior.
Estos han sido introducidos en hojas de cálculo Excel para su manipulación y representación en gráficos.
El tipo de gráfico que se ha creado es: Número de threads concurrentes VS tiempo de respuesta de los threads, porque se cree que es un buen indicador del rendimiento de los buscadores. De esta manera, se puede ver cómo varían los tiempos de respuesta de las búsquedas, a medida que aumenta la carga de peticiones concurrentes. Con la comparativa de las gráficas, se puede concluir cuál de ellos es más rápido, es decir, cuál tiene un tiempo de respuesta más bajo.
Como preparación de los datos, se ha calculado el tiempo promedio de respuesta de todos los threads activos durante un período de tiempo. En este tiempo, el número de threads activos se mantiene igual, así que se puede asignar un tiempo de respuesta promedio a una etiqueta "Número de threads concurrentes" (por ejemplo, 10 threads). Esta etiqueta conformará el eje X de los gráficos, y el tiempo de respuesta promedio (en ms) el eje Y.

B. Gráficas

Previamente al proceso de comparativa entre ElasticSearch y Sphinx, se han seguido unos primeros pasos para cada escenario de test.
Se ha escogido qué heap size de ElasticSearch va mejor para comparar con Sphinx, y se ha comparado también el rendimiento de 1 nodo y de 3 nodos.
En todos los casos el heap que size que ha ofrecido mejor tiempo de respuesta ha sido 512 MB para 1 nodo, y 256 MB para el cluster.

Y en todos los casos también, ElasticSearch con 1 nodo ha tenido mejor rendimiento que cluster de 3.

Se pueden ver gráficos de estas comparativas en las figuras 1 y 2. Son las ejecuciones en el escenario 1, cuyos detalles son descritos a continuación.

Escenario 1
Núm. Máx. Threads: 25
Ramp-up: 150s
Duración: 250s

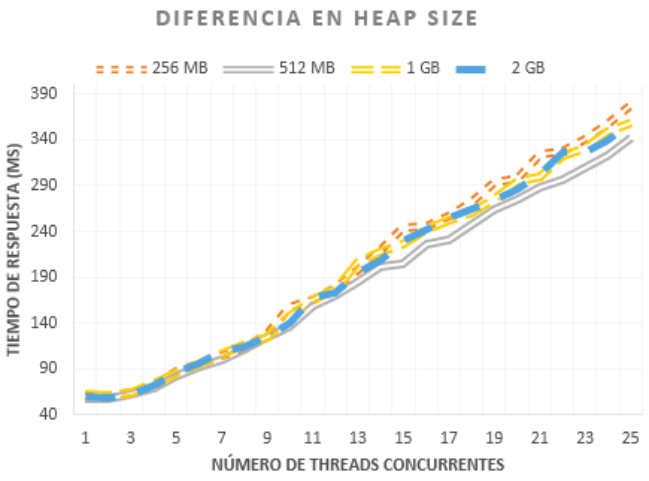

Fig.1 Tiempo de respuesta de ElasticSearch (1 nodo) con distintos valores de heap size, VS núm. threads concurrentes. Escenario 1

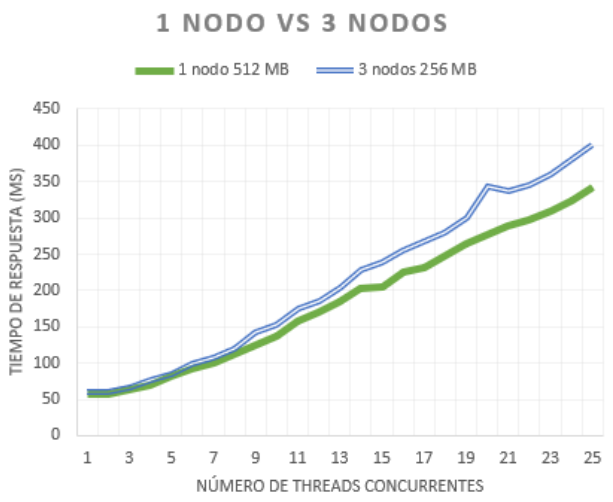

Fig.2 Tiempo de respuesta de ElasticSearch con cantidad distinta de nodos, VS núm. threads concurrentes. Escenario 1.

Como consecuencia de estos resultados, se ha escogido ElasticSearch con 1 nodo y 512 MB de heap size para comparar con SphinxSearch. En las figuras 3, 4 y 5 se pueden observar los resultados de la comparativa en los distintos escenarios.

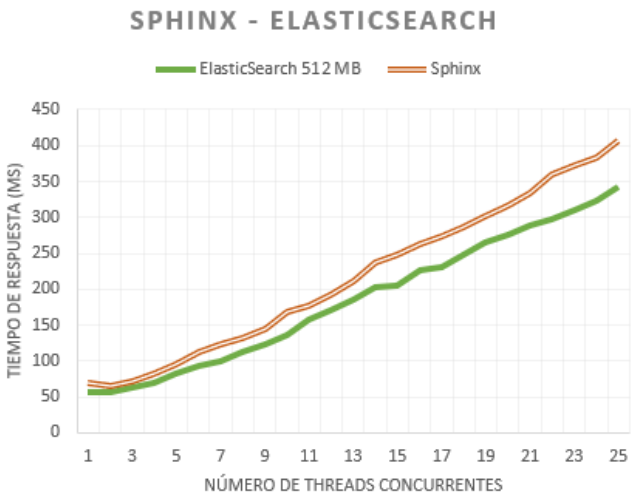

Fig.3 Tiempo de respuesta de ElasticSearch y SphinxSearch VS núm. threads concurrentes. Escenario 1.

Escenario 2
Núm. Máx. Threads: 40
Ramp-up: 300s
Duración: 500s

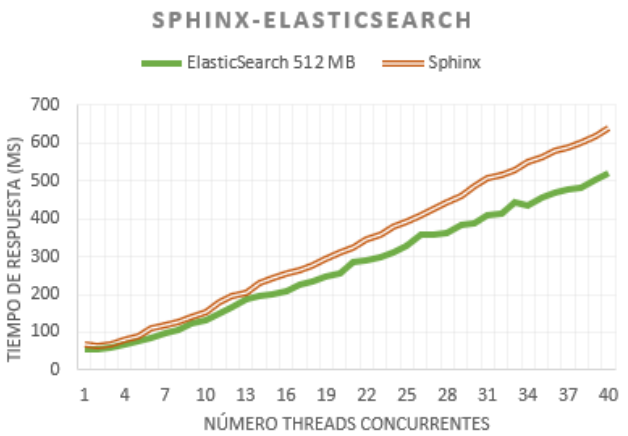

Fig.4 Tiempo de respuesta de ElasticSearch y SphinxSearch VS núm. threads concurrentes. Escenario 2.

Escenario 3
Núm. Máx. Threads: 70
Ramp-up: 700s
Duración: 900s

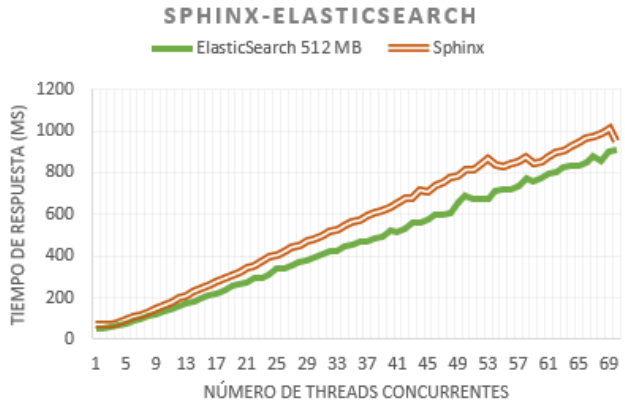

Fig.5 Tiempo de respuesta de ElasticSearch y SphinxSearch VS núm. threads concurrentes. Escenario 3.

C. Análisis de los resultados

Si estudiamos las gráficas, observamos que en los 3 escenarios Elastic es un poco más rápido que Sphinx.
Por tanto, se desmiente la hipótesis inicial que partía del estudio cualitativo, que planteaba que Sphinx sería más rápido que ElasticSearch.
Por otra parte, no parece haber mucha diferencia de rendimiento si se modifican los parámetros del heap size de la JVM, tal y como se ve en la figura 1. Sin embargo, aunque por un margen pequeño, 512 MB ha sido mejor y se ha escogido para la comparativa con Sphinx.

Se ha probado la ejecución con el cluster de Elastic, y ha resultado en un tiempo de respuesta peor que con un nodo.
Como posible explicación se puede deducir que el sharding de Elastic, puede haber afectado al test de rendimiento. Sharding consiste en dividir un índice en trozos que pueden ser distribuidos entre los distintos nodos del cluster. Por tanto, para recuperar todos los trozos de un índice, los nodos deben comunicarse entre ellos, y es posible que este tiempo de comunicación y montaje del índice penalice el tiempo de la búsqueda.
Otra posible explicación, es que tener los 3 contenedores de Docker en una misma máquina puede provocar un overhead que empeore el tiempo de respuesta. Un cluster está pensado para ejecutar cada nodo en una instancia separada, pero aún así un core debería poder aprovechar este tipo de distribución.

7. CONCLUSIONES

A. Acerca del trabajo realizado

Se ha comprobado que, a nivel de integración en el código, es posible añadir ElasticSearch como nuevo buscador de la web. Usando el conector es una labor de relativa facilidad, e incluso se le puede dar uso al código que se ha hecho para este TFG.
Después de las pruebas de stress realizadas, se puede observar que Elastic tiene un tiempo de respuesta menor que Sphinx. No es una mejora sustancial, pero sí apreciable, así que se cree que sería recomendable a nivel de rendimiento que fuera el nuevo buscador.
Pensaba que iba a ir peor pero no, la hipótesis no se ha cumplido.

A nivel cualitativo, también se considera recomendable cambiar de Sphinx a ElasticSearch. Se ha visto que los puntos expresados en la investigación inicial se aplican en la práctica:
Hay mucho más soporte en Elastic cuando se trata de resolver dudas o problemas, y se ha comprobado que era mucho más fácil encontrar soluciones que en la comunidad de Sphinx.
Uno de los principales atractivos de ElasticSearch para la empresa es su capacidad de soportar la indexación real-time, y además se ha visto que su implementación en PHP no es complicada.
En cuanto a la escalabilidad de la solución, gracias al desarrollo con Docker del cluster de Elastic, se ha podido comprobar la facilidad con que se pueden añadir nuevos nodos para aumentar su capacidad y así escalar horizontalmente. Además, el sharding (particionamiento de índices) entre los distintos nodos se hace automáticamente y es totalmente transparente al usuario.
Las posibilidades de consulta parecen menos limitadas que con Sphinx. Elasticsearch Query DSL (Domain Specific Language) es muy expresivo y ofrece mucha flexibilidad para crear consultas complejas. La capacidad de plantear preguntas más sofisticadas sobre el set de datos puede permitir la creación de nuevas páginas en la web, usando nuevos criterios para ordenar la información.

Cabe destacar la importancia de Docker en este proyecto, sin el cual no hubiera sido posible simular todo el entorno. Se ha visto que es una herramienta de gran utilidad y se ha podido desarrollar con ella sin complicaciones.
Por otro lado, la herramienta Kibana, que se ha usado sobretodo en el TFG como interfaz para tratar con datos

de test, ha permitido familiarizarse con las consultas de ElasticSearch de manera rápida.

En relación a la empresa, Bodas.net, han expresado su satisfacción con los resultados de la comparativa, ya que les ofrecen un gran motivo para implantar ElasticSearch, el plan que ya tenían en mente.
Además, la parte realizada de integración en PHP les permitirá acelerar el cambio, y ahorrar tiempo en desarrollo y configuración del nuevo buscador.

B. Trabajo futuro

En el futuro, hay que experimentar más con el DSL de ElasticSearch para encontrar más funcionalidades beneficiosas para la empresa, a nivel de búsqueda y ordenación de contenido.

La integración en PHP tendrá que hacerse de forma completa, así como la migración de datos a ElasticSearch. Queda pendiente de ver si habría problemas en este sentido, el problema del bulk descrito en este artículo. En ese caso, habría que solucionándolo aumentando la memoria del cluster dedicada a la indexación o usar un tamaño más pequeño de batch (como se ha hecho en este TFG).

Finalmente, no se ha dimensionado el hardware en detalle para un entorno de producción. Habrá que decidir la implementación concreta, y tomar decisiones como el número de nodos que se usarán, y también si se usarán máquinas físicas para alojarlos o un entorno virtualizado como Docker.

8. AGRADECIMIENTOS

Como agradecimientos me gustaría mencionar a mis tutores del trabajo, a Andrew Koster en la universidad y a Gregorio Martínez en la empresa. Ambos han podido encontrar tiempo en sus agendas para poder ir evaluando el proyecto y ofrecer sugerencias y correcciones para que este saliera adelante.

9. BIBLIOGRAFÍA

[1] Sphinx 2.2.11-release reference manual[online]. 2016. Disponible en: http://sphinxsearch.com/docs/current.html
[2] Elasticsearch [online]. 2018. Disponible en: https://www.elastic.co/products/elasticsearch
[3] Apache Lucene Core [online]. 2014. Disponible en: https://lucene.apache.org/core/
[4] Apache Solr 7.4.0 [online]. 2018. Disponible en: http://lucene.apache.org/solr/
[5] System Properties Comparison Elasticsearch vs. Sphinx [online]. Disponible en: https://db-engines.com/en/system/Elasticsearch;Sphinx
[6] HALILI, Emily H. Apache JMeter: A practical beginner's guide to automated testing and performance measurement for your websites. Packt Publishing Ltd, 2008.
[7] THACKER, Urvi; PANDEY, Manjusha; RAUTARAY, Siddharth S. Performance of elasticsearch in cloud environment with nGram and non-nGram indexing. En Electrical, Electronics, and Optimization Techniques (ICEEOT), International Conference on. IEEE, 2016. p. 3624-3628.
[8] What is Docker [online]. 2018. Disponible en: https://www.docker.com/what-docker
[9] Overview of Docker Compose[online]. 2018. Disponible en: https://docs.docker.com/compose/overview/
[10] Configuring Elasticsearch [online]. 2016. Disponible en: https://www.elastic.co/guide/en/elasticsearch/reference/current/settings.html
[11] Kibana: Explore, Visualize, Discover Data [online]. 2018. Disponible en: https://www.elastic.co/products/kibana
[12] ElasticSearch-PHP [online]. 2017. Disponible en: https://www.elastic.co/guide/en/elasticsearch/client/php-api/current/index.html
[13] ab - Apache HTTP server benchmarking tool [online]. 2018. Disponible en: https://httpd.apache.org/docs/2.4/programs/ab.html
[14] Apache JMeter[online]. 2018. Disponible en: https://jmeter.apache.org/
[15] Symfony [online]. 2018. Disponible en: https://symfony.com/what-is-symfony