

# COEN 242 Final Project Report (Team 10)

*Shivanisanjay Tergaonkar, Shreesh Shivaprakash, Swetha Valluru, Vikram Ramanna*

## ***Report Cover Letter:***

Dear professor, thank you for offering this course and helping us better understand the subject. In our final project, we have implemented the PageRank algorithm using PySpark over the Hadoop Distributed File System on Berkley-Stanford web dataset. We have addressed the project report based on the following topics as per your request.

## ***A clear project introduction, including background and motivation:***

In Section I, we have a detailed discussion on PageRank, section II describes the motivation behind using this algorithm, along with the explanation of the associated backgrounds from sections III to VI.

## ***Formulating/solving the proposed problem, PageRank algorithm using power iteration and Google PageRank algorithm based on dataset:***

Referring to the ‘*A Comparative Study of Page Ranking Algorithms for Information Retrieval*’ [1] conference paper provided us a detailed insight on the different algorithms used and in comparing the different PageRank algorithms. We used the Google PageRank algorithm for our final project and compared different results.

## ***Highlighting the parallel computation:***

We have showcased the different approaches to enable parallel computation and speed up the runtime for our PageRank algorithm implementation, in Figure 4.

## ***Presenting necessary experimental results (quantitative and qualitative), including tables, charts, and analyses:***

In section X, we have provided our observations and plotted the results for different combinations and proved the presence of improvements via the results. The optimized results are displayed, tabulated and plotted in graphs.

## ***Project conclusion:***

We observed that by optimizing the algorithm with an optimal number of cores, executors, executors’ memory and multiple partitions, we could optimize the run time for our implemented PageRank algorithm. More details are provided in the conclusion section.

## ***Questions in the experiment which respond to the project motivation and highlight parallelized implementation***

We initially found the time taken to run the algorithm was more than an hour for just 50 iterations and made us realise that we need to optimize further and led us to learn more about the different configuration options on yarn. This helped reduce the time from seventy minutes to four minutes overall in 210 iterations for the ranks to converge. Further, using the Spark/HDFS feature of persistence helped further reduce the latency for running the algorithm. Overall we learnt the PageRank algorithm, how to run this algorithm over a huge data set having more than 600K of records in less than five mins to compute the top ranked URLs from this dataset.

# Google PageRank

**Abstract** — This report is our final project for Big Data (COEN 242) class at Santa Clara University, Spring 2021. In this project, we implement the PageRank algorithm over the Berkeley-Stanford web graph data using the Spark framework implemented over the Hadoop Distributed File System (HDFS).

**Keywords** — *BigData, PageRank, pySpark, yarn, cores, partitions.*

## I. INTRODUCTION

It is a well-known fact that computers have led to the third revolution of the human civilization, with the information revolution taking its place alongside the agricultural and industrial revolutions. The process of retrieving information in the present times has been simpler than it had ever been. When a person wishes to gather information on a specific topic, all that needs to be done is to fire up a search engine in a browser, and enter that query for which the user is interested in getting the answers of. It goes without saying that Google is colloquial with the term ‘search engine’, moreso among the less technically inclined populace. This observation can be attributed to the fact that the predominant search engine of the present (and from a long time from the past) is Google, with it enjoying a market share of well over 90%, leaving other competitors to scramble among themselves to attain a paltry share of single-digit percentage values. The reason for the stratospheric rise of Google as the behemoth of search engines can be attributed to (among other factors) an algorithm known as PageRank, invented by Larry Page, one of the founders of Google [2].

Prior to the emergence of Google and PageRank, earlier search engines utilized the mechanism of web-crawling so as to develop a data structure known as ‘inverted index’, which was a collection of terms found in each of the considered webpages. Upon issuing a search query to such search engines, a list of corresponding webpages is returned, wherein the position (or rank) of a webpage in relation to the supplied search query was a function of the nature of terms (comprising the search query) as present in the said webpage. The nature of terms can be attributed to its count and importance, i.e., its

positioning in the webpage relative to other terms. It was soon obvious that such a mechanism was prone to malpractices; webpages were made to include numerous general, non-relevant terms by their owners, in order to arrive at a higher probability of a user landing at their webpage, even if the search query supplied by the user and the primary content of the webpage were at odds with each other. Such dubious implementations led to high network traffic for these deceiving webpages, much to the dismay of users [3].

It was in lieu of the above situation that heralded the coming of Google and the based PageRank algorithm; the above mentioned issues were successfully alleviated, thereby rendering an era of efficient and accurate web searches [3]. This paper is organized as follows. In section II, we describe the factors that motivated us in taking up this specific topic as our final project implementation. The section III offers a detailed elucidation on the algorithm of PageRank. Sections IV to VI put forth the theoretical background of the platform over which the project was implemented. Sections VII to X describes in detail the technical aspects of the project.

## II. MOTIVATION

A search engine, especially Google, is now a quintessential part of our lives. Pretty much anything and everything we humans wish to gain information about, can be answered by just providing our query as an input, only to obtain various, pertinent results, within a matter of milliseconds. As a whole, knowledge has become much more accessible than it ever was before. Behind this fancy front-end, lies an intermingling of diverse algorithms, primary of which is PageRank, which work in harmony to provide the results in response to the provided search query. We therefore chose to go ahead with this topic for our final project; owing to its implicit importance in real life.

Furthermore, the dimensionality of the data to be worked on, was on the higher end of the scale. This lended an amazing opportunity for an implementation in a big data based platform. The advantages offered by a distributed computing paradigm and the usage of Spark in coding the requisite program were visibly made evident, in contrast to the regular paradigm of local

computing. These are the motivating factors which influenced us in going ahead with this project.

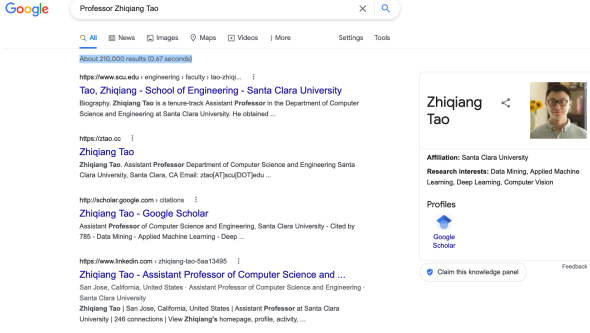


Figure 1: An example of the results (approximately about 2 million in number) returned by Google when supplied with a search query, within one second.

### III. PAGERANK ALGORITHM

As is suggested by the name, the PageRank algorithm aims to assign a (real number) rank to each webpage, directly proportional to its importance relative to other webpages. The algorithm utilizes a directed graph data structure to emulate the world wide web. Each node represents a unique webpage, and a directed edge/arc from one node to another indicates the presence of a hyperlink in the former which leads to the latter.

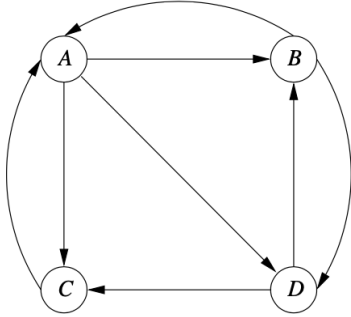


Figure 2: An illustrative example of the Web as a directed graph; with the nodes representing unique web pages, and directed edges indicating a hyperlink, navigating from one node to another [3].

The PageRank algorithm consists of the following computational steps:

$$\begin{aligned}
 s_w &= 1/N \\
 \text{for } w &= 0 \text{ to } n - 1: \\
 s_w &= \gamma/N + (1 - \gamma) \cdot \sum_{w': w' \rightarrow w} \frac{s_{w'}}{d_{w'}}
 \end{aligned}$$

Figure 3: The PageRank algorithm in pseudocode.

The above algorithm is executed until the scores of all webpages are essentially unchanged. The algorithm comprises of the following terms:

$s_w$ : The score/rank of a webpage  $w$

$N$ : The total number of webpages in the web

$w'$ : A webpage which contains a hyperlink to website  $w$

$d_{w'}$ : The out-degree of webpage  $w'$

$1 - \gamma$ : The damping factor indicates the probability of a user continuing to surf the web; usually equated to 0.85.

### IV. BIG DATA

Big data is a term that describes the large volume of structured and/or unstructured data that inundates an organization on a day-to-day basis. It's not the amount of data that's important; it's what organizations do with the data that matters. Big data can be analyzed for insights that lead to better decisions and strategic business moves. The traditional definition of big data is articulated as the three V's:

**Volume:** Organizations collect data from a variety of sources, such as business transactions, smart (IoT) devices, industrial equipment, social media and more. Economical storage media on platforms like data lakes and Hadoop help manage these gigantic magnitudes of data.

**Velocity:** With the growth in IoT, data streams into organizations at an unprecedented rate and renders it essential to be handled in a timely manner.

**Variety:** Data comes in all types of formats – from structured, numeric data in traditional databases to unstructured text documents, emails, videos, audios, stock ticker data and financial transactions.

More commonly, organizations of today furthering the definition with two additional metrics:

**Variability:** Data flows are challenging and unpredictable – changing often and varying greatly. Organizations need to know when something is trending in social media, and how to manage daily, seasonal and event-triggered peak data loads.

**Veracity:** Veracity refers to the quality of data. Because data comes from so many different

sources, it's difficult to link, match, cleanse and transform data across systems. Organizations need to connect and correlate relationships, hierarchies and multiple data linkages to prevent data from spiralling out of control [4].

## V. DISTRIBUTED COMPUTING AND FILE SYSTEMS - HADOOP

A distributed computer system consists of multiple software components that are on multiple computers, but run as a single system. The computers in a distributed system can either be physically close together in a local network, or be geographically distant and connected by a wide area network. A distributed system can consist of any number of possible individual configurations. The end goal of distributed computing is to make such a network work as a single computer.

Distributed systems offer many benefits over centralized systems, such as:

*Scalability:* The system can easily be expanded by adding more machines as needed.

*Redundancy:* Several machines can provide the same services, so if one is unavailable, work does not stop. Additionally, because many smaller machines can be used, this redundancy does not need to be prohibitively expensive.

Distributed computing systems can run on hardware that is provided by many vendors, and can use a variety of standards-based software components. Such systems are independent of the underlying software. They can run on various operating systems, and can use various communications protocols [5].

The Apache Hadoop software library is a framework that allows for the distributed processing of large data sets across clusters of computers using simple programming models. It is designed to scale up from single servers to thousands of machines, each offering local computation and storage. Rather than rely on hardware to deliver high-availability, the library itself is designed to detect and handle failures at the application layer, so delivering a highly-available service on top of a cluster of computers, each of which may be prone to failures.

The main modules provided are as follows:

*HDFS (Hadoop Distributed File System):* A distributed file system that provides high-throughput access to application data.

*Hadoop YARN (Yet Another Resource Negotiator):* A framework for job scheduling and cluster resource management.

*Hadoop MapReduce:* A YARN-based system for parallel processing of large data sets [6].

## VI. SPARK

Apache Spark is a data processing framework which accelerates processing tasks on very large data sets, and can also distribute data processing tasks across multiple computers, either on its own or in tandem with other distributed computing tools. These two qualities are key to big data, which require the marshalling of massive computing power to crunch through large data stores [7].

Spark is built on the concept of *distributed datasets*, which contain arbitrary Java or Python objects. A dataset is created from external data, over which parallel operations are applied. The building block of the Spark API is its RDD (Resilient Distributed Datasets) API, which consist of two types of operations: *transformations*, which define a new dataset based on previous ones, and *actions*, which kick off a job to execute on a cluster. On top of Spark's RDD API, other high level APIs are incorporated, providing a concise way to conduct certain data operations.

Spark is related to Hadoop in the sense that, it can run in Hadoop clusters through YARN (or Spark's standalone mode), and it can process data in HDFS, and any Hadoop InputFormat. It is designed to perform both batch processing (similar to MapReduce) and new workloads like streaming, interactive queries, and machine learning [8].

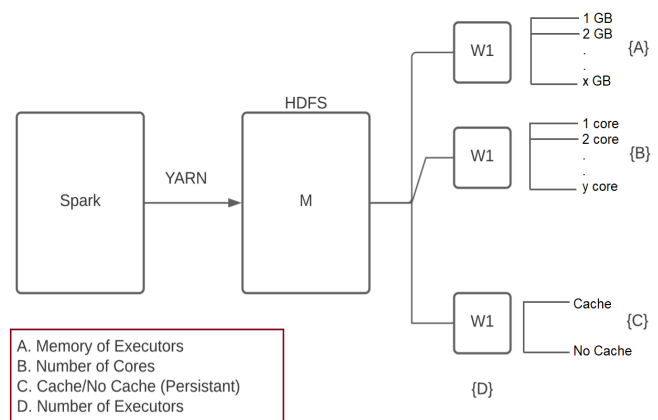


Figure 4: The layout of the platform of implementation, comprising the Spark framework established over HDFS.

## VII. DATASET

The dataset utilized in this project is the popular Berkeley-Stanford web graph, used often in research pertaining to PageRank algorithms and related applications. It comprises numerous nodes (denoting pages in berkeley.edu and stanford.edu domains), and directed edges (representing hyperlinks between them) [9].

Dataset statistics	
Nodes	685230
Edges	7600595
Nodes in largest WCC	654782 (0.956)
Edges in largest WCC	7499425 (0.987)
Nodes in largest SCC	334857 (0.489)
Edges in largest SCC	4523232 (0.595)
Average clustering coefficient	0.5967
Number of triangles	64690980
Fraction of closed triangles	0.002746
Diameter (longest shortest path)	514
90-percentile effective diameter	9.9

Figure 5: Characteristics of the Berkeley-Stanford web graph dataset.

## VIII. SCU COMPUTING CLUSTER DETAILS

### Cluster Info

#### Overall Cluster

- Environment: Cloudera CDH-5.16 - YARN (MapReduce v2) and Spark (2.4)
- Worker Nodes: 24
- Cores: 96
- Threads: 192
- RAM: 768GB
- HDFS Storage (Raw): 261TB
- HDFS Storage (Usable): 80TB (After factoring replication overhead)

#### Specific Nodes

- NameNode
  - Hostname: name1.hadoop.dc.engr.scu.edu
  - Cores: 4
  - RAM: 32GB
- SecondaryNameNode
  - Hostname: name2.hadoop.dc.engr.scu.edu
  - Cores: 4
  - RAM: 32GB
- 24x Worker Nodes (DataNode/NodeManager)
  - Hostname: worker-M-NN.hadoop.dc.engr.scu.edu (Where M is 1-2 and NN is 01-12)
  - Cores: 4
  - Threads: 8
  - RAM: 32GB
  - Storage: 12TB
  - Bandwidth: 1Gbit

Figure 6: Configurations of the various nodes within the SCU cluster system.

## IX. ALGORITHM IMPLEMENTATION

### A. Algorithm

The PageRank algorithm is executed in an iterative manner. In each iteration, the contributions are calculated, followed by recalculations of ranks based on these contributions. The algorithm consists of the following steps:

1. Loading of input\_file, of the format: node,neighbour\_node:  

```
dataSet=spark.sparkContext.textFile(input_file)
```
2. Initializing nodes and neighbour\_node:  

```
pairs = dataSet.map(lambda x: tuple([float(n) for n in re.split('\t', x)]).distinct().groupByKey())
```
3. Loading all node and associated neighbour\_node from input\_file and initializing ranks of each node to 1:  

```
ranks = pairs.mapValues(lambda x: 1.0)
```
4. The continuous calculation/updating of node ranks with the PageRank algorithm:

```
# Calculate/Update node ranks with PageRank algorithm continuously
for iteration in range(iterations):

    # Calculates node contributions to the rank of other nodes.
    node_contrib=
    pairs.join(ranks).flatMap(
        lambda nodes_rank:
        computeContributions(nodes_rank[1][0],
        nodes_rank[1][1]))

    ranks=
    node_contrib.reduceByKey(add).mapValues(
    lambda x: x * 0.85 + 0.15)
```

The pairs RDD and ranks RDD are joined together to form an RDD, the values of which are extracted to form another RDD that is flat mapped to generate node\_contrib RDD. The rank of each node is distributed evenly across the other node it references to. The function named computeContributions calculates the contributions of each node to the ranks of other nodes.

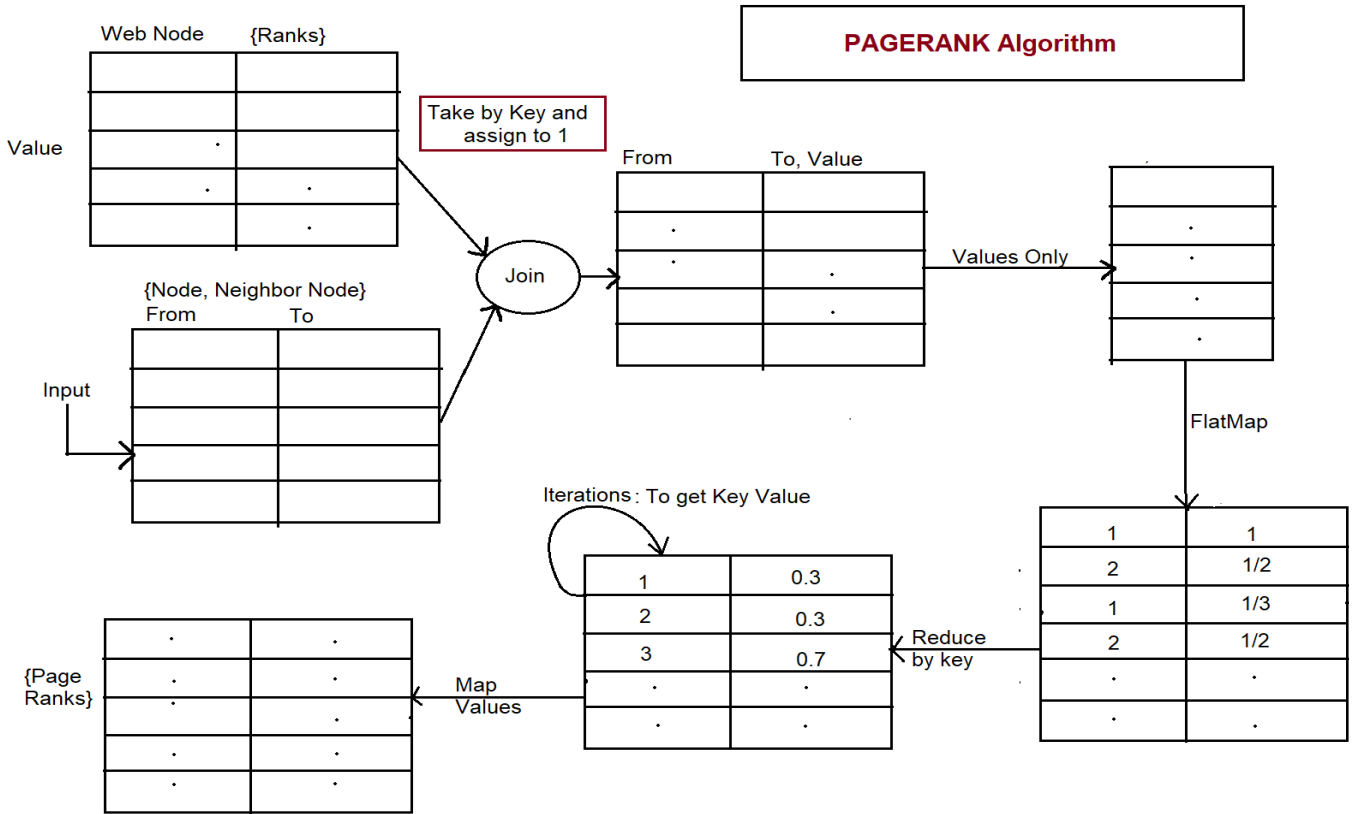


Figure 7: Illustration of RDDs generated w  
itera

## B. With Default partitioning

The PageRank implementation without any user-provided custom partitioning has 2 partitions by default. The default configuration for spark has 4 cores and 4 executors. The implementation is explained in detail in sub-section A - 'Algorithm' under this section. The experiment results are captured in section XIV, B - 'Results from Default setup.'

## C. With External partitioning

One of the important aspects in Spark is to control the partition of the dataset between the nodes. Controlling the distribution of data can help in improving the overall performance as the communication cost is very high in distributed systems. By controlling the RDD partitioning, Spark can reduce the communication overhead. A custom partitioning has been implemented to increase the overall performance.

The below configuration is applied to this experiment:

```
spark = SparkSession\
    .builder\
    .appName("PythonPageRank")\
    .config("spark.driver.memory", "1g")\
    .config("spark.eventlog.enabled", "true")\
    .config("spark.executor.memory", "1g")\
    .config("spark.executor.cores", "4")\
    .config("spark.task.cpus", "1")\
    .config("spark.executor.instances", "4")\
    .config("spark.default.parallelism", "16")\
    .getOrCreate()\
```

Figure 8: Spark configuration for running Pagerank  
with external partitioning.

- To increase the parallelism between the executors, the parallelism is set to 16 as we have 4 cores and 4 executors.

```
.config("spark.default.parallelism", "16")\
```



- Applied co-partitioning using `partitionBy(numPartitions)` to improve the overall performance and to efficiently parallelize the I/O while reading the input dataset from HDFS, the partitions are specified and observed to have improved performance.

To maintain the same partitioning for the ranks RDD from the pairs RDD, `mapvalues` has been used.

```
# Loading in input_file. It should be in
format of: node, neighbor_node
```

```
dataSet=
spark.sparkContext.textFile(input_file,pa
rtitions)
```

```
# Loading all nodes from input_file and
initializes the neighbor_node.
```

```
pairs = dataSet.map(lambda x:
tuple([float(n) for n in re.split('[
\\t]',
x)])) .distinct().groupByKey().partitionBy
(partitions)
```

```
# Loading all nodes with other node(s)
link to from input_file and initializes
ranks of every node to 1
```

```
ranks = pairs.mapValues(lambda x:
1.0).partitionBy(partitions)
```

```
ranks = ranks.partitionBy(partitions)
```

`flatMap` is used on the RDD that is resulted by joining pairs and ranks RDDs. As it loses the partitioning of its parent, `partitionBY(numPartitions)` is used on ranks RDD after every iteration.

```
# Calculate/Update node ranks with
PageRank algorithm continuously
```

```
for iteration in range(iterations):
```

```
# Calculates node contributions to the
rank of other nodes.
```

```
node_contrib =
pairs.join(ranks).flatMap(
lambda nodes_rank:
computeContributions(nodes_rank[1][0],
nodes_rank[1][1]))
```

```
ranks =
node_contrib.reduceByKey(add).mapValues(1
ambda x: x * 0.85 +
0.15).partitionBy(partitions)
```

- Upon allocating more cores on external co-partitioning, the execution time has improved and the results showed a significant increase in performance. This experiment is tested with 8 cores and 16 partitions.

```
.config("spark.executor.cores","8")\
```

The results are captured in section X, C - ‘Results from changing the configuration parameters of yarn.’

## D. With Persistence

Considering the efficiency of an iterative computation, applying persistence on the pairs and ranks RDDs are helpful, as they are loaded in every iteration. This saves the time it takes to load the data from the disk or while shuffling it across the network.

```
pairs = dataSet.map(lambda x: tuple([float(n) for
n in re.split('[ \\t]',
x)])) .distinct().groupByKey().partitionBy(partiti
ons).cache()
```

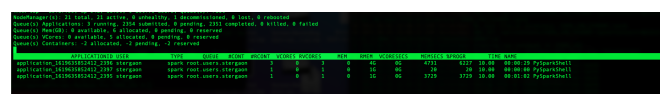
```
ranks =
node_contrib.reduceByKey(add).mapValues(lambda x:
x * 0.85 + 0.15).partitionBy(partitions).cache()
```

## X. EXPERIMENTAL RESULTS

### A. Utilities used in gathering statistics

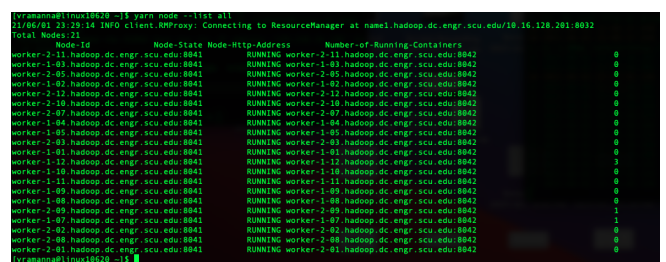
i. `yarn top`

Lists all the tasks spawned by user



ii. `yarn node --list all`

List all partitions instances used by everybody



## B. Results from default Setup

Below is an illustration of the experimental results from the default setup for PageRank algorithm, i.e., without partitions against iterations in multiples of 10 upto 90.

Default Configuration:-----  
 Number of cores : 4  
 Partitions(default) : 2  
 Executors : 4 by default

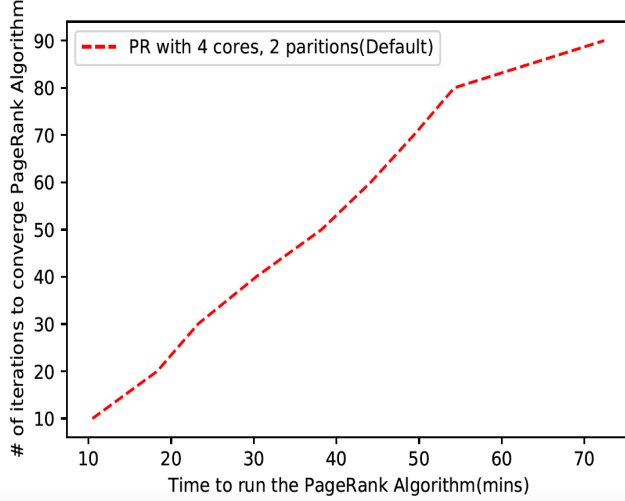


Figure 8: Experimental statistics of the time taken for the PageRank algorithm, with default setup against iterations in multiples of 10 upto 90.

## C. Results from changing the configuration parameters of yarn

Below is an illustration of the experimental results from the default setup for pagerank algorithm with the below configurations:

- Different Partitions against multiple iterations
- Different number of core configurations:
  - a. 4 core and 22 partitions
  - b. 8 cores and 16 partitions**
  - c. 4 cores and 2 partitions(Default setup)

On running with 8 cores and 16 partitions we converged after 210 iterations and validated all the ranks having similar values using. This was the most optimized configuration we could achieve.

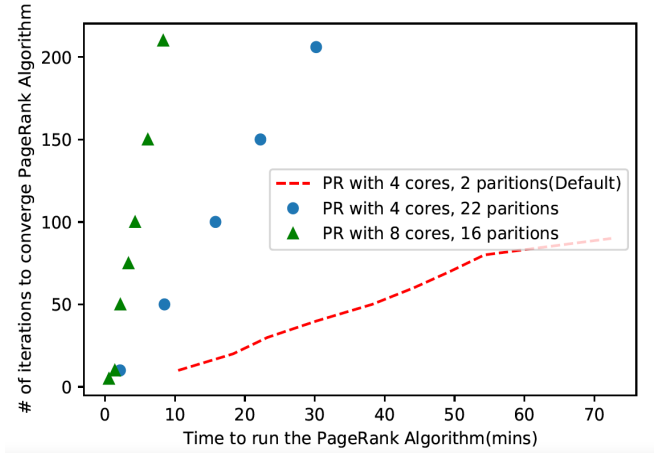


Figure 9: Experimental statistics of the time taken for pagerank with default setup against iterations with different partitions and different Core configurations.

## D. Results from applying persistence on PageRank

Below are the results captured for with/without persistence on default and custom partitioning:

# of Iterations =100	With persistence	Without persistence
With Partitioning and 8 cores	4m4.314s	4m30.251s
With Partitioning and 4 cores	12m27.32s	15m18.13s

Table 1: Experimental statistics of the time taken for the PageRank algorithm, with a focus on the improvements brought forth by enabling persistence, i.e., enabling storing RDDs in cache memory.

## XI. CONCLUSION

Looking at the results, it is quite evident that big data offers tremendous advantage for organizations required to crunch through massive hordes of data so as to obtain valuable information which can assist in decision making of utmost importance. The benefits offered by utilizing Spark over a distributed file system clearly showcases the superiority of distributed computing paradigm over the localized computing paradigm. The magnitude of observed speedups is a precursor of the fact that tremendous computational efficiency stands to be attained, and lends itself to a compelling prospect in implementing a platform for large-scale data processing, moreso with digital content being generated at quicker rates than ever before.



## XII. REFERENCES

- [1] <https://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.359.9835&rep=rep1&type=pdf>
  - [2] L. Page, S. Brin, R. Motwani, and T. Winograd, "The PageRank Citation Ranking: Bringing Order to the Web," 1999
  - [3] J. Leskovec, A. Rajaraman, and J. D. Ullman, "Mining of Massive Datasets," 2014.
  - [4] "Big Data - What it is and why it matters," [https://www.sas.com/en\\_us/insights/big-data/what-is-big-data.html](https://www.sas.com/en_us/insights/big-data/what-is-big-data.html)
  - [5] "What is distributed computing - IBM Documentation," <https://www.ibm.com/docs/en/txseries/8.1.0?topic=overview-what-is-distributed-computing>
  - [6] "Apache Hadoop," <http://hadoop.apache.org>
  - [7] "What is Apache Spark? The big data platform that crushed Hadoop," <https://www.infoworld.com/article/3236869/what-is-apache-spark-the-big-data-platform-that-crushed-hadoop.html>
  - [8] "Apache Spark," <https://spark.apache.org/faq.html>
  - [9] "Berkeley-Stanford web graph," <https://snap.stanford.edu/data/web-BerkStan.html>
-