BÚSQUEDA DE DOCUMENTOS BASADA EN EL USO DE ÍNDICES ONTOLÓGICOS CREADOS CON MAPREDUCE

DOCUMENT SEARCH SUPPORTED ON AN ONTOLOGICAL INDEXING SYSTEM CREATED WITH MAPREDUCE

Sonia Jaramillo Valbuena[1], Jorge Mario Londoño[2]

Fecha de recepción: 6 de mayo de 2014
Fecha de aprobación: 9 de septiembre de 2014

Referencia: S. Jaramillo Valbuena, J.M. Londoño. (2014). Búsqueda de documentos basada en el uso de índices ontológicos creados con MapReduce. Ciencia e Ingeniería Neogranadina, 24 (2), pp. 57 - 75

RESUMEN

Este artículo presenta un sistema de búsqueda soportado en un sistema de indexación ontológico. La técnica presentada utiliza emparejamiento de retículos. El proceso de emparejamiento se realiza entre el retículo podado con el espacio de búsqueda del corpus y el retículo con el espacio de búsqueda de la consulta. Dicho proceso permite realizar un filtrado con los documentos que deben presentarse al usuario. El sistema propuesto fue implementado utilizando el modelo de programación MapReduce. Los resultados experimentales reflejan la eficacia del sistema, al brindar al usuario una mayor correspondencia de los resultados con el dominio de búsqueda. Además, se evidencian mejoras en el rendimiento y mayor precisión en los resultados mostrados al usuario. La evaluación realizada se incluye al final del artículo.

Palabras Clave: Algoritmo apriori, MapReduce, ontología, emparejamiento, semántica.

ABSTRACT

This paper presents a search system supported on an ontological indexing system. The approach we present is based on lattice matching. The system performs a matching operation that takes as input a pruned lattice of ontological terms associated with the documents in a corpus and a

1. Ingeniero de Sistemas y Computación, M.Sc, Estudiante de Doctorado en Ingeniería UPB, Profesor asociado, Facultad de Ingeniería. Universidad del Quindío, Armenia, Colombia, sjaramillo@uniquindio.edu.co
2. Ingeniero Electrónico, PhD, Facultad de Ingeniería Informática y Telecomunicaciones. Universidad Pontificia Bolivariana, Medellín, Colombia, jorge.londono@upb.edu.co

query lattice. The matching process selects relevant documents and ranks them in accordance to their closeness to the user's query. We have implemented our approach on top of MapReduce. The experimental results support the efficacy of the system by providing users with a greater consistency among the results and the search domain. The results displayed to the users also show performance enhancements and improved accuracy. The test results are included at the end of the paper.

Keywords: Apriori algorithm, MapReduce, ontology, matching, semantics

INTRODUCCIÓN

El volumen de información que se maneja de forma sistematizada ha experimentado un crecimiento exponencial a lo largo del tiempo. Este rápido incremento ha llevado a reconsiderar los métodos actuales de búsqueda de información para lograr que permitan recuperar los documentos teniendo en cuenta su relevancia para el usuario. La técnica propuesta en este trabajo está motivada por dos problemáticas. La primera es que es posible que en algunas ocasiones se devuelvan una gran cantidad de resultados, algunos de los cuales no son significativos para el usuario (falsos positivos), y la segunda, que no se detecten otros resultados que deberían hacer parte de la respuesta (falsos negativos). Una propuesta para abordar esta problemática es hacer uso de ontologías para garantizar que los resultados arrojados realmente correspondan al dominio de búsqueda. Según Gruber [1], una ontología es una especificación formal de una conceptualización compartida.

La técnica expuesta en este artículo, denominada Algoritmo de Búsqueda Semántica Indexada basada en combinaciones de Conceptos Ontológicos (ABSICO), utiliza emparejamiento de retículos para presentar los documentos al usuario. ABSICO propone una metodología alternativa y una implementación eficiente que usa un conjunto de documentos anotados semánticamente para crear un retículo podado del espacio de búsqueda del corpus. En ABSICO, los términos de la consulta son buscados dentro del árbol n-ario del corpus y no documento a documento. Se define consulta como el conjunto de términos requeridos por el usuario.

Para efecto de las pruebas se han usado documentos pertenecientes al corpus CRAFT V0.9 con anotaciones de diferentes ontologías. Se han seleccionado solamente dos ontologías: Cell Type Ontology (CL) y Protein Ontology (PO). Cuando el usuario ingresa una consulta, también se construye un retículo a partir de los términos ingresados y este retículo se empareja con el del corpus, con el fin de identificar los documentos más cercanos a la consulta. ABSICO se apoya en MapReduce para simplificar la construcción de los retículos de índices inversos. En Dean et al. [2] se presenta a MapReduce como un modelo de programación que permite trabajar con enormes conjuntos de datos en paralelo. Este tipo de trabajos demanda mayor capacidad computacional y grandes cantidades de memoria. ABSICO ha sido aplicado al dominio de búsqueda biomédico. No obstante, la metodología utilizada es general y puede emplearse en otros contextos.

Este artículo se estructura de la siguiente forma: En la sección 1, se presentan los materiales y métodos utilizados para la construcción del sistema propuesto. En la sección 2, se hace una evaluación experimental y se muestran los resultados. Finalmente, en la sección 3 se dan las conclusiones.

1. MATERIALES Y MÉTODOS

1.1. REVISIÓN DE TRABAJOS RELACIONADOS

Existen diversas aproximaciones que hacen uso del conocimiento ontológico para recuperar información. En general, dichas propuestas extraen nuevos términos de la ontología para ampliar la consulta del usuario, haciendo uso de relaciones léxicas dentro de las cuales se destaca la sinonimia y términos relacionados (relaciones jerárquicas).

En Zou et al. [3] se presenta un marco teorico basado en ontologías para expansión semántica, que se apoya en una ontología de dominio, un algoritmo de anotación semántica y un algoritmo de razonamiento de expansión.

OntoSearch, sistema propuesto por Jiang et al. [4], construye una red semántica partiendo de conceptos semilla y aplica la teoría de difusión de activación de Anderson [5] para inferir la relevancia de los conceptos en el dominio ontológico respecto a la consulta dada. OntoSearch se apoya en documentos anotados semánticamente. Después de que el usuario ingresa una consulta, el sistema recupera una lista de documentos que contienen las palabras clave, y también, obtiene un conjunto de conceptos ontológicos asociados. Los términos ontológicos hallados dentro del corpus son utilizados como semillas para construir la red semántica. Mediante una aproximación heurística se determina qué tan fuertes son las relaciones semánticas en el dominio ontológico. Para calcular la medida de ranking, OntoSearch utiliza la medida de similitud de coseno.

ORank, descrito en Shamsfard et al. [6], utiliza un modelo conceptual basado en ontologías para anotar los documentos y realizar la expansión de la consulta. Recibe como entrada un grupo de documentos, sobre los cuales hace anotaciones semánticas y aplica técnicas conceptuales y estadísticas para construir, a partir del corpus, vectores de documentos que almacena en una base de datos. Cada vector de documento contiene etiquetas y pesos. Luego de que el usuario ingresa la consulta, ORank empieza un proceso de extracción de frases, que tiene como objetivo determinar si las mismas existen en la ontología o en las etiquetas de los documentos. Si esto ocurre, tanto palabras como frases son seleccionadas y se envían como entrada al algoritmo de difusión de activación. Para obtener mayor precisión se asignan pesos más altos a frases. Luego, la consulta se expande haciendo uso de hipónimos, sinónimos y de WordNet. En Fellbaum [7] se indica que WordNet es una base de datos léxica para el idioma inglés. Los pesos en la consulta pueden ser definidos por el usuario o computados por el sistema. ORank se mueve en la ontología, a la cual previamente le asignó pesos, para extraer los conceptos ontológicos relacionados y obtener los valores de activación. Finalmente, ORank convierte la consulta en un vector para compararla con los vectores de los documentos y poder calcular el grado de relevancia existente entre ambos. ORank propone como mejora utilizar las propiedades de las clases ontológicas para enriquecer el proceso de expansión de la consulta.

En Wang et al. [8] se describe un sistema para la recuperación de objetos de aprendizaje,

que tiene dos importantes características: un procedimiento para remoción de ambigüedad y un algoritmo para expansión de consulta basado en ontologías, completamente automático, que permite inferir y agregar la intención del usuario. Para resolver la consulta, dicho sistema sigue 6 pasos. En primer lugar, efectúa una etapa de preprocesamiento, durante la cual elimina palabras vacías y obtiene las raíces de los términos (stemming). A continuación, determina la coincidencia de conceptos y evaluación de pesos, para lo cual se proponen medidas tales como puntaje del concepto base, del concepto base pesado y del concepto base propagado. El tercer paso corresponde a la construcción del árbol de intención del usuario (puede ser uno o varios), que corresponde a un subárbol de la ontología. Posteriormente, se efectúa la fase de remoción de ambigüedad, para lo cual se selecciona el árbol de intención con mayor puntaje. Luego se lleva a cabo la expansión de los términos de la consulta; aquí se tienen en cuenta correlaciones semánticas y se calcula la distancia semántica. Finalmente, se apunta a los metadatos de los objetos de aprendizaje.

En Velardi et al. [9] se plantea la utilización de ontologías para capturar el dominio semántico de una palabra. La expansión de la consulta se apoya en las definiciones de los términos consultados. La consulta es representada mediante una red semántica.

Por su parte, Song et al. [10] propone un modelo híbrido, denominado SemanQE, que combina ontologías, reglas de asociación y procesamiento de lenguaje natural para la expansión. Dicha técnica hace uso de propiedades contextuales de términos descubiertos mediante reglas de asociación, propiedades lingüísticas de corpus de texto no estructurado y ontologías (utiliza WordNet para el proceso de desambiguación). En SemanQE, la ampliación de la consulta es realizada por un motor de expansión basado en reglas de asociación.

En Ranwez et al. [11] se muestra un sistema de recuperación de información que considera relaciones de proximidad semántica entre conceptos de la ontología y modelos de agregación. Además, ofrece una representación gráfica de los resultados de la consulta para permitir interacciones con el usuario y favorecer la exploración de los mismos. El sistema permite navegar dentro de la jerarquía de conceptos ontológicos para apoyar la formulación de la consulta.

Por último, se tiene la aproximación presente en Alipanah et al. [12], que se desarrolla sobre una arquitectura federada haciendo uso de MapReduce. El modelo descrito realiza varios pasos. Primero se determinan los Términos de expansión Básica (BET) en cada ontología haciendo uso de MapReduce. Luego, se efectúa una alineación ontológica para determinar Nuevos Términos de Expansión (NET) basados en entidades similares entre cada par de ontologías. NET es determinada mediante alineación ontológica. Finalmente, mediante MapReduce se realiza el cálculo de las rutas más cortas en cada ontología individual. Estos resultados son usados para obtener los pesos de rutas de similitud semántica y medidas de densidad. Lo anterior permite elegir los k-términos relacionados a través de la arquitectura federada. En general, se obtiene una lista de términos alternativos a la consulta del usuario desde diferentes ontologías.

El sistema propuesto en este artículo, ABSICO, hace uso de ontologías para afrontar dos importantes problemáticas en lo referente a recuperación de información: la primera,

devolver resultados que no son significativos para el usuario, y la segunda, no hallar resultados que sí deberían mostrarse. La aproximación presentada utiliza emparejamiento de retículos para listar los documentos al usuario. Luego de que el usuario ingresa las palabras que desea, a la consulta se le asocian términos ontológicos y posteriormente se construye un retículo con el espacio de búsqueda de la consulta. Este retículo se compara con el retículo del corpus, el cual se generó a partir de todos los árboles que surgen de cada uno de los documentos que hacen parte del corpus. Las relaciones ontológicas existentes entre los subárboles se extraen directamente de las ontologías. Para la construcción de ambos retículos, ABSICO realiza combinaciones de términos ontológicos con el objetivo de proporcionar mayor precisión en los resultados, y por ende, reducir la ambigüedad al prestar importancia al contexto. A menos que existan cambios en el corpus, estos subárboles no necesitan ser recalculados.

En ABSICO los términos de la consulta no se buscan documento a documento, sino dentro del árbol n-ario del corpus, logrando de esta forma mejoras en el rendimiento. Luego de que se hace el proceso de emparejamiento entre ambos retículos, el sistema se apoya en los retículos de índices inversos para obtener los documentos relacionados semánticamente. MapReduce se utiliza para paralelizar y hacer eficiente la construcción de los índices semánticos de grandes volúmenes de datos. ABSICO se vale de un proceso de doble ranqueo para garantizar que los resultados de la consulta correspondan más con la pertinencia de los documentos. El ranking se efectúa por niveles (ver sec. 3.3).

ABSICO hace uso de algunas de las propiedades de las clases ontológicas, en especial las propiedades que permite obtener los sinónimos y las que reflejan las relaciones jerárquicas. Para su funcionamiento el sistema usa documentos con anotaciones semánticas. Finalmente, es de anotar que ABSICO no requiere la intervención del usuario para resolver la consulta.

1.2. METODOLOGÍA

La metodología seguida en esta investigación contempla en primer lugar definir la estructura del sistema de búsqueda. Dicha estructura se compone por tres módulos fundamentales, a saber: el módulo de procesamiento de documentos, el módulo de procesamiento de consultas y el módulo de emparejamiento–ranking, ver Figura 1.

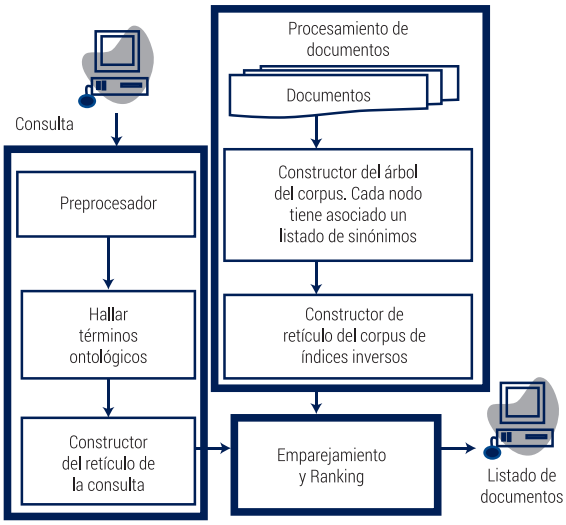

Figura 1. Arquitectura general del modelo propuesto.

Dado que el sistema propuesto está soportado en un sistema de indexación ontológico, se hace necesario seleccionar previamente algunas ontologías. Se eligen Cell Type Ontology (CL) y Protein Ontology (PO), ambas pertenecientes al corpus CRAFT.

1.2.1. El módulo de procesamiento de documentos

Este módulo es el encargado de generar 3 importantes estructuras, a saber: el árbol del corpus con todos los términos ontológicos encontrados y sus relaciones jerárquicas, un listado de sinónimos oficiales para cada uno de los términos ontológicos y un retículo de índices inversos. El módulo de procesamiento de documentos para su funcionamiento recibe documentos anotados semánticamente. Cada anotación tiene la forma <term sem= "NombreOntologia:idTermino"> termino</ term> , por ejemplo, la etiqueta <term sem="CL:0000540">neuronal</term> indica que el término neuronal pertenece a la ontología CL y dentro de esta ontología tiene el código 0000540.

Para extraer los términos de cada uno de los documentos se obtienen cada una de las etiquetas y se saca de ellas el término ontológico (que puede ser una palabra o frase), el identificador y la ontología asociada. Posteriormente, se extrae de la ontología el subárbol cuyas hojas son los términos encontrados en el documento, que permite evidenciar las relaciones jerárquicas entre los términos ontológicos, ver Figura 2.

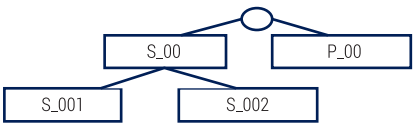

Figura 2. Subárbol con las relaciones entre todos los términos ontológicos de un documento. Cada nodo es un término ontológico.

Las hojas del subárbol corresponden a los términos hallados dentro del documento. Los nodos internos (que pueden o no estar presentes en el documento) son los padres de los términos que están en el nivel inmediatamente inferior. A cada término perteneciente al subárbol se le asigna un listado de sinónimos oficiales, que se extrae directamente de la ontología. Este listado es utilizado durante el preprocesamiento de consultas. Con todos los subárboles obtenidos de los documentos se crea un árbol global del corpus, que es almacenado por el sistema.

A partir del árbol global del corpus se construye un retículo podado con el espacio de búsqueda del corpus. Cada nodo del retículo está formado por uno o más términos ontológicos (dependiendo de la profundidad). Para la generación de los nodos del retículo ontológico se utiliza un método inspirado en el Algoritmo Apriori, Agrawal et al. [13]. El sistema propuesto requiere que la versión original de este algoritmo sea modificada. Lo anterior obedece a que en Apriori la cantidad total de nodos generados crece exponencialmente de acuerdo al número de términos identificados, generándose en total 2n-1 nodos (siendo n la cantidad de diferentes términos ontológicos que hay en el corpus), razón por la cual se hace necesario podar el retículo.

Por ello, se combina Apriori con Eclat, este último descrito en Zaki [14] permite descomponer el espacio de búsqueda usando el concepto de prefijos basado en clases de equivalencia, donde los miembros de una relación comparten el mismo prefijo k. Por ejemplo si se debe generar el conjunto C={ab, ad, bc, bd, cd}, las clases de equivalencia serían [a] ={b, d},[b] = {c, d}, [c]={d}. A partir de cada clase de equivalencia se originan los nuevos nodos al ser descompuesto recursivamente, Ceglar et al. [15]. No se generan nodos que no tengan documentos relacionados

semánticamente. Los nodos con (*) no fueron creados por no existir documentos que cayeran en dichas categorías. El retículo podado con el espacio de búsqueda del corpus se muestra en la Figura 3.

El generador del índice inverso toma como entrada un conjunto de archivos. Cada uno de ellos hace referencia a un documento específico del corpus y contiene los términos ontológicos existentes dentro del mismo, como se ilustra en la Figura 4a.

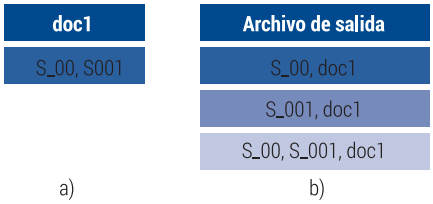

Figura 4. En (a) se muestra la relación de los documentos con los términos ontológicos que utiliza el generador de índice inverso. En (b) se ilustra el índice inverso después de procesarse con MapReduce.

1.2.2. El procesamiento de la consulta

El preprocesamiento de la consulta se lleva a cabo en dos etapas. En la primera, el preprocesador se encarga de eliminar las palabras vacías (stopwords), tales como artículos, preposiciones y pronombres. También, en caso de ser necesario, detecta posibles errores de digitación por parte del usuario mediante la aplicación del concepto de similitud entre cadenas. Existen diversas técnicas para realizar emparejamiento entre cadenas, entre ellas Levenshtein [16], Jaro, Jaro-Winkler, Smith-Waterman, Monge-Elkan y N-Gramas. De estas aproximaciones, una de las más usadas es la distancia de Levenshtein. Levenshtein permite hallar la cantidad mínima de operaciones (inserción,

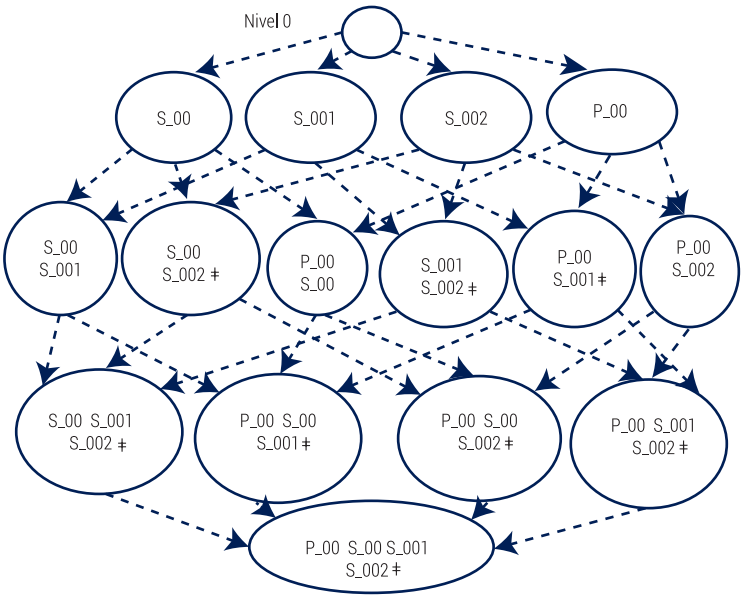

Figura 3. Espacio de búsqueda del corpus.

eliminación o sustitución) necesarias para convertir una cadena de caracteres en otra. Es decir, se pretende lograr que las palabras digitadas por el usuario que no correspondan a términos existentes en la ontología, se puedan relacionar con el término más cercano acorde con una medida de similitud. Nuestra implementación hace uso del algoritmo de Levenshtein, principalmente por su sencillez, facilidad de implementación (haciendo uso de programación dinámica) y por permitir comparar cadenas de diferente tamaño. No se trata de un módulo fundamental de nuestro sistema y se deja abierta la posibilidad de implementar otras técnicas.

Durante la segunda etapa, se asocian los términos ontológicos a cada uno de los criterios de búsqueda que quedaron después del preprocesamiento. Esto se lleva a cabo mediante un recorrido por niveles del árbol del corpus, de abajo hacia arriba. Por ejemplo, el usuario ingresa las siguientes palabras: "cell dna cycle" y el procesador devuelve que cell está asociado con S_00, dna con P_00 y cell cycle con S_002. En el caso de ingresar algún sinónimo, el sistema debe retornar el término oficial correspondiente al mismo. Si el usuario introduce un término que no existe dentro del corpus, se debe devolver el término padre más cercano dentro de la relación ontológica. Si al finalizar la etapa de procesamiento de consulta, ésta no tiene ningún término ontológico asociado el resultado sería la búsqueda vacía. Con los términos obtenidos en el paso anterior se genera un retículo con el espacio de búsqueda de la consulta, tal como se ilustra en la Figura 5.

1.2.3. Emparejamiento-Ranking

Este módulo es el encargado de realizar el proceso de emparejamiento entre el retículo

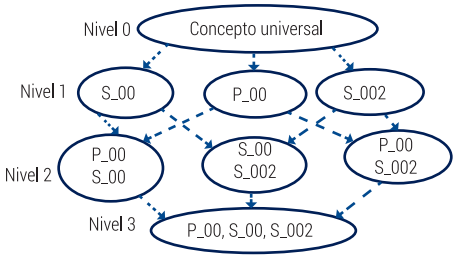

Figura 5. Espacio de búsqueda de la consulta.

de espacio de búsqueda (corpus) y el retículo de la consulta. Para realizar el proceso de emparejamiento se recorre el retículo de la consulta de abajo hacia arriba (por niveles) y se determina si cada nodo de la consulta equivale exactamente con uno dentro del retículo del corpus. Para que se dé la equivalencia exacta ambos nodos deben tener la misma etiqueta. Una etiqueta se define como el vector de todos los términos ontológicos que pertenecen al nodo. En este vector los elementos están ordenados lexicográficamente.

El pseudocódigo del algoritmo de emparejamiento se presenta a continuación. En primer lugar, se muestra la función emparejar, ver Algoritmo 1. Éste se encarga de realizar el proceso de emparejamiento entre el retículo de la consulta y el del corpus. Devuelve el retículo resultado del emparejamiento.

La función emparejarNivel, ver Algoritmo 2, recibe el nivel i de la consulta y el nivel i del corpus y devuelve un vector que contiene los nodos que coinciden en ambos niveles.

El método verificarExisteEtiquetaEnCorpus, que se muestra en el Algoritmo 3, recibe una etiqueta e informa si la misma existe en el nivel del corpus analizado.

El método verificarIgualdadEtiquetas, ver Algoritmo 4 4, recibe la etiqueta de un nodo

Algoritmo 1. Función emparejar.

```
Función emparejar(reticuloConsulta: Retículo de la consulta, reticuloCorpus: Retículo del corpus )
Inicio
/*reticuloEmparejamiento es el retículo resultante del proceso de emparejamiento*/
para i=maximoNivelDeReticuloConsulta hasta 1 con decremento 1 hacer
   /*se recorre el retículo de abajo hacia arriba*/
   nivelConsulta= Vector de nodos del nivel i de reticuloConsulta;
   nivelCorpus= Vector de nodos del nivel i de reticuloCorpus,
   /*emparejarNivelI  recibe el nivel i de la consulta y el nivel i  del corpus*/
   vectorEmparejamientoNivel= emparejarNivelI(nivelConsulta, nivelCorpus, i) ;
   Si vectorEmparejamientoNivel!=null entonces
        agregarNuevoNivelAlReticuloEmparejamiento(vectorEmparejamientoNivel)
   Fin Si
Fin para
retornar reticuloEmparejamiento;
Fin Función emparejar
```

Algoritmo 2. Función emparejar nivel.

```
Funcion emparejarNivel (nivelConsulta: Vector de nodos correspondientes a un nivel de la consulta,  nivelCorpus:
Vector de nodos correspondientes a un nivel del corpus, nivel: entero )
Inicio
para j=1 hasta cantidad de elementos de nivelConsulta con incremento 1 hacer
   /*terminoQ contiene la  etiqueta de un nodo. La etiqueta puede estar formada por uno o más términos ontológicos*/
   terminoQ= nivelConsulta [j];
   res= verificarExisteEtiquetaEnCorpus(terminoQ, nivelCorpus, nivel)
   Si res!=null entonces
        agregarAVectorEmparejamientoNivel(res)
   Fin Si
Fin para
retornar vectorEmparejamientoNivel
Fin función emparejarNivel
```

Algoritmo 3. Función para verificar si una etiqueta existe en el corpus.

```
Función  verificarExisteEtiquetaEnCorpus (terminoQ: Vector, nivelCorpus: Vector de nodos correspondientes a un nivel del
corpus)
Inicio
para k=1 hasta cantidad de elementos de nivelCorpus con incremento 1 Hacer
   Si verificarIgualdadEtiquetas (terminoQ, nivelCorpus [k].obtenerEtiqueta())  entonces
        /*Si las etiquetas son iguales se devuelve el nodo del nivel del corpus que coincide con el términoQ. */
        retornar nivelCorpus [k]
    Fin Si
Fin para
retornar null;
Fin función verificarExisteEtiquetaEnCorpus
```

Algoritmo 4. Función para verificar si dos etiquetas son iguales.

```
Funcion verificarIgualdadEtiquetas (terminoQ: Vector, terminoOntologicoCorpus: Vector)
Inicio
centinela2=true
/* términoQ y terminoOntologicoCorpus están ordenados lexicográficamente. */
para i=0  hasta i<cantidad de elementos de terminoQ con incremento 1 hacer
    Si terminoQ [i]!= terminoOntologicoCorpus [i] entonces
            centinela2=false;   retornar centinela2;
    Fin Si
Fin para i
retornar  centinela2;
Fin Funcion verificarIgualdadEtiquetas
```

del retículo de la consulta y la compara con la etiqueta de un nodo del retículo del corpus. Se devuelve true en caso de que sean iguales.
La operación del algoritmo de matching y ranking se ilustra en la Figura 6. El retículo de la consulta, ver Figura 5, y el retículo podado del corpus, ver Figura 6a, son los argumentos de la función emparejar.

El proceso de emparejamiento se realiza nivel a nivel, iniciando en el nivel inferior del retículo de la consulta y asciende hasta llegar al nivel 1. La función emparejarNivel recibe todas etiquetas pertenecientes al nivel i de la consulta y una a una, a través de la función verificarExisteEtiquetaEnCorpus, las compara con las etiquetas de los nodos del retículo de corpus pertenecientes al mismo nivel.

La función verificarIgualdadEtiquetas es la encargada de determinar si las etiquetas de dos nodos cumplen con el criterio de igualdad. Para que exista igualdad ambas etiquetas deben tener los mismos componentes en el mismo orden. Si se determina que existe correspondencia exacta, se examinan los documentos pertenecientes al nodo del corpus. Es de anotar que si otro nodo apunta hacia un documento que ya fue examinado, ese documento no se agrega múltiples veces al resultado de la consulta.

Los nodos presentes en un nivel inferior que coincidan con los criterios de búsqueda tendrán mayor relevancia semántica, esto se traduce en un mayor peso dentro del proceso de ranking. En el ejemplo ilustrado en la Figura 6 se buscaban los términos S_00, S_002 y P_00, por lo tanto, los nodos seleccionados para listar por nivel serían:

Nivel 4 del corpus: Ninguno (el nodo contiene 4 términos ontológicos y sólo se requieren 3)
Nivel 3: Ninguno (aunque está el nodo S_00

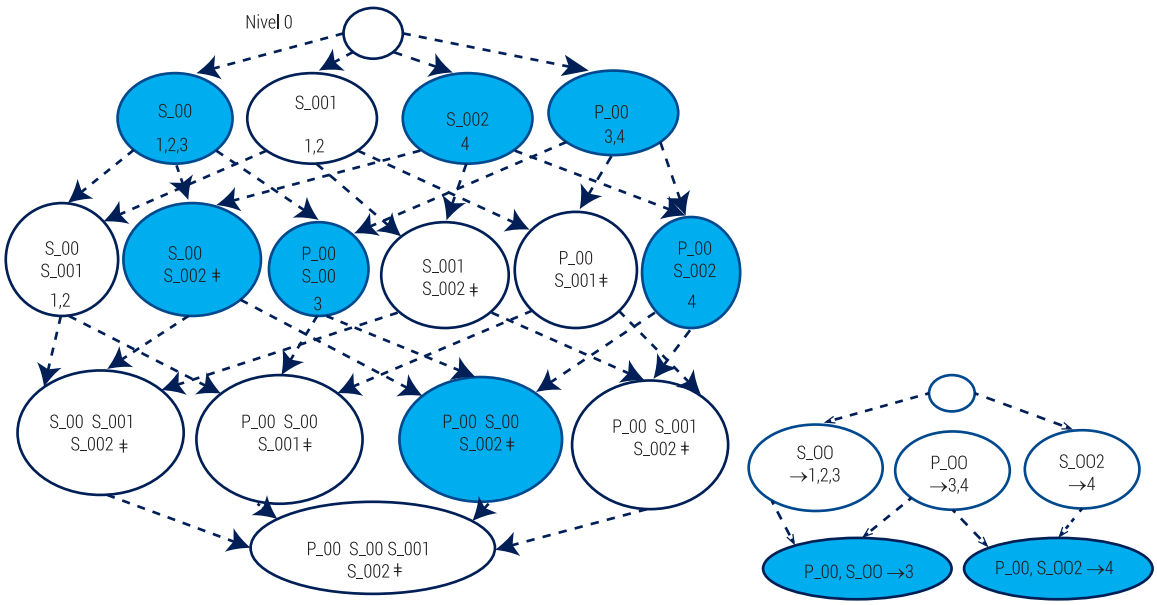

Figura 6. En a. Retículo del corpus, en él se encuentran sombreados los nodos que tienen correspondencia exacta con la consulta. En b. Resultado de emparejar el retículo de la consulta (Figura 5) con el del corpus (6a).

S_002 P_00, éste no tiene documentos asociados)

Nivel 2: El nodo S_00 P_00 está asociado con doc3 y el nodo P_00 S_002 asociado con doc4
Nivel 1: El nodo S_00 está asociado con doc1 y doc2 (doc3 ya fue seleccionado), del nodo P_00 no se lista ningún documento (estos ya fueron seleccionados), tampoco del nodo S_002.

Además del ranking establecido por el nivel donde se logra la correspondencia con el documento, se hace necesario definir un ranking dentro de cada uno de los niveles. Este ranking intranivel requiere que se calcule la similitud entre cada uno de los documentos y la consulta. Lo anterior se realiza midiendo la distancia existente entre los vectores que los representan. Para construir el vector correspondiente a un documento se requiere calcular por cada término ontológico buscado un peso (equivalente a la cantidad de veces que el término ocurre en el documento). Si un término no fue hallado exactamente, sino que se le asoció el término padre o cualquier otro ancestro, el peso asociado es 1/(di+1), siendo di>la cantidad de aristas existentes entre el término ontológico buscado y ancestro asignado. Esto implica que cuando se logra una coincidencia exacta se tiene un ranking mayor que cuando se utilizan los nodos padre.

El sistema de recuperación de información propuesto se basa en el Modelo de Espacio Vectorial, por lo cual para hallar la distancia entre el vector del documento y el de la consulta se aplica la distancia del coseno, que se describe en Porta [17]. En la distancia de coseno el documento se representa mediante un vector de pesos e igual ocurre con la consulta. La distancia de coseno se utiliza para medir el ángulo entre dos vectores en el espacio vectorial. La distancia angular o del coseno es ampliamente reconocida en la literatura como una métrica de la similitud entre documentos [18-19]. El resultado de este cálculo arroja valores entre cero y uno, donde 1 indica que hay mayor cercanía entre la consulta y el documento. En este caso, vectorDoc es el vector correspondiente a cada documento y vectorQ es el vector de la consulta.

La ecuación 1 realiza el cálculo de similitud de coseno, donde p es el vector del documento, q el de la consulta, pi es el peso del término i dentro del documento y qi es el peso del término i de la consulta.

$$\text{SimilaridadCos}\,(p,q) = \frac{\sum_{i=1}^{n}(p_i \cdot q_i)}{\sqrt{\sum_{i=1}^{n} p_i^2} \cdot \sqrt{\sum_{i=1}^{n} q_i^2}} \quad (1)$$

El algoritmo de ranking intranivel se muestra en el Algoritmo 5. En primer lugar se presenta la función ranking, ésta recibe un grafo acíclico dirigido (DAG) y lo recorre por niveles, de abajo hacia arriba. Por cada nivel calcula un ranking. Al final retorna el listado total de documentos que se le mostrarán al usuario.

La función calcularRankingNivel, mostrada en el Algoritmo 6, es la encargada de realizar el ranking intranivel. Retorna el listado de documentos del nivel examinado que deben presentarse al usuario. El grafo acíclico dirigido (DAG) resultante del proceso del emparejamiento se envía como argumento a la función ranking. El proceso de ranking se realiza nivel a nivel, iniciando en el nivel inferior del DAG y ascendiendo hasta llegar al nivel 1. Para cada uno de los nodos del nivel analizado se obtienen los documentos relacionados semánticamente y a cada uno de ellos se le

Algoritmo 5. Función que calcula el ranking.

```
Función ranking(grafo: DAG resultado del emparejamiento)
Inicio
/*arregloGlobal es un vector donde se almacenan todos los documentos que se van a listar al usuario (por orden de ranking)*/
para i=obtenerCantidadNiveles(grafo) hasta 1 con decremento 1 hacer
   /*se recorre el DAG nivel a nivel, de abajo hacia arriba*/ arregloEmparejamientoNivel=arreglo de nodos del nivel i del grafo,
     arregloDocumentosGlobal es utilizado para verificar que cada documento solo se muestre en el resultado una sola vez*/
   arregloAuxiliarDocumentosNivel=calcularRankingNivel (arregloEmparejamientoNivel,
arregloDocumentosGlobal)
   Si arregloAuxiliarDocumentosNivel!=null
      /* agregarAlArregloGlobal recibe un arreglo con los documentos existentes en el ArregloAuxiliarDocumentosNivel y los
        agrega uno a uno conservando el orden*/
        agregarAlArregloGlobal (arregloAuxiliarDocumentosNivel)
   Fin Si
Fin para
devolver arregloGlobal
Fin Función Ranking
```

Algoritmo 6. Función que calcula el ranking intranivel.

```
Funcion calcularRankingNivel (arregloEmparejamientoNivel: vector de nodos , arregloDocumentosGlobal:
vector de documentos)
Inicio
para j=1 hasta cantidad de nodos de arregloEmparejamientoNivel con incremento 1 hacer
   Vector aux= obtenerDocumentosNodo(arregloEmparejamientoNivel[j])
   para k=1 hasta cantidad de documentos almacenados en aux con incremento 1 hacer
      Si estaContenidoEnArregloDocumentosGlobal (aux[k])==false entonces
         puntaje=calcularSimilitudCos (aux[k])
         asignarPuntaje(aux[k])
         agregarAlArregloAuxiliarDocumentosNivel (aux[k])
         agregarAlArregloDocumentosGlobal(aux[k])
      Fin Si
   Fin para
ordenarPorPuntaje(ArregloAuxiliarDocumentosNivel)
devolver ArregloAuxiliarDocumentosNivel
Fin función calcularRankingNivel
```

calcula la similitud existente con la consulta mediante la función calcularSimilitudCos. Esta función hace uso de la ecuación definida en la Figura 6. Finalmente, se ordenan los documentos del nivel analizado de mayor a menor grado de similitud.

1.3. IMPLEMENTACIÓN DEL SISTEMA DE BÚSQUEDA

El prototipo fue desarrollado en Java. Se utilizaron las bibliotecas Hadoop, Jena y Lucene Analyzer de Apache Software Foundation, las cuales se describen respectivamente en [20-22]. La librería Hadoop es una implementación abierta de MapReduce, soporta la creación de funciones de mapeo y reducción, además de permitir enviar trabajos y monitorear la ejecución de los mismos. Jena provee una serie de herramientas para construir aplicaciones de la web semántica y posibilita la realización de consultas a ontologías mediante SPARQL (Protocolo SPARQL y Lenguaje de consultas RDF).

Es de anotar que para evaluar el sistema se utilizaron las ontologías pertenecientes al

corpus CRAFT presentado en Bada et al. [23], todas ellas con extensión OWL (Lenguaje de Ontologías Web, W3C [24]). Jena proporcionó el soporte para dicho lenguaje. Fue usado para el proceso de carga y navegación del modelo ontológico, obtener los nodos padres de los términos ontológicos y las propiedades declaradas de las clases ontológicas (principalmente los sinónimos). El proceso de carga e indexación de las ontologías se realiza una vez al inicio del proceso (una simplificación del prototipo experimental), a partir de allí los retículos e índices semánticos del corpus se mantienen en la memoria para ser reutilizados en cada consulta. Por lo anterior, no se utilizó el Lenguaje de Consultas sobre Datos RDF (RDQL) ni SPARQL, puesto que las consultas se efectuaban mediante Java. Lucene Analyzer se utilizó para hallar las raíces de los términos, Lucene Analyzer contiene la clase EnglishStemmer que implementa la versión inglesa del Algoritmo de Porter, presente en Rijsbergen et al. [25]. Este algoritmo incluye un conjunto de reglas mediante las cuales es posible eliminar terminaciones de palabras con una raíz común y que, por ende, pueden ser tratadas como un sólo término.

El árbol del corpus contiene todos los términos ontológicos hallados dentro del corpus y los ancestros de cada uno de ellos. Este árbol es el punto de partida para la generación del retículo del corpus. Este retículo no se genera todo al mismo tiempo, inicialmente se crean los nodos del primer nivel. Luego en tiempo de ejecución, después de que el usuario ingresa los términos de la consulta y de que a estos se les asocian términos ontológicos, se agregan los nodos de los demás niveles, es decir, estos niveles se crean siguiendo un enfoque perezoso.

La generación del índice inverso es realizada mediante MapReduce. Para la construcción del índice, MapReduce toma como entrada un conjunto de archivos, cada uno de los cuales contiene el nombre del documento y los términos ontológicos relacionados con el mismo. El procesamiento hecho por MapReduce sobre estos archivos es realizado principalmente en 2 fases: Map y Reduce. Cada trabajador Map lee línea por línea el contenido del bloque de entrada y a partir de allí genera parejas de tipo clave-valor, la clave corresponde a un documento y el valor a la etiqueta del nodo del retículo asociado a dicho documento. Para cada una de estas parejas se llama a la función Map, esto produce un listado de parejas intermedias clave-valor, que es almacenado en el sistema de archivos distribuidos (HDFS). El trabajador Reduce itera sobre los datos intermedios clasificados y, para cada única clave intermedia encontrada, pasa la clave y el correspondiente conjunto de valores intermedios a la función Reduce del usuario. La salida de la función Reduce se añade al archivo final de salida. El procesamiento realizado en MapReduce da como resultado un índice inverso que mapea cada nodo con un grupo de documentos relacionados semánticamente, ello con el objetivo de hacer eficientes las búsquedas dentro de esta red.

Sobre un corpus estático, el proceso de indexación solo necesita ejecutarse una vez. Si se agregan documentos al corpus es posible actualizar incrementalmente los índices con los nuevos documentos, aunque la implementación de esta funcionalidad se deja para futuros trabajos.

2. RESULTADOS Y ANÁLISIS

Para la evaluación del sistema se realizaron 18 consultas, cada una de ellas con términos pertenecientes a las ontologías del corpus CRAFT. Dicho corpus, que consta de 67

artículos, es considerado uno de los más grandes etiquetados con el estándar de oro. En estas ontologías cada término tiene como mínimo un identificador y una etiqueta. En algunas ocasiones posee una descripción, sinónimos relacionados y relaciones de jerarquía. Las clases de interés para realizar la evaluación fueron las mostradas en la Tabla 1.

Para obtener resultados con mayor validez estadística, para cada uno de estos identificadores se generaron 5 consultas. Cada una de estas no necesariamente es igual a la etiqueta correspondiente al identificador, obteniendo un total de 90 mediciones. Se tuvieron en cuenta algunas variantes tales como singular – plural, variaciones en el orden de los términos, raíces y sinónimos oficiales. Las consultas se cargan mediante un archivo de texto con el fin de automatizar el proceso de evaluación.

La evaluación del sistema se efectuó sobre un computador con procesador Intel Core i5 de 2.5GHz, 8 GB memoria RAM y sistema operativo Linux Ubuntu. Todas las consultas se llevaron a cabo sobre este mismo hardware y se tuvo la precaución de verificar que no estuvieran otros procesos ejecutándose.

La evaluación consistió en medir en ABSICO la eficiencia en tiempos de respuesta de cada consulta. Estos resultados se compararon con los tiempos de respuesta obtenidos con el algoritmo de línea de base (baseline) no indexado. Este último corresponde al enfoque en el que se ejecuta una búsqueda exhaustiva documento a documento. Los resultados de la evaluación realizada se muestran en la Figura 1.

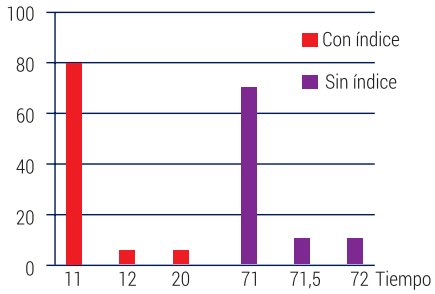

Figura 1. Histograma de los tiempos de respuesta.

En promedio, con ABSICO el tiempo de búsqueda es de 10,86 segundos, la mediana es de 10,3 segundos, la varianza de 4.8 y la desviación estándar de 2,19. En el segundo caso, con el algoritmo de línea de base no indexado, el tiempo es de 71,05 segundos, la mediana es de 71, la varianza de 0.12 y

Tabla 1. Clases de interés para la evaluación.

Id	Etiqueta exacta dentro de la ontología	Id	Etiqueta exacta	Id	Etiqueta exacta	Id	Etiqueta exacta
SO_0000452	thyroid hormone secreting cell	CL_0000015	male germ cell	CL_0000187	muscle cell	CL_0000287	eye photoreceptor cell
SO_0000476	thyroid stimulating hormone secreting cell	CL_0000589	Inner hair cell	CL_0000561	amacrine neuron	SO_0001025	Polymorphic sequence_variant
CL_0000457	biogenic amine secreting cell	CL_0000216	Sertoli cell	CL_0000636	Muller cell	CL_0000125	glial cell
SO_0000061	restriction_enzyme_ binding_site	CL_0000228	Multinucleate cell	CL_0000125	glial cell		
CL_0000066	epithelial cell	CL_0000445	apoptosis fated cell	CL_0000178	Leydig cell		

la desviación estándar de 0,4. Se resalta que en ambos casos los documentos obtenidos fueron los mismos.

Para determinar si las diferencias halladas con ambas técnicas eran significativas estadísticamente se quiso aplicar análisis de varianza (ANOVA). Sin embargo, al realizar la prueba de prueba Bowman – Shelton para ver si los datos se ajustaban a una distribución normal se concluyó que no cumplían dicho supuesto. De igual forma, se consideró usar la prueba no paramétrica de Kruskal-Wallis, que no requiere supuesto de normalidad, pero ésta solo es aplicable cuando se tienen más de dos poblaciones. Finalmente se aplica el test no paramétrico de las medianas, obteniendo un p <= 6.08e-40. Como el valor de p es menor que .05 se rechaza la hipótesis nula y se concluye que hay evidencia estadística de que el rendimiento en ABSICO es mejor.

Adicionalmente, para evaluar la calidad de los resultados se utilizó FileSeek de Binary Fortress Software [26]. Esta herramienta permitió obtener los archivos que contenían el identificador del término buscado. Las consultas realizadas se basaron únicamente en coincidencia entre los términos ingresados por el usuario y la etiqueta exacta dentro de la ontología de las clases de interés. El listado proporcionado por FileSeek sirvió de base para determinar si los documentos retornados por el modelo propuesto contenían la clase de oro buscada, y por lo tanto podían considerarse como una respuesta correcta.

Basado en este supuesto, se calculó la precisión promedio del sistema, haciendo uso de la fórmula presente en la Ecuación 2. La precisión obtenida fue de 0.89. Este valor se obtiene directamente de los resultados hallados con FileSeek. En Frakes et al. [27] se describe la precisión como la probabilidad de que un documento recuperado sea relevante. Adicionalmente, se calculó la exhaustividad y la medida F. La exhaustividad se define como la proporción de documentos relevantes que se recuperan realmente y la medida F, como una media ponderada y armónica en la que se integran las medidas de precisión y exhaustividad [27]. Generalmente, la exhaustividad y la precisión se relacionan de forma inversa, puesto que a mayor especificidad en los téminos ingresados en la consulta del usuario, los resultados serán más precisos, y esto hará que se dejen de obtener documentos debido a ese alto nivel de especificación.

Sin embargo, en ABSICO esta premisa no se cumple, ya que la exhaustividad obtenida es de 1.0, esto indica que el sistema asegura que siempre se retornen todos los documentos relevantes. Lo anterior se debe a que cuando el usuario ingresa la consulta, ésta es transformada en un retículo dentro del cual se contemplan todos los términos ontológicos asociados con la consulta y las combinaciones de los mismos. Al no buscar los términos de la consulta documento a documento, sino dentro del retículo del corpus (que ya tiene los documentos relacionados semánticamente) se garantiza que si se requiere una combinación de términos ontológicos y ésta se encuentra presente dentro de árbol n-ario del corpus, el proceso de emparejamiento siempre informará sobre su existencia. Al tomar en consideración para la construcción de los retículos las relaciones ontológicas y de sinonimia se amplían las posibilidades de búsqueda, y al usar combinaciones de términos ontológicos se eleva la precisión de los resultados, haciendo que todos los documentos recuperados correspondan más con la pertinencia requerida por el usuario. Finalmente, la medida-F fue

de 0.9. Es de anotar que cuando el sistema retornaba otras clases del estándar de oro, se requirió realizar una evaluación manual, documento a documento, para determinar si la respuesta era o no correcta.

La ecuación 2 permite determinar el valor de la precisión , la 3 permite hallar exhaustividad y la 4 se utiliza para determinar la medida F.

$$P = \frac{\#\ docRelevantesRecuperados}{\#totalDocRecuperados} \quad (2)$$

$$E = \frac{\#docRelevantesRecuperados}{\#totalDeDocRelevantes} \quad (3)$$

$$F = 2 * \frac{P * E}{P + E} \quad (4)$$

Es de destacar que la parte de la indexación es la más beneficiada por el uso de MapReduce. Para analizar el desempeño del sistema en este aspecto se realizó una simulación con el software Matlab de Mathworks [28] que evaluó el tiempo que tarda la generación del índice inverso en función del número de Mappers que participan en su construcción, ver Figura 2.

Durante la simulación se generaron en total 10 000 documentos. La cantidad de reductores se calculó con base en el tamaño de la distribución de datos de las salidas de los mappers [29-30]. Los resultados obtenidos indican que para este corpus el buscador es eficiente con 3 niveles.

En lo referente a complejidad computacional del componente de procesamiento de documentos, el cual se basa en MapReduce, se utiliza la metodología descrita en Goel et al. [31], se calculan dos medidas: clave y secuencial. Ambas medidas dependen exclusivamente del algoritmo, pero omiten detalles de instalación tales como número de mappers y reducers y cantidad de máquinas. La complejidad clave examina tres aspectos: el tamaño máximo de una pareja de entrada Clave-Valor, el tiempo de ejecución máximo para un Mapper/Reducer aplicado a una pareja Clave-Valor y la memoria máxima utilizada por un Mapper/Reducer para procesar una pareja Clave-Valor. En el caso del índice inverso descrito se asumen N documentos, M términos ontológicos, un tamaño total de documento S y unas frecuencias de términos ontológicos f1, f2,..., fM. Tanto tamaño, tiempo y memoria son todos O (fMAX) siendo fMAX = maxi fi. Por su parte, la complejidad secuencial captura el total de recursos consumidos por el sistema. Los aspectos evaluados en este caso son: el tamaño de todas las parejas Clave-Valor de entrada y salida por los los mappers y los reducers y el tiempo total de ejecución para todos los mappers y reducers. En el caso de la complejidad secuencial, el tamaño total y el tiempo de ejecución son ambos O(S). Se omite el total de la memoria de la medida de complejidad secuencial, pues depende del número de reducers operando.

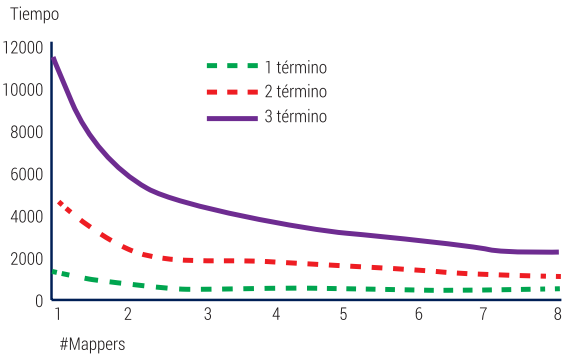

Figura 2. Tiempo de procesamiento en función del número de nodos.

3. CONCLUSIONES

La técnica de búsqueda presentada en este artículo se apoya en ontologías para buscar los documentos relevantes teniendo como base la consulta del usuario. El uso de ontologías hizo posible ampliar la consulta y resolverla utilizando términos ontológicos y las relaciones lógicas que los enlazan, con lo cual se redujo la ambigüedad al prestar importancia al contexto. La jerarquía de conceptos fue importante durante dicho proceso. La aproximación expuesta, a diferencia de las ya existentes, utiliza emparejamiento de retículos para poder presentar los documentos al usuario.

Este nuevo sistema ofrece ventajas entre las cuales se destacan: resultados con mayor correspondencia con el dominio de búsqueda, mejoras en el rendimiento debido al proceso de indexación, mayor precisión en los resultados (debido a la combinación de términos semánticos), mayor eficiencia al resolver consultas (al contar con combinaciones más grandes de términos ontológicos dentro del retículo y utilizar el índice inverso para obtener los documentos semánticamente relacionados a tales combinaciones), los resultados de la consulta corresponden más con la pertinencia de cada uno de los documentos, esto se debe a la utilización de un ranking doble.

El modelo propuesto se implementó completamente. En la actualidad, sobre un corpus estático, el proceso de indexación solo se realiza una vez. Se plantea como trabajo futuro la actualización incremental de los índices cuando se agregan documentos al corpus. Los resultados experimentales demuestran la eficacia del sistema para recuperar información.

REFERENCIAS BIBLIOGRÁFICAS

[1] Gruber, T. (1993). A Translation Approach to Portable Ontology Specifications. Knowledge Acquisition, 5(2), pp.199-220.

[2] Dean, J. & Ghemawat, S. (2008). MapReduce: simplified data processing on large clusters. Communications of the ACM - 50th anniversary issue: 1958 - 2008, 51(1), pp.107-113.

[3] Zou, G., Zhang, B., Gan, Y. & Zhang, J. (2008). An Ontology-Based Methodology for Semantic Expansion Search. FSKD '08: Proceedings of the 2008 Fifth International Conference on Fuzzy Systems and Knowledge Discovery, 5, pp.453-457.

[4] Jiang, X. & Tan, A. (2006). OntoSearch: a full-text search engine for the semantic web. AAAI'06 proceedings of the 21st national conference on Artificial intelligence, 2 , pp.1325-1330.

[5] Anderson, J. (1983). A spreading activation theory of memory. Journal of Verbal Learning and Verbal Behavior, 22, pp.261-295.

[6] Shamsfard, M., Nematzadeh, A. & Motiee S. (2006). ORank: An Ontology Based System for Ranking Documents. International Journal of Electrical and Computer Engineering, pp.225-231.

[7] Fellbaum, C. (2005). WordNet and wordnets. En Encyclopedia of Language and Linguistics, pp.665-670.

[8] Wang, L. & Tsai. (2008). A practical ontology query expansion algorithm for semantic-aware learning objects retrieval. Computers Education, 50(4), pp.1240-1257.

[9] Velardi, P. & Navigli, R. (2003)..An Analysis of Ontology-based Query Expansion Strategies. Workshop on Adaptive Text Extraction and Mining.

[10] Song, M., Song, Hu X., Allen, R. (2007). Integration of association rules and ontologies for semantic query expansion Data. Knowledge Engineering, 63(1), pp.63-75.

[11] Ranwez, S., Sy, M., Montmain, J., Regnault, A., Crampes, M. & Ranwez V. (2012). User Centered and Ontology Based Information Retrieval System for Life Sciences. BMC Bioinformatics, Vol 13(1).

[12] Alipanah, N., Khan, L., & Thuraisingham, B. (2011). Optimized Ontology-Driven Query Expansion Using Map-Reduce Framework to Facilitate Federated Queries. Proceeding ICWS '11 Proceedings of the 2011 IEEE International Conference on Web Services, pp. 712-713.

[13] Agrawal, R. & Srikant R. (1995). Mining sequential patterns. Proceeding ICDE '95 Proceedings of the Eleventh International Conference on Data Engineering, pp.3-14.

[14] Zaki, M. (2000). Scalable algorithms for association minin. IEEE Trans.Knowl.Data Eng, 12(3), pp.372–390.

[15] Ceglar, A. & Roddick J. F. (2006). Association mining. ACM Computing Surveys (CSUR) Surveys Homepage archive, 38(2), artículo 5.

[16] Levenshtein. (1966). Binary codes capable of correcting deletions, insertions, and reversals. Soviet Physics Doklady , 10(8), pp.707–10.

[17] Porta, J., (2006). Clasificación de patrones. Recuperado en abril de 2014, de http://arantxa.ii.uam.es/~jporta/iula/unsupervised.slides.pdf

[18] Salton, G. (1971). The Smart retrieval system–experiments, Inc. Upper Saddle River, NJ, EE.UU.: Prentice–Hall.

[19] Salton, G., Wong, A. & Yang C.S. (1975). A vector space model for automatic indexing. Communications of the Association for Computing, 18(11), pp.613–620.

[20] Apache software foundation. (2012). Apache Hadoop. Recuperado en abril de 2014, de http://hadoop.apache.org

[21] Apache software foundation. (2012). Apache Jena. Recuperado en abril de 2014, de http://jena.apache.org/

[22] Apache software foundation. (2012). Apache Lucene Core. Recuperado en abril de 2014, de http://lucene.apache.org/core/

[23] Bada, M., Eckert, M., Evans, D., Garcia, K., Shipley, K., Sitnikov, D., Baumgartner, W. A. & Cohen K. (2012). The CRAFT Colorado Richly Annotated Full-Text Corpus. Recuperado en abril de 2014, de http://sourceforge.net/projects/bionlp-corpora/files/CRAFT/v0.9/

[24] W3C. (2009). OWL 2 Web Ontology Language Document Overview. Recuperado en abril de 2014, de http://www.w3.org/TR/owl2-overview/

[25] Rijsbergen, R., Robertson, S.E. & Porter, M.F. (1980). New models in probabilistic information retrieval. British Library Research and Development Report, Vol 5587.

[26] Binary fortress software. (2014). Fileseek Fast and Free File Search. Recuperado en abril de 2014, de http://www.fileseek.ca/

[27] Frakes, W. & Baeza, R. (1992). Information Retrieval: data structures and Algorithms. México: Prentice-Hall.

[28] Mathworks. (2014). Matlab. Recuperado en abril de 2014, de http://www.mathworks.com/products/matlab

[29] Babu, S.(2010). Towards automatic optimization of MapReduce programs. SoCC10 Proceedings of the 1st ACM symposium on Cloud computing, pp.137–142.

[30] Paravastu, R., Scarlat, R. & Chandrasekaran, B. (2012). Adaptive Load Balancing in MapReduce. Recuperado en abril de 2014, de https://www.cs.duke.edu/courses/fall12/cps216/Project/Project/projects/Adaptive_load_balancer/adaptive-load-balancing.pdf

[31] Goel, A. & Munagala, K. (2012). Complexity measures for MapReduce, and comparison to Parallel computing. Recuperado en abril de 2014, de: http://www.stanford.edu/~ashishg/papers/mapreducecomplexity.pdf