



Universidad Católica  
**San Pablo**

# Ciencia de la Computación

Motor de Búsqueda Hadoop

## Big Data

Docente: Alvaro Mamani Aliaga

Casusol Escalante, Joaquin  
Delgado Vidal, Paolo  
Torres Sotomayor, Rodrigo

9no SEMESTRE

2024

“Los alumnos declaran haber realizado el presente trabajo de  
acuerdo a las normas de la Universidad Católica San Pablo”

## **Introducción**

Este informe presenta la implementación de un motor de búsqueda utilizando técnicas avanzadas como el índice invertido y el algoritmo PageRank. El índice invertido optimiza la velocidad de búsqueda al indexar términos y sus ubicaciones en documentos, mientras que PageRank evalúa la importancia de las páginas web en función de su popularidad y relevancia. Este enfoque combinado busca mejorar la precisión y la rapidez de los resultados de búsqueda, proporcionando así una herramienta efectiva para la recuperación de información en grandes conjuntos de datos en línea.

## **Algoritmo de Índice Invertido con Hadoop**

Primero se implementó un proceso de indexación invertida utilizando Apache Hadoop, que es un framework de procesamiento distribuido para grandes conjuntos de datos. La función principal del código es realizar el conteo de palabras por documento y generar un índice invertido que relacione cada palabra con los documentos en los que aparece y su frecuencia.

El proceso se divide en dos etapas principales: la fase de Map y la fase de Reduce. En la fase de map, la clase `InvertedIndexMapper` recibe como entrada un conjunto de documentos de texto y emite pares clave-valor donde la clave es la palabra y el valor es el nombre del documento seguido de un 1 para indicar una ocurrencia.

El paso siguiente es la fase de Reduce, donde la clase `InvertedIndexReducer` suma los recuentos de palabras para cada documento y genera la salida final, que consiste en la palabra seguida de una lista de documentos y sus respectivas frecuencias separadas por comas.

La configuración del trabajo Hadoop se realiza en el método `main`, donde se definen las clases mapper y reducer, así como el formato de entrada y salida de datos.

## **Algoritmo de Page Rank**

El segundo algoritmo a implementar fue un PageRank bajo el contexto de procesamiento distribuido. El algoritmo de PageRank es utilizado por los motores de búsqueda para clasificar páginas web en función de su importancia en la web global. En este caso, se aplica el algoritmo a una colección de archivos de texto para

determinar la importancia relativa de cada archivo en función de su "conexión" (link) con otros archivos en la colección.

El código recibe un archivo de texto procesado previamente por Hadoop, que es el índice invertido, con esta información, ya es muy fácil calcular el PageRank.

El cálculo del PageRank se realiza mediante iteraciones que actualizan las puntuaciones de importancia de cada archivo. Se utiliza el índice invertido para determinar qué documentos están relacionados entre sí. Esto es lo que sucede:

- Iteración sobre cada documento: Para cada documento en el directorio, se obtiene una lista de palabras clave que aparecen en ese documento particular, extrayendo esta información del índice invertido.
- Identificación de documentos relacionados: Para cada palabra clave que aparece en un documento, el código examina todos los otros documentos en los que también aparece esa palabra clave. Esto se hace accediendo a la entrada correspondiente en el índice invertido para esa palabra clave.
- Evitar duplicados y auto-referencias: El código se asegura de que un documento no se vincule consigo mismo y que cada relación se registre una sola vez, evitando así la autorreferencia y duplicación en las listas de conexiones.

El proceso de cálculo se repite hasta que la diferencia entre las puntuaciones de PageRank en iteraciones consecutivas sea menor que un valor epsilon predefinido o hasta que se alcance un número máximo de iteraciones, lo que garantiza la convergencia del algoritmo y nos evita bucles infinitos.

Finalmente, los resultados del PageRank se escriben en un archivo de texto junto con los resultados del índice invertido. Esto proporciona una visión general de la importancia de cada archivo en la colección en función de su conexión con otros archivos y las palabras clave relevantes encontradas en ellos. Este archivo txt será el que conectaremos con nuestra interfaz para poder realizar la búsqueda o consulta de la palabra deseada.

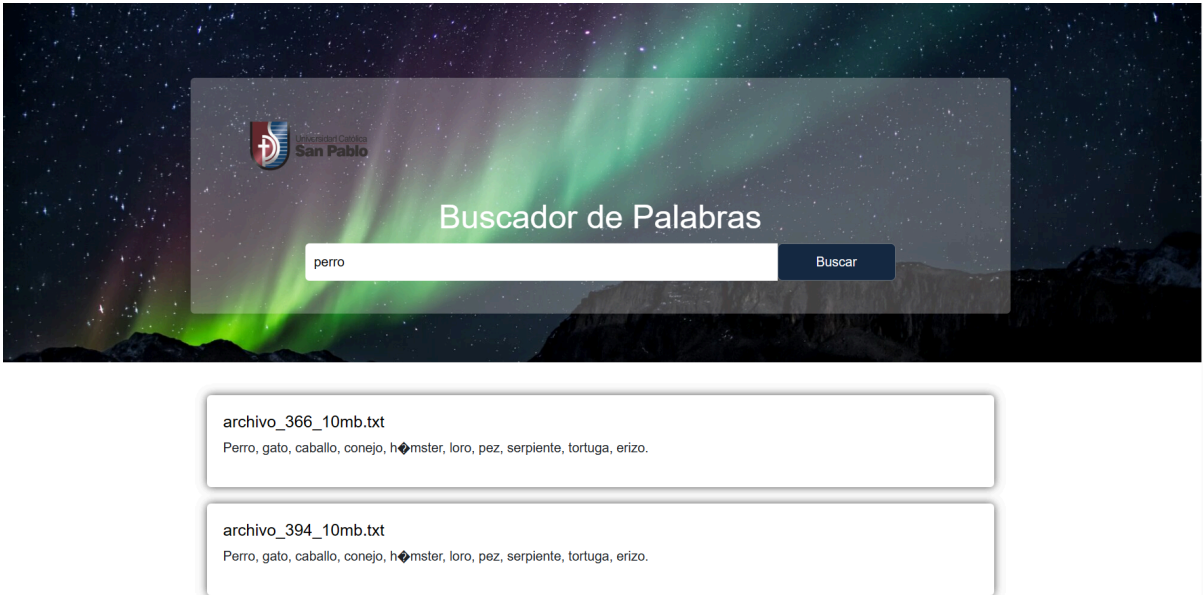
## **Problemas y desafíos**

Durante la fase de implementación y ejecución del código surgieron algunos problemas relacionados con Hadoop, siendo uno de los desafíos principales la configuración y la disponibilidad de los recursos para su uso, ya que la conexión entre los nodos del clúster era inestable y complicada de mantener en funcionamiento constante. Además, se debía tener en cuenta los permisos de

acceso y la configuración de red para garantizar la comunicación adecuada entre los nodos del clúster, lo que dificulta más su uso.

Otro problema que surgió fue durante el cálculo del PageRank debido al gran tamaño de la colección de archivos y la complejidad de las relaciones entre ellos. Si la colección es demasiado grande o si los archivos tienen enlaces demasiado complejos entre sí, el proceso de cálculo del PageRank podría consumir una cantidad significativa de recursos computacionales y tiempo de ejecución, o en caso de ser demasiado simples el cálculo del PageRank no mostraría diferencias significativas entre los documentos. Para ello, se consideró realizar una evaluación previa sobre los datos, para poder verificar la partición óptima de los mismos, así como también el ajuste de parámetros dentro de la implementación de PageRank, con el fin de poder optimizar al máximo la eficiencia y capacidad del algoritmo.

### **Buscador:**



The image shows a web application interface for a word search. At the top, there is a header with the logo of Universidad Católica San Pablo. Below the header, the title "Buscador de Palabras" is displayed. A search bar contains the text "perro", and a "Buscar" button is next to it. Below the search bar, two results are shown, each in a separate box. The first box is for "archivo\_366\_10mb.txt" and the second box is for "archivo\_394\_10mb.txt". Both boxes contain the same list of animals: "Perro, gato, caballo, conejo, hamster, loro, pez, serpiente, tortuga, erizo."

Se realiza una consulta en el archivo `inverted_index` para buscar los documentos relacionados con la palabra buscada:

```

<script>
function search() {
  const searchTerm = document.getElementById('searchInput').value.toLowerCase();
  const searchResults = document.getElementById('searchResults');
  searchResults.innerHTML = ''; // Limpiar resultados anteriores

  // Leer el archivo inverted_index.txt
  fetch('inverted_index.txt')
    .then(response => response.text())
    .then(text => {
      console.log('Contenido de inverted_index.txt:', text); // Depuración

      const data = {};
      text.split('\n').forEach(line => {
        const parts = line.split(':');
        if (parts.length >= 2) {
          const word = parts[0].trim();
          const files = parts.slice(1).join(':').trim().replace(/[{}]/g, '').split(', ').map(item => item.split('=')[0]);
          data[word] = files;
        }
      });

      console.log('Datos de inverted_index.txt:', data); // Depuración
    })
}

```

Se ordenan los documentos relacionados aplicando el pagerank de esos documentos.

```

    .then(text => {
      // Leer el archivo ranking.txt
      // Leer el archivo ranking.txt
      fetch('pagerank_results.txt')
        .then(response => response.text())
        .then(rankingText => {
          console.log('Contenido de ranking.txt:', rankingText); // Depuración

          const rankingOrder = rankingText.trim().split('\n');
          console.log('Orden de ranking.txt:', rankingOrder); // Depuración

          rankingOrder.forEach(line => {
            const fileName = line.split(':')[0].trim(); // Obtener solo el nombre del archivo
            console.log('Nombre del archivo:', fileName); // Depuración

            if (data.hasOwnProperty(searchTerm) && data[searchTerm].includes(fileName)) {
              // Crear un div para cada archivo
              const fileDiv = document.createElement('div');
              fileDiv.classList.add('file-info');

              // Crear un elemento para mostrar el nombre del archivo
              const fileNameElement = document.createElement('h5');
              fileNameElement.textContent = fileName;
              fileNameElement.title = "Ver Archivo";
              fileNameElement.style.cursor = 'pointer'; // Agregar estilo para indicar que es clickeable
              fileNameElement.addEventListener('click', () => openFile(fileName));
              fileDiv.appendChild(fileNameElement);
            }
          });
        })
    })
}

```

Mostramos una parte de los documentos en el buscador:

```

fileNameElement.textContent = fileName;
fileNameElement.title = "Ver Archivo";
fileNameElement.style.cursor = 'pointer'; // Agregar estilo para indicar que es clickeable
fileNameElement.addEventListener('click', () => openFile(fileName));
fileDiv.appendChild(fileNameElement);

// Leer las primeras 2 líneas de cada archivo
fetch(fileName)
  .then(response => response.text())
  .then(text => {
    const lines = text.split('\n').slice(0, 1); // Obtener las primeras 2 líneas
    lines.forEach(line => {
      const p = document.createElement('p');
      p.textContent = line;
      fileDiv.appendChild(p);
    });
    // Agregar el div del archivo al contenedor de resultados
    searchResults.appendChild(fileDiv);
  })
  .catch(error => console.error('Error al cargar el archivo:', error));
});
});
}).catch(error => console.error('Error al cargar el archivo ranking.txt:', error));
}).catch(error => console.error('Error al cargar el archivo inverted_index.txt:', error));

```

### **Link repositorio:**

<https://github.com/RodATS/BigData.git>