UNIVERSIDAD CENTRAL DE VENEZUELA
FACULTAD DE CIENCIAS
POSTGRADO EN CIENCIAS DE LA COMPUTACIÓN

ANTEPROYECTO:
Solución Sistema Complementario Saber UCV (SSCSU): extracción de datos desde PDF, clasificación y construcción de un sistema de recuperación de información en línea

Anteproyecto presentado
por el Econ. José Miguel Avendaño Infante
Tutor: Dr. Andrés Sanoja

Caracas, Octubre de 2021

Resumen:

Se presenta la propuesta de un sistema denominado **Solución Sistema Complementario Saber UCV** para hacer búsquedas de texto completo sobre los resúmenes de las tesis que se encuentran alojadas en el repositorio institucional Saber UCV (www.saber.ucv.ve).

Se aplican técnicas de Procesamiento de Lenguaje Natural (NLP), de Minería de Texto y de indexación en base de datos sobre los textos y se muestran los resultados con tablas interactivas, visualizaciones con gráficos y gráfos de coocurrencias de palabras.

Esta propuesta se basa principalmente en que mediante la búsqueda de palabras o frases se genere un (*query*) y con la técnica conocida como ”*full text search*” se puedan extraer los trabajos en que se encuentran contenidas tales palabras y a partir de ahí enriquecer la experiencia del usuario con la presentación de la información recuperada.

La aplicación se diseña como un sistema distribuido bajo la arquitectura cliente-servidor en distintos contenedores soportando cada uno un proceso para el funcionamiento, teniendo entre los principales el de base de datos, el servidor de la aplicación y otro con los distintos procesamientos que son efectuados sobre los textos.

También se propone un algoritmo que permite clasificar las tesis por el área académica donde cursó estudios el autor del correspondiente trabajo y así resolver el problema que actualmente presenta Saber UCV donde no están disponibles estas clasificaciones.

Índice general

Índice de figuras

Índice de cuadros

Every important aspect of programming arises somewhere in the context of sorting or searching.

— Donald Knuth, The Art of Computer Programming, Volume 3

Capítulo 1

Introducción

En este Capítulo se presenta la propuesta 1.1 denominada **Solución Sistema Complementario Saber UCV** junto con el objetivo principal de la misma 1.2. Posteriormente se introducen los sistemas de recuperación de información 1.3 y se presenta el repositorio institucional Saber UCV 1.4 junto con la exposición donde se muestran los principales problemas 1.5 que tiene el Sistema que son los que motivan la presentación de esta Solución. Igualmente se hace un repaso a investigaciones realizadas 1.6 en la Universidad Central de Venezuela que parcialmente han abordado problemas similares al que acá presentaremos. Finalmente se expone la estructura 1.7 por capítulos de este Anteproyecto.

1.1. Propuesta:

En este trabajo se formula la propuesta de una Solución que permite realizar **busquedas de texto completo** sobre los Resúmenes de las Tesis que están alojadas en el repositorio institucional de documentos digitales Saber U.C.V. de la Universidad Central de Venezuela.

La propuesta se basa en el desarrollo e implementación de un sistema distribuido con arquitectura cliente-servidor que funcionará como una aplicación web y recibirá el nombre del **Solución Sistema Complementario Saber UCV (SSCSU)**, a la cual se accederá por medio de un navegador web y permitirá que ante la formulación de un ***query*** que contendrá una frase y la selección de distintos parámetros y granularidades como el intervalo de fechas, la selección de la(s) facultad(es) y/o escuela(s)-postgrado(s) donde se generó la investigación, sean recuperados del Sistema los textos que cumplen con las restricciones establecidas. Los resultados se mostrarán con distintos gráficos, tablas interactivas, coocurrencias de palabras **??** y recomendaciones de documentos basados en las similitudes que pueda presentar un documento 2.9 con los otros.

El SSCSU será desarrollado basándose en los principios que definen a los Sistema de Recuperación de Información 2.3 y se usará el *Corpus* [1] definido en 1.4.1.1 sobre el cual serán aplicadas técnicas de Minería de Texto 2.7 y de Procesamiento de Lenguaje Natural 2.6.1, para detectar relaciones y extraer conocimiento (Aggarwal and Zhai, 2012a). La implementación se hará en contenedores recayendo en cada uno de estos componentes los procesos principales del Sistema: gestor de base de datos, servidor de la aplicación, procesamientos de textos entre otros.

[1] Conjunto cerrado de textos o de datos destinado a la investigación científica.

Uno de los aspectos relevantes a resolver es realizar la clasificación de las Tesis por área de conocimiento, asociándolos a la escuela o el postgrado donde curso estudios el autor de cada investigación. Actualmente esta información no está disponible para el público ni en el repositorio Saber UCV.

Previo a la implementación de la Solución se realizará una investigación documental para generar un listado con todas las etiquetas candidatas con que las que se ejecutará el proceso de clasificación. La rutina que se propone incluye la lectura de los archivos adjuntos en distintos formatos y el uso de diversas técnicas y algoritmos que permitan generar la correcta clasificación de los documentos.

También el SSCSU permitirá obtener indicadores cuantitativos sobre los trabajos que reposan en Saber UCV,pudiendo establecer distintas dimensiones y granularidades para la obtención de estas cifras.

El SSCSU contará con una base de datos estructurada indexada **??** y estará diseñado para actualizarse periódicamente, incorporando así los nuevos trabajos que sean ingresados en Saber UCV.

El desarrollo de esta propuesta dará solución a los problemas planteados en 1.5 y se podrá convertir en una aplicación de utilidad para la comunidad universitaria y en general para los investigadores que usan como fuente de información los documentos que reposan en Saber UCV.

1.2. Objetivo General:

Generar un sistema de recuperación de información que funcione bajo una arquitectura distribuida cliente-servidor, que permita clasificar los trabajos que reposan en Saber UCV y a los usuarios finales generar consultas de “búsqueda de texto completo” sobre los textos que constituyen el *Corpus* mediante el acceso a una página web.

1.3. Sistemas de Recuperación de Información:

Es conocida la gran diversidad de información en distintos formatos que tienen a su disposición los investigadores. Ejemplo de esto son libros en las bibliotecas o los documentos que se encuentran accesibles en formatos digitales como artículos publicados en revistas arbitradas, libros, páginas de internet especializadas en algún tema o los repositorios digitales. Es en esta abundancia donde recaen varios de los problemas a los cuales se enfrenta la labor de investigación (Hernández Orallo et al., 2004) (p.565), especialmente cuando se realiza la fase exploratoria de selección de aquellos documentos que resultan de mayor interés.

El investigador debe contar con herramientas al enfrentarse al proceso de búsqueda de información siendo una de estas los ***Sistemas de Recuperación de Información (SRI)***, que son los que permiten que el investigador formule, con el nivel de detalle deseado, la búsqueda mediante un texto al cual denominaremos ***query***, siendo el sistema el encargado de detectar la aparición de dicho texto en los documentos que previamente han sido agrupados como un ***Corpus*** (Manning et al., 2008). Conocemos que tal búsqueda se debe ejecutar con ciertos niveles de flexibilidad y no restringirse a la aparición exacta del ***query***. También se puede complementar con distintos criterios a filtrar

como un período restringido de fechas o que esté asociado a una determinada clasificación, como por ejemplo un área del conocimiento específica.

Para ilustrar lo anterior usemos el siguiente ejemplo: supongamos que un ingeniero quiere buscar información sobre la "investigación de operaciones" dentro de un ***Corpus*** de textos académicos que contiene diversas áreas de conocimiento. Al hacer el *query* el sistema que usemos va a determinar cuáles documentos dentro del *corpus* hacen mención a ese tema. Es probable que encontrásemos textos asociados a medicina donde la "investigación de operaciones" tiene una semántica distinta a la rama del conocimiento que estudia problemas de optimización que se denomina "investigación de operaciones". Esto es gracias a que en diferentes áreas de estudio se pueden tener comprensiones que divergen sobre una misma búsqueda de términos.

Visto lo anterior es necesario que el Sistema de Recuperación de Información que usemos permita separar de alguna forma la información recuperada acorde a un contexto de interés del investigador, jerarquizando aquellos documentos que puedan resultar de mayor interés. Incluso, las visualizaciones con las que representemos los resultados obtenidos de un *query* pueden aportar conocimiento en sí mismo haciendo intuitiva la comprensión (Zhang, 2008) o dando aportes hacia dónde es viable dirigir futuras indagaciones en un proceso que claramente es iterativo (Zhai and Massung, 2016) y adicionalmente el sistema puede generar la recomendación de otros trabajos (Aggarwal, 2018) que presenten alguna similitud sobre el tema investigado. En la sección 2.3 se exponen otros detalles relevantes sobre los **SRI**.

1.4. Saber UCV:

La Universidad Central de Venezuela cuenta con un Sistema de Recuperación de Información denominado **Saber UCV**, al cual se puede acceder en la dirección www.saber.ucv.ve que funciona a modo de repositorio digital institucional [2] de los distintos documentos de textos académicos generados por la comunidad ucevista. Este sistema fue "creado para alojar, gestionar y difundir de manera gratuita y en texto completo: tesis, artículos de investigación, libros, revistas electrónicas, presentaciones que conforman la producción académica" de esta casa de estudios.

Saber UCV es una implementación del software de código abierto llamado **DSpace** que fue creado con la finalidad de ser un repositorio digital institucional bibliográfico, para uso académico de colecciones digitales según se indica en la página web oficial del proyecto https://duraspace.org.

Este software fue diseñado e implementado en un esfuerzo conjunto entre desarrolladores del *Massachusetts Institute of Technology (MIT)* y *Hewlett-Packard Labs (HP Labs)* en el año 2002. En junio de 2021 se lanzó la 7 versión que es la más reciente, no obstante la versión que está implementada en Saber UCV es la 1.7.1. que data del año 2013.

1.4.1. Delimitación de documentos con los cuales se trabajará:

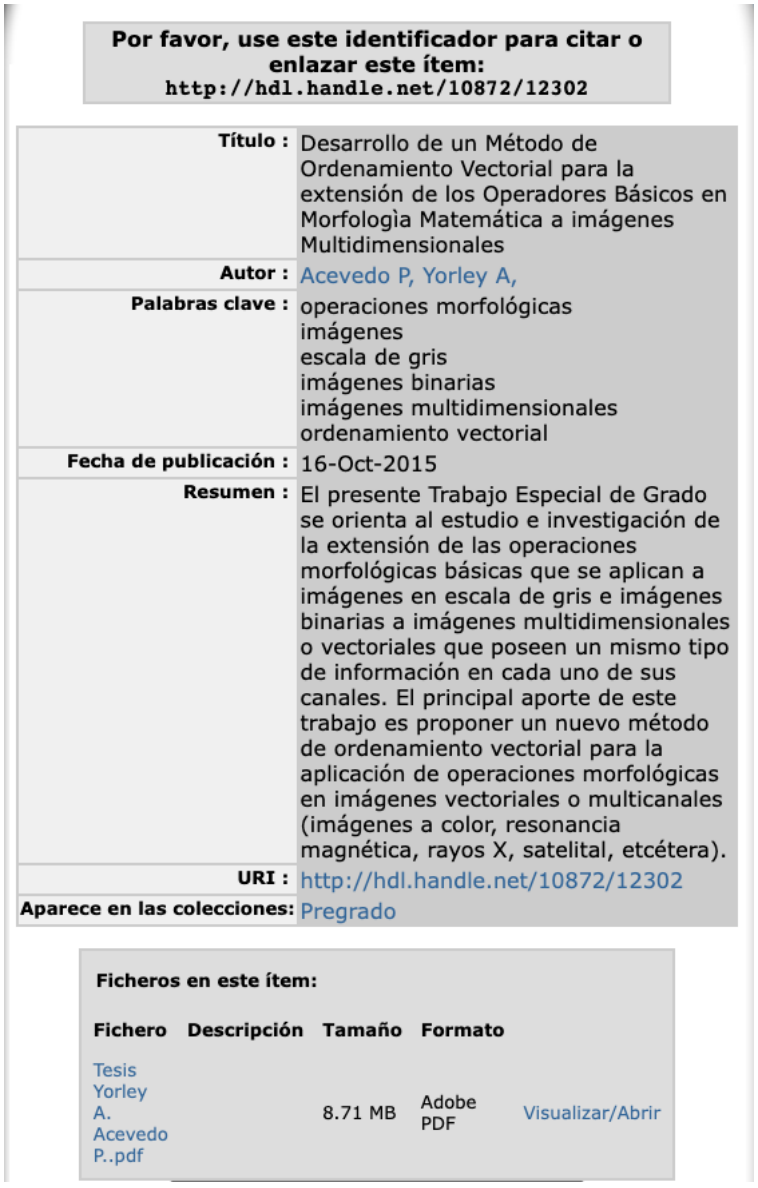
Si bien en el sitio web están almacenados artículos de prensa, artículos preimpresos y publicados, fotografías, infografías y tesis de distintos niveles académicos es exclusivamente con este último subconjunto con el cual se trabajará.

[2] "base de datos electrónica que aloja una colección de pequeñas unidades de información educativas o actividades que pueden ser accedidas para su obtención y uso" (Lehman, 2007)

1.4.1.1. Corpus seleccionado:

En la categoría **Tesis** del repositorio se encuentran los trabajos especiales de grado de pre y postgrado junto con las tesis de doctorado. Cada documento de la categoría Tesis, indistintamente del nivel académico al que pertenezca, tiene uno o más archivos anexos en formato *word* o *PDF* donde está contenida la investigación y adicionalmente cada documento cuenta con una ficha en *html*. En nuestra investigación usaremos la denominación "Tesis" indistintamente de si son trabajos especiales de grado de pre y postgrado o tesis de doctorado.

En esta ficha se muestran ciertos detalles sobre el trabajo seleccionado como lo son: el título, el nombre del autor, las palabras claves, la fecha de publicación y el texto **Resumen**. En la figura 1.1 podemos ver la ficha que corresponde a un trabajo almacenado en el repositorio.



Por favor, use este identificador para citar o enlazar este ítem:
http://hdl.handle.net/10872/12302

Título :	Desarrollo de un Método de Ordenamiento Vectorial para la extensión de los Operadores Básicos en Morfologìa Matemática a imágenes Multidimensionales
Autor :	Acevedo P, Yorley A,
Palabras clave :	operaciones morfológicas imágenes escala de gris imágenes binarias imágenes multidimensionales ordenamiento vectorial
Fecha de publicación :	16-Oct-2015
Resumen :	El presente Trabajo Especial de Grado se orienta al estudio e investigación de la extensión de las operaciones morfológicas básicas que se aplican a imágenes en escala de gris e imágenes binarias a imágenes multidimensionales o vectoriales que poseen un mismo tipo de información en cada uno de sus canales. El principal aporte de este trabajo es proponer un nuevo método de ordenamiento vectorial para la aplicación de operaciones morfológicas en imágenes vectoriales o multicanales (imágenes a color, resonancia magnética, rayos X, satelital, etcétera).
URI :	http://hdl.handle.net/10872/12302
Aparece en las colecciones:	Pregrado

Ficheros en este ítem:

Fichero	Descripción	Tamaño	Formato	
Tesis Yorley A. Acevedo P..pdf		8.71 MB	Adobe PDF	Visualizar/Abrir

Figura 1.1: Modelo de ficha registro en Saber UCV

Con los textos **Resumen**, con la **fecha**, el **nombre del autor**, las **palabras claves** de todos los trabajos catalogados como Tesis en el repositorio Saber UCV es con los que conformaremos nuestro ***Corpus***.

Para el momento en que se presenta esta Investigación bajo la categoría **Tesis** reposan 10.481 documentos distribuidos con las jerarquías que se muestran en el cuadro 1.1.

Cuadro 1.1: Total documentos por jerarquía

jerarquía	cantidad
doctorado	305
maestria	729
otras	1.239
pregrado	8.208
Total	**10.481**

La disponibilidad de trabajos en el repositorio por fecha de creación la presentamos en el histograma que se muestra en la figura 1.2 .

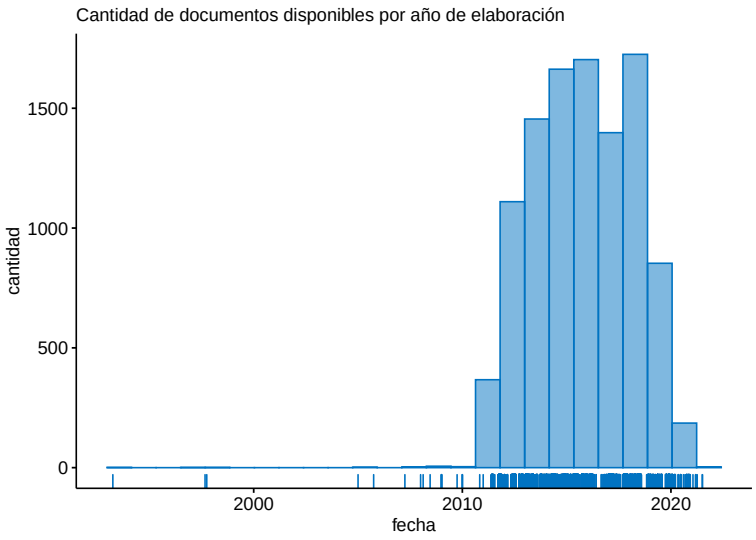


Figura 1.2: Cantidad de documentos disponibles por año de elaboración

Analizando la distribución de la frecuencia de la figura 1.2 por año en que fue generada la investigación es necesario destacar que en el repositorio no se encuentran todas las investigaciones que se han realizado en la historia de la U.C.V. Actualmente se tienen incorporados solo 20 trabajos creados entre 1993 y 2011, siendo a partir de este último año cuando se empiezan a realizar la ingesta de forma periódica y en mayor volumen los trabajos. No obstante también queremos resaltar que desde el 2011 no se encuentran incorporados todos los trabajos generados dentro de la comunidad. Las razones que explican omisiones o retardos en las incorporaciones queda fuera de los objetivos de este trabajo investigarlas.

1.4.1.2. Archivos complementarios:

Por razones que se expondrán en 1.5.1 igualmente será necesario trabajar con los documentos anexos que indicamos en 1.4.1.1 que disponen un vínculo de descarga en cada ficha *html*. La descarga y el procesamiento de estos textos será necesario para complementar el corpus descrito en 1.4.1.1.

1.5. Planteamiento del Problema:

Si bien este repositorio con que cuenta la Universidad sirve para realizar búsqueda de textos mediante procesos de recuperación de información, hay algunas características que pueden ser complementadas y mejoradas.

1.5.1. I. No se posee clasificación de los documentos:

El ***Corpus*** que reposa en Saber UCV para ninguno de los documentos posee la etiqueta que clasifique a cual postgrado o carrera de pregrado pertenece. En caso de realizar una búsqueda y querer delimitarla a documentos que se circunscriban a una determinada área de conocimiento, actualmente es imposible y por ende resulta inviable evaluar la escuela o el postgrado y la facultad donde se hizo la investigación.

Tampoco se dispone de un listado con las etiquetas candidatas a asignar a cada uno de los documentos. Según lo investigado no es de conocimiento público un único directorio que contenga los nombres de las carreras de pregrado y de los distintos postgrados, significando esto que es un doble proceso de construcción el que se necesita hacer para realizar la clasificación de los documentos.

Relativo a este problema también surge el determinar cómo se realizará la comparación de cada una de las etiquetas con el texto de cada documento, incluso detectar dónde se localiza el texto con el cual se efectuará la comparación, ya que el texto denominado **Resumen** que vimos en la figura 1.1 no contiene la información necesaria para extraer dónde fue realizada la investigación (escuela o postgrado), haciendo necesario que sean analizados los documentos anexos a cada ficha que es propiamente el archivo, o los archivos, que componen cada una de las investigaciones.

En detalle será necesario consolidar los archivos disponibles para cada investigación, los cuales no necesariamente se encuentran ordenados, por ejemplo por capítulo. Una vez realizada la consolidación será necesario tener el archivo en texto plano teniedo presente que en algunos casos en viable que sean imágenes lo que contiene el archivo PDF procediendo a realizar la lectura OCR [3] para obtener el texto. En este proceso de clasificación también se presentarán otros problemas relativos a los sistemas de *encodings* y a la introducción de caracteres no deseados dentro del texto, lo que será expuesto a detalle en el capítulo que contiene el desarrollo de la propuesta.

[3] OCR: "*optical character recognition*" traducido como reconocimiento óptico de caracteres

1.5.2. II. Problemas en los resultados que generan los *Querys*:

Existe un tipo de *query* denominado *"Full Text Search"* o búsqueda de texto completo **??** que es donde se manifiesta una de las principales deficiencias que posee el Sistema Saber UCV.Aunque en él se pueden ejecutar búsquedas de texto, los métodos con los que se implementan estas búsquedas llevan a que sean generados una gran cantidad de resultados, en especial al usar más de una palabra en la búsqueda, presumimos que por usar el operador lógico *OR* entre cada una de las palabras solicitadas en el *query*.

1.5.2.1. Ejemplo *Query* en Saber UCV:

Evaluemos el problema descrito con un ejemplo. Supongamos que queremos ver en el sistema Saber UCV los trabajos que un autor de nombre "Jesús Fajardo" ha realizado.

Figura 1.3: Búsqueda de un autor "Jesus Fajardo" en www.saber.ucv.ve

Al revisar los resultados que se muestran en la figura 1.3 podemos apreciar que la primera posición de los 30 resultados que fueron encontrados, no muestra un autor que se llame "Jesús Fajardo" sino el apellido Fajardo. Incluso, al realizar la consulta del vínculo del primer resultado tampoco aparece la palabra "Jesús" dentro de todos los datos que muestra la ficha. Al revisar el segundo resultado vemos que sí coincide con el término que se estaba buscando, pero de resto ningún otro de los 28 resultados incluyen un "Jesús Fajardo" o un "Jesús".

Revisemos otro ejemplo de búsqueda esta vez con cuatro palabras que son "simultánea de múltiples metástasis" las cuales se encuentran en el título del trabajo visto anteriormente en la posición número dos. El resultado del *query* lo podemos ver en la figura 1.4.

La cantidad de resultados asciende a 10.449, cifra similar a la cantidad total de trabajos de la categoría "Tesis" que se encuentran en el repositorio, ya que como fue explicado anteriormente el Sistema Saber UCV debe aplicar el operador lógico *OR* entre cada una de las palabras al momento

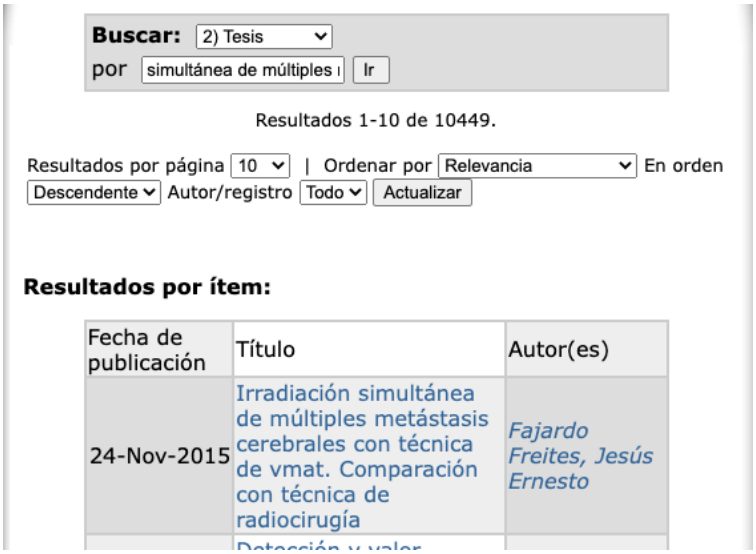


Figura 1.4: Búsqueda de ”simultánea de múltiples metástasis” en www.saber.ucv.ve

de generar el *query*. A manera de referencia podemos explicar que si hubiésemos querido realizar la búsqueda exacta de la frase, con haberle colocado comillas al inicio y al final obtendríamos la referencia al trabajo que previamente sabemos que es el que queremos encontrar.

Sin embargo el uso de comillas para buscar una frase exacta genera una compensación entre la precisión en la búsqueda, lo cual se motiva en que al ser exacta, se cierra el umbral para encontrar otros resultados que pudiesen resultar de interés, así que el uso de este método puede presentar considerables desventajas.

Posterior a evaluaciones realizadas sobre el comportamiento de Saber UCV presumimos que los textos al ser procesados para su estructuración e ingesta en la base de datos no le son aplicadas técnicas con el algoritmo de Porter que permite realizar la extracción del ***stemming*** [4] de cada una de las palabras. La importancia de aplicar este procesamiento será explicado en 2.6.1.4

1.6. Antecedentes:

Se hizo una revisión bibliográfica de trabajos producidos dentro de la Universidad Central de Venezuela que puedan haber realizado investigaciones similares o complementarias a la que proponemos en este desarrollo. De particular interés resultó la “Elaboración de un prototipo de buscador de documentos académicos de la Facultad de Ciencias” (Sánchez, 2008), que fue una investigación que implementó el motor de búsqueda y repositorio denominado BUSCONEST 1 que sirve exclusivamente a los documentos de investigación generados en la Facultad de Ciencias de la U.C.V. Este sistema sí cuenta con los trabajos clasificados por área del conocimiento donde fueron generados, no obstante no resuelve el problema planteado sobre el resto de documentos que reposan en Saber UCV.

Posteriormente (Guevara, 2015) presenta otra propuesta de motor de búsqueda con una versión modificada de BUSCONEST 1, denominada BUSCONEST 2, no obstante sólo se basa sobre textos

[4] proceso heurístico que remueve letras al final de las palabras con el objetivo de encontrar la raíz de la misma.

producidos en la Facultad de Ciencias y en ninguno de estos desarrollos se realizan procesos de minería de texto.

Otra investigación que podemos señalar es “Sistema de recomendación para el Buscador Académico Venezolano” (Rodríguez Laguna, 2016). En ella se señalan algunos repositorios digitales institucionales que se encuentran en Venezuela y se hace una propuesta de unificación de todos los contenidos y así poder generar recomendaciones de investigaciones a partir de una determinada búsqueda.

Evaluando estos antecedentes consideramos que nuestra propuesta resuelve el problema de la clasificación de documentos y permite efectuar las búsquedas sobre todo el conjunto de textos, así que podemos tener en perspectiva que nuestro desarrollo no solapará las propuestas mencionadas y generará un valor agregado favoreciendo a la actividad de investigación.

1.7. Estructura del Trabajo:

Este trabajo está estructurado de la siguiente forma: en un Capítulo 2 se expone el Marco Teórico. En el Capítulo 3 se describirá la propuesta técnica, el objetivo general y específicos de la Solución.

Tentativamente se propone que en el Proyecto de Trabajo de Grado en el Capítulo 4 se exponga el proceso de desarrollo de la Solución y las distintas pruebas que se realizaron para elegir una determinada tecnología por sobre otras, así como los resultados obtenidos, mientras que en el Capítulo 5 se presentarán las conclusiones y recomendaciones, posterior a la implementación del Sistema.

Capítulo 2

Marco teórico-referencial:

En este Capítulo se muestra el marco teórico en que se sustentan los aspectos de mayor relevancia para el desarrollo de la Solución. Principalmente se enuncian una serie de conceptos que involucran algoritmos de búsqueda, la recuperación de información, la minería de texto, el procesamiento del lenguaje natural, la estructuración de la base de datos y lo referente a la arquitectura distribuida en que se soporta la SSCSU.

2.1. Reseña histórica:

El profesor Donald Knuth señala, dentro del campo de las ciencias de la computación, que la **búsqueda** *es el proceso de recolectar información que se encuentra en la memoria del computador de la forma más rápida posible, esto cuando tenemos una cantidad N de registros y nuestro problema es encontrar el registro apropiado de acuerdo a un criterio de búsqueda* (Knuth, 1997) (p. 392) .

Iniciamos con esta cita porque la recuperación de información gira en torno a un problema central que es la **búsqueda**. A continuación mencionaremos una serie de algoritmos que abordan este problema, no necesariamente resultando óptimos para dar solución a lo planteado en 1.5.2.

En la década de 1940 cuando aparecieron las computadoras las búsquedas no representaban mayor problema debido a que estas máquinas disponían de poca memoria *RAM* pudiendo almacenar solo moderadas cantidades de datos. En realidad estaban diseñadas para realizar cómputos y arrojar los resultados más no para tenerlos almacenados en memoria.

No obstante con el desarrollo del almacenamiento en memoria *RAM* o en dispositivos de almacenamiento permanentemente ya en la década de 1950 empezaron a aparecer los problemas de búsqueda y los primeras investigaciones para afrontarla. En la década de 1960 se adoptan por ejemplo estrategias basadas en arboles.

Los primeros algoritmos que sirvieron para localizar la aparición de una frase dentro de un texto, o expresado de forma más abstracta, como la detección de una subcadena *P* dentro de otra cadena *T*, fueron los algoritmos de *Pattern-Matching* (Goodrich et al., 2013) (p. 584). Así tenemos dentro de la literatura el algoritmo *Fuerza Bruta* donde dado un texto T y una subcadena P, se va recorriendo cada elemento de la cadena T para detectar la aparición de la subcadena P. Si bien este algoritmo no

presentaba el mejor desempeño, por contar con ciclos anestados en su ejecución, creó una forma válida de enfrentar el problema de la búsqueda de subcadenas de texto.

El algoritmo *Knuth-Morris-Pratt* que se introdujo en 1976 tenía como novedad que se agregó una función que iba almacenando previas coincidencias parciales en lo que eran fallos previos y así al realizar un desplazamiento tomaba en cuenta cuántos caracteres se podían reusar. De esta forma se logró considerablemente mejorar el rendimiento en los tiempos de ejecución de $O(n+m)$ que son asimptóticamente óptimos.

Posteriormente en 1977 el problema se enfrenta con un nuevo algoritmo que es el de *Boyer-Moore* en el cual se implementan dos heurísticas (*looking-glass y character-Jump)* que permiten ir realizando algunos saltos en la búsqueda, ante la no coincidencia de la subcadena con la cadena y adicionalmente el orden en el que se va realizando la comparación se invierte. Estas modifacaciones permitieron obtener un mejor desempeño.

Sobre una modificación al algoritmo *Boyer-Moore* se sustenta la utilidad *grep* de la línea de comandos UNIX que igualmente le da soporte a diversos lenguajes que la usan para ejecutar búsquedas de texto con un proceso que comúnmente es conocido como *grepping*. Esta utilidad fue ampliamente usada para resolver parcialmente lo expuesto en en 1.5.1.

Los algoritmos mencionados anteriormente pueden ser usados en procesos de recuperación de información en conjunto con técnicas que pueden mejorar considerablemente los tiempos en la ejecución de las rutinas.

Otra estrategia que surgió para enfrentar las búsquedas de texto fue el uso de la programación lineal donde bajo la premisa *divida et impera* los problemas que requieren tiempo exponencial para ser resueltos son descompuestos en polinomios y por lo tanto se disminuye la complejidad en tiempo para ser resueltos. Entre este tipo de algoritmos podemos mencionar los de *alineación del ADN y las secuencias de texto* que originalmente se planteó para resolver problemas de alineación de cadenas de ADN de forma parcial o total dentro de una cadena mayor. Posteriormente se identificó que este tipo de procedimiento era extrapolable a los subcadenas de texto. Una de las versiones de estos algoritmos es la denominada ***Smith-Waterman*** que resultó de gran utilidad para resolver el problema planteado en 1.5.1 que no pudo ser resuelto con la utilidad *grep*.

Un gran paso para aproximarnos a la aparición de los Sistemas de Recuperación de Información lo representó el enfoque que presentan los algoritmos *Tries*. Este nombre proviene del proceso de *Information Retrieval* y principalmente se basa en hacer una serie de preprocesamientos a los textos para que al momento de ejecutar la búsqueda de texto, es decir, de la subcadena dentro de la cadena, ya tengamos una parte del trabajo realizado previamente y no tener que ejecutarlo todo *"on the fly"*, es decir, sobre la marcha.

Un *Trie* (Fredkin, 1960) es una estructura de datos que se crea para almacenar textos para así poder ejecutar más rápido la coincidencia en la búsqueda. En la propuesta del SSCSU todos los textos van siendo procesados con distintas técnicas a medida que son insertados en la base de datos. Esto claramente representa una mejora en el desempeño en los tiempos de búsqueda.

2.2. Recuperación de Información:

El eje central sobre el cual gira el proceso de recuperación de información (RI) es satisfacer las necesidades de información relevante que sean expresadas por un usuario mediante una consulta

(***query***) de texto. El investigador Charu Aggarwal en su libro sobre Minería de Texto (Aggarwal and Zhai, 2012b) (p.14) menciona que el objetivo del proceso de RI es conectar la información correcta, con los usuarios correctos en el momento correcto, mientras que otro de los autores con mayor dominio sobre el tema, Christopher Manning en su libro *Information Retrieval* indica que "es el proceso de encontrar materiales (generalmente documentos) de una naturaleza no estructurada (generalmente texto) que satisface una necesidad de información dentro de grandes colecciones (normalmente almacenada en computadores)" (Manning et al., 2008) (p. 25).

Satisfacer una necesidad de recuperación de información no sólo se circunscribe a un problema **búsqueda** de un texto dentro de un *corpus*. En la mayoría de los casos se deberá cumplir con ciertos criterios, o restricciones, como por ejemplo que el *query* esté dentro de un período de fechas, o que se encuentre comprendido en un subconjunto del corpus que es a lo que se denomina **búsqueda múlti atributo**.

La información que se recolecte en una búsqueda tendrá distintos aspectos que aportarán peso en el orden en que sea presentada al usuario y no sólo vendrá dado por la aparición de las palabras sino también por otros elementos como lo puede ser la aparición de la frase del ***query*** dentro del título, la proximidad (la cercanía entre dos palabras) que tengan los términos que conforman el *query* o por otra parte la frecuencia que una palabra, o varias, se repitan dentro un determinado documento que compone el *corpus*. Igual puede aportar un peso mayor a la recuperación de un documento las referencias (citas) que contengan otros documentos a ese determinado documento. El fin último será la extracción de los documentos que resulten de mayor relevancia para el usuario.

Incluso es válido incorporar documentos, en los resultados que arroje la búsqueda, que propiamente no coincidan con los términos buscados sino que contengan términos que sean sinónimos o también añadiendo a los resultados documentos que presenten alguna similitud con el texto del ***query***. Lo que acabamos de mencionar incorporará formalmente dentro del proceso de extracción de información algo de imprecisión con la intención última de enriquecer el proceso de ***Information Retrieval*** (Kraft and Colvin, 2017).

Evaluando el proceso con cierto nivel de abstracción tenemos que el proceso de recuperación de información está compuesto principalmente por: un *query*, por un corpus y por una función de *ranking* para ordenar los documentos recuperados de mayor importancia a menor.

El desarrollo de los algoritmos expuestos en 2.1 sumado a la necesidad de resolver los problemas asociados a la búsqueda de un texto dentro de un *corpus´´* con múltiples atributos en tiempos aceptables y la abundante cantidad de información digital potenciada por el uso generalizado de los computadores abonó las condiciones para la creación de los **Sistemas de Recuperación de Información**.

2.3. Sistemas de Recuperación de Información (SRI) :

Los Sistemas de Recuperación de Información (*Information Retrieval Systems-IRS*) son los dispositivo (software y/o hardware) que median entre un potencial usuario que requiere información y la colección de documentos que puede contener la información solicitada (Kraft and Colvin, 2017) 1. El SRI se encargará de la representación, el almacenamiento y el acceso a los datos que estén estructurados y se tendrá presente que las búsquedas que sobre él recaigan tendrán distintos costos siendo uno de estos el tiempo que tarde en efectuarse.

Es de nuestro conocimiento que generalmente los datos estructurados son gestionados mediante un sistema de base de datos pero en el caso de los textos estos se gestionan por medio de un motor de búsqueda motivado a que los textos en un estado crudo carecen de estructura (Aggarwal and Zhai, 2012b) (p. 2). Son los motores de búsqueda (*search engines*) los que permiten que un usuario pueda encontrar fácilmente la información que resulte de utilidad mediante un *query*.

El SSCSU está diseñado como un IRS donde se pueden ejecutar querys que son procesados y los resultados que se obtienen son sometidos a una función de ranking que será expuesta en ????.

2.4. Ejemplos de IRS:

Profundizando en el tema de esta Investigación mencionaremos un par de páginas de internet que funcionan con IRS. Una es el proyecto denominado Arvix alojado en www.arvix.com, que es un repositorio de trabajos de investigación. También tenemos el portal de la Asociation Computery Machine (ACM) que incorpora motores de búsqueda con particulares características facilitando la labor de investigación y extracción de información ante una determinada necesidad.

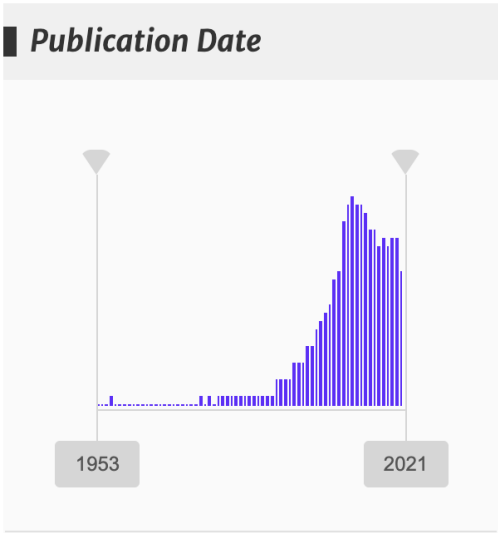


Figura 2.1: Gráfico que acompaña resultados de búsqueda de un término en la biblioteca digital de la Association for Computing Machinery (https://dl.acm.org/)

2.5. Modelos de Recuperación de Información:

2.5.1. Recuperación boleana:

Ante una búsqueda de información se recorre linealmente todo el documento para retornar un valor boleano indicando la presencia o no del término buscado. Es uno de los primeros modelos que se uso y está asociado a técnicas de *grepping* (Manning et al., 2008) (p.3). El desarrollo apareció entre 1960 y 1970.

El usuario final obtendrá como respuesta a su *query* solo aquellos textos que contengan el término. Es un modelo muy cercano a los típicos *querys* de bases de datos con el uso de operadores "AND", "OR" y "NOT". En el procesamiento de los textos se genera una matriz de incidencia binaria termino-documento, donde cada término que conforma el vocabulario, ocupa una fila *i* de la matriz mientras que cada columna *j* se asocia a un documento. La presencia de el término *i* en el documento *j* se denotará con un valor verdadero o *1*.

La recuperación boleana si bien representa una buena aproximación a la generación de *querys* más rápidos, presenta una gran desventaja y es que al crecer la cantidad de documentos y el vocabulario se obtiene una matriz dispersa de una alta dimensionalidad, que hace poco efectiva su implementación.

Los documentos y los *querys* son vistos como conjuntos de términos indexados que luego fueron llamados "bolsa de términos" *(bag of terms)*. Las deficiencias de este modelo recaen en que los resultados, no tienen ningún ranking. Si por ejemplo el término sobre el cual se realiza el *query* aparece 100 veces en un documento y en otro aparece sólo una vez, en la presentación de los resultados ambos documentos aparecerán al mismo nivel, ninguno tendrá alguna preferencia sobre el otro.

Una de las desventajas es que no se registra el contexto semántico de las palabras, incluso se pierde el orden en que aparecen las palabras en cada texto.

Este modelo se presume que en el cual se basa la implementación de Saber UCV y por eso es que en general se termina presentando el problema de que al usar el operador *AND* en las búsquedas exactas mencionadas en 1.5.2.1 se obtiene una alta ***precisión*** en los resultados pero un bajo ***recall*** mientras que al usar el operador *OR* da una baja ***precisión*** y un gran ***recall***. Con la propuesta del SSBSU se quiere una versión de recuperación de información que aplica métodos de mayor eficiencia, que permiten mejorar el desempeño (tiempo) y la calidad de los resultados.

2.5.2. Indices Invertidos:

Se denominan índices invertidos porque en vez de guardar los documentos con las palabras que en ellos aparecen, en ellos se guarda cada palabra y se indica los documentos en los cuales aparece y adicionalmente se puede guardar la posición en que aparece cada palabra, con distintas granularidades pudiendo ser estas, dentro del documento, del capítulo, del párrafo o de la oración. También pueden contener la frecuencia con que se presenta determinada palabra. Toda esta información nos permite mejorar los tiempos de búsqueda pero con ciertos costos.

El primero es el espacio en disco que implica guardar estos datos adicionales, que puede oscilar del 5 % al 100 %, mientras que el segundo costo lo representa el esfuerzo computacional de actualizarlos una vez que se incorporan nuevos documentos (Mahapatra and Biswas, 2011).

Existen diversos tipos y constantemente se están realizando investigaciones que permitan mejorar su desempeño motivado en que sobre ellos recae gran parte de la efectividad que podamos obtener ejecutando los *querys*.

El espacio que ocupa la implementación se ve afectado, en la reducción por el preprocesamiento que hacemos de las palabras buscando su raiz mediante el steamming {#steam} y en total el peso se incrementa. En el transcurso del desarrollo de nuestra investigación indicaremos en cuánto se incremento el espacio de almacenamiento en disco con la aplicación de este índice.

Incluso se pueden generar dos índices inversos para un sistema conteniendo, uno de estos la lista de documentos y la frecuencia de la palabra, mientras que el otro registra la lista con las posiciones de la palabra.

El uso de los índices invertidos permite la denominada "busqueda de texto completa" (*full text search*) que es uno de los pilares que sustenta a los motores de búsqueda. Adicionalmente podemos introducir el criterio de búsqueda de texto aproximado *(approximate text searching)*, que permite flexibilizar la coincidencia entre el texto requerido y el resultado.

En la Solución propuesta la optimización en la generación de este índice quedará bajo la administración del propio manejador de base de datos que es *postgreSQL*, por lo cual el manejo de métodos como ***non-byte-aligned*** o ***simple byte-aligned*** quedan sujetos a los algoritmos que use el manejador.

Cuando la base de datos que registra el índice invertido crece y no es viable almacenarla en un solo computador, es necesario acudir al uso de técnicas que permitan distribuir la base de datos con el uso de tecnologías como *spark, hadoop, Apache Storm* entre otras.

2.5.3. Scoring Model:

Es un modelo aplicado en amplias colecciones de documentos donde es necesario solo mostrar al usuario que realizó la búsqueda, una fracción de los documentos encontrados, pero es necesario que estos sean los de mayor puntuación (*score*), de acuerdo a distintos criterios que permitan determinar cuales son los que tienen mayor relevancia como lo puede ser la cantidad de veces que se repite una palabra aparecida en el *query* dentro del documento, o la distancia de aparición de cada una de las palabras que conforman el query (cantidad de palabras que median entre una y otra).

También se puede asignar un peso mayor en la generación del ranking, si una, o varias, de las palabras que generan el *query* aparece dentro del título, o en otros campos que se registrasen en la base de datos.

2.6. Procesamientos de texto:

En esta sección mostramos métodos de trabajo con los textos. Para el idioma español no son de abundante literatura, a diferencia de aquellos que están en el idioma inglés. Incluso hasta hace unos pocos años las herramientas computacionales para el procesamiento de los textos **??**nlproc) tampoco eran abundantes y sabiendo que son justamente los textos, el insumo que recibe de nuestro sistema de recuperación de información, la calidad en los procesamientos que sobre ellos hagamos, marcarán en gran medida la propia calidad del sistema que tengamos.

Como decíamos la literatura y herramientas disponibles para el NLP en el idioma español, fueron escasas durante un considerable período. Se tenían disponibles algunas como el coreNLP de la Universidad de Stanford pero no incluía todas las utilidades, tales como la identificación de parte del discuro *(Part of Speech Tagging),* ni el análisis morfológico (*Morphological Analysis)* o el reconocimiento de entidades nombradas (*Named Entity Recognigtion)*, sino algunas pocas como el tokenizador y el separador de oraciones (Sentences Splitting).

Casos similares se presentaban con otras herramientas, siendo un caso aparte el esfuerzo del CLiC-Centre de Llenguatge i Computación quienes hicieron la anotación del Corpus AnCora [1] . También la Universidad Politécnica de Cataluña creó la herramienta FreeLing [2] que permite realizar algunas de las funcionalidades mencionadas en el parrafo anterior. No obstante su integración en cadenas de trabajo y la actualización de sus modelos de entrenamiento presenta rezagos en comparación a otros modelos que actualmente se están usando, los cuales serán evaluados en el transcurso de esta investigación y serán señalados en el Capítulo del Desarrollo.

2.6.1. Procesamiento del Lenguaje Natural (natural language processing- NLP):

Son las técnicas computacionales desarrolladas para permitir al computador "comprender" el significado de los textos de lenguaje natural. En esta Investigación son aplicadas una serie de estos métodos sobre los textos que componen el Corpus y podemos mencionar las siguientes:

2.6.1.1. Tokenizador:

Básicamente es separar el documento en palabras las cuales se llaman *tokens*. Para el idioma español no representa un mayor reto, ya que se puede usar el espacio como delimitador de palabras, no así en otros idiomas como el chino donde el problema se aborda de manera distinta.

Al obtener las palabras como entidades separadas de un texto nos permite calcular la frecuencia de uso de las mismas.

Posteriormente se elaborará un cuadro comparativo de resultados entre distintos *tokenizadores* con una muestra aleatoria de palabras.

Es común que las librerías de procesamiento de lenguaje natural contengan tokenizadores que presentan un 100 % como métrica de precisión.

2.6.1.2. Entidades Nombradas *(named entity reconigtion-NER)*:

Son los procesos que permiten la extracción de las distintas entidades contenidas dentro de los textos. Las entidades son: nombres, lugares, organizaciones. También permite detectar relaciones entre entidades. Las medidas de precisión en los módulos de NER alcanzan una precisión cercana al 89 % en modelos entrenados con *machine learning*, tal es el caso de spacy, que es una de las librerías propuestas para realizar estos procesamientos en nuestro desarrollo.

2.6.1.3. Etiquetado de Partes del Discurso *(Part of speech tagging-POS)*:

Una de las técnicas usadas en el Procesamiento del Lenguaje Natural es el part-of-speech (POS) y consiste en asignar un rol sintáctico a cada palabra dentro de una frase (Eisenstein, 2019) siendo

[1] **AnCora** es un corpus del **catalán (AnCora-CA)** y del **español (AnCora-ES)** con diferentes niveles de anotación como lema y categoría morfológica, constituyentes y funciones sintácticas, estructura argumental y papeles temáticos, clase semántica verbal, tipo denotativo de los nombres deverbales, sentidos de WordNet nominales, entidades nombradas (NER), relaciones de correferencia (http://clic.ub.edu/corpus/es/ancora)[http://clic.ub.edu/corpus/es/ancora]

[2] https://nlp.lsi.upc.edu/freeling/node/1

necesario para ello evaluar cómo cada palabra se relaciona con las otras que están contenidas en una oración y así se revela la estructura sintáctica.

Los roles sintácticos principales de interés en la elaboración de esta Investigación son los sustantivos, adjetivos y verbos.

- Los sustantivos tienden a describir entidades y conceptos.
- Los verbos generalmente señalan eventos y acciones.
- Los adjetivos describen propiedades de las entidades

Igualmente dentro del POS se identifican otros roles sintácticos como los adverbios, nombres propios, interjecciones entre otros.

El POS es un procesamiento que sirve de insumo para la coocurrencia de palabras, que es una de las formas en que se representan los resultados de los *querys*.

En el estado del arte este etiquetado alcanza un 98 % de precisión.

2.6.1.4. Steaming:

Es el proceso en que se consigue el lema de una palabra, entendiendo que el lema es la forma que por convenio se acepta como representante de todas las formas flexionadas de una misma palabra.

Al buscar el lema se tiene presente la función sintáctica que tiene la palabra, es decir que se evalúa el contexto en el que ocurre. Una de las ventajas de aplicar esta técnica es que se reduce el vocabulario del Corpus y eso conlleva a que también se reduce el espacio de búsqueda en los documentos.

En el estado del arte este etiquetado alcanza un 96 % de precisión.

2.7. Minería de Texto:

La extracción de ideas útiles derivadas de textos mediante la aplicación de algoritmos estadísticos y computacionales, se conoce con el nombre de minería de texto, analítica de texto o aprendizaje automático para textos (text mining, text analytics, machine learning from text). Se quiere con ella representar el conocimiento en una forma más abstracta, detectar relaciones.

El uso de las técnicas de minería de texto ha ganado atención en recientes años motivado a las grandes cantidades de textos digitales que están disponibles. La minería de texto, o *text mining* surge para dar respuesta a la necesidad de tener métodos y algoritmos que permitan procesar estos datos no estructurados (Aggarwal and Zhai, 2012b) . Una vez que mediante el proceso de recuperación de información tenemos un cúmulo de textos, es posible que necesitemos dar un paso más allá y usar un conjunto de técnicas que nos permitan analizar y digerir estos textos, mediante la detección de patrones.

Uno de los desafíos al trabajar con textos es darles estructura para que resulte viable trabajar con ellos desde la perspectiva de procesos computacionales. Una de las primeras fases consiste en agrupar todos los textos en un Corpus. Posteriormente se procederá a conformar una matriz dispersa

de una alta dimensionalidad que se denominará *"Sparce Term-Document Matrix)* de tamaño *n X d,* donde *n* es el número total de documentos y *d* es la cantidad de términos (palabras distintas) presentes entre todos los documentos. Formalmente sabemos que la entrada *(i,j)* de nuestra matriz es la frecuencia (cantidad de veces que aparece) de la palabra *j* en el documento *i* .

El problema de la alta dimensionalidad de la matriz mencionada motiva ir aplicando otras técnicas que en principio puedan colaborar a reducirla por medio de los simplificar atributos, es decir, disminuyendo el vocabulario por ejemplo aplicando el algoritmo de Porter (stemming).

Igualmente a nivel de minería de texto se hace deseable poder contar con la identificación semántica de cada una de las palabras que conforman nuestro vocabulario, para así obtener representaciones que aportan un mayor significado. Los procesamientos inherentes al NLP mencionados anteriormente son insumo para la minería de texto.

En esta investigación se usará una técnica denomimada "Coocurrencia de Palabras" para la detección de patrones en los textos. Esto consiste en evaluar las palabras que coocurren dentro de un texto y se puede hacer con distintas granularidades. Por ejemplo: las palabras que coocurren una seguida de otra. También pudiesemos definir que coocurran pero dentro de la misma oración, o parrafo y así sucesivamente. Luego se podrá ver cuáles son las palabras que coocurren en distintos textos. En la figura 2.2 podemos ver la coocurrencia de palabras sobre unos textos obtenidos de los resumenes de la Escuela de Física de la Universidad Central de Venezuela.

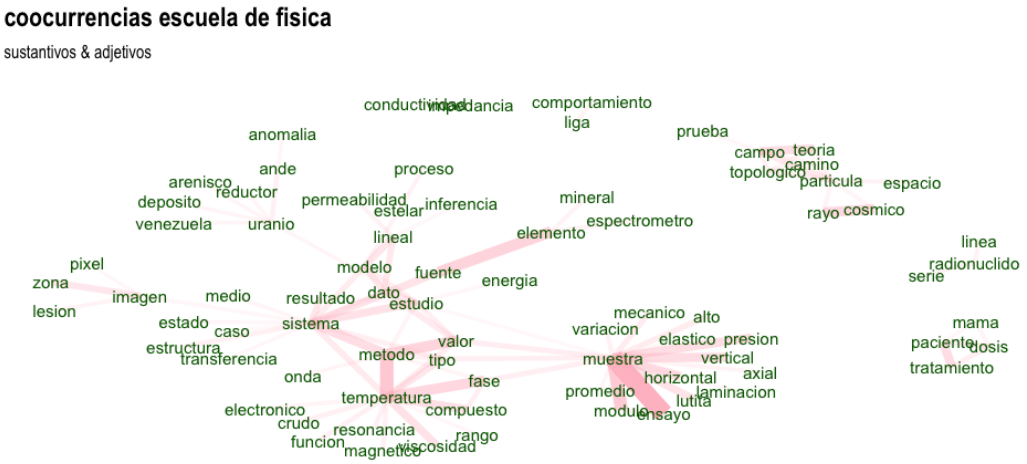


Figura 2.2: Coocurrencia de Palabras

2.8. Sistemas Distribuidos:

Los distintos procesos y componentes de la Solución propuesta han sido diseñados e implementados como un sistema distribuido y por eso haremos una mención a este tema.

Una definición formal que se le puede dar a los sistemas distribuidos es "cuando los componentes de hardware y/o sofware se encuentran localizados en una red de computadores y estos coordinan sus acciones solo mediante el pase de mensajes" (Coulouris, 2012).

Algunas de las principales características que tienen los sistemas distribuidos es la tolerancia a fallos, compartir recursos, concurrencia, ser escalables (Czaja, 2018) entre otras. Mencionamos estas en particular al ser propiedades que están presentes en la propuesta acá descrita.

1. Fiabilidad o la llamada tolerancia a fallos: en caso de fallar un componente del sistema los otros se deben mantener en funcionamiento.
2. Compartir recursos: un conjunto de usuarios pueden compartir recursos como archivos o base de datos.
3. Concurrencia: poder ejecutar varios trabajos en simultaneo.
4. Escalable: al ser incrementada la escala del sistema se debe mantener en funcionamiento el sistema sin mayores contratiempos.

2.8.1. Contenedores:

Un contenedor es una abstracción de una aplicación que se crea en un ambiente virtual, en el cual se encuentran "empaquetados" todos los componentes (sistema operativo, librerías, dependencias, etc.), que una aplicación necesita para poder ejecutarse. En su diseño se tiene presente que sean ligeros y que con otros contenedores pueden compartir el *kernel*, usando un sistema de múltiples capas, que también pueden ser compartidas entre diversos contenedores, ahorrando espacio en disco del *host* donde se alojan los contenedores.

El uso de los contenedores permite crear, distribuir y colocar en producción aplicaciones de software de una forma sencilla, segura y reproducible. También a cada contenedor se le puede realizar una asignación de recursos (memoria, cpu, almacenamiento) que garantice un óptimo funcionamiento de la aplicación que contienen.

Es importante señalar que el uso de esta tecnología añade un entorno de seguridad al estar cada contenedor en una ambiente isolado.

Para cada contenedor será necesario usar una imagen donde se definen las dependencias necesarias para su funcionamiento.

2.8.2. Orquestador:

Al tener diversos contenedores, donde cada uno aloja una aplicación distinta, puede resultar necesario que todos se integren en un sistema. Para que esta integración sea viable es necesario contar con un orquestador. Su uso permitirá lograr altos grados de portabilidad y reproducibilidad, pudiendo colocarlos en la nube o en centros de datos garantizando que se pueda hacer el *deploy* de forma sencilla y fiel a lo que se implementó en el ambiente de desarrollo.

En el caso de la Solución propuesta se adoptará el uso de Docker como orquestador y en el Capítulo que contiene la Propuesta Técnica 1.1 serán expuestas las funcionalidades de cada contenedor y se apreciará la integración que brindará el orquestador.

Explicación de la contenerización y el uso de *lahyers* en cada contenedor asociando a Docker.

2.9. Similitud de documentos:

Para poder realizar la recomendación de documentos, una de las técnicas que se usa es medir la similitud que presenta un documento con los otros contenidos en el corpus. Un ejemplo de esta técnica es el uso de la similitud coseno que se explica con esta formula.

$$\cos(\mathbf{t}, \mathbf{e}) = \frac{\mathbf{te}}{\|\mathbf{t}\|\|\mathbf{e}\|} = \frac{\sum_{i=1}^{n} \mathbf{t}_i \mathbf{e}_i}{\sqrt{\sum_{i=1}^{n} (\mathbf{t}_i)^2} \sqrt{\sum_{i=1}^{n} (\mathbf{e}_i)^2}} \tag{2.1}$$

En la formula *t* representa un documento y *e* representa otro documento. Ambos documentos se asumen que están en un espacio con *i* atributos, o dimensiones, y la intención es calcular un índice de similitud entre ambos documentos.

Este es uno de los métodos más usados para detectar similitudes en los textos, aunque existen otras formulas para el cálculo de la similitud como es el índice de jaccard.

El otro elemento de gran importancia en obtención de esta medición, es la representación que se haga del documento. Son distintas las técnicas que existen estando entre ellas la representación mediante "bolsas de palabras" o *bag of words*. Recientemente se han creado formas más complejas para la representación como lo son los *words embeddings* que son obtenidos mediante el entrenamiento de redes neuronales de aprendizaje profundo.

Estas formas de representación de textos serán investigadas más a fondo en futuras fases de la elaboración de este trabajo.

2.9.0.1. Tendencias actuales Sistemas de Información:

Si bien anteriormente las búsquedas de información dentro de un conjunto de textos se procesaba determinando la aparición o no de palabras o de frases dentro de un determinado texto este método ha ido evolucionando para llegar hoy en día a un elevado nivel de abstracción, donde a partir de la necesidad de obtener una determinada información, es decir, de aquello que necesitamos buscar que antes consistía en hacer *match* con un objeto de información, ahora hemos pasado de los motores de búsqueda (*search engines*) a motores de respuestas (*answering engines*) (Balog, 2018), donde el sistema ante un determinada consulta del usuario va a retornar una serie de resultados que pretenden enriquecer el resultado, mostrando la identificación de entidades, hechos y cualquier otro dato estructurado que esté de forma explícita, o incluso implícita, mencionado dentro de los textos que conforman el corpus.

Para lograr la generación de estos resultados se han tenido que conformar las llamadas bases de conocimiento o *knowlodege bases*, que son repositorios donde previo a la búsqueda de la información dentro del sistema, se logra ir estructurando la organización de la información alrededor de objetos o datos específicos que se denominan **entidades**. Estos conceptos y métodos se asocian directamente a los que también se proponen, de manera más amplia, en la denominada *web semántica* [3] .

[3] Se basa en la idea de añadir metadatos semánticos y ontológicos a la *World Wide Web*. Esas informaciones adicionales —que describen el contenido, el significado y la relación de los datos— se deben proporcionar de manera formal, para que así sea posible evaluarlas automáticamente por máquinas de procesamiento. Tomado de Wikipedia

De ejemplo de *knowledge bases* se puede mencionar a DBpedia, que se encuentra en la dirección www.www.dbpedia.org.com y es un proyecto en donde puede acceder a una red global y unificada de grafos de conocimiento, la cual cubre más de 20 idiomas y principalmente genera sus grafos de conocimiento a partir de las publicaciones del Proyecto Wikipedia.

2.9.0.2. Temas en proceso de investigación:

Por estar aún en curso esta investigación a continuación se mencionan algunos temas que se está evaluando incluir en este marco teórico

- Métodos usados para el almacenamiento o la indexación como crear agrupamientos *(clusters)* de aquellos documentos que compartan algunas características, por ejemplo, en la temática que aborden. Otra de las innovaciones que se están añadiendo a los sistemas de recuperación de información, es generar resúmenes con técnicas de procesamiento de lenguaje natural soportadas en el uso de arquitecturas de aprendizaje profundo *(deep learning)* .
- Resultados ante una búsqueda personalizados: al existir mecanismos como la sesión de usuario o las *cookies* que guardan información contextual, permitiendo que ante un mismo *query* en distintos equipos, los resultados sean distintos en función de la persona que hizo la búsqueda.
- En cuanto al estado del arte existen distintas librerías y modelos de representación de documentos, palabras y caracteres. Algunos de estos modelos son fastext, word2vec, GPT3. Para este momento aún estamos haciendo la evaluación del uso de ellos para nuestra investigación, por lo cual, sólo los mencionamos referencialmente.

Capítulo 3

Propuesta Técnica:

En este Capítulo se detalla la propuesta técnica de la Solución en la sección 3.1. Posteriormente se indica el Objetivo General 3.2 y los Objetivos Específicos 3.3 que se aspiran cubrir en el desarrollo. A continuación se muestra el esquema general de la aplicación 3.4, junto con el esquema de funcionamiento en los contenedores 3.4.2 y se describe el funcionamiento de cada contenedor.

Se realiza un análisis de factibilidad en la sección 3.5 para la implementación y se muestra un cronograma 3.6 de actividades.

3.1. Propuesta Técnica:

Crear una Solución que funcione como una aplicación web distribuida bajo la arquitectura cliente-servidor. Del lado del servidor se encontrarán contenedores que alojarán los distintos servicios necesarios para el funcionamiento.

Del lado del cliente se podrán formular los *querys* con múltiples atributos (la frase a buscar, la jerarquía académica, intervalo de fechas, facultad y/o escuela de adscripción).

La Solución permitirá generar procesos de extracción de información (*information retrieval*) y los resultados se mostrarán en tablas interactivas, gráficos y grafos de co-ocurrencias de palabras. Adicionalmente se indicarán recomendaciones de textos basados en la similitud que presente un documento con los otros.

Los textos del resumen de cada tesis con los cuales se conformará el *Corpus*, serán obtenidos y actualizados mediante técnicas de *web crawling* realizadas al repositorio Saber UCV.

Se opta por la estrategia de hacer el *web crawling* a los documentos contenidos en Saber UCV al no ser posible, al momento de realizar esta propuesta, la obtención de la base de datos de los documentos ahí alojados.

La Solución propuesta está diseñada para irse actualizando periódicamente incorporando los nuevos documentos que sean alojados en el repositorio Saber UCV.

Contará con un procedimiento que permite clasificar los documentos (tesis) por área académica, según donde haya efectuado estudios el autor del trabajo, al ser la tesis el requisito necesario para obtener el título de grado. Actualmente el repositorio Saber UCV no dispone de esta clasificación.

3.2. Objetivo:

Crear una Solución que permita realizar la clasificación de documentos y la búsqueda de información sobre los trabajos de investigación que se encuentran en el Repositorio SABER UCV usando técnicas de extracción de información (*information retrieval*).

Para el procesamiento de las búsquedas será usado un sistema distribuido con distintos componentes (contenedores) que permitan la ingesta, procesamiento y transformación de los datos, al igual que el alojamiento de los mismos en una base de datos.

3.3. Objetivos Específicos

- Querys multi atributo en distintas dimensiones: tiempo, jerarquías (pregrado, especializaciones, maestría y/o doctorado), facultad ,carreras o postgrados.
- A cada resultado que arroje la búsqueda se le debe asignar un nivel de relevancia. El conjunto de los textos recuperados se debe mostrar ordenadamente de mayor a menor relevancia. La asignación de la relevancia será hecha mediante una función que cuente con distintos parámetros.
- Los datos: textos, fechas, clasificaciones, y demás que sean necesarios para el funcionamiento de la Solución, deben registrarse en una base de datos estructurada y la ingesta y consulta se efectuará mediante un manejador de base de datos.
- Generar recomendaciones de textos basados en la similitud coseno que presente un documento con cada uno de otros que conforman el *Corpus*.
- En los documentos extraídos se debe contar con el enlace a los documentos que reposan en Saber UCV.
- Permitir la concurrencia de accesos al Sistema.
- La tolerancia a fallas en los contenedores.
- Contar con el certificado SSL para acceso seguro por parte de los visitantes.
- Procesar las coocurrencias de las palabras más comunes las cuales se deben visualizar mediante grafos.
- En la representación de coocurrencias de palabras mediante grafos, se debe poder filtrar documentos interactivamente al hacer *click* con el *mouse* sobre los arcos que unen un par de nodos, donde cada uno de estos representa la co-ocurrencia de una dupla de palabras. El *click* generará el evento para el query sobre los documentos que contienen esa determinada co-ocurrencia.

3.4. Esquema de la Aplicación WEB:

Podemos representar el Sistema y sus interacciones mediante un esquema con un elevado grado de abstracción. Ver figura 3.1.

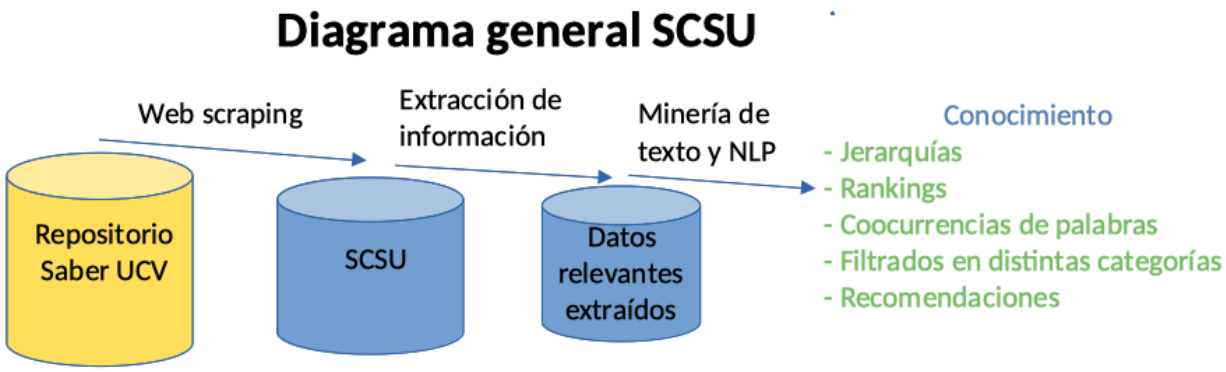


Figura 3.1: Esquema General

En la figura 3.1 se muestra como el SSCSU está diseñado para que mediante técnicas de *web crawling* se pueda extraer la información del repositorio Saber UCV. Posteriormente el sistema ante la consulta del usuario va a recuperar los documentos que sean relevantes, representando el conocimiento con la generación de jerarquías, rankings, coocurrencias de las palabras más mencionadas y recomendaciones de texto según similitudes.

El SSCSU se implementará mediante el uso de Docker. En la figura 3.2 se muestra el esquema que con los contenedores.

3.4.1. Cliente - Servidor:

La arquitectura **cliente-servidor** en esta Propuesta se basa en:

3.4.1.1. 1 - Cliente - Navegador web:

El acceso del cliente, el navegador web, realiza la petición al enlace que direcciona al servidor que aloja la Solución. Actualmente se cuenta con un prototipo al que se puede acceder en la dirección https://proyecta.me/.

El sistema no está diseñado para usarse desde dispositivos móviles, aunque igualmente es viable el acceso.

3.4.1.2. 2 - Servidor:

El prototipo del sistema requiere para funcionar al menos estos recursos:

- 2 CPU virtual
- 2 GB de memoria RAM
- 50 GB de disco duro

En el servidor se instala el software Docker sobre el cual se despliegan los siguientes contenedores:

3.4. ESQUEMA DE LA APLICACIÓN WEB:

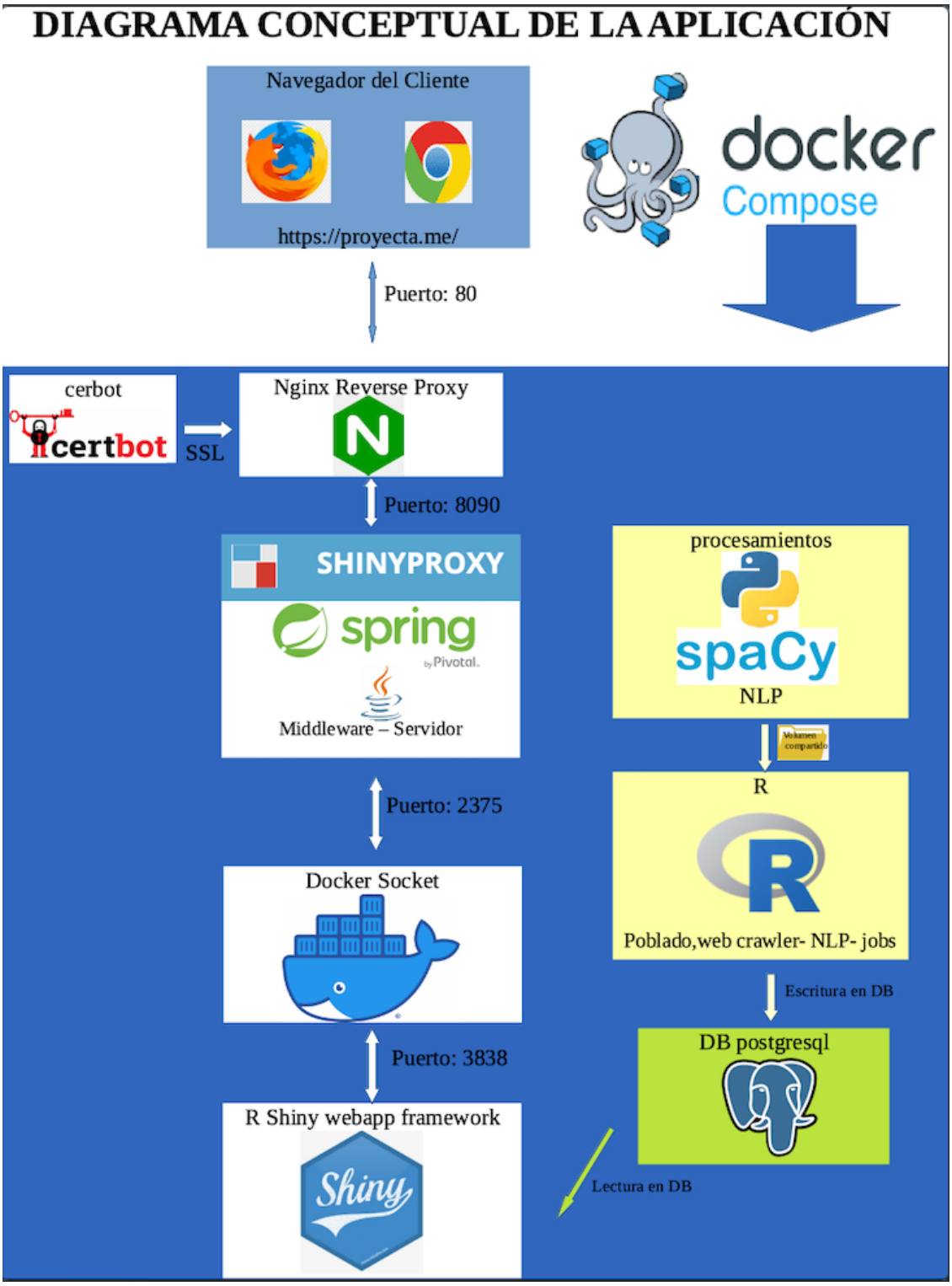


Figura 3.2: Diagrama Aplicación

3.4.2. Contenedores:

3.4.2.1. Docker Compose:

Funciona como un orquestador para correr aplicaciones distribuidas en múltiples contenedores usando un archivo en formato yml donde se establecen las imágenes, los puertos y los volúmenes que serán usados y compartidos por cada uno de los contenedores.

3.4.2.2. Contenedor con Nginx:

Es el servidor web/proxy inverso de código abierto que sirve para redireccionar las peticiones del puerto 80 al puerto 8090. Mediante este contenedor también se define el certificado SSL para permitir conexiones por el protocolo HTTPS.

Este contenedor fue generado desde una imagen oficial de NGINX que se encuentra en el ***docker hub*** [1] sin añadir ninguna capa (lahyer) adicional.

3.4.2.3. Contenedor con Cerbot:

Cerbot es una herramienta de código abierto que permite habilitar las conexiones mediante el protocolo HTTPS con el uso de un certificado "Let´s Encrypt". El uso de este certificado está asociado al uso de un dominio en el *deploy* de la aplicación.

Este contenedor fue generado desde una imagen de CERBOT del ***docker hub*** sin ninguna modificación posterior.

3.4.2.4. Contenedor con Shinyproxy:

Es una implementación del servidor "*Spring boo*t" que dará servicio a las aplicación web desarrolladas en *shiny* [2] 3.4.2.5. Con el uso de este *middleware* se obtienen las siguientes ventajas:

- Ante cada petición de acceso al servidor se despliega un *workspace* completamente aislado, es decir, un contenedor distinto. Las aplicaciones desarrolladas en *shiny* son *single threaded* y adoptar esta estrategia representa una ventaja, motivado a que se pueden controlar los recursos de memoria y cpu asignados a cada contenedor que se despliega.
- Permite establecer *login* en el uso de la aplicación y grupos de usuarios. También da soporte a distintos métodos de autenticación. Si bien en estos momentos la aplicación es de libre acceso, en algún momento se pudiese restringir y no sería necesaria ninguna modificación en la arquitectura, más allá de cambiar el archivo de configuración.
- Uso de una tecnología estable y probada.

[1] repositorio de imágenes de contenedores de docker.
[2] Shiny un framework para crear aplicaciones web en el lenguaje de programación R.

Es necesario destacar que originalmente *Shiny* como *framework* cuenta con su propio software que actúa como servidor, pero para tener acceso a ciertas funcionalidades es necesario pagar por el servicio directamente a la Fundación RStudio Software y usar el alojamiento que ellos proveen, no siendo todos los componentes de código abierto. Por ejemplo, el acceso mediante *login* no está disponible en la versión libre del *Shiny Server* sino en la *Shiny Server Professional*.

Ciertas configuraciones de librerías e incluso la propia contenerización de la aplicación, no es posible usando el servicio pago, así que la propuesta acá adoptada, si bien representa un mayor esfuerzo en la configuración, claramente implica que se obtienen una serie de ventajas, por todas las adaptaciones y control que es posible realizar al y sobre el sistema.

Este contenedor funciona en el puerto 2375 y fue generado desde la imagen del docker hub *Shinyproxy* sin ninguna modificación.

3.4.2.5. Contenedor con "*R Shiny Web App framework*":

En este contenedor es donde reposa la aplicación web con todas las librerías necesarias para generar las visualizaciones y remitir los *querys* al contenedor del manejador de la base de datos 3.4.2.6 . Como comentamos anteriormente, cada vez que ocurre desde el navegador del cliente una petición de acceso, desde el contenedor *shinyproxy* 3.4.2.4, se crea una replica de este contenedor con todos los elementos necesarios para que la app funcione correctamente.

En caso de presentar alguna falla, el sistema sería tolerante, porque se pueden seguir recibiendo peticiones que replicarían una imagen nueva del contenedor sin afectar al que presentase el fallo, o viceversa.

Desde este contenedor se realiza el acceso de lectura al contenedor que contiene PostgreSQL 3.4.2.6 donde reposa la base de datos que contiene los textos ya procesados.

Por los momentos no hay escritura de datos en las tablas, pero está contemplado que se registren los querys formulados en alguna tabla, junto con los documentos que el usuario revisa mediante las interacciones, para así generar métricas de calidad del sistema y del uso que se le da.

La imagen que se usa en este servidor fue definida a medida con todos los recursos necesarios.

En un posterior Capítulo donde se mostrará el proceso de desarrollo de la Solución, serán descritas todas las librerías que se encuentran incluidas en este contenedor.

Varios de los procesos que se ejecutan en este contenedor ocurren al momento de recibir un *query,* no obstante todos los procesos que puedan ser pre computados, se trata de ejecutarlos previamente en el contenedor "R Servicios" 3.4.2.7, para lograr así la disminución de los tiempos.

Funcionalidades principales: **ENTRADAS**

- Contiene un campo para la entrada de texto que generará el query.
- Contiene un selector para indicar si se quiere generar la coocurrencia de palabras
- Contiene tablas para seleccionar:

    1) Nivel académico del trabajo. Opciones (pregrado, especialización, maestría, doctorado).
    2) Facultad o Centro de adscripción. Opciones: 11 Facultades más un centro (CENDES).

3) Nombre del pregrado o postgrado. En total son 412 las opciones.

Cada una de las tablas anteriores se actualiza según se vayan seleccionando las relaciones y la disponibilidades. Por ejemplo, al seleccionar pregrado solo se mostrarán los nombres de las carreras de pregrado, pero si se selecciona también el nombre de la Facultad, sólo se mostrarán las carreras de pregrado dentro de la Facultad seleccionada. Para una determinada tabla también se permiten selecciones múltiples dando una total flexibilidad al momento de ejecutar los *querys*.

SALIDAS - Ante el *query* se genera:

- Una tabla creada con la librería *reactable*. En la misma se muestra cada uno de los documentos recuperados con los distintos atributos disponibles: autor, fecha, palabras claves, texto resumen. El orden en que aparece está generado con una función de ranking. Adicionalmente se muestra un enlace al repositorio Saber UCV que es donde se encuentra alojado el respectivo trabajo (el documento en PDF). Igualmente se presentan los textos que tienen mayor similitud con el texto seleccionado.
- Un gráfico con la frecuencia por año de los trabajos extraídos mediante el *query*. El gráfico se generá con la librería *apexcharter* que es un wraper para Javascript, por lo cual el gráfico tiene ciertas interactividades. Con el *hoover* muestra el valor de cada columna.
- Gráfico de coocurrencia interactivo de palabras: se genera mediante la librería de *VisNetwork* que también es un *wrapper* de javascript. Este gráfico permite seleccionar un arco de unión entre dos palabras coocurrentes. Al realizar la selección se filtran un subconjunto de los documentos que contienen ambas palabras representadas por nodos. Los documentos filtrados se mostrarán en una tabla contigua, también generada en *reactable*, donde sólo se incluye el texto resumen de cada trabajo. En un próximo Capítulo a desarrollar será mencionado el diseño y funcionamiento a detalle de esta visualización.
- Gráfico de coocurrencia estático de palabras: mediante la librería *ggraph* son generados un par de gráficos. El primero con la misma coocurrencia de palabras vista anteriormente pero esta vez generada en una visualización estática. El segundo gráfico también muestra la coocurrencia, pero solo de palabras que ocurren una seguida de otra dentro del texto resumen.

Con la librería *UdPipe* se generan las estructuras de datos necesarias para generar los grafos (arcos y nodos).

3.4.2.6. Contenedor con PostgreSQL:

En este contenedor tenemos una imagen de PostgreSQL versión 13.3. No fue realizada ninguna otra modificación distinta a la definición de usuarios y poblado desde base de datos incluyendo un volumen compartido para garantizar que tengamos “datos persistentes. Este contenedor recibe consultas del contenedor”R Shiny Web App framework” 3.4.2.5 y escritura desde el contenedor “R imagen Servicios” 3.4.2.7.

En este contenedor ocurre la indexación de la base de datos y la generación del ranking al procesar el resultado con la función tsrank [^propuesta-3]. Se cuenta con dos tablas no relacionales.

[^propuesta-3]: esta función mide la relevancia de los documentos para una consulta en particular, de modo que cuando haya muchas coincidencias, las más relevantes puedan mostrarse primero tomando en cuenta la información léxica, de proximidad y estructural del documento (título, cuerpo del documento, etc).

En una tabla se encuentra la identificación del documento, la fecha de creación y propiamente los textos que mediante el Tsvector [3] almacena el título, el resumen, el autor y las palabras claves.

La otra tabla contiene otro procesamiento que se le hace a los textos, clasificando cada una de las palabras mediante el *part of speach* , el documento al que está asociada y el lemma.

El TSvector es la estructura de datos que permite la búsqeda de texto completa (*Full Text Seach*) mediante la función de *postgresql* denominada *tsquery*. En un futuro capítulo a desarrollar será mostrado el diseño de las tablas con sus campos.

3.4.2.7. Contenedor con "R Imagen Servicios":

En este contenedor se creó una imagen con todos los servicios necesarios para realizar el web crawling, procesamientos de textos y archivos descargados desde Saber.UCV, poblado y llamar a las funciones para la escritura en la base de datos, programación de rutinas de actualización (cron) y demás servicios que se debe ejecutar periódicamente. La imagen usada es la del proyecto ***Rocker*** (Boettiger and Eddelbuettel, 2017) ,la cual es una versión ampliamente probada y optimizada en, y por, la comunidad de usuarios de R.

Posteriormente serán descritas todas las librerías que fueron añadidas mediante una capa (lahyer) a este contenedor. Por los momentos podemos citar que se encuentran agregadas las siguientes:

3.4.2.7.1. Text Mining y NLP: Generalmente las distintas librerías que permiten realizar procesos de *Natural Language Processing* también hacen procesos de *Text Mining* parcial o totalmente. En la investigación fueron evaluadas múltiples librerías como *Spacy, Quanteda, OPENLP, CoreNLP, Freeling y Udpipe*.

En una futura entrega se hará una breve mención a cada una y la razón por la cual se adoptó Spacy para varios de los procesos de NLP.

Para ejecutar *Spacy* es necesario usar los archivos que se encuentran en el contenedor *"Python Spacy"* , 3.4.2.8 donde está instalada la libraría *Spacy* que corre en *Python*.

Procesos que se ejecutan llamando al contenedor "Python Spacy": Tokenización, POS y Lematización.

También en este contenedor se realizan los siguientes procesos:

1. Poblado base de datos: se hace el *web crawling* para el poblado inicial y adicionales de la base de dato, con los textos de los Resúmenes.
2. Clasificación de los documentos: mediante una rutina compuesta por varios algoritmos serán clasificados los distintos trabajos según lo antes expuesto.

[3] El Tsvector es una función de postgresql que crea una estructura de datos que es una lista ordenada de distintos lexemas, que son palabras que se han normalizado para fusionar diferentes variantes de la misma palabra mediante el algoritmo de Porter.

3. Cálculo de la Similitud de los documentos: cuando se ha realizado la lematización de las palabras se procede a generar una matriz de tipo Td-Idf (term document- inverse document frecuency), que sirve de insumo para el cálculo de similitud entre los documentos. Este cálculo de similitud se realiza con la librería Quanteda.texstats y se usa la medida de similitud coseno, ya que varios autores la sitúan como una de las mejores formas de comparar la similitud entre un documento y otro.

3.4.2.8. Contenedor con "Python Spacy":

Se creó una imagen que contiene un ubuntu con python, spacy y el modelo de Spacy es_core_news_sm . Su función es que mediante un volumen compartido pueda ser invocado desde el contenedor "R Imagen Servicios" 3.4.2.7 para así realizar los procesamientos de NLP antes descritos.

3.5. Factibilidad:

Para la propuesta del SCSU se hizo una evaluación de la factibilidad del desarrollo del proyecto que consistió en hacer pruebas de ***web crawling*** sobre el repositorio Saber UCV al no ser viable la obtención del conjunto de datos por otro medio. Igualmente se realizaron pruebas sobre el hardware disponible con arquitecturas similares a la que se propondrá más adelante en nuestro desarrollo.

Estas pruebas fueron exitosas por lo cual no se estima que exista algún factor que impida la implementación del Sistema acá expuesto.

3.6. Cronograma:

El cronograma de desarrollo propuesto es el siguiente:

Cuadro 3.1: Cronograma de desarrollo de la Solución

ID	Tarea	cantidad	medida	predecesora
1	web craling inicial	2	semanas	0
2	diseño algoritmo clasificación	3	semanas	1
3	diseño componentes de la aplicación	4	semanas	1
4	Implementación de componentes	4	semanas	3
5	Pruebas contenedores Docker	1	semanas	4
6	Pruebas de software	1	semanas	5
7	implementación en servidor	1	semanas	6
8	Elaboración de trabajo especial de grado	8	semanas	7

3.6. CRONOGRAMA:

ID Tarea	cantidad medida	predecesora
TOTAL	**24 semanas**	

Bibliografía

Aggarwal, C. C. (2018). *Machine Learning for Text*. Springer International Publishing : Imprint: Springer, Cham, 1st ed. 2018 edition. DOI: 10.1007/978-3-319-73531-3.

Aggarwal, C. C. and Zhai, C., editors (2012a). *Mining text data*. Springer, New York Heidelberg. DOI: 10.1007/978-1-4614-3223-4.

Aggarwal, C. C. and Zhai, C., editors (2012b). *Mining text data*. Springer, New York Heidelberg. DOI: 10.1007/978-1-4614-3223-4.

Balog, K. (2018). *Entity-Oriented Search*. Number 39 in The Information Retrieval Series. Springer International Publishing : Imprint: Springer, Cham, 1st ed. 2018 edition. DOI: 10.1007/978-3-319-93935-3.

Boettiger, C. and Eddelbuettel, D. (2017). An introduction to rocker: Docker containers for r. *The R Journal*, 9(2):527–536.

Coulouris, G. F., editor (2012). *Distributed systems: concepts and design*. Addison-Wesley, Boston, 5th ed edition.

Czaja, L. (2018). *Introduction to Distributed Computer Systems: Principles and Features*. Number 27 in Lecture Notes in Networks and Systems. Springer International Publishing : Imprint: Springer, Cham, 1st ed. 2018 edition. DOI: 10.1007/978-3-319-72023-4.

Eisenstein, J. (2019). *Introduction to natural language processing*. Adaptive computation and machine learning. The MIT Press, Cambridge, Massachusetts.

Fredkin, E. (1960). Trie memory. *Communications of the ACM*, 3(9):490–499.

Goodrich, M. T., Tamassia, R., and Goldwasser, M. H. (2013). *Data structures and algorithms in Python*. Wiley, Hoboken, NJ.

Guevara, J. (2015). *Desarrollo de un Motor de Búsquedas para la recuperación de documentos académicos de la Facultad de Ciencias*. PhD thesis. Accepted: 2015-05-25T14:29:29Z.

Hernández Orallo, J., Ramírez Quintana, M., and Ferri Ramírez, C. (2004). *Introducción a la minería de datos*. Pearson Educación, Madrid. OCLC: 989816167.

Knuth, D. E. (1997). *The art of computer programming*. Addison-Wesley, Reading, Mass, 3rd ed edition.

Kraft, D. H. and Colvin, E. (2017). Fuzzy information retrieval. *Synthesis Lectures on Information Concepts, Retrieval, and Services*, 9(1):i–63.

Lehman, R. (2007). Learning object repositories. *New Directions for Adult and Continuing Education*, 2007(113):57–66.

Mahapatra, A. K. and Biswas, S. (2011). Inverted indexes: Types and techniques. *International Journal of Computer Science Issues*, 8.

Manning, C. D., Raghavan, P., and Schütze, H. (2008). *Introduction to information retrieval*. Cambridge University Press, New York. OCLC: ocn190786122.

Rodríguez Laguna, J. (2016). *Sistema de recomendación para el buscador académico venezolano*. PhD thesis. Accepted: 2016-03-16T17:26:20Z.

Sánchez, J. (2008). *Elaboración de un prototipo de buscador de documentos académicos de la Facultad de Ciencias*. PhD thesis, Caracas, Venezuela.

Zhai, C. and Massung, S. (2016). *Text data management and analysis: a practical introduction to information retrieval and text mining*. Number #12 in ACM Books. ACM Books, New York, first edition edition. OCLC: ocn957355971.

Zhang, J. (2008). *Visualization for information retrieval*. The information retrieval series. Springer, Berlin. OCLC: ocn173239487.