Base de datos distribuida de grafos con Neo4j para el Record Linkage de Redes Sociales.

Adrià Agudo Merino

Resumen—Este proyecto buscaba colaborar con el *Centre de Visió per Computador* en su proyecto de investigación *XARXES*, que tiene como objetivo construir redes sociales históricas a partir de datos demográficos de diferentes localidades. De esta manera se conseguiría una herramienta para analizar el comportamiento de las personas de una época y zona determinadas, así como la evolución social y económica de nuestros antepasados. Así pues, con el objetivo de dotar al proyecto *XARXES* de un sistema capaz de almacenar los datos demográficos e históricos que permita consultarlos de manera gráfica para su posterior análisis, se desarrolló una base de datos distribuida de grafos con *Neo4j*. Esta base de datos se desarrolló en un entorno de máquinas virtuales creado con *Vagrant*, de tal manera no se produjeran problemas de compatibilidades durante el despliegue en el clúster del *Centre de Visió per Computador*. Además, para garantizar que el sistema fuera escalable y extensible, se utilizaron contenedores *D*ocker, de tal manera que el sistema pudiera crecer a la par que el proyecto *XARXES*.

Palabras clave—almacenamiento de datos, base de datos de grafos, clúster, contenedores, Docker, Neo4j, sistema distribuido, Vagrant.

Abstract— This project sought to collaborate with the *Centre de Visió per Computador* in its *XARXES* research project, which aims to build historical social networks based on demographic data from different locations. In this way, a tool would be obtained to analyze the behavior of people from a certain time and area, as well as the social and economic evolution of our ancestors. Thus, with the aim of providing the *XARXES* project with a system capable of storing demographic and historical data that allows them to be consulted graphically for subsequent analysis, a distributed graph database with *Neo4j* was developed. This database was developed in a virtual machine environment created with *Vagrant*, in such a way that compatibility problems would not occur during the deployment in the cluster of the *Centre de Visió per Computador*. Also, to ensure that the system was scalable and extensible, *Docker* containers were used, so that the system could grow along with the *XARXES* project.

Index Terms—cluster, containers, data storage, distributed system, Docker, graph database, Neo4j, Vagrant.

1 Introducción

LOS documentos históricos de carácter demográfico permiten estudiar no sólo el comportamiento de un determinado grupo de personas en una determinada zona geográfica, sino también entender y explicar mejor la evolución social y económica de distintos momentos del pasado.

Es por ello por lo que, desde el *Centre de Visió per Computador*, se está llevando a cabo el proyecto de investigación *XARXES* [1], cuyo objetivo consiste en desarrollar técnicas informáticas para construir redes sociales históricas a partir de la vinculación de los habitantes registrados en los padrones y en otras fuentes demográficas de localidades vecinas. De esta manera se conseguiría una visión clara de los movimientos de la población en el pasado y de los vínculos que unían a las personas, permitiendo trazar una línea que una el presente con el pasado.

Para lograr esto, en primer lugar, se incorporarán mecanismos que permitan la contextualización de los contenidos de las fuentes originales, asociando la información semántica necesaria a los datos extraídos. En segundo lugar, se vincularán mediante *Record Linakge* [2] diferentes fuentes demográficas para la construcción de las redes sociales del pasado.

Finalmente, se incluirá la participación de la ciudadanía y de los archiveros, tanto en el proceso de extracción de los datos como en el diseño de nuevas y más atractivas formas de interacción de la plataforma final con el usuario, de manera que se facilite su uso y el conocimiento histórico sea accesible de manera gráfica y pedagógica.

- *E-mail de contacto: adria.agudom@e-campus.uab.cat*
- *Mención realizada: Ingeniería del Software.*
- *Trabajo tutorizado por: Oriol Ramos Terrades (Departament de Ciències de la Computació)*
- *Curso 2019/20*

Por la naturaleza del proyecto *XARXES*, se necesita una infraestructura sea capaz de manejar un gran volumen de datos. Para hacernos una idea, se trata de datos demográficos, de diferentes épocas y territorios (en este caso, localidades y municipios), por lo que la cantidad de datos puede llegar a ser realmente grande en el futuro. Así pues, el sistema debe ser extensible y escalable. Además, los datos pueden proceder de distintas fuentes y deben poder ser consultados desde diferentes lugares para conseguir que el sistema sea lo más accesible posible. Siguiendo en la línea de asegurar la accesibilidad, se busca que la infraestructura sea lo más resistente posible a caídas y fallos del sistema, de manera que, en caso de desastre, los datos no se vean comprometidos y continúen siendo accesibles.

Todo esto hace pensar que el sistema de almacenamiento debe ser una base de datos distribuida [3]. Si bien es cierto que una base de datos centralizada [4] otorgaría más integridad y evitaría problemas de redundancia e inconsistencia de datos, en este caso, al realizar posteriormente el *Record Linkage*, se palian parte de estos problemas.

Una vez aclaradas las motivaciones del trabajo y definido el problema a resolver, en las siguientes secciones de este documento se presentarán los objetivos del trabajo y se expondrá la planificación y metodología de desarrollo. A continuación, se hará un repaso del estado del arte, continuando por la exposición de los resultados obtenidos para, finalmente, presentar las conclusiones que se pueden extraer de ellos.

2 OBJETIVOS

El objetivo principal de este trabajo es crear una infraestructura escalable, extensible y distribuida para poder tratar grandes cantidades de datos, lo que conlleva una serie de objetivos secundarios o derivados:

- El diseño de una base de datos de grafos para almacenar los datos, así como la configuración del servidor de la base de datos.
- El diseño y desarrollo de la arquitectura de base de datos distribuida y el despliegue de la infraestructura en el clúster del *Centre de Visió per Computador*.
- La preparación de los datos a cargar para facilitar su posterior análisis.

Con estos objetivos se pretende contribuir al proyecto *XARXES*, desarrollando la infraestructura distribuida de almacenamiento de datos y realizando el tratamiento adecuado a los datos de tal manera que resulte sencillo su análisis.

3 PLANIFICACIÓN Y METODOLOGÍA

Con los objetivos descritos anteriormente, se definieron las siguientes tareas a realizar para obtener un desarrollo exitoso del proyecto:

1. Clusterización de la BD:
   a. Investigación sobre las herramientas a utilizar: búsqueda de información y realización de pruebas de funcionamiento.
   b. Diseño de la arquitectura de la BD: modelaje de la arquitectura distribuida de la BD.
   c. Creación y contenerización de la BD: configuración de la BD.
   d. Implementación de los clústeres: configuración de los clústeres y definición de las réplicas.
   e. Orquestación de los contenedores: configuración y gestión de los contenedores de los diferentes clústeres.
   f. Monitorización y pruebas: monitorizar el rendimiento de la base de datos y realizar pruebas de consistencia de datos y rendimiento.

2. Desarrollo de la BD:
   a. Estudio sobre criterios de fragmentación de datos: búsqueda de información sobre definición de índices y criterios de fragmentación de datos en *Neo4j*, realizar una comparativa y escoger la opción más adecuada.
   b. Volcado de datos: fragmentar y cargar los datos demográficos a la BD en base a la nueva arquitectura.
   c. Monitorización y pruebas: monitorizar el rendimiento de la base de datos y realizar pruebas de consistencia de datos y rendimiento.

3. Despliegue y entrega del sistema: instalación y configuración del sistema en el clúster del *Centre de Visió per Computador*.

La idea inicial era llevar a cabo el desarrollo del proyecto mediante el seguimiento de una estrategia iterativa o incremental, basada en la división del trabajo en pequeñas etapas repetitivas y aproximadamente del mismo tamaño en cuanto a horas de trabajo. Con ello se pretendía mitigar riesgos, mejorar los procesos de trabajo a medida que el proyecto avanza y conocer su progreso real.

Sin embargo, el proyecto contaba con demasiadas tareas que no podían empezar hasta haber acabado las anteriores. Por ejemplo, no se podía empezar con el tratamiento y volcado de los datos hasta que no se hubiera finalizado el desarrollo de la infraestructura de la base de datos.

Además, el hecho de no estar familiarizado y/o no dominar algunas tecnologías utilizadas para el desarrollo hacía que cada fase nueva implicara un tiempo de investigación y otro de prueba antes de empezar con el desarrollo real.

Estos factores hicieron que resultase más práctico cambiar el enfoque y seguir una estrategia de desarrollo secuencial, siempre en base a la planificación de tareas detallada anteriormente.

4 ESTADO DEL ARTE

Partiendo de la necesidad de que el sistema esté implementado sobre una arquitectura distribuida y sea capaz de escalar horizontalmente y tratar un gran volumen de datos, resulta idóneo que el sistema de almacenamiento de los datos históricos sea una base de datos no relacional [5].

El hecho de que los datos contengan información altamente relacionada y se pretenda saber cómo se relacionan las personas entre sí en un determinado lugar y período histórico hace que el modelo en forma de grafo [6] destaque por encima del resto, pues este tipo de BBDD representa de manera explícita las interrelaciones. Esto conlleva que la recuperación de elementos relacionados entre sí resulte muy sencilla y eficiente.

Sin embargo, las bases de datos de este tipo tienen un inconveniente, y es que no son tan fácilmente escalables como otros modelos no relacionales. Esto sucede porque, al estar los datos tan relacionados entre sí, se corre el riesgo de "romper" dichas relaciones durante la fragmentación de los datos. Además, está la posibilidad de tener elementos iguales en diferentes fragmentos, lo que dificulta el mantenimiento de los datos y su consistencia. A pesar de esto último, los beneficios de utilizar este tipo de base de datos no relacional eran mayores que los inconvenientes, por lo que el sistema de almacenamiento que se desarrollaría para almacenar los datos históricos sería una base de datos no relacional en forma de grafo.

Una vez determinado que la base de datos sería en forma de grafo, era necesario escoger cuál sería la base de datos que mejor se adaptaría a necesidades del proyecto. Entre las bases de datos más populares de este tipo [7] se encontraban: *Neo4j, Azure Cosmos DB, Orient DB y Arango DB.*

Neo4j [8] es una potente base de datos que hace tiempo que lidera el sector de BBDD de grafos. Está diseñada para almacenar, consultar, analizar y administrar datos altamente conectados con un nivel de optimización elevado para realizar operaciones gráficas como recorridos, agrupamiento de datos y cálculos de rutas óptimas, entre otras funciones.

La potencia de *Neo4j* se pone de manifiesto especialmente en la analítica gráfica de los datos y los algoritmos gráficos que ayudan a comprender los conjuntos de datos sin importar el nivel de profundidad y complejidad. Esto era perfecto para conseguir que el sistema fuese accesible de manera gráfica y pedagógica, tal y como se pretendía en el proyecto *XARXES*.

Además, es altamente escalable, tanto vertical como horizontalmente. Las operaciones son de alta fiabilidad (garantizan ACID) y no introduce problemas de integridad o consistencia de datos utilizando su arquitectura de *Causal Cluster*, que ahora también permite *multi-clustering*. Justo lo que queremos para nuestro sistema, que sea escalable. Sin embargo, cuenta con un lenguaje de consulta que requiere un nivel previo de capacitación, por lo que podía significar un inconveniente a la hora de implementar el sistema.

Azure Cosmos DB [9] es un servicio de datos multimodelo distribuido de forma horizontal que admite modelos de datos de documentos, pares clave-valor, grafos y familias de columnas en función de la API utilizada. Se podría clasificar como una base de datos NoSQL como servicio. Si se utiliza la API de *Gremlin*, se consigue un modelo de bases de datos de grafos que presenta muchas ventajas gracias a su sincronización por región de almacenamiento, lo que la hace una buena opción para proyectos de alta complejidad, como puede llegar a ser *XARXES*.

Azure Cosmos DB se fragmenta automáticamente de forma horizontal, crea y gestiona índices, escalas y sincronización por regiones, lo que nos asegura la escalabilidad que buscamos para nuestro sistema. Además, se hacen copias de seguridad de todos los grafos y los protege ante errores regionales. Estas garantías permiten a los desarrolladores centrarse en proporcionar valor a la aplicación en lugar de dedicarse a la operación y administración de sus bases de datos de grafos. Sin embargo, su duración es limitada (sólo se mantienen las dos últimas copias de seguridad durante un período de 8 horas). La restauración de las copias de seguridad sólo se puede conseguir reportando un ticket de soporte y esperando la asistencia del equipo de soporte de *Microsoft*.

Por otra parte, *Cosmos DB* requiere de un pago por uso en función de máquinas aprovisionadas, horas, región y capacidad de almacenamiento.

OrientDB [10] es un sistema de gestión de *BBDD NoSQL* de código abierto en *Java.* Al igual que *Cosmos DB*, soporta diferentes modelos no relacionales, pero en este caso las relaciones se gestionan como se hace en las bases de datos de grafos, con conexiones directas entre registros. Tiene un fuerte sistema de perfiles basado en usuarios y roles y admite consultas con *SQL* y *Gremlin* para el recorrido de grafos. Se basa en el modelo *master-master*, que incluye clústeres distribuidos geográficamente. Además, permite el particionado de datos mediante *sharding*, lo que aseguraba la escalabilidad que se buscaba para el proyecto. Sin embargo, es importante comprender un poco el motor de la base de datos y cómo funciona en

general (enlaces, por ejemplo). Esto implica una pequeña curva de aprendizaje si no está familiarizado con las bases de datos de grafos. Por otra parte, las inserciones masivas pueden provocar un bloqueo de memoria insuficiente, por lo que habría que tener cuidado y realizar las inserciones de una en una.

Arango DB [11] es una base de datos de código abierto que está diseñada desde una perspectiva multimodelo que se diferencia del estándar. Esta base de datos contempla módulos de datos de todo el universo *NoSQL*. Este modelo hibrido permite almacenar y analizar datos en forma de documentos, grafos y claves valor sin problemas. La flexibilidad y potencia de este motor resultaba bastante interesante. Cumple con las propiedades ACID y, además, pueden trabajarse grafos sociales para entender las relaciones entre usuarios y conjuntos de datos, cosa que hubiese ayudado con la red social histórica del proyecto *XARXES*. Además, cuenta con una versión de código abierto y una de licencia comercial que incorpora herramientas y funciones especiales para el análisis de grafos como *SmartGraphs*, que gestiona la división de nodos en un gráfico de atributos conocidos. Sin embargo, las limitaciones que presenta su tienda de valores clave y algunas restricciones de implementación de API dificultan la migración de datos que se encuentran en otro tipo de bases de datos *NoSQL* de tipo híbrido, lo que representaba una limitación para la evolución de proyectos en marcha hacia este gestor.

En base a la información recabada sobre las principales bases de datos de grafos comentadas anteriormente, se consideró que la mejor opción para este trabajo era *Neo4j*, por tratarse del sistema más expandido y mejor valorado por la comunidad de usuarios.

Para conseguir una base de datos distribuida, *Neo4j* cuenta con un sistema llamado *Fabric* [12], una forma de almacenar y recuperar datos de múltiples bases de datos utilizando una única consulta, pues utiliza una "base de datos virtual" (*Fabric*) como punto de entrada a la infraestructura distribuida/fragmentada. Esto resultaba idóneo puesto que el entorno de despliegue (el clúster del *CVC*) contaba con un nodo de "contención" desde el cual se acceden a los servicios del clúster, mientras que otros tres nodos internos se encargan de contener los datos y mantener los servicios.

Estos datos se almacenarían en *Causal Clusters* [13], una arquitectura de *Neo4j* que se compone de servidores principales (*core servers*), que salvaguardan los datos, y de réplicas de lectura (*read replicas*), que se encargan de escalar las cargas de trabajo de lectura. El número de servidores principales viene determinado en base al número de fallas que queremos que el sistema tolere siguiendo la formula

$$M = 2 * F + 1 \qquad (1),$$

dónde M es el número de *core servers* y F el número de fallas.

En cuanto al desarrollo de la base de datos, se creyó conveniente crear un entorno de pruebas basado en el uso de máquinas virtuales y contenedores, puesto que de esta manera se conseguiría una mayor consistencia entre los entornos de prueba y de producción y se obtendría una mayor modularidad, lo que facilitaría el posterior mantenimiento.

Para crear las máquinas virtuales que simulasen el entorno de despliegue se decidió utilizar *Vagrant* [14], una herramienta que permite crear y configurar entornos virtuales de desarrollo para evitar problemas de compatibilidades del producto desarrollado derivados de las diferencias existentes entre el entorno de desarrollo y el de despliegue. Puesto que la máquina desde la cual se desarrollaría el proyecto era un *Windows* 10, mientras que la de despliegue era una *CentOS/7*, esta herramienta podría evitar cualquier tipo de incompatibilidad o imprevisto durante el despliegue final de la BD.

En cuanto a los contenedores, *Docker* [15] fue la opción escogida ya que facilitaba el despliegue y entrega a los usuarios al no tener que programar ni configurar el software para diferentes tipos de plataformas hardware o sistemas operativos en los que debía poder ejecutarse el sistema.

Por otra parte, el uso de contenedores implicaba también la necesidad de utilizar un orquestador de contenedores. *Kubernetes* y *Docker Swarm* [16] son las opciones más conocidas y utilizadas por los desarrolladores, por lo que eran los principales candidatos.

Kubernetes [17] admite demandas más altas y con más complejidad, mientras que *Docker Swarm* [18] ofrece una solución bastante sencilla que resulta más rápida para comenzar. Así pues, este segundo resulta bastante popular entre los desarrolladores que prefieren implementaciones rápidas y sencillas. *Kubernetes*, en cambio, es más complejo de manejar, pero también es la opción que utilizan en entornos de producción varias empresas de Internet de alto perfil que ofrecen servicios populares como *Spotify*, *eBay* o *Pinterest*.

En base a esto último, y teniendo en cuenta el tiempo limitado para realizar el proyecto, *Docker Swarm* resultó en la opción preferida para ser el orquestador de contenedores.

5 DESARROLLO

Con el objetivo de cumplir con la planificación descrita en la sección 3 de este documento, los primeros pasos del desarrollo del proyecto se dieron en una dirección muy concreta: conseguir la clusterización de la base de datos.

Así pues, se empezó a realizar pruebas de familiarización con *Docker* y *Neo4j* [19][20]. Estas pruebas consistieron en levantar un contenedor con la imagen de *Neo4j* para *Docker* en la máquina de trabajo del estudiante para, seguidamente, realizar los tutoriales que propone la misma base de datos para familiarizarse con su propio lenguaje de gestión y consultas, *Cypher*.

Una vez realizadas las pruebas en local, se elaboró el diseño de la arquitectura de la base de datos distribuida reflejada en la Fig. 1.

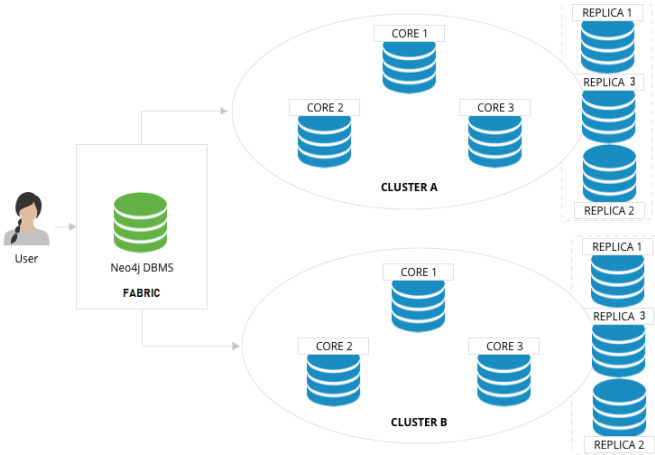

Fig. 1. Modelo de la arquitectura final del sistema de base de datos Neo4j

La arquitectura diseñada se componía de una "base de datos virtual" *Fabric* conectada a dos *Causal Clusters* (Clústers A y B) típicos, cada uno de los cuales almacenaría un fragmento de los datos. En este caso, como la cantidad de datos demográficos de la que se disponía no era demasiado grande (unos 500 MB aproximadamente), se decidió optar por una estructura típica de *Causal Cluster* (Fig. 2) compuesta por tres *cores* y tres *read replicas*, de tal manera que el clúster fuera capaz de tolerar una falla.

Con este diseño (Fig. 1), la arquitectura *Fabric* proporcionaría una alta escalabilidad y disponibilidad sin un solo punto de falla. Además, los *Causal Clusters* podrían dimensionarse en el futuro en función de la carga de trabajo esperada y los datos podrían ubicarse en el mismo clúster o fragmentarlos para proporcionar un mayor rendimiento.

Esta arquitectura debía desplegarse en el clúster del *Centre de Visió per Computador* concretamente en cuatro de sus nodos, uno de los cuales era el "nodo de contención". Este nodo tiene la función de servir como puerta de acceso a los servicios del clúster, por lo que no almacena datos de aplicaciones, simplemente realiza peticiones y ofrece

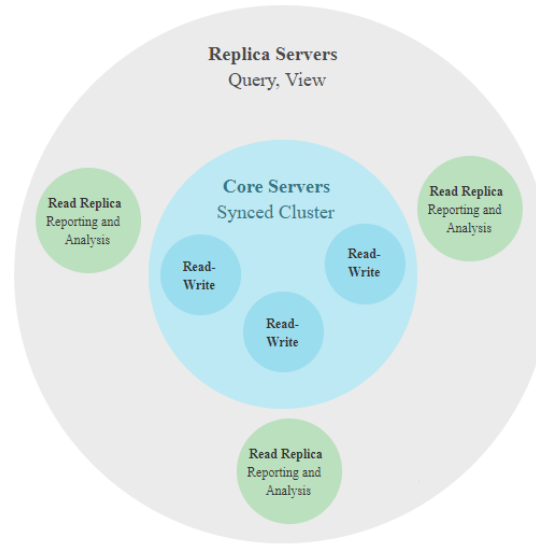

Fig. 2. Arquitectura de un Causal Cluster típico

las correspondientes respuestas. Son el resto de los nodos los que se encargan de procesar las peticiones realizadas por el "nodo de contención" y almacenar los datos. En este caso, el *CVC* proporcionó tres de estos nodos para el servicio de la base de datos distribuida.

Este diseño, por lo tanto, se ajusta a la perfección con el entorno de despliegue, puesto que *Fabric*, que no contiene datos de la BD, podría desplegarse en el nodo de contención y actuar de puerta de acceso a la base de datos, mientras que el resto de los contenedores, que sí que almacenan datos y los procesan, se encontrarían en los otros tres nodos disponibles.

Con el diseño de la arquitectura determinado, el siguiente paso era la creación y contenerización de la base de datos.

En primer lugar, se creó el entorno de desarrollo. Con *Vagrant* se crearon cuatro máquinas virtuales *CentOS/7* que simularían ser el "nodo de contención" y los nodos de procesamiento y almacenamiento de datos. Con *Docker Swarm* se estableció un clúster con ellas, lo que permitía que están cuatro máquinas virtuales se comportasen verdaderamente como nodos de un clúster. Finalmente, con el objetivo de que los contenedores que se levantarían estuvieran conectados entre sí y fueran capaces de intercambiar información, se creó una red *Docker* de tipo *overlay* y un servicio de *nginx* [21]. La red *overlay* permitiría tener una red privada interna que abarcase todos los nodos del clúster, mientras que el servicio de *nginx* se encargaría del enrutamiento entre contenedores.

En segundo lugar, se decidió que los contendores correspondientes al diseño de la Fig. 1 se distribuirían de la siguiente manera en el clúster del *CVC*: *Fabric* en el nodo de "contención" y un *core* y un *read replica* de cada *Causal Cluster* en cada uno de los tres nodos restantes. El siguiente paso era la creación de los contenedores *Docker* de los diferentes componentes de la arquitectura diseñada.

Primeramente, se levantaron los contenedores del *Causal Cluster* A en los tres nodos de almacenamiento y procesamiento del clúster de desarrollo, de tal manera que cada nodo tuviese un *core* y un *read replica*.

Con los contenedores trabajando en la red *overlay* y conectados entre sí, se pasó a realizar pruebas en él. En estas pruebas se puedo observar cómo cada *core* y *read replica* tenía una réplica de los datos totales de la base de datos y que sólo se permitía realizar operaciones de escritura a los *cores*. Así pues, se pudo determinar que, el Clúster A respondía según lo esperado. El siguiente paso fue realizar las mismas acciones para levantar el Clúster B.

En este punto, estaban levantados y funcionando los dos *Causal Cluster* que contendrían los fragmentos de los datos del sistema, por lo que el siguiente paso era levantar el *Fabric* en el "nodo de contención" del clúster de máquinas virtuales. Para ello, durante la configuración de este, se estableció que *Fabric* estaría conectado a los Clústers A y B y que cada uno contendría un fragmento de los datos de la BD total.

Se levantó el contenedor de *Fabric* en el "nodo de contención" del clúster de desarrollo y se comprobó que este podía recuperar los datos alojados en los Clústers A y B, por lo que la arquitectura distribuida de base de datos con *Neo4j* quedó funcionando y se finalizó así la primera de las tareas de la planificación, la clusterización de la BD.

La siguiente tarea era el desarrollo de la BD e incluía una investigación sobre criterios de fragmentación de datos en *Neo4j*, así como el tratamiento y volcado de los datos demográficos al sistema desarrollado. Estos datos se encontraban almacenados en una base de datos del *Centre de Visió per Computador* y se esperaba poder realizar una exportación de los mismos. Sin embargo, tras varios intentos sin éxito, se pudo comprobar que era una tarea más complicada de lo previsto en un primer momento, por lo que se tuvo que optar por un plan B: importar al sistema desarrollado *datasets* públicos de distintos tamaños y realizar las pruebas de rendimiento con estos datos.

Esto, lejos de suponer un problema, permitió observar cómo se comportaba el sistema en función del tamaño del dataset importado y compararlo con el comportamiento de un sistema no distribuido.

6 Pruebas y Resultados

Con el objetivo de medir el rendimiento de la arquitectura distribuida desarrollada formada por un *Fabric* con dos *Causal Clusters* de tres *cores* y tres *read replicas* respecto al de una arquitectura no distribuida, se realizaron una serie de consultas sobre dos tipos de datasets de distintos tamaños, considerando el tiempo de respuesta a dichas consultas el factor que determinara el rendimiento del sistema.

Puesto que el tiempo de respuesta puede estar sujeto a variaciones en función de diferentes factores (como puede ser la memoria caché), se realizaron cuatro mediciones por cada consulta con el objetivo de evaluar su evolución en función del número de veces que realiza la consulta y obtener así una medición más fiable y ajustada a la realidad.

En un principio, estas pruebas iban a realizarse sobre los datos demográficos del *CVC,* por lo que durante la fase de desarrollo se preparó un conjunto de 10 consultas sobre los datos demográficos disponibles y que se consideraba que podían ser habituales entre los potenciales usuarios de la base de datos, de tal manera que se pudiera tener una idea del rendimiento general del sistema. Estas consultas fueron expuestas a Joana Maria Pujades Mora, investigadora del *Centre d'Estudis Demogràfics* [22], quién las validó y propuso cinco nuevas consultas, las cuales fueron añadidas al conjunto de consultas anterior.

Sin embargo, tal y como se ha explicado anteriormente, debido a la complejidad de la exportación de estos datos, estas pruebas tuvieron que realizarse sobre dos *datasets* públicos: *EURO 2016* [23], que contenía datos sobre los jugadores de futbol que participaron en la pasada *UEFA EURO 2016*, y el segundo, *Contratación pública de México* [24], que contenía datos sobre la contratación pública de México.

Hay que añadir que esta prueba de rendimiento se llevó a cabo en el equipo del estudiante y que no se realizaron otras acciones sobre la base de datos mientras tenían lugar las consultas.

6.1 Dataset EURO 2016

En primer lugar, se realizaron las pruebas sobre un pequeño y sencillo dataset que recogía los datos de los jugadores de fútbol que participaron en la pasada *UEFA EURO 2016*.

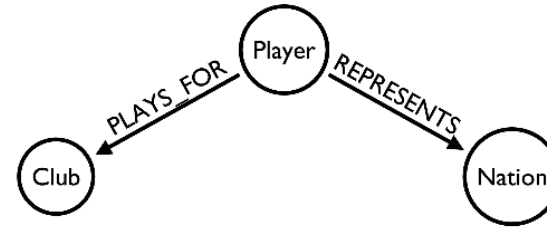

Fig. 3. Data Model del dataset EURO 2016

Como se puede observar en la Fig. 3, en este dataset hay datos sobre los jugadores participantes, los clubes para los que juegan y las naciones a las que representan. En total, se trata de un conjunto de datos que contiene 1030 nodos, 1027 propiedades y 1513 relaciones, lo que suponían 1,19MB de almacenamiento.

En cuanto a la fragmentación que se realizó sobre estos datos para cargarlos en el sistema distribuido, se decidió poner los datos sobre los jugadores y los clubes en los que juegan en un clúster (*graphA* para el servidor de *Fabric*) y

los datos sobre los jugadores y las naciones a las que representan en otro clúster (*graphB* para el servidor de *Fabric*).

Así pues, una vez se fragmentaron los datos, se cargaron a la base de datos y se ejecutaron las consultas de la prueba, se obtuvieron los resultados recogidos en la Fig. 4.

	Consulta 1		Consulta 2		Consulta 3	
	Sist. Distribuido	Sist. No Distribuido	Sist. Distribuido	Sist. No Distribuido	Sist. Distribuido	Sist. No Distribuido
Intento 1	1936	63	618	24	4091	229
Intento 2	440	29	60	8	942	4
Intento 3	121	21	55	7	137	3
Intento 4	51	12	38	6	35	2

Fig. 4. Comparativa de tiempos de respuesta en ms entre sistemas para las consultas del dataset EURO 2016

En estos resultados se puede observar como el sistema distribuido desarrollado ofrece un peor rendimiento que un sistema no distribuido en todas las consultas realizadas.

Se esperaba que esto fuera así para la consulta 1, puesto que sólo necesita los datos de uno de los fragmentos, por lo que el sistema no distribuido debía ofrecer un mejor resultado. Sin embargo, esto se traslada también al resto de consultas. Estos resultados pues, son negativos e inesperados, ya que el sistema distribuido, que cuenta con balanceo de carga, debería comportarse mejor que el no distribuido.

6.2 Dataset Contratación pública de México

En segundo lugar, se realizaron las pruebas sobre un dataset considerablemente mayor y más complejo que recoge datos de contratación pública que reflejan el ciclo de vida de la contratación pública en México.

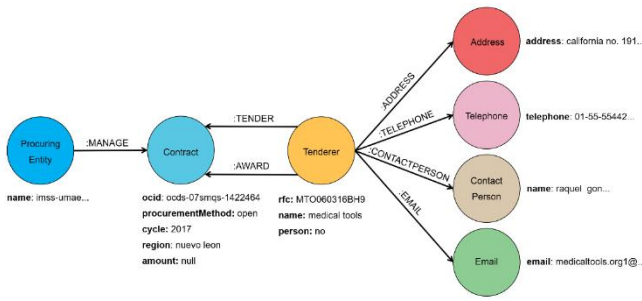

Fig. 5. Data Model del dataset Contratación pública de México

En este *dataset* se recogen los procesos de contratación (*Contract*), las empresas o personas que ofrecen servicios (*Tenderer*), las entidades públicas que gestionan los procesos de contratación (*ProcutingEntity*) y la dirección, número de teléfono, persona de contacto y correo electrónico de cada *Tenderer*.

Hay dos posibles tipos de relación para capturar los roles que las empresas *y* las personas juegan en un proceso de contratación. (*TENDER*) identifica a todas las entidades que compiten en la fase de licitación, y (*AWARD*) solo aquellos contratos ganadores en el proceso.

Otras relaciones capturan la conexión entre las empresas y su respectiva información de contacto: (*ADDRESS*), (*TELEPHONE*), (*CONTACTPERSON*) y (*EMAIL*). Finalmente, (*MANAGE*) conecta las entidades contratantes con sus respectivos contratos. En total, el dataset está compuesto por 128748 nodos, 186827 propiedades y 242789 relaciones, lo que supone un total de 99,30 MB.

En cuanto a la fragmentación de los datos, se decidió tener por un lado las entidades procuradoras, los contratos y los tenderer, y por otro todo lo referente a los tenderers y sus datos personales. La distribución queda reflejada en la Fig. 6.

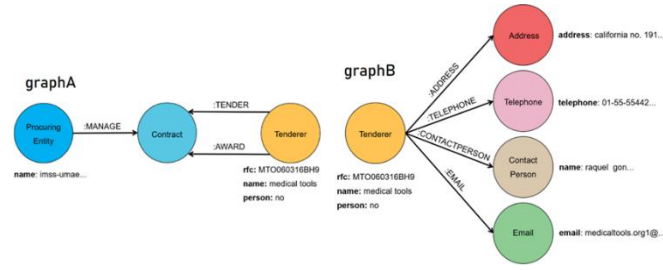

Fig. 6. Data Model de la fragmentación de los datos del dataset Contratación pública de México

Así pues, tal y como se hizo con el *dataset EURO 2016*, en cuanto se fragmentaron los datos y se cargaron a la base de datos, se procedió a realizar las pruebas de rendimiento.

	Consulta 1		Consulta 2		Consulta 3	
	Sist. Distribuido	Sist. No Distribuido	Sist. Distribuido	Sist. No Distribuido	Sist. Distribuido	Sist. No Distribuido
Intento 1	68	27	762	22	722	17
Intento 2	31	10	66	11	370	4
Intento 3	11	7	42	3	32	4
Intento 4	6	4	32	3	15	4

Fig. 7. Comparativa de tiempos de respuesta en ms entre sistemas para las consultas del dataset Contratación pública de México

Los resultados de las pruebas se recogen en la Fig. 7. Estos resultados vuelven a ser negativos. El sistema distribuido siempre ofrece un peor rendimiento que el no distribuido y, en este caso, la diferencia de tiempo entre ambos es mucho más abrupta en términos absolutos, llegando a tardar 3862ms más en responder para el primer intento de la consulta 4.

En este caso, esto es más inesperado, puesto que al tratarse de un *dataset* con muchos más nodos y relaciones que el anterior, el balanceo de carga del sistema distribuido debería suponer una ventaja sobre el no distribuido.

Con el objetivo de ver si esto estaba relacionado con el uso de máquinas virtuales limitadas y el número de contenedores de cada una, se realizaron nuevas pruebas.

Aprovechando que la consulta 1 de las pruebas al *datasets EURO 2016* que sólo necesitaba los datos de uno de los fragmentos (*graphA*), se utilizó esta misma consulta para ver cómo variaba el tiempo de respuesta en función del número de contenedores del clúster y del uso de *Fabric*. De esta manera se podría determinar si los resultados negativos obtenidos anteriormente se debían a las limitaciones de las máquinas virtuales y/o al uso de *Fabric*.

	3 cores y 3 read replicas		3 cores y 2 read replicas		3 cores y 1 read replica		3 cores sin read replicas	
	Fabric	Sin Fabric	Fabric	Sin Fabric	Fabric	Sin Fabric	Fabric	Sin Fabric
Intento 1	1146	90	1546	37	989	28	1552	21
Intento 2	50	17	46	20	27	8	43	14
Intento 3	22	12	24	11	23	8	29	9
Intento 4	14	6	13	5	12	5	13	5

Fig. 8. Comparativa de tiempos de respuesta en ms del sistema distribuido pasando por Fabric o sin pasar

En los resultados de estas pruebas (Fig. 8) se puede observar claramente cómo el hecho de recuperar los datos mediante *Fabric* tiene una incidencia negativa en cuanto al tiempo de respuesta de la base de datos. Sin embargo, el número de contenedores no tiene tanto impacto en el tiempo de respuesta, pues los resultados son bastante similares cuando hay tres *read replicas* y cuando hay uno sólo.

6.3 Análisis de resultados

Tal y como se ha comentado anteriormente, los resultados fueron claramente negativos. El sistema distribuido ofreció siempre peores resultados que el no distribuido.

Para el *dataset EURO 2016*, en la Fig. 4 se puede observar como para la Consulta 1 el sistema no distribuido resulta un factor de 2.51x más rápido que el distribuido (41ms menos). Sin embargo, esta diferencia se rebaja conforme la consulta se realiza más veces, siendo al final un factor de 1,5x (2ms menos).

Esta situación se dio también en el resto de las consultas y en ambos sistemas. El origen de esto puede estar en la memoria cache de la BD, que guarda los caminos para llegar a los datos consultados recientemente. Por otra parte, puede observarse que esta tendencia a la baja es mucho más abrupta en el sistema distribuido. La explicación a este hecho reside en el balanceo de carga que se realiza en los *Causal Clusters* del sistema distribuido entre los diferentes nodos que lo componen.

Sin embargo, existe una diferencia más importante en cuanto al rendimiento en las consultas 2 y 3 (Fig. 4). Esto se debe a que en estas consultas el sistema distribuido debe acceder a los dos fragmentos de los datos y esperar la respuesta salida de uno de ellos para procesar la del otro (tal y como se puede ver en la figura anterior), mientras que en la consulta 1 sólo debe acceder a una parte. A pesar de ello y al igual que sucede en el caso anterior, la diferencia se ve reducida a medida que se realizan las consultas más veces, pasando de unos factores 42x i 36x para las consultas 2 y 3, a unos factores 10,67x i 3,75x, respectivamente.

Los resultados tampoco fueron demasiado distintos para las pruebas con el *dataset Contratación pública de México*. Como se puede observar en la Fig. 7, el sistema no distribuido también ofrece un mayor rendimiento que el distribuido en la realización de las consultas de la prueba. En este caso, como se ha comentado anteriormente, se trata de un resultado aún más inesperado, puesto que en este caso el dataset era mayor y más complejo que el anterior. Esto debía suponer una ventaja para para el sistema distribuido, ya que debería ser capaz de dividir la carga de trabajo de una consulta que implica gran cantidad de nodos y relaciones. Sin embargo, esto no es así.

En la consulta 1 (Fig. 7) se observa que el sistema no distribuido resulta 30,73 veces más rápido que el no distribuido la primera vez que se realiza la consulta. De la misma manera que sucedió para el dataset anterior, la diferencia entre ambos sistemas se ve reducida a medida que realizamos más veces la consulta, puesto que pasa de un factor 30,73x a un 4,25x.

En la consulta 2 se puede observar cómo, en primera instancia, la proporción diferencial de 25,75x, la cual pasa a ser de 6,33x para la cuarta vez que se realiza la consulta. Y este resultado empeora aún más en la consulta 3, dónde pasamos de un factor 17,86x a un 17,5x, por lo que apenas mejora.

En vista de los resultados, se puede concluir que el sistema distribuido desarrollado se comporta peor que un sistema no distribuido tanto para datasets pequeños como para datasets mayores.

En cuanto a las pruebas que se realizaron posteriormente para estudiar el efecto del número de contenedores activos y el uso de máquinas virtuales, los resultados tampoco fueron demasiado esperanzadores.

Claramente el hecho de que el sistema esté desplegado en máquinas virtuales debe afectar al rendimiento del sistema, pues los recursos de la máquina sobre la que se están ejecutando deben repartirse entre cada una de ellas (4 en total). Esto lleva a pensar que, a su vez, cuantos más contenedores corriendo en las máquinas virtuales, peor rendimiento debe ofrecer el sistema distribuido desarrollado. Además, el hecho de utilizar Fabric también debería tener cierto impacto en cuanto al tiempo de respuesta, pues cada clúster debe enviar los datos encontrados al Fabric.

Como se ha comentado anteriormente y se puede observar en la Fig. 8, el hecho de pasar o no por Fabric sí que tiene un impacto real sobre el tiempo de respuesta. De hecho, si no se utiliza Fabric, los resultados se acercan más a los del sistema no distribuido. Sin embargo, reducir el número de contenedores no afecta demasiado al rendimiento, pues se pueden observar unos resultados bas-

tante similares cuando hay tres read replicas y cuando hay uno sólo, siendo un poco mejor en este último caso. Por otra parte, se espera que estos resultados sean distintos cuando el sistema esté desplegado en el clúster del *Centre de Visió per Computador*, pues los nodos serán más potentes en términos de computación que las máquinas virtuales utilizadas para el desarrollo.

7 CONCLUSIONES

El objetivo principal del trabajo era desarrollar una base de datos distribuida de grafos con *Neo4j* para contribuir al proyecto *XARXES* que se lleva a cabo en el *Centre de Visió per Computador*.

Ese objetivo es una realidad a día de hoy. Se ha desarrollado la base de datos y se ha hecho de tal manera que sea fácilmente escalable y extensible, pues en caso de experimentar problemas de rendimiento simplemente sería cuestión de añadir más *read replicas* al *Causal Cluster* en cuestión, de manera que se realice un mayor balanceo de carga, y en caso de querer extender la base de datos, simplemente sería cuestión de añadir otro *Causal Cluster* en el que almacenar los nuevos datos.

Se trata de un sistema que puede tener un largo recorrido en el tiempo y que puede llegar a ser mucho mayor en el futuro.

Si bien todavía quedan cosas por hacer, como realizar el despliegue en el clúster del *Centre de Visió per Computador* y cargar los datos demográficos en la base de datos, se espera que en futuro cercano esto pueda hacerse.

AGRADECIMIENTOS

En primer lugar, quiero agradecer a mi tutor de TFG, Oriol Ramos Terrades, el apoyo prestado durante el desarrollo del trabajo, así como su guía en los momentos complicados del mismo.

Por otra parte, también quería agradecer a Joana Maria Pujades Mora, investigadora del *Centre d'Estudis Demogràfics*, que dedicara parte de su tiempo a este trabajo.

BIBLIOGRAFÍA

[1] "INFO | XARXES", Dag.cvc.uab.es, 2020. [En línea]. Disponible: http://dag.cvc.uab.es/xarxes/info/. [Accedido: 12- Feb- 2020].

[2] "Record linkage", En.wikipedia.org, 2020. [En línea]. Disponible: https://en.wikipedia.org/wiki/Record_linkage. [Accedido: 12- Feb- 2020].

[3] "¿Qué es una base de datos distribuida y porqué puede ser interesante?", Informática para tu negocio, 2020. [En línea]. Disponible: https://www.informaticaparatunegocio.com/blog/una-base-datos-distribuida-puede-interesante/. [Accedido: 03- Mar- 2020].

[4] "Base de datos centralizada", Es.qwe.wiki, 2020. [En línea]. Disponible: https://es.qwe.wiki/wiki/Centralized_database. [Accedido: 03- Mar- 2020].

[5] "Bases de datos NoSQL. Qué son y tipos que nos podemos encontrar", Acens.com, 2020. [En línea]. Disponible: https://www.acens.com/wp-content/images/2014/02/bbdd-nosql-wp-acens.pdf. [Accedido: 15- Feb- 2020].

[6] P. UOC, "Introducción a las bases de datos NoSQL en grafo", Informática ++, 2020. [En línea]. Disponible: http://informatica.blogs.uoc.edu/2018/05/28/introduccion-a-las-bases-de-datos-nosql-en-grafo/. [Accedido: 15- Feb- 2020].

[7] "DB-Engines Ranking of Graph DBMS", DB-Engines, 2020. [En línea]. Disponible: https://db-engines.com/en/ranking/graph+dbms. [Accedido: 18- Feb- 2020].

[8] "Neo4j System Properties", Db-engines.com, 2020. [En línea]. Disponible: https://db-engines.com/en/system/Neo4j. [Accedido: 20- Feb- 2020].

[9] "Introducción a Gremlin API de Azure Cosmos DB", Docs.microsoft.com, 2020. [En línea]. Disponible: https://docs.microsoft.com/es-es/azure/cosmos-db/graph-introduction. [Accedido: 20- Feb- 2020].

[10] "OrientDB", En.wikipedia.org, 2020. [En línea]. Disponible: https://en.wikipedia.org/wiki/OrientDB. [Accedido: 23- Feb- 2020].

[11] D. Naranjo, "ArangoDB un sistema de base de datos multimodelo de código abierto", Linux Adictos, 2020. [En línea]. Disponible: https://www.linuxadictos.com/arangodb-un-sistema-de-base-de-datos-multimodelo-de-codigo-abierto.html. [Accedido: 23- Feb- 2020].

[12] "8.1. Introduction - Chapter 8. Fabric", Neo4j.com, 2020. [En línea]. Disponible en: https://neo4j.com/docs/operations-manual/current/fabric/introduction/. [Accedido: 21- May- 2020].

[13] "6.1. Introduction - Chapter 6. Clustering", Neo4j.com, 2020. [En línea]. Disponible en: https://neo4j.com/docs/operations-manual/current/clustering/introduction/. [Accedido: 01- Abr- 2020].

[14] "Introducción a Vagrant: ¿Qué es? ¿Para qué sirve? - Guiadev", Guiadev, 2020. [En línea]. Disponible en: https://guiadev.com/vagrant/. [Accedido: 21- Mar- 2020].

[15] T. Rodríguez, "De Docker a Kubernetes: entendiendo qué son los contenedores", Xataka.com, 2020. [En línea]. Disponible: https://www.xataka.com/otros/docker-a-kubernetes-entendiendo-que-contenedores-que-mayores-revoluciones-industria-desarrollo. [Accedido: 26- Feb- 2020].

[16] S. Ravindra, K. Farmer and R. Staatz, "Kubernetes vs. Docker Swarm: What's the Difference?", The New Stack, 2020. [En línea]. Disponible: https://thenewstack.io/kubernetes-vs-docker-swarm-whats-the-difference/. [Accedido: 26- Feb- 2020].

[17] "Neo4j Considerations in Orchestration Environments", Medium, 2020. [En línea]. Disponible en: https://medium.com/neo4j/neo4j-considerations-in-orchestration-environments-584db747dca5. [Accedido: 05- Abr- 2020].

[18] "Neo4j Container Orchestration with Kubernetes, Docker Swarm & Mesos", Neo4j Graph Database Platform, 2020. [En línea]. Disponible en: https://neo4j.com/blog/neo4j-container-orchestration-kubernetes-docker-swarm-mesos/. [Accedido: 08- Abr- 2020].

[19] "How-To: Run Neo4j in Docker - Neo4j Graph Database Platform", Neo4j Graph Database Platform, 2020. [En línea]. Disponible en: https://neo4j.com/developer/docker-run-neo4j/. [Accedido: 14- Mar- 2020].

[20] "Chapter 3. Docker - The Neo4j Operations Manual v4.0", Neo4j.com, 2020. [En línea]. Disponible en: https://neo4j.com/docs/operations-manual/current/docker/. [Accedido: 14- Mar- 2020].

[21] "Networking with overlay networks", Docker Documentation, 2020. [En línea]. Disponible en: https://docs.docker.com/network/network-tutorial-overlay/#use-a-user-defined-overlay-network. [Accedido: 21- May- 2020].

[22] "Centro de Estudios Demográficos", Centro de Estudios Demográficos, 2020. [En línea]. Disponible en: https://ced.uab.cat/es/. [Accedido: 26- Mar- 2020].
[23] "graphgist - UEFA 2016", Neo4j Graph Database Platform, 2020. [En línea]. Disponible en: https://neo4j.com/graphgist/uefa-euro-2016. [Accedido: 15- May- 2020].
[24] "Graph databases for journalists", Medium, 2020. [En línea]. Disponible en: https://medium.com/neo4j/graph-databases-for-journalists-5ac116fe0f54. [Accedido: 17- May- 2020].

APÉNDICE

A1. CONSULTAS DE LA PRUEBA DE RENDIMIENTO PARA EL DATASET EURO 2016

Consulta 1

Consulta	MATCH (c:Club)<-[PLAYS_FOR]-(Player) RETURN c.name AS Club, count(*) AS Internationals ORDER BY Internationals DESC LIMIT 10;
Consulta Sist. dist	use fabric.graphA MATCH (c:Club)<-[PLAYS_FOR]-(Player) RETURN c.name AS Club, count(*) AS Internationals ORDER BY Internationals DESC LIMIT 10;
Resultados Sist. No Distribuido	Started streaming 10 records after 12 ms and completed after 27 ms. Started streaming 10 records after 1 ms and completed after 10 ms. Started streaming 10 records after 1 ms and completed after 7 ms. Started streaming 10 records in less than 1 ms and completed after 4 ms.
Resultados Sist. Distribuido	Started streaming 10 records after 10 ms and completed after 68 ms. Started streaming 10 records after 1 ms and completed after 31 ms. Started streaming 10 records after 1 ms and completed after 11 ms. Started streaming 10 records in less than 1 ms and completed after 6 ms.

Consulta 2

Consulta	MATCH (c:Club)<-[PLAYS_FOR]-(p:Player)-[REPRESENTS]->(n:NationalTeam) WHERE c.name='Manchester United' RETURN n.name AS Country, count(*) AS Count, collect(p.name) AS Internationals ORDER BY Count DESC;

Consulta Sist. dist	CALL{
	use fabric.graphA
	MATCH (c:Club)<-[PLAYS_FOR]-(p:Player)
	WHERE c.name='Manchester United'
	RETURN collect(p.name) AS Players}
	CALL{
	use fabric.graphB
	with Players
	MATCH (p:Player)-[REPRESENTS]->(n:NationalTeam)
	WHERE p.name in Players
	RETURN n.name AS Country, count(*) AS Count, collect(p.name) AS Internationals
	ORDER BY Count DESC
	}
	RETURN Country, Count, Internationals

Resultados Sist. No Distribuido	Started streaming 7 records after 15 ms and completed after 22 ms. Started streaming 7 records after 1 ms and completed after 11 ms. Started streaming 7 records after 1 ms and completed after 3 ms. Started streaming 7 records after 1 ms and completed after 3 ms.
Resultados Sist. Distribuido	Started streaming 7 records after 31 ms and completed after 762 ms. Started streaming 7 records after 1 ms and completed after 66 ms. Started streaming 7 records after 1 ms and completed after 42 ms. Started streaming 7 records after 1 ms and completed after 32 ms.

Consulta 3

Consulta	MATCH (c:Club)<-[PLAYS_FOR]-(p:Player)-[REPRESENTS]->(n:NationalTeam) WHERE n.name='Spain' RETURN c.name AS Club, count(*) AS Count, collect(p.name) AS Internationals ORDER BY Count DESC;

Consulta Sist. dist	CALL{
	use fabric.graphB
	MATCH (p:Player)-[REPRESENTS]->(n:NationalTeam)
	WHERE n.name='Spain'
	RETURN collect(p.name) AS Players}
	CALL{
	use fabric.graphA
	with Players
	MATCH (c:Club)<-[PLAYS_FOR]-(p:Player)
	WHERE p.name in Players
	RETURN c.name AS Club, count(*) AS Count, collect(p.name) AS Internationals
	ORDER BY Count DESC
	}
	RETURN Club, Count, Internationals

Resultados Sist. No Distribuido	Started streaming 18 records after 15 ms and completed after 17 ms. Started streaming 18 records in less than 1 ms and completed after 4 ms. Started streaming 18 records in less than 1 ms and completed after 4 ms. Started streaming 18 records in less than 1 ms and completed after 4 ms.
Resultados Sist. Distribuido	Started streaming 18 records after 74 ms and completed after 722 ms. Started streaming 18 records after 2 ms and completed after 370 ms. Started streaming 18 records after 1 ms and completed after 32 ms. Started streaming 18 records in less than 1 ms and completed after 15 ms.

A2. CONSULTAS DE LA PRUEBA DE RENDIMIENTO PARA EL DATASET CONTRATACIÓN PÚBLICA DE MÉXICO

Consulta 1

Consulta	MATCH (a:Tenderer {name: 'acondicionamiento en potencia y comunicaciones sa de cv'})-[:TENDER]->(b:Contract)<-[:TENDER]-(c:Tenderer) RETURN a,b,c
Consulta Sist. dist	use fabric.graphA MATCH (a:Tenderer {name: 'acondicionamiento en potencia y comunicaciones sa de cv'})-[:TENDER]->(b:Contract)<-[:TENDER]-(c:Tenderer) RETURN a,b,c
Resultados Sist. No Distribuido	Started streaming 2 records after 1 ms and completed after 63 ms. Started streaming 2 records after 1 ms and completed after 29 ms. Started streaming 2 records after 1 ms and completed after 21 ms. Started streaming 2 records after 1 ms and completed after 12 ms.
Resultados Sist. Distribuido	Started streaming 2 records after 1 ms and completed after 1936 ms. Started streaming 2 records after 1 ms and completed after 440 ms. Started streaming 2 records after 1 ms and completed after 121 ms. Started streaming 2 records after 1 ms and completed after 51 ms.

Consulta 2

Consulta	MATCH (c:Contract{ocid:'ocds-07smqs-1520579'})-[*1..2]- (t:Tenderer) -[]-(p) RETURN t, p

Consulta Sist. dist	call{
	use fabric.graphA
	MATCH (c:Contract{ocid:'ocds-07smqs-1520579'})-[*1..2]-(t:Tenderer)
	RETURN t.rfc as TendRfc
	}
	call{
	use fabric.graphB
	WITH TendRfc
	MATCH (t:Tenderer {rfc:TendRfc})-[]-(p)
	RETURN t as Tenderer, p as personalData
	}
	RETURN Tenderer, personalData

Resultados Sist. No Distribuido	Started streaming 61 records after 2 ms and completed after 24 ms. Started streaming 61 records in less than 1 ms and completed after 8 ms. Started streaming 61 records after 1 ms and completed after 7 ms. Started streaming 61 records in less than 1 ms and completed after 6 ms.
Resultados Sist. Distribuido	Started streaming 14 records after 29 ms and completed after 618 ms Started streaming 14 records after 1 ms and completed after 60 ms. Started streaming 14 records after 1 ms and completed after 55 ms. Started streaming 14 records after 1 ms and completed after 38 ms.

Consulta 3

Consulta	MATCH (tel:Telephone {telephone: "3323014185"}) -[]-(t:Tenderer) -[]-(c:Contract)-[]-(pe:ProcuringEntity) RETURN t as Tenderer, c as Contract, pe as ProcuringEntity

Consulta Sist. dist	call{
	use fabric.graphB
	MATCH (t:Tenderer)-[]-(tel:Telephone {telephone: "3323014185"})
	RETURN t.rfc as TendRfc
	}
	call{
	use fabric.graphA
	WITH TendRfc
	MATCH (t:Tenderer {rfc:TendRfc})-[]-(c:Contract)-[]-(pe:ProcuringEntity)
	RETURN t as Tenderer, c as Contract, pe as ProcuringEntity
	}
	RETURN Tenderer, Contract, ProcuringEntity

Resultados Sist. No Distribuido	Started streaming 4 records after 220 ms and completed after 229 ms. Started streaming 4 records after 1 ms and completed after 4 ms. Started streaming 4 records after 1 ms and completed after 3 ms. Started streaming 4 records after 1 ms and completed after 2 ms.
Resultados Sist. Distribuido	Started streaming 14 records after 2 ms and completed after 4091 ms. Started streaming 14 records after 1 ms and completed after 942 ms. Started streaming 14 records after 2 ms and completed after 137 ms. Started streaming 14 records after 1 ms and completed after 35 ms.