TP BigData Report

Supervised by : Dr John AOGA

Group 8 member :

AGBOTON Josué Esperance

NSELE KISAMBU Grace

AKADIRI Faiçole

Topic: MapReduce implementation for analyzing web graph structures, including PageRank calculation of web pages (imagining the realization of an efficient search engine using MapReduce and PageRank).

Introduction

In this report, we describe how we modeled and solved the PageRank calculation problem for a set of web pages using the MapReduce programming model. We explain the various stages of the process, including data collection, pre-processing, web graph construction, PageRank calculation, page indexing and user interface. We also highlight how MapReduce facilitates parallel execution to improve system efficiency and scalability.

A- Let's explain how we've modeled our problem in the sense of parallel execution

1. Data collection and pre-processing

To begin with, we collected HTML files representing web pages. Each HTML file contains links to other pages, forming a graph of connections between pages..

- Pre-treatment step :
    - Reading HTML files from a directory.
    - Extract text and links from each HTML file using BeautifulSoup.

- Standardize text by removing accents and converting to lower case to ensure case insensitivity.

```
import glob
from bs4 import BeautifulSoup

def read_html_files(directory):
    pages = {}
    page_contents = {}
    all_links = set()
    for filepath in glob.glob(f"{directory}/*.html"):
        with open(filepath, 'r', encoding='utf-8') as file
            soup = BeautifulSoup(file, 'html.parser')
            page_name = os.path.basename(filepath)
            body_text = soup.body.get_text(separator=' ', 
            page_contents[page_name] = body_text
            links = [a['href'] for a in soup.find_all('a', 
            pages[page_name] = links
            all_links.update(links)
    return pages, page_contents, all_links
```

2. Building the Web Graph

We've built a web graph where each node represents a web page and each edge represents a link from the source page to a destination page. The graph is modeled as a dictionary in Python.

3. PageRank calculation

PageRank calculation is performed using an iterative algorithm, modeled with the MapReduce paradigm to exploit parallel execution.

- **Map Phase :**
  - Each mapping task reads a web page and issues a key-value pair for each outgoing link, representing the PageRank contribution of the source page to the destination page.

```
def map_page_rank(pages, page_ranks):
contributions = defaultdict(float)
```

```
for page, links in pages.items():
num_links = len(links)
if num_links > 0:
for link in links:
contributions[link] += page_ranks[page] / num_links
return contributions
```

Reduce Phase :

- The PageRank contributions of all source pages for a destination page are combined to calculate the new PageRank for that page.

```
def reduce_page_rank(contributions, damping_factor=0.85, n
    new_ranks = {page: (1 - damping_factor) / num_pages fo
    for page, contribution in contributions.items():
        new_ranks[page] += damping_factor * contribution
    return new_ranks
```

Iterations :

- The MapReduce process is repeated iteratively until the PageRank values converge on a stable solution.

```
def calculate_page_rank(pages, all_links, iterations=10):
    page_ranks = initialize_page_ranks(pages, all_links)
    num_pages = len(page_ranks)
    for _ in range(iterations):
        contributions = map_page_rank(pages, page_ranks)
        page_ranks = reduce_page_rank(contributions, num_p
    return page_ranks
```

4. Page indexing

We've built an inverted index to enable efficient searching of web pages based on their content. Each word is associated with a list of pages containing that word.

```
def create_inverted_index(page_contents):
    inverted_index = defaultdict(list)
    for page, content in page_contents.items():
        words = content.split()
        for word in words:
            normalized_word = unidecode(word.lower())
            inverted_index[normalized_word].append(page)
    return inverted_index
```

5. User interface

The user interface is built with Streamlit, allowing users to enter a search query and obtain a list of web pages ranked by their PageRank.

```
import streamlit as st

st.set_page_config(page_title="Simple Search Engine", page_
st.title('🔍 Simple Search Engine with PageRank')

directory = 'data'
if os.path.exists(directory):
    st.success("Directory exists. Ready to search!")
    pages, page_contents, all_links = read_html_files(dire
    page_ranks = calculate_page_rank(pages, all_links)
    inverted_index = create_inverted_index(page_contents)

    st.write("---")
    st.subheader("Search")
    query = st.text_input('Enter your search query:')

    if st.button('Search'):
        if query:
            results = search(query, inverted_index, page_r
            if results:
                st.write('### Search Results:')
                for page, rank in results:
                    percentage_rank = round(rank * 100, 2)
                    st.write(f"- **{page}**: {percentage_r
            else:
                st.warning('No results found.')
else:
    st.error(f'The directory {directory} does not exist. P
st.write("---")
st.write("Developed using Streamlit and the PageRank algor
```

B- Let's explain what our mappers and reducers represent

In the context of PageRank calculation and web page indexing using the MapReduce model, the concepts of mappers and reducers are used to divide and process data in a parallel and distributed way. Here's what each of these components represents:

Mapper

A mapper is a function or process that is responsible for processing part of the input data and producing intermediate key-value pairs. In the context of PageRank and web page indexing :

1. Mapper function :
    - Each mapper is responsible for processing a specific web page or set of pages.
    - For PageRank calculation, each mapper analyzes a web page and issues PageRank contributions to other related pages.
    - For page indexing, each mapper processes the textual content of a page and issues key-value pairs, where the key is a standardized word and the value is a reference to the page containing that word.
2. **Example:**
    - For PageRank calculation, a mapper could analyze a web page and issue PageRank contributions to pages linked to it, taking into account the number of outgoing links and the current PageRank of the source page.
    - For page indexing, a mapper could analyze the textual content of a web page, extract all significant words after normalization (in lower case, without accents), and issue key-value pairs where the key is a word and the value is the page name.

Reducer

A reducer is a function or process that aggregates and processes the intermediate key-value pairs produced by mappers. It consolidates these pairs to produce a set of final results.

1. **Reducer function:**
    - Each reducer receives a group of intermediate key-value pairs that share the same key.
    - In the PageRank calculation, the reducers aggregate the PageRank contributions received for each destination page, calculating the new PageRank for each page.
    - For page indexing, reducers aggregate all references to pages containing the same word, creating a list of pages associated with that word.
2. **Example:**
    - In PageRank calculation, a reducer could aggregate all PageRank contributions for a particular landing page, apply the PageRank formula and produce the new PageRank for that page.
    - For page indexing, a reducer could aggregate all references to the same web page (or several pages) that contain a specific word, creating a list of pages where that word appears.

In summary, mappers and reducers in the context of MapReduce play essential roles in data transformation and aggregation to solve complex problems such as PageRank calculation and web page indexing in an efficient and scalable way.

C- Here is a detailed step-by-step explanation of the code:

Importing libraries

```
import os
import glob
from collections import defaultdict
from bs4 import BeautifulSoup
```

```
import streamlit as st
from unidecode import unidecode
```

- `os` and `glob`: Used for file manipulation and searching for HTML files in a specific directory.
- `defaultdict`: A variant of the basic dictionary that returns a default value when a key does not exist.
- `BeautifulSoup`: Used to parse and extract information from HTML files.
- streamlit`: Used to create an interactive web interface.
- `unidecode`: Used to normalize text by removing accents and putting everything in lower case.

Function `read_html_files(directory)`

This function reads HTML files from a specified directory, and extracts links and text content from the pages.

```
def read_html_files(directory):
    pages = {}
    page_contents = {}
    all_links = set()  # Utiliser un ensemble pour éviter
    for filepath in glob.glob(os.path.join(directory, '*.h
        with open(filepath, 'r', encoding='utf-8') as file
            soup = BeautifulSoup(file, 'html.parser')
            page_name = os.path.basename(filepath)

            # Extraire le texte du body
            body_text = soup.body.get_text(separator=' ',
            page_contents[page_name] = body_text

            # Extraire les liens
            links = [a['href'] for a in soup.find_all('a',
            pages[page_name] = links
            all_links.update(links)  # Ajouter tous les li
    return pages, page_contents, all_links
```

- `pages` : A dictionary where each key is the name of the page and each value is a list of outgoing links.
- `page_contents` : A dictionary where each key is the name of the page and each value is the textual content of the page.
- `all_links` : A set of all extracted links to avoid duplication.
- Using `BeautifulSoup` to parse HTML files and extract body text and links.

Function `initialize_page_ranks(pages, all_links)`

This function initializes PageRanks for each extracted link.

```
def initialize_page_ranks(pages, all_links):
    num_pages = len(all_links) if all_links else 1
    page_ranks = {page: 1.0 / num_pages for page in all_li
    for links in pages.values():
        for link in links:
            if link not in page_ranks:
                page_ranks[link] = 1.0 / num_pages
    return page_ranks
```

- `page_ranks` : A dictionary where each key is a page and each value is the page's initial PageRank, evenly distributed across all pages.

Function `map_page_rank(pages, page_ranks)`

This function implements the mapping phase of the PageRank algorithm.

```
def map_page_rank(pages, page_ranks):
    contributions = defaultdict(float)
    for page, links in pages.items():
        num_links = len(links)
        if num_links > 0 and page in page_ranks:  # Vérifi
            for link in links:
                contributions[link] += page_ranks[page] /
    return contributions
```

- `contributions` : A dictionary where each key is a destination page and each value is the sum of the PageRank contributions of all source pages pointing to it.

Function `reduce_page_rank(contributions, damping_factor=0.85, num_pages=1)`

This function implements the reduce phase of the PageRank algorithm.

```
def reduce_page_rank(contributions, damping_factor=0.85, n
    new_ranks = {page: (1 - damping_factor) / num_pages fo
    for page, contribution in contributions.items():
        new_ranks[page] += damping_factor * contribution
    return new_ranks
```

- `iterations`: Number of iterations required for PageRanks to converge.

Function `create_inverted_index(page_contents)`

This function builds an inverted index from the textual content of pages.

```
def create_inverted_index(page_contents):
    inverted_index = defaultdict(list)
    for page, content in page_contents.items():
        words = content.split()
        for word in words:
            normalized_word = unidecode(word.lower())  # N
            inverted_index[normalized_word].append(page)
    return inverted_index
```

- `inverted_index`: A dictionary where each key is a standardized word and each value is a list of pages containing that word.

Function `search(query, inverted_index, page_ranks)`

This function performs a search based on the inverted index and PageRanks.

```
def search(query, inverted_index, page_ranks):
    terms = [unidecode(term.lower()) for term in query.spl
    results = defaultdict(float)
    for term in terms:
        if term in inverted_index:
            for page in inverted_index[term]:
                results[page] += page_ranks.get(page, 0)
    return sorted([(page, rank) for page, rank in results.
```

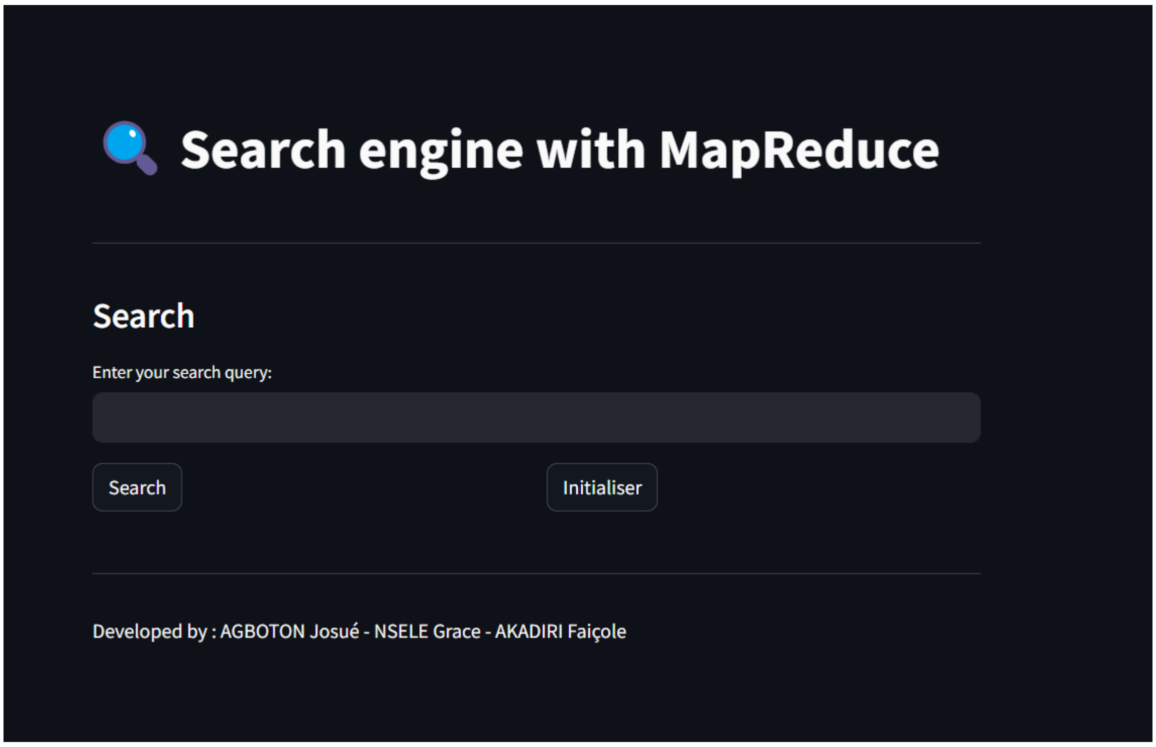
- `terms`: The search query is normalized and broken down into individual terms.
- `results`: A dictionary where each key is a page and each value is the sum of the PageRanks for the query terms found in the page.

User interface with `Streamlit`

```
st.set_page_config(page_title="Simple Search Engine", page_

st.title('🔍 Search engine with MapReduce')

directory = 'data'

if os.path.exists(directory):
    st.success("Directory exists. Ready to search!")
    pages, page_contents, all_links = read_html_files(dire
    page_ranks = calculate_page_rank(pages, all_links)
    inverted_index = create_inverted_index(page_contents)

    st.write("---")
    st.subheader("Search")
    query = st.text_input('Enter your search query:')

    if st.button('Search'):
        if query:
            results = search(query, inverted_index, page_r
            if results:
                st.write('### Search Results:')
                for page, rank in results:
                    percentage_rank = round(rank * 100, 2)
                    st.write(f"- **{page}**: {percentage_r
            else:
                st.warning('No results found.')
else:
    st.error(f'The directory {directory} does not exist. P
st.write("---")
st.write("Développé par : AGBOTON Josué - NSELE Grace - AK
```

- Page configuration and title:**
    - `st.set_page_config`: Sets page parameters, including page title (`"Simple Search Engine"`) and page icon (`"🔍"`).
    - `st.title`: Defines the title of the web application displayed at the top of the page.
- Directory Existence Check:**
    - directory = 'data'`: Defines the directory containing the HTML files to be analyzed.
    - `os.path.exists(directory)`: Checks whether the specified directory exists.
    - `st.success`: Displays a success message if the directory exists.
    - `read_html_files(directory)`: Reads HTML files in the specified directory and extracts links and page content.
    - `calculate_page_rank(pages, all_links)`: Calculates PageRanks for pages using extracted links.
    - `create_inverted_index(page_contents)`: Creates an inverted index from page content to facilitate searching.
- Search interface:**
    - `st.write("---")`: Adds a visual separator to the interface.
    - st.subheader`: Adds a "Search" subheader to the interface.
    - st.text_input`: Displays a search bar where the user can enter a search query.
- Search Button and Results Display:**

D- Let's give an example of its execution and explain the result obtained

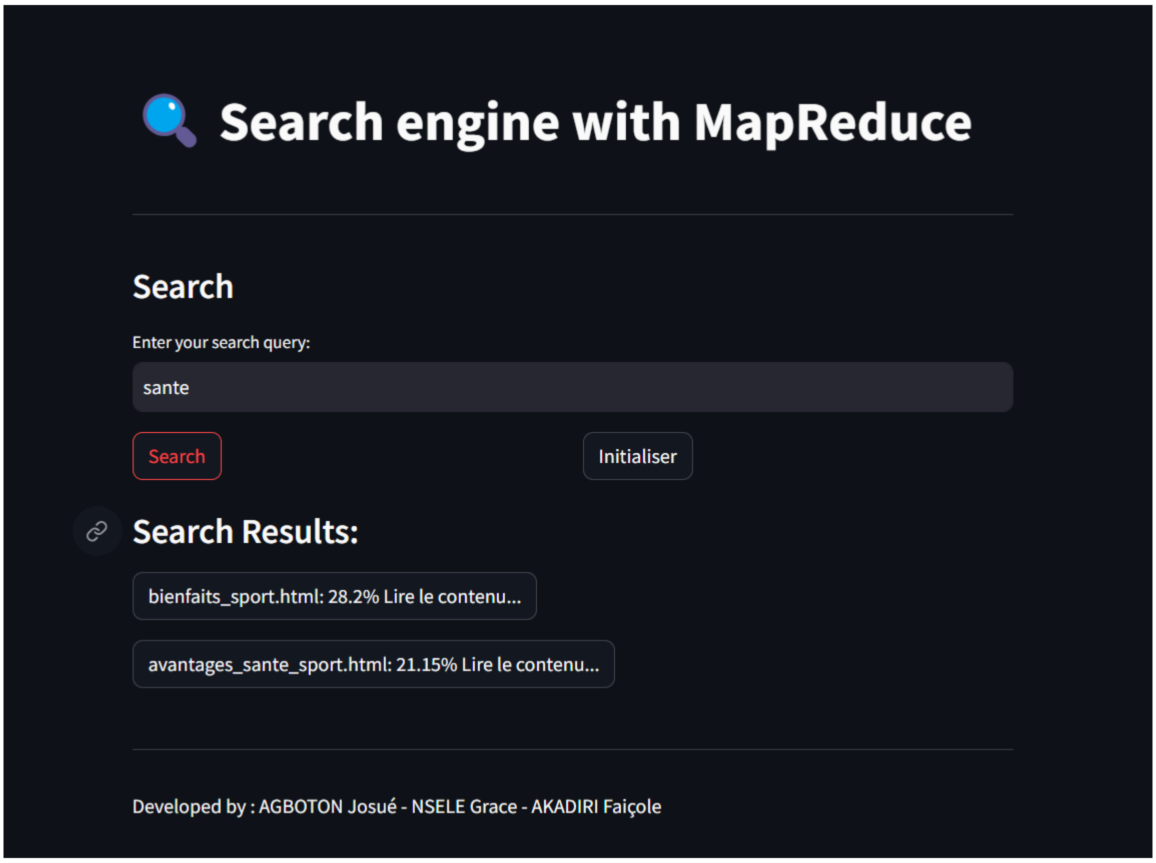


Search engine with MapReduce

Search

Enter your search query:

Developed by : AGBOTON Josué - NSELE Grace - AKADIRI Faiçole



Search engine with MapReduce

Search

Enter your search query:

sante

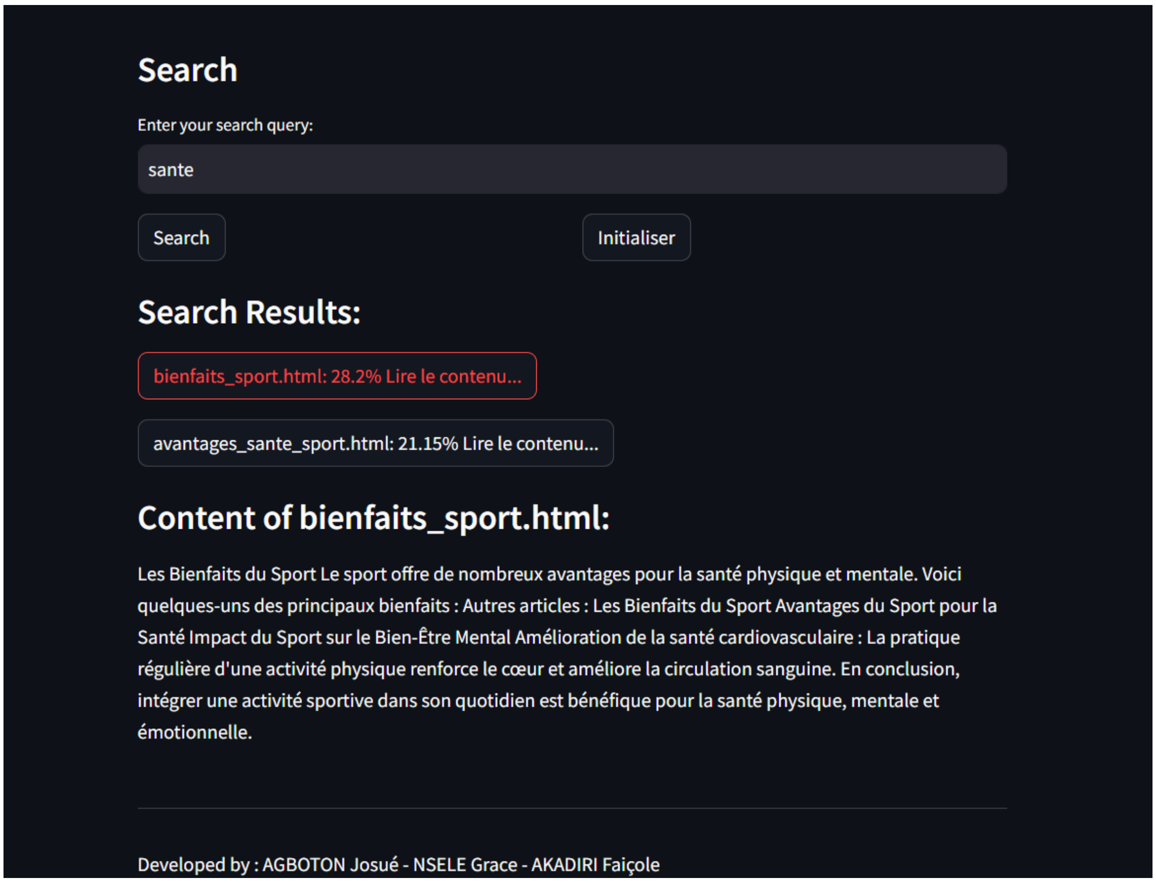
Search Results:

bienfaits_sport.html: 28.2% Lire le contenu...

avantages_sante_sport.html: 21.15% Lire le contenu...

Developed by : AGBOTON Josué - NSELE Grace - AKADIRI Faiçole

To use our search engine, enter one or more keywords. The system uses PageRank and MapReduce algorithms to process and analyze these keywords, checking the relationships between web pages. The aim is to present the most relevant pages at the top of the results, thanks to the PageRank calculation. The user can consult the content of the page displayed by means of a hypertext link, as illustrated in the image above.



Search Results:

bienfaits_sport.html: 28.2% Lire le contenu...

avantages_sante_sport.html: 21.15% Lire le contenu...

Content of bienfaits_sport.html:

Les Bienfaits du Sport Le sport offre de nombreux avantages pour la santé physique et mentale. Voici quelques-uns des principaux bienfaits : Autres articles : Les Bienfaits du Sport Avantages du Sport pour la Santé Impact du Sport sur le Bien-Être Mental Amélioration de la santé cardiovasculaire : La pratique régulière d'une activité physique renforce le cœur et améliore la circulation sanguine. En conclusion, intégrer une activité sportive dans son quotidien est bénéfique pour la santé physique, mentale et émotionnelle.

Developed by : AGBOTON Josué - NSELE Grace - AKADIRI Faiçole