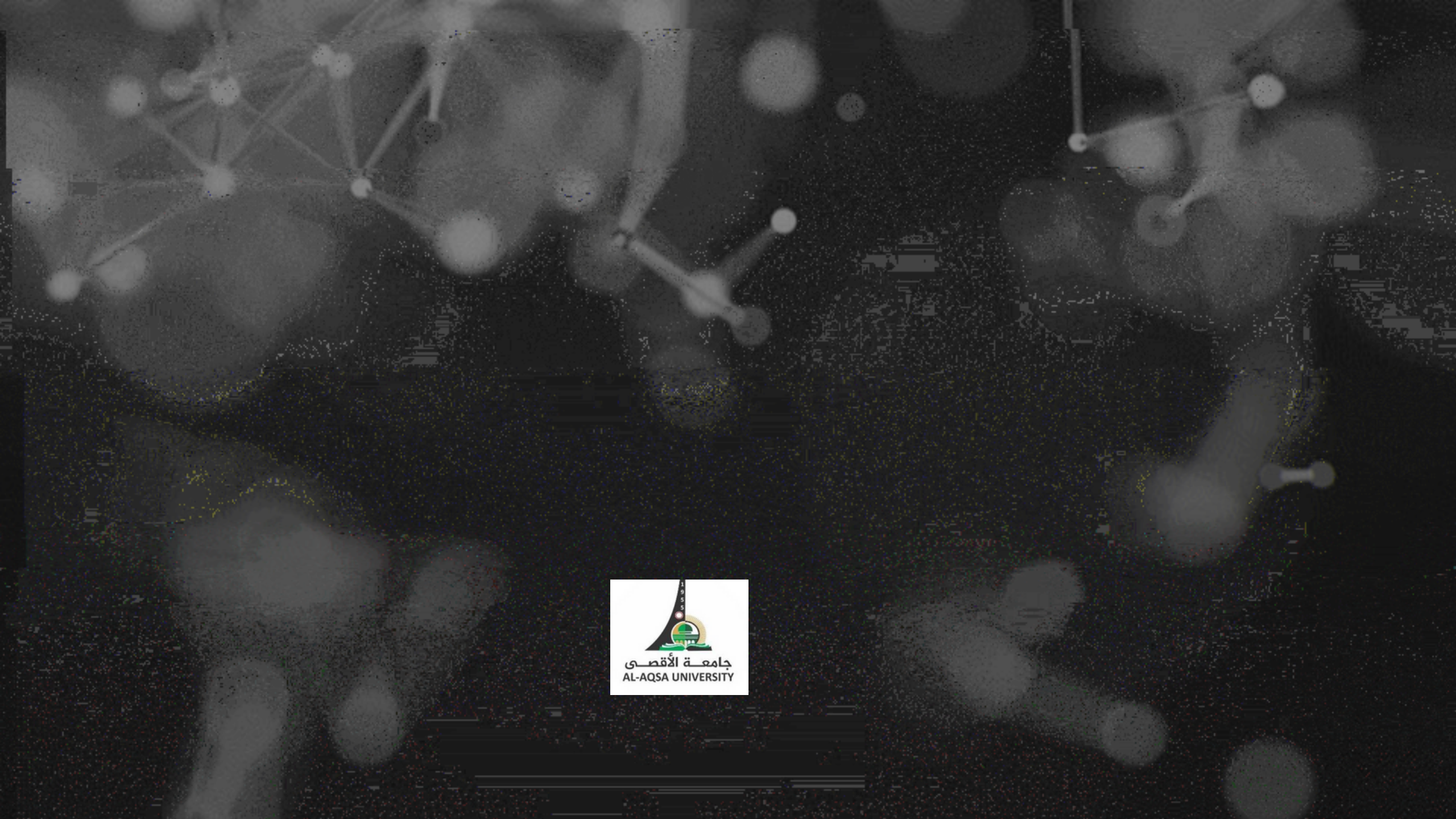

1955
جامعة الأقصى
AL-AQSA UNIVERSITY

INTRODUCTION

- We conducted a thorough comparativeanalysisofHadoop MapreduceandSpark, two prominent data processing platforms, in the context of the current era of exponential data growth.
- Initially, Hadoop's MapReduce revolutionized distributed computing but faced limitations, paving the way for Spark as a promising alternative.
- Spark's strength lies in efficient handling of iterative tasks through result caching, though it grapples with memory constraints.
- The study explores architectural and theoretical disparities before meticulously evaluating their performances across key dimensions. Word count operations, join efficiency, and Page Rank algorithm execution serve as benchmarks, considering factors such as speed, resource utilization, scalability, and memory constraints.
- This comprehensive analysis aims to guide practitioners and organizations in selecting the optimal platform tailored to their specific data processing needs.

HADOOP

- Framework to store and process large set of data across computer clusters.

- Components of Hadoop:

  - Storage unit : HDFS (Hadoop Distributed File System)
    Primary data storage system that manages large data sets.
    Splits data into multiple blocks and stored it into name node and data nodes.

  - Processing unit : Map – Reduce
    Splits data and processes everything separately in each datanode.

  - Resource manager unit : Yarn (Yet Another Resource Negotiator)
    Processes job request and handles cluster of nodes.
    Cluster resource manager that schedules tasks and allocates resources (e.g., CPU and memory) to applications.

MAP-REDUCE

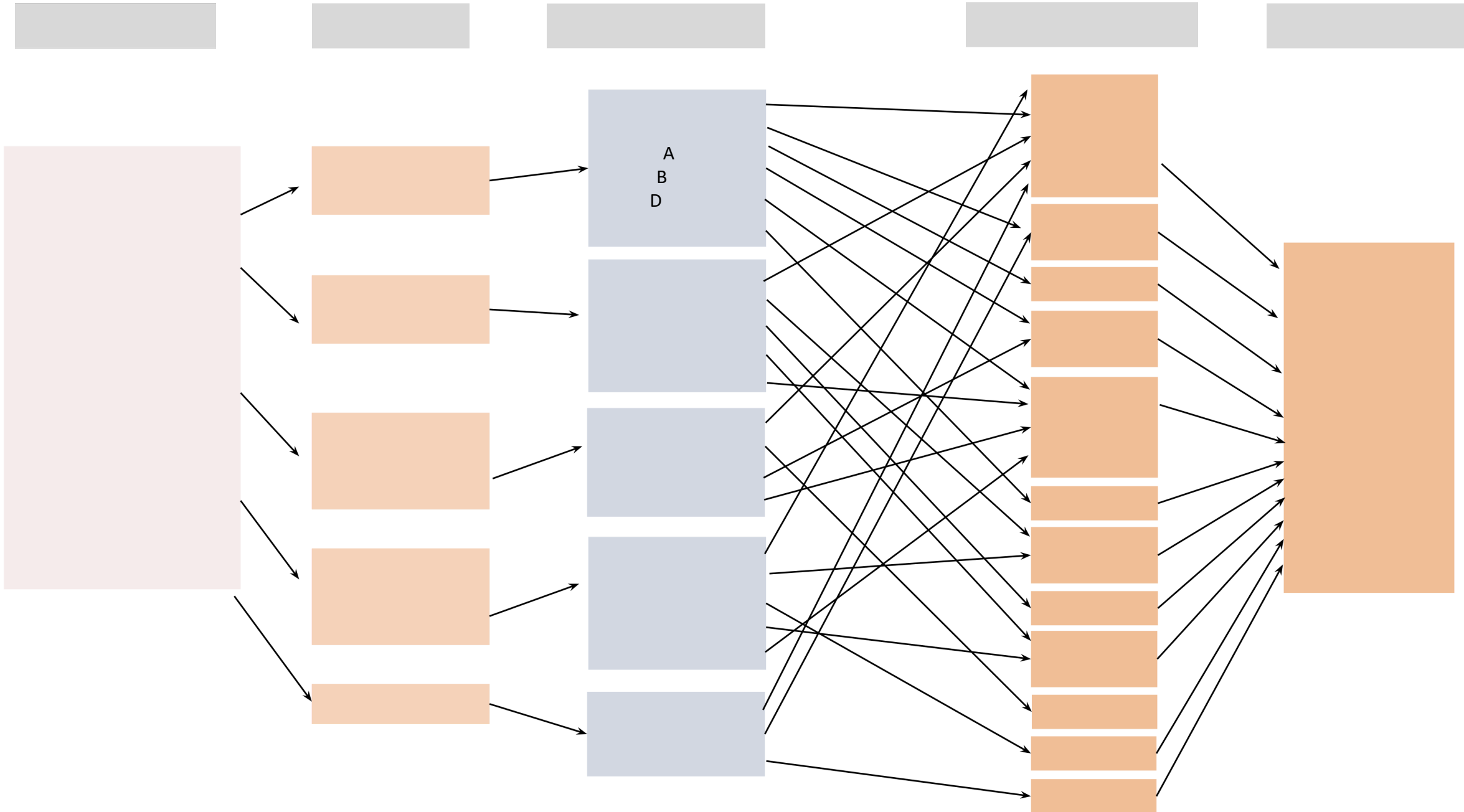
- MapReduce is a programming modeland an associated implementation for processing big data sets with a parallel, distributed algorithm on a cluster of the computer.

- MapReduce programs:

For analyzing, large volumes of data from databases and data warehouses by implementing specific business logic to derive insights.

- Phases of MapReduce:

  - Input
  - Split
  - Mapper phase
  - Shuffle and Sort
  - Reduce phase.


A
B
D

CHARACTERISTICS OF MAP-REDUCE

- Data moves through disk and network.
- Very difficult to code. Requires some time to learn the syntax.
- Comparatively less costly because of hard disk storage.
- Data is collected processed and then results are produced at later stage.
- Hadoop MapReduce uses enjoy all Hadoop security benefits. Therefore, better than spark.
- MapReduce use replication for fault tolerance.


5px #ccc}.gbrtl .gbm{
display:block;position
pacity:1;*top:-2px;*lef
/;top:-4px\0/;left:-6px
e-box;display:inline-b
display:block;list-styl
e-block;line-height:27p
pointer;display:block;t
tive;z-index:1000}.gbt
padding-right:9px}#gbz

-
-
-
-
-

SPARK

- Spark is a distributed computing platformthatprovidesin-memory processing for large-scale data processing and analytics.
- Components of Spark:
  - Storage Unit : Depends on external storage systems like HDFS,S3 or others
  - Processing Unit : Utilizes RDDs, DataFrames, Datasets for distributing data processing
  - Cluster manager Unit : Can use its built-in cluster manager or integrate with others like Mesos or YARN
  - Resource manager Unit : Utilizes its built-in resource manager or integrates with external resource managers for task scheduling and resource allocation.

USING SPARK IN PYTHON

- PySpark is the PythonAPIforApacheSpark,providingahigh-levelandexpressiveframeworkfordistributeddata processing and analytics.

- PySpark employs RDDs, which are fault-tolerant collections of items that serve as an abstraction for distributed data processing.
- SparkContext represents the connection to a Spark cluster and can be used to create RDDs and perform various operations, including MapReduce operations
  - The map transformation applies a specified function to each element of the RDD, producing a new RDD of the same size.
  - The reduceByKey transformation is used on key-value pair RDDs. It groups elements by key and applies a specified function to reduce the values associated with each key.
  - The flatMap transformation is similar to map but can produce multiple output elements for each input element. It flattens the results.
  - The mapValues transformation is specifically used on key-value pair RDDs. It applies a function to the values while keeping the keys unchanged.

CHARACTERISTIC OF SPARK

- Performance:

  Minimizes datamovementthroughin-memoryprocessing

- Ease of use:

  The syntax is more approachable and provides more user-friendly API with higher-level abstractions like DataFrames

- Cost:

  In-memory processing can lead to better performance, potentially reducing the overall cost of processing

- Data Processing:

  It is versatile, supporting both batch and real-time processing

- Security:

  It has built-in security features, It also supports integration with Hadoop’s secure storage

- Fault Tolerance:

  It provides a mechanism to recover lost data


5px #ccc}.gbrtl .gbm{
display:block;position
pacity:1;*top:-2px;*lef
/;top:-4px\0/;left:-6px
e-box;display:inline-b
display:block;list-styl
e-block;line-height:27p
pointer;display:block;t
tive;z-index:1000}.gbt
padding-right:9px}#gbz

-
-
-
-
-

EXPERIMENT

MAP.py

```
import sys

for line in sys.stdin:

words=line.strip().split()
for word in words:
    print(word + "\t1")
```

REDUCE.py

```
import sys


current_word = None
current_count = 0


for line in sys.stdin:
word, count = line.strip().split("\t")


if current_word == word:
    current_count += int(count) # If the same word,
increment the count
else:

    if current_word:
      print(current_word + "\t" + str(current_count)) # Print
the word and its accumulated count

    current_word = word
    current_count = int(count)


if current_word:
print(current_word + "\t" + str(current_count))
```

EXPERIMENT

spark_word_count.py

```
#Import libraries
from pyspark import SparkContext, SparkConf
import time
#Configuring the setup
conf =
SparkConf().setAppName("WordCount").setMaster('local')
sc = SparkContext(conf=conf)
#Read the text file
text_file = sc.textFile("dataset_wordcount/input_300000.txt")

#Main code -> It splits the line into words and it counts the
words
word_counts = text_file \
.flatMap(lambda line: line.split()) \
.map(lambda word: (word, 1)) \
.reduceByKey(lambda a, b: a + b)

#Saves the output in a text file
word_counts.saveAsTextFile("word_count_output_300000")
time.sleep(60)
sc.stop()
```

MapReduce job execution command :

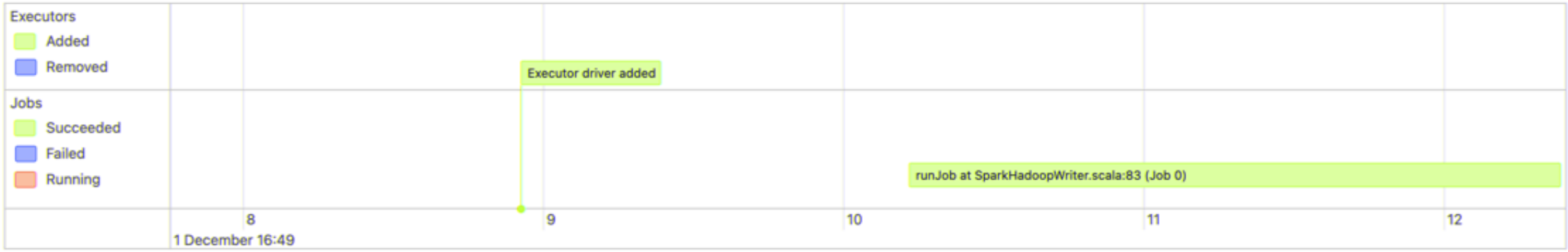
```
hadoop@hadoop:~/Project1$ hadoop jar
/usr/local/hadoop/share/hadoop/tools/lib/hadoop-streaming-
3.3.6.jar -input /proj/dataset_wordcount/input_10.txt -output
/proj/output -mapper
/home/hadoop/Project1/word_count/map.py -reducer
/home/hadoop/Project1/word_count/reduce.py
```

Spark Jobs (?)

User: apoorvareddy
Total Uptime: 13 s
Scheduling Mode: FIFO
Completed Jobs: 1

▾ Event Timeline

☐ Enable zooming



▾ Completed Jobs (1)

Page: 1

1 Pages. Jump to 1 . Show 100 items in a page. Go

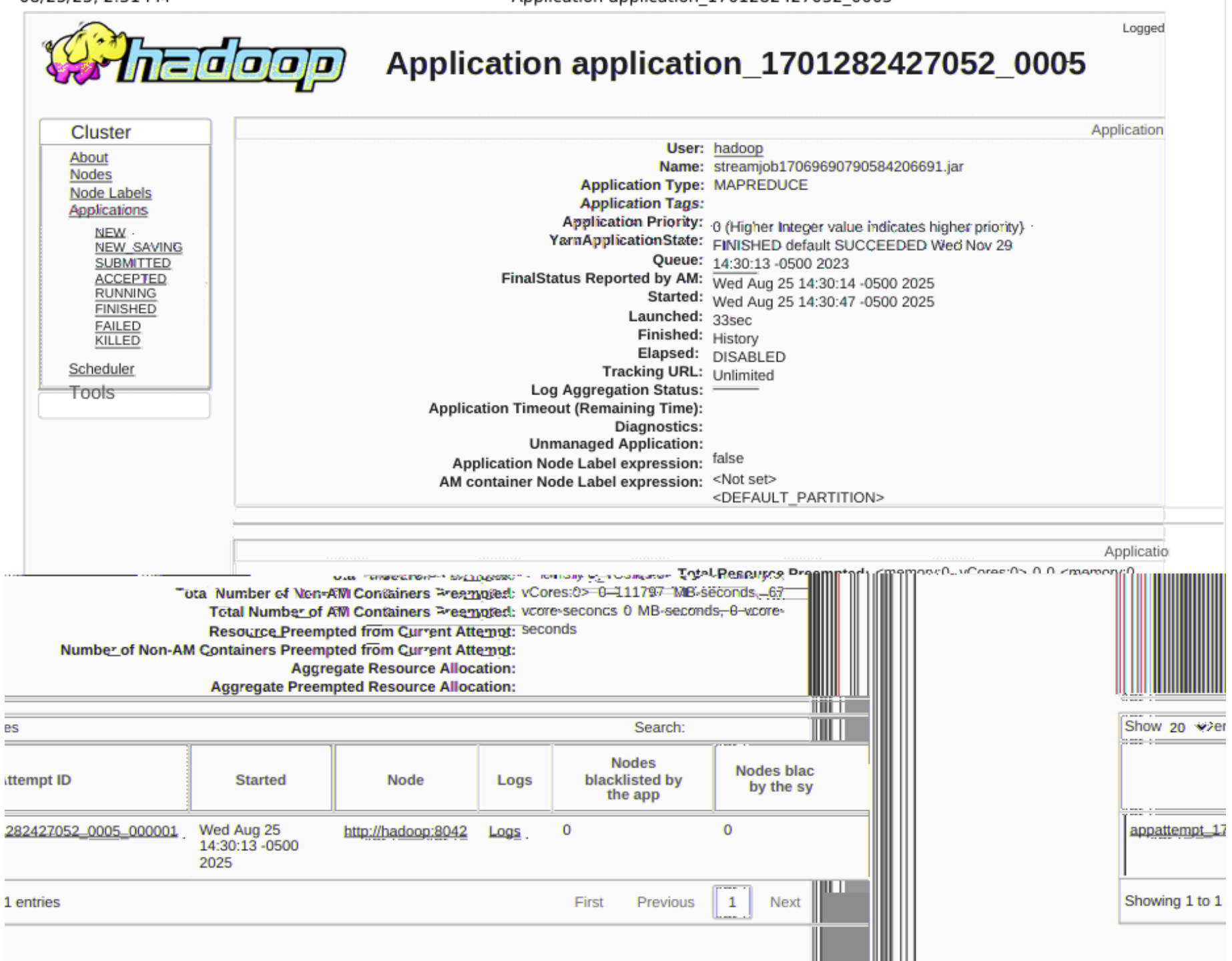
Job Id ▾	Description	Submitted	Duration	Stages: Succeeded/Total	Tasks (for all stages): Succeeded/Total
0	runJob at SparkHadoopWriter.scala:83	2025/08/25 16:49:10	2 s	2/2	2/2

runJob at SparkHadoopWriter.scala:83

100 items in a page. Go

Page: 1

1 Pages. Jump to 1 . Show


Logged
Application application_1701282427052_0005
Cluster
About
Nodes
Node Labels
Applications
NEW
NEW_SAVING
SUBMITTED
ACCEPTED
RUNNING
FINISHED
FAILED
KILLED
Scheduler
Tools
Application
User: hadoop
Name: streamjob1706969079058420691.jar
Application Type: MAPREDUCE
Application Tags:
Application Priority: 0 (Higher Integer value indicates higher priority)
YarnApplicationState: FINISHED default SUCCEEDED Wed Nov 29 14:30:13 -0500 2023
Queue:
FinalStatus Reported by AM: Wed Aug 25 14:30:14 -0500 2025
Started: Wed Aug 25 14:30:47 -0500 2025
Launched: 33sec
Finished: History
Elapsed: DISABLED
Tracking URL: Unlimited
Log Aggregation Status:
Application Timeout (Remaining Time):
Diagnostics:
Unmanaged Application: false
Application Node Label expression: <Not set>
AM container Node Label expression: <DEFAULT_PARTITION>
Applicatio
Total Number of Non-AM Containers Preempted:
Total Number of AM Containers Preempted:
Resource Preempted from Current Attempt:
Number of Non-AM Containers Preempted from Current Attempt:
Aggregate Resource Allocation:
Aggregate Preempted Resource Allocation:
Search:
Attempt ID
Started
Node
Logs
Nodes blacklisted by the app
Nodes blac by the sy
282427052_0005_000001
Wed Aug 25 14:30:13 -0500 2025
http://hadoop:8042
Logs
0
0
1 entries
First
Previous
1
Next
Show 20
appattempt_17
Showing 1 to 1

- C

Words	MapReduce execution time	Spark Execution time
10	31 secs	2.45 secs
100	33 secs	2.41 secs
1000	28 secs	2.39 secs
10000	33 secs	2.34 secs
100000	32 secs	2.59 secs
200000	32 secs	2.64 secs
300000	31 secs	2.66 secs

MAP.py

```
#!/usr/bin/python3

# Get values from input dataset using streaming.

# Remove leading and trailing whitespaces from the input line

    # Split the input line into key and value1 using the tab ('\t') as a delimiter

    # Print the key and value1, separated by a tab
```

REDUCE.py

```
#!/usr/bin/python3

# Initialize an empty dictionary to store key-value pairs

 # Iterate throughthe lines of input received from standard input (stdin) streaming

     # Remove leading and trailing whitespaces from the input line

    # Split the input line into key and value using the tab ('\t') as a delimiter

    # Check if the key is already present in the dictionary

        # If the key exists, print the key, its stored value, and the new value - joining the 2 datasets. - key is the primary key connecting the tables.


        # If the key is not present, store the key-value pair in the dictionary
```

EXPERIMENT

```
df1 = spark.createDataFrame(data1, columns1)
df2 = spark.createDataFrame(data2, columns2)

# Step 3: Perform Inner Join
inner_join_df = df1.join(df2, df1['code1'] == df2['code1'], 'inner')

# Step 4: Show Inner Join Result
print("Inner Join Result:")
inner_join_df.show(truncate=False)

# Step 5: Perform Outer Join
outer_join_df = df1.join(df2, df1['code1'] == df2['code1'], 'outer')

# Step 6: Show Outer Join Result
print("Outer Join Result:")
outer_join_df.show(truncate=False)
```

EXPERIMENTS

- Join Program

Rows	MapReduce execution time	Spark Execution time
10	25 secs	5.5 secs
20	21 secs	5.7 secs
30	28 secs	5.75 secs
50	26 secs	5.55 secs
75	29 secs	5.45 secs
100	31 secs	5.46 secs

MAP.py

```
#!/usr/bin/python3

# Get values from input dataset using streaming.

# Split the input line into node and neighbors using the tab ('\t') as a delimiter
node, neighbors = line.strip().split("\t")
# Split the neighbors into a list of neighbor nodes
neighbors = neighbors.split(',')
# Initialize the rank for the current node to 1.0

# Check if the current node has neighbors

    # Calculate the number of links from the current node

    # Distribute the rank evenly among the neighbors and print the contributions
```

REDUCE.py

```
#!/usr/bin/python3

# Initialize variables to keep track of the current node and its contributions


# Obtain the input directory from the command line arguments


# calculate the number of lines in dataset.

# Open the input file and count the number of lines

# Split the input line into node and value using the tab ('\t') as a delimiter

 # Convert the value to a floating-point number


  # Check if the current node is the same as the previous one

    # If the same node, add the contribution to the list


    # If a new node is encountered or it's the first node

      # Calculate the new rank for the current node based on contributions

      # Normalize the rank by dividing it by the total number of lines

      # Print the node and its normalized rank

    # Update the current node and contributions for the new node


# After processing all lines, print the last node and its normalized rank
```

EXPERIMENT

```
ranks = sc.parallelize(link_data.keys()).map(lambda x : (x, 1.))
links = sc.parallelize(link_data.items()).cache()
sorted(links.join(ranks).collect())
def computeContribs(node_urls_rank):

T


 _, (urls, rank) = node_urls_rank
nb_urls = len(urls)
for url in urls:
    yield url, rank / nb_urls
c = links.join(ranks).flatMap(computeContribs)
print(c.toDebugString().decode("utf8"))
from operator import add
for iteration in range(10):
# compute contributions of each node where it links to
contribs = links.join(ranks).flatMap(computeContribs)
# use a full outer join to make sure, that not well connected nodes aren't dropped
contribs = links.fullOuterJoin(contribs).mapValues(lambda x : x[1] or 0.0)
# Sum up all contributions per link
ranks = contribs.reduceByKey(add)
# Re-calculate ranks
ranks = ranks.mapValues(lambda rank: rank * 0.85 + 0.15)
```

EXPERIMENTS

- Page Rank program

Nodes	MapReduce execution time	Spark Execution time
10	19 secs	60 secs
20	21 secs	80 secs
30	22 secs	110 secs
50	31 secs	115 secs
75	31 secs	110 secs
100	22 secs	130 secs
175	25 secs	120 secs
300000	47 secs	234 secs

Conclusion

In conclusion, the three experiments conducted to compare the performance of MapReduce and Spark for different algorithms - word count, join operations, and PageRank - reveal significant insights into the efficiency and scalability of these frameworks.

- Experiment 1: The word count program showed a consistent performance in MapReduce across different data sizes, with a slight increase in execution time as the number of words increased. In contrast, Spark demonstrated a superior performance with significantly lower execution times, although it experienced a slight increase in time as the data size grew to 300,000 words. This indicates Spark's better efficiency in handling large datasets for word count operations.
- Experiment 2: In the join operations, MapReduce showed a fluctuating performance but generally increased in execution time with the growth in data size. Spark, however, maintained a comparatively stable and considerably lower execution time across all data sizes. This underscores Spark's proficiency in managing join operations, especially with larger datasets.
- Experiment 3: For the PageRank algorithm, MapReduce demonstrated a relatively stable performance, but Spark's execution time increased with the growth in the number of pages, indicating a higher resource demand for larger datasets.

Overall, these experiments highlight Spark's superiority in handling word count and join operations with greater efficiency, especially for larger datasets. However, for more complex algorithms like PageRank, MapReduce may offer a more stable performance, particularly when dealing with extremely large data sizes. This comparative analysis provides valuable insights for selecting the appropriate framework based on the specific requirements of data processing tasks.


THANK YOU