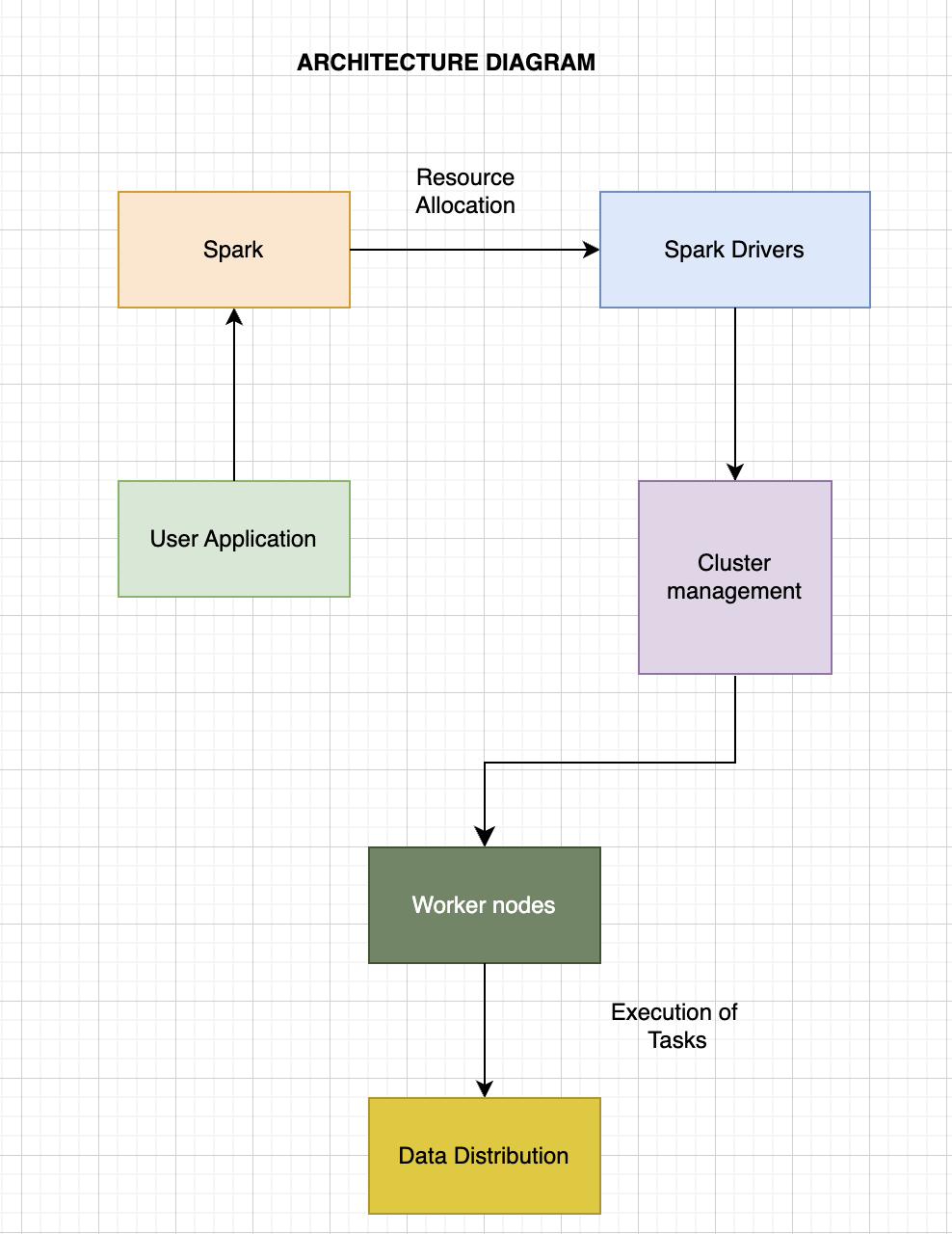
|  |  |  |
| --- | --- | --- |
| **UMBC – CSEE Department** |  | Dr. Waleed Youssef |
| **Data Science Program** |  | youssef1@umbc.edu |
| **Fall 2023** |  |  |
| **DATA 603 – Big Data Platforms** | | |
|  |  |  |
| **Homework #5 – Apache Spark** | | |
| Name: Akshay Reddy Gone  Id: AM60898 |  |  |

**Questions:**

1. **[10 points]** Draw a detailed architecture diagram describing the operation of the Spark platform?



**User Application:** This is where a user or developer writes their Spark application. It typically includes defining the operations to be performed on a distributed dataset, configuring the execution environment, and setting various parameters.

**Spark:** The framework for distributed computing used to process data is called Spark. Developers can more easily work with big datasets across clusters because it offers a high-level API for distributed data processing.

**Resource Allocation:** This step is crucial for distributing resources efficiently. It involves tasks like allocating memory and CPU resources to different parts of the application. Resource allocation ensures that various components of the Spark application get the resources they need to run efficiently.

**Spark Drivers:** The Spark driver program is the entry point of a Spark application. It's responsible for translating the high-level operations defined in the user application into a series of distributed tasks. The driver program interacts with the cluster manager to coordinate task execution.

**Cluster Management:** Cluster management is a critical component of Spark. It oversees the allocation and tracking of resources in the cluster. It can work with various cluster managers like Apache Mesos, Hadoop YARN, or Spark's standalone cluster manager. The cluster manager makes sure that resources are allocated appropriately to the Spark driver and worker nodes.

**Worker Nodes:** Worker nodes are the machines in the cluster responsible for executing tasks. They receive and run the tasks scheduled by the driver program. The driver assigns tasks to worker nodes based on the data partitioning strategy.

**Execution of Tasks:** On each worker node, tasks are executed in parallel. These tasks can include operations like data transformation, filtering, or computation. They work on distributed datasets, and the results are sent back to the driver or may trigger further tasks.

**Data Distribution:** Data distribution is a key aspect of Spark's operation. Large datasets are divided into partitions, and these partitions are distributed across the worker nodes in the cluster. This data distribution enables parallel processing, which is one of Spark's strengths.

By following these steps, Spark can efficiently process large volumes of data across a cluster of machines, making it a powerful framework for big data processing and analytics.

1. **[10 points]** Describe RDDs? How are they created? What types of processing can be performed on them?

* Give 3 examples of each type of RDD processing.

RDDs, or resilient distributed datasets, are the fundamental data structures of Apache Spark, a distributed computing framework for gathering large sets of data. RDDs are intended to provide maximum fault tolerance, parallelism and ease of use when processing distributed data. RDDs are immutable and scattered data collections that can be processed at the same time in a group of machines.

**RDDs can be created in two ways:**

**Parallelizing an existing collection:** You can create an RDD from a collection in your driver program. For example, you can create an RDD from a list of numbers or an array.

data = [1, 2, 3, 4, 5]

rdd = sc.parallelize(data) # sc is the SparkContext

**Loading external datasets:** RDDs can be created by loading data from external storage systems such as HDFS, HBase, or local file systems.

rdd = sc.textFile("hdfs://data/input.txt")

**Types of Processing on RDDs:**

RDDs support two types of operations: transformations and actions.

**Transformations:** These are operations where the RDD is created from an existing one, usually by using a function to every element of the parent RDD. The transformation is lazy, which means it doesn't run instantly but only computes once an event has been called.

**Examples of transformations:**

map(func): Applies a function to each element of the RDD.

filter(func): Returns a new RDD containing only the elements that satisfy a given condition.

reduceByKey(func): Aggregates data based on key, for example, to perform word count.

**Actions:** These are operations that trigger the computation and return values to the driver program or write data to an external storage system.

**Examples of actions:**

collect(): Retrieves all the elements of the RDD to the driver program (use with caution for large RDDs).

count(): Returns the number of elements in the RDD.

saveAsTextFile(path): Writes the RDD's content to a text file in HDFS or a local file system.

**Examples of RDD Processing:**

**Transformations:**

* **Map Transformation:**

rdd = sc.parallelize([1, 2, 3, 4, 5])

squared\_rdd = rdd.map(lambda x: x \* x)

* **Filter Transformation:**

rdd = sc.parallelize([1, 2, 3, 4, 5])

even\_rdd = rdd.filter(lambda x: x % 2 == 0)

* **ReduceByKey Transformation:**

rdd = sc.parallelize([(1, 2), (3, 4), (1, 6)])

result\_rdd = rdd.reduceByKey(lambda x, y: x + y)

**Actions:**

* **Collect Action:**

rdd = sc.parallelize([1, 2, 3, 4, 5])

result = rdd.collect()

* **Count Action:**

rdd = sc.parallelize([1, 2, 3, 4, 5])

count = rdd.count()

* **SaveAsTextFile Action:**

rdd = sc.parallelize(["Hello, world!", "This is Spark.", "RDDs are cool."])

rdd.saveAsTextFile("hdfs://data/output/")

RDDs are a powerful abstraction for distributed data processing in Spark, and they enable efficient, fault-tolerant, and parallel execution of data manipulation tasks across clusters of machines.

1. **[10 points]** Compare Scala, Java, and Python in details? Describe which language you plan on using and why?

Scala, Java, and Python are three popular programming languages used in various domains, including software development, data analysis, and big data processing. Below, I'll provide a detailed comparison of these languages and explain which one might be a suitable choice depending on your use case.

**Scala:**

Conciseness: Scala is known for its concise syntax, which allows developers to express complex ideas with relatively fewer lines of code. This can enhance productivity.

Type System: Scala employs a strong, static type system, which means that type-related errors are detected at compile-time, making it suitable for building robust and reliable software.

Functional Programming: Scala supports functional programming paradigms, making it a solid choice for tasks like data processing and distributed computing (e.g., Apache Spark).

JVM Compatibility: Scala runs on the Java Virtual Machine (JVM), enabling seamless integration with Java libraries.

Concurrency: Scala provides powerful concurrency features like Akka, which is beneficial for developing highly concurrent and fault-tolerant systems.

Community and Ecosystem: While the Scala community is growing, it may not be as extensive as that of Java or Python.

**Java:**

Conciseness: Java is relatively more verbose compared to Scala and Python, which can lead to larger codebases.

Type System: Java employs a strong, static type system, ensuring type safety and reliability in code.

Platform Independence: Java follows the "Write Once, Run Anywhere" philosophy, making it compatible with various platforms.

Enterprise Usage: Java is widely used in enterprise-level software development, web applications, and Android app development.

Community and Ecosystem: Java boasts a vast and mature ecosystem with a wide range of libraries and frameworks for different applications.

**Python:**

Conciseness: Python is celebrated for its simplicity and readability, allowing developers to write concise code that is easy to understand.

Type System: Python uses dynamic typing, meaning you don't have to specify variable types explicitly. However, this can result in runtime errors if not handled with care.

Versatility: Python is an incredibly versatile language used in web development, scientific computing, data analysis, artificial intelligence, and more.

Data Science and Machine Learning: Python is the language of choice for many data scientists due to libraries like NumPy, pandas, and scikit-learn.

Community and Ecosystem: Python enjoys a large and active community, along with an extensive ecosystem comprising numerous third-party packages.

**Choice of Language:**

The choice of language depends on your specific use case:

Scala: If you're working on big data processing, building highly concurrent systems, or value the expressiveness of functional programming, Scala is an excellent choice.

Java: For enterprise-level applications, Android development, or scenarios where platform independence is crucial, Java remains a solid option.

Python: Python is the go-to option if you need a flexible, simple-to-learn language for data science, machine learning, web development, or general-purpose scripting.

"I prefer Python for my projects due to its exceptional versatility, user-friendliness, and extensive library support. Python's readability and clear syntax make development and maintenance straightforward. Its diverse library ecosystem, especially in data science and AI, aligns with my project requirements. Additionally, the strong Python community and comprehensive documentation provide robust support. Python's rapid prototyping capabilities are invaluable for my creative and research-focused projects. Considering these factors, Python best suits my current needs and preferences compared to Scala or Java."

1. **[20 points]** Write a Spark program to count the number of images in a URL and then display the URLs of these images.

For example, the program should read the URL 🡪 [www.usmd.edu](http://www.usmd.edu)

Then, it should display the output as:

1. There are <n> images at the <<URL>> site
2. The images are:

<<Image URL 1>>

<<Image URL 2>>

<<Image URL 3>>

…

<<Image URL n>>

**Code:**

#Install PySpark

!pip install pyspark

#Import necessary libraries

from pyspark import SparkContext, SparkConf

from bs4 import BeautifulSoup

import requests

#Create a Spark configuration and context

conf = SparkConf().setAppName("ImageCountApp")

sc = SparkContext(conf=conf)

#Function to count images and retrieve their URLs from a given URL

def count\_and\_get\_image\_urls(url):

try:

#Send an HTTP GET request to the provided URL

response = requests.get(url)

#Check if the request was successful (status code 200)

if response.status\_code == 200:

#Get the HTML content of the page

html\_content = response.text

#Parse the HTML content with BeautifulSoup

soup = BeautifulSoup(html\_content, 'html.parser')

#Find all image (img) tags in the HTML

img\_tags = soup.find\_all('img')

#Extract the 'src' attribute (URL) from each image tag

img\_urls = [img['src'] for img in img\_tags]

#Count the number of images found

num\_images = len(img\_urls)

#Return a tuple with the URL, number of images, and the list of image URLs

return (url, num\_images, img\_urls)

else:

#If the request was not successful, return 0 images and an empty list of URLs

return (url, 0, [])

except Exception as e:

#Handle any exceptions (e.g., network errors) and return 0 images and an empty list of URLs

return (url, 0, [])

#Input URL

url = "http://www.usmd.edu"

#Count images and retrieve their URLs using Spark

result = sc.parallelize([url]).map(count\_and\_get\_image\_urls).collect()

#Display the results

for (url, num\_images, img\_urls) in result:

print(f"(a) There are {num\_images} images at the {url} site")

print("(b) The images are:")

for img\_url in img\_urls:

print(img\_url)

#Stop the Spark context

sc.stop()

(a) There are 21 images at the http://www.usmd.edu site

(b) The images are:

/images/logo/USMLogo.png

/images/home/CAI2.png

/images/home/USMF\_logo\_CMYK.png

/images/featured\_institutions/new/UMGC-Color.png

/images/featured\_institutions/new/UMB-Color.png

/images/featured\_institutions/new/UMCP-Color.png

/images/featured\_institutions/new/UMSG-Color.png

/images/featured\_institutions/new/FSU-Color.png

/images/featured\_institutions/new/UMCES-Color.png

/images/featured\_institutions/new/UOB-Color.png

/images/featured\_institutions/new/UMES-Color.png

/images/featured\_institutions/new/CSU-Color.png

/images/featured\_institutions/new/BSU-Color.png

/images/featured\_institutions/new/USMSM-Color.png

/images/featured\_institutions/new/TU-Color.png

/images/featured\_institutions/new/SU-Color.png

/images/featured\_institutions/new/UMBC-Color.png

/images/featured\_institutions/new/USMHG-Color.png

/images/home/OppsForAll\_Logo.png

/images/logo/USM-Logo-stacked.png

/images/redesign/site/md-flag.gif (Also Attached my python file)