MAYNOOTH UNIVERSITY

MASTERS THESIS

Publication Citation Network Analysis Using Spark and GraphX

Author:
Ramesh SURAGAM

Supervisor:
Dr. Dapeng DONG

A thesis submitted in fulfillment of the requirements for the degree of Master of Science

in the

Data Science

August 10, 2020

Declaration of Authorship

I, Ramesh SURAGAM, declare that this thesis titled, "Publication Citation Network Analysis Using Spark and GraphX" and the work presented in it are my own. I confirm that:

- This research was conducted wholly or mainly while at this university in candidacy for a research degree.
- Where any part of this thesis has previously been submitted at this University or any other institution for a degree or other qualification, this has been stated clearly.
- Where I have consulted the published work of others, this is always clearly attributed.
- Wherever I have quoted from others' work, there is always the source given. Except for quotes like these, this article is entirely my own work.
- I have acknowledged all main sources of help.
- Where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself.

Signed: Ramesh Suragam

Date: August 10,2020

""In my perception, the world wasn't a graph or formula or an equation. It was a story.""

Cheryl Strayed

Abstract

The popularity of the scientific publications keeps on increasing day by day and the number of publications is also immensely increasing. Thus, it becomes more challenging for any researcher to search for a topic, review the literature, follow research trends in that field. The use of online search engines help to a certain extent but the number of results is vast. In the past, many solutions are proposed to analyse the document similarities using Natural Language Processing(NLP) which makes use of pattern matching but these results are not so accurate.

In this dissertation, we propose a system to analyse the document similarities by making use of the connectedness between them which can be known by using the citations and references that are given by the original authors of the document. Our system makes use of the semantic scholar publication data to form a citation network graph. This network thus gives a much better insight into a publication among a network of publications. The graph that is constructed with this approach contains publications as nodes of the graph and citations as directed edges of the graph. Besides, this approach helps in searching for publications, journals as a separate query and dynamic graphs for each of the entities publications, journal are created. The interactive system is cable of getting the most influential publications/journals connected to any selected publication.

Another key aspect of our dissertation is the use of emerging big data analytics system, i.e. "Apache Spark" platform and "GraphX" framework for citation network analysis. The project was implemented in national supercomputing infrastructure (ICHEC) with the 47 GB raw data of publication citation network and references.

Acknowledgements

It is a pleasant task to express my thanks to all those who contributed in many ways to the successful completion of this thesis.

I would first like to express my deep and sincere gratitude to my supervisor Dr. Dapeng Dong, for the conceptualization of the subject and guidance, motivation, and patience all the way through my master's thesis. He taught me the methodology of conducting the research and of presenting the research work as clearly as possible. For me, the joy and enthusiasm he has towards research has been contagious and motivational. Sir... your valuable contributions will always be treasured.

I gratefully acknowledge the funding sources that made this masters thesis project possible. This work was funded by the Department of Computer Science of NUI Maynooth to whom I owe a great deal of appreciation. The technical contributions and support were provided by the Irish Centre for High-End Computing(ICHEC) which are truly appreciated. Without their support, this project cannot have reached its goal.

And most of all, I would like to acknowledge my family's support and great love where the most fundamental source of strength in my life resides. They kept me motivated and without their help, the research would not have been possible.

Contents

List of Figures

List of Tables

List of Listings

List of Abbreviations

IR	**I**nformation **R**etrieval
CN	**C**itation **N**etwork
DAG	**D**irected **A**cyclic **G**raph
RDD	**R**esilient **D**istributed **D**atasets
RDG	**R**esilient **D**istribute **G**raph
JSON	**J**ava **S**cript **O**bject **N**otation
PR	**P**age **R**ank
HDFS	**H**adoop **D**istributed **F**ile **S**ystem
YARN	**Y**et **A**nother **R**esource **N**egotiator
ETL	**E**xtract **T**ransform **L**oad
JAR	**J**ava **A**rchive
API	**A**pplication **P**rogramming **I**nterface
NPPC	**N**ode **P**air **P**rojection **C**ount
SPLC	**S**earch **P**ath **L**ink **C**ount
SPNP	**S**earch **P**ath **N**ode **P**air
ICHEC	**I**rish **C**entre for **H**igh **E**nd **C**omputing
HEI	**H**igher **E**ducation **I**nstitutions
NS	**N**ational HPC **S**ervice
HPC	**H**igh- **P**erformance **C**omputing
GPU	**G**raphics **P**rocessing **U**nit
GHz	**G**iga **H**ertz
RAM	**R**andom **A**ccess **M**emory
CPU	**C**entral **P**rocessing **U**nit
SSD	**S**olid **S**tate **D**rive
S2 ORC	**S**emantic **S**cholar **O**pen **R**esearch **C**orpus
EDA	**E**xploratory **D**ata **A**nalysis
Scala	**S**calable **L**anguage
JVM	**J**ava **V**irtual **M**achine
IDE	**I**ntegrated **D**evelopment **E**nvironment
Bash	**B**ourne **A**gain **S**hell

List of Symbols

N	Citation Network
G	Graph
V	Vertex
A	Arc
E	Edge
R	Citation relation
P	Vertex properties
W	Weights of line values
v	Set of vertices or nodes
ℓ	Lines or Links
w	weight of a single arc
n	Number of vertices
m	Number of lines

Chapter 1

Introduction

1.1 Introduction to Publication Citation Network Analysis

The main aim of this research is to examine the *publication citation network* and to determine the importance of a collection of documents.

It has been used in many practical applications such as tracing prior art and citation of early patents, calculating citation indexes, and referring back to judgements made in the previous cases by judges. A citation network can be formulated in a directed graph of relevant documents, however, given that the ever-increasing numbers of documents (e.g., IEEE Xplore contains more than five million technical articles), analysing citation networks at large scale becomes a challenging task.

The focus of much of my work has been on developing a generic framework for citation network analysis based on the "Apache Spark platform" and "GraphX APIs". The outcomes of the project are twofold.

- The first section deals with the development of a generic framework for citation analysis based on Spark and "GraphX" on the big data set that we have.
- The second section deals with the development a set of basic functions that can be used by non-technical personnel.

1.2 Motivations of the project

A lot of scholars worldwide are working on science projects and writing academic papers. As a result, many papers with various scientific contributions and impacts are published daily. It is therefore overwhelming that these published papers need to be evaluated and their quality assessed. While identifying the key criteria for this evaluation, we must understand that any publication in scientific periodicals is based on the knowledge of others and acknowledge this by referring to earlier publications is called *citation*. Although the literature contains a variety of criteria for evaluating the quality of a scientific paper, the most important evaluation metrics can be considered as the number of citations to the paper[See 1]. This number of citations is an important indicator, as it is widely used to measure the impact of a paper(Garfield, 1998; Moed, 2005; Oppenheim,1995).

The network of scientific journal publications started to grow rapidly as the referencing of older publications became standard practice. It may be helpful to attempt to imagine the spatial expansion of the entire publication citation network as an ever-growing spear that adds a new "growth ring" each year in which papers are found through the sources they quote [See 2, p3]. That said, given the enormous results, determining the citation count for various papers is a huge issue.

This problem has multiple applications in various domains. Researchers need to identify the most important articles in advance with the the amount of published papers, so that they can prepare their research direction(Yan et al., 2012; Amjad et al., 2017). In addition, evaluating citation counts of paper will help us understand that this paper can help us assess the future impact of paper authors, with potential applications in hiring researchers and faculties, and awards and funding. There have been various efforts to gain such insights into the future impact of researchers in the literature (Chan et al., 2018; Havemann and Larsen, 2015; Revesz, 2014; Fialaand Tutoky, 2017). This motivates us to build a system that helps us create a citation network for analyzing and retrieving information using the “GraphX” framework in combination with Apache spark to handle big data loading.

1.3 Thesis Organization

The rest of the thesis is organized as follows.

- In Chapter 2, we elaborates the literature review which involves the concepts of Information Retrieval systems and how they have evolved and their drawbacks. Following which we discuss the basics of a bibliometric network and move towards citation networks. After we get a fair understanding of these networks we shall dig deep into understanding the various similarity metrics that help us analyse the citations among them. Later on, we get into the technical aspects of the proposed graph theory and how it can be applied to big data.
- In Chapter 3, we deal with the analysis of citation networks and about the design of the system for this project. This section also talks about the system work-flow and architecture.
- In Chapter 4, we describe the computation environment configuration, dataset being used, and their sources, the software, and programming languages used in the project.
- In Chapter 5, we discuss how the implementation of the project is carried out and how various algorithms were designed to build the system discussed in Chapter 4.
- In Chapter 6, we discuss how the objectives are achieved and the work is evaluated by checking the performance, accuracy, and resource usage.
- In Chapter 7, we conclude our thesis and discuss the author’s contribution and future works and improvements.

Chapter 2

Background and Literature Review

In this chapter, we shed light on several methods related to the main concept of the thesis, which is bibliographic network analysis focusing on the citation networks.

One of the key aspects of citation networks is topic evolution. Topic evolution analysis over some time helps us understand and objectively evaluate the contribution of one publication or an article to another. Moreover, topic evolution analysis may lead to information retrieval tools that can recommend citations[See 3]. Apart from this many other information retrieval communities make use of the information retrieval for the analysis of topic evolution or trends. It is this aspect of information retrieval that this work is connected to. This chapter is composed of the following sections: information retrieval, citation networks, document similarity measurements, and graph theory.

2.1 Information Retrieval

In the current world of Internet of Things where every device is connected and with the amount of time we spend in the digital space, the amount of data generated each nano-second is very huge. Anyone who searches for information is required to make more decisions about searching and expected to engage with an increased number and variety of search systems. The most famous example of Information Retrieval systems(IR systems) is the web search where users can access any information of interest including current events, to locate people and organizations, find out answers to queries, on-line shopping, and much more. The second most example of IR services are digital libraries. These help students and academic researchers to discover and obtain the required information about journals, scientific articles, conferences, and to connect them to recent research news in their specific domain.

A digital library is a focused collection of documents stored in organized electronic text forms available on the internet or digital resources such as multimedia sources including audio and video materials. The main purpose of these digital libraries is to assist users in seeking the required information. Furthermore, a certain distinction is made between content that was created in a digital format and the information that has been converted from a physical medium like paper. There may be

different kinds of service provide of digital libraries. They include Academic repositories and Digital Archives[1].

- The Academic repositories are the repositories that institutions build by making use of their books, papers, thesis, and other works that can be digitized. Many of these are made public with certain restrictions.
- On the other hand, a digital archive contains sources of information that are produced directly by individuals or organizations rather than the secondary sources found in a library (books, papers, journals, and so on). Furthermore, a digital archive always contains an organized group of items rather than individual items. And lastly, the contents of these digital archives are unique, unlike books that may be found in different libraries.

Although these IR systems have been highly efficient and organized there are still certain aspects in which they fall behind.

2.2 Drawbacks of current IR systems

Even though the current web IR systems use sophisticated and complex methods, the users still face several issues. These web-based IR systems often provide the users with results that often contain a lot of irrelevant documents and the documents actually matching user's needs are stashed in the rest of the results. This is mainly caused because of the keyword-based or pattern match or textual similarity-based search mechanisms of the IR systems, both operational and experimental to display relevant results to the users. Some IR systems may include a form of normalisation and some sort of weighting. Some others use distributional information to measure the strength of relationships between keywords or between the keyword descriptions of documents. Yet another aspect of the drawback is due to the subjectivity of human understanding.

The user queries are often vague and the notion of what should actually be returned as relevant results is very indefinite. To improve this condition, different approaches of graph analysis for the data retrieval combined with big data which can be achieved by incorporating the usage of "graphX" and "Apache Spark" to build citation networks and ranking the results was proposed. In this paper, we describe how different citation network graphs can be created based on the user's requirement and provide better results. To reach this point, we need to first understand the very basics of how articles, papers, publications are connected to form networks and how this science of bibliometrics was formed.

2.3 Bibliometric Networks

As discussed in section 2.1, the information we deal with on-line has a fundamental network structure. Links among web pages, for example, can help us understand

[1] *Digital library - New World Encyclopedia* https://www.newworldencyclopedia.org/entry/digital_library#Types_of_digital_libraries.

how these papers are related, how they are grouped into different communities, and which pages are the most prominent or important.This way they build a bibliometric network.

Bibliometrics is a field of study devoted to statistical analysis of bibliographic information. The bibliographic summary of a published work includes a variety of elements such as the writer's names, paper title, keywords, and details required to identify the text such as the journal title or edited volume in which an article appears, year of publication, volume / issue number, page numbers. They also store and preserve this information in databases. Each of these elements constitute a document's bibliographic attributes, also known as the metadata[See 2] comprising the main aspects of designing a bibliometric network.

A bibliometric network is composed of nodes and links, or edges. The nodes can be instances of publications , journals, authors, while the edges indicate the relation between the nodes that can be citation relationships, co-authorship relationships. The bibliometric networks that contain the citation relations are named *Citation Networks* and those with co-authorship relations are named as *Co-authorship relations Networks*. These two are the most common types of bibliometric networks of which we shall now discuss the Citations Networks and its technical definition in the next section.

2.4 Citation Networks

Before we understand the citation networks let us first understand what a network is all about. A network is a graph along with its properties. Informally, a ***Network = Graph + Data***. Formally, a network $\mathbf{N} = (v, \ell, P, W)$ consists of:

- A graph $\mathbf{G} = (v, \ell)$ where v is the set of vertices, $\boldsymbol{A}$ is the set of arcs, $\boldsymbol{E}$ is the set of edges, with $\ell = \boldsymbol{E} \cup \boldsymbol{A}$ the set of lines, $\boldsymbol{E} \cap \boldsymbol{A} = \emptyset$. If the line is directed, then it is defined as an *Arc*($\boldsymbol{A}$) and if the line is undirected then it is defined as an *Edge*($\boldsymbol{E}$). On the same note, a graph in which $\boldsymbol{A} = \emptyset$ is called an undirected graph. On the other hand, when $\boldsymbol{E} = \emptyset$ it is a directed graph. The number of vertices is denoted by $\mathbf{n} = |v|$ and the number of lines by $\mathbf{m} = |\ell|$.
- $\boldsymbol{P}$ is a set of vertex value functions, called properties, mapping vertices to a domain $A_p : v \to A_p$.
- $\boldsymbol{W}$ is a set of line value functions, called weights, mapping lines to a domain $B_w : w : \ell \to B_w$.

A sample graph can be shown in Figure 2.1[See 5] which shows the basic attributes of a graph. This graph is not to be called a citation network because it contains edges that are undirected while a citation network comprises of only directed edges which define the citing relations.

A Citation network is a set of citing relations between various units of a given set. For instance, if we have a set of units or vertices v (articles, books, works, publications, ...) we introduce a citing relation $R \subseteq v \times v$. The citation relation R can

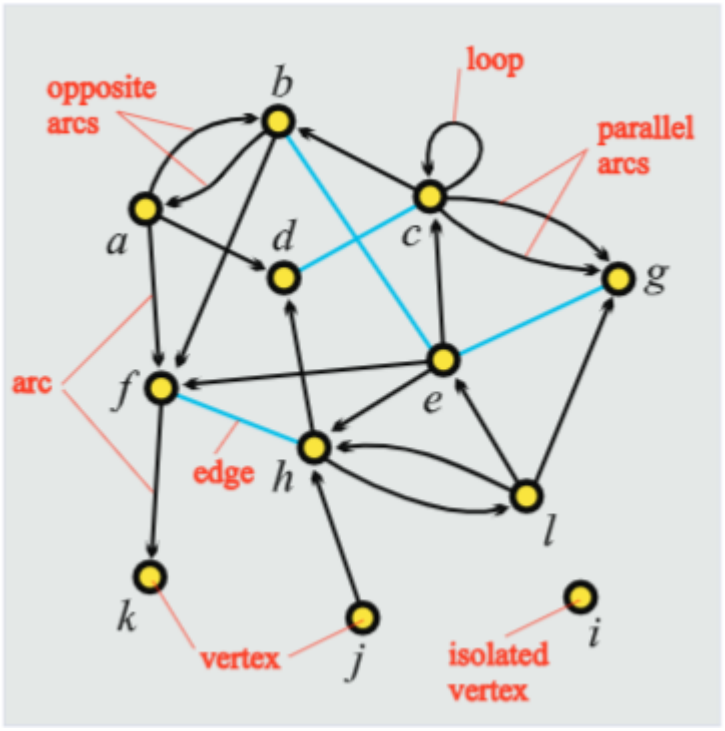

FIGURE 2.1: Some basic terms for graphs.

also be termed as an arc or as a directed edge which determines a citation network $\mathbf{N} = (\mathbf{v}, \mathbf{R})$ [See 5]. The equation for a citation network can be explained with the below equation:

$$v\mathbf{R}\nu \equiv \nu \text{ cites } v$$

A sample citation network graph can be shown in the Figure 2.2 which contains vertices representing the units and edges representing the citing relations.

Since the edges are directed, they can be either citing from or citing to another vertex. These are termed as *in-degrees* and *out-degrees*. Every vertex can have an *in-degree* or an *out-degree* or *both*. For instance, if we have a vertex v, then considering the in-degree of v will be the number of edges going towards v as the terminal node. Similarly, the out-degree of v will be the number of edges going away from v as the initial node.

In our thesis, we discuss about publications and how they are interconnected with citations. These citations are modelled as edges directed according to the flow of information, i.e. cited to citing publication. Accordingly, the importance of a citation edge is based on the number of nodes that it connects in the network. In directed acyclic graphs (DAGs) like citation networks, this corresponds to the extent an edge is stressed when information flows from source nodes (early publications) to sink nodes (latest publications). The citation relationships can be better understood by categorising them.

When a publication is cited inside another publication, we can consider that there is direct citation relation between those two publications. On the other hand,

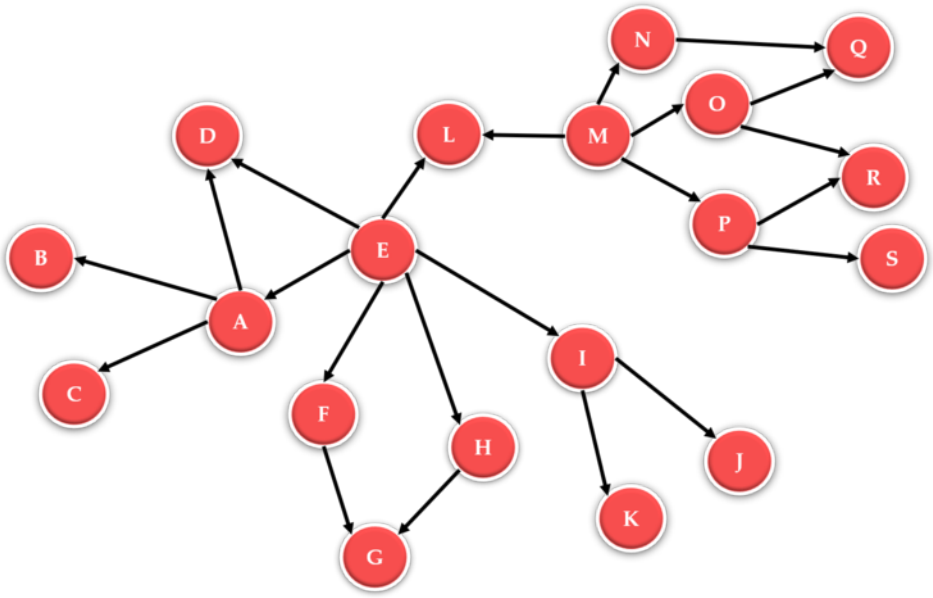

FIGURE 2.2: Sample Citation Network.

when two publications are cited in a third article but there is no direct citation between the first two publications then they are said to have a co-cited relationship. As more and more publications cite these two publications, they are said to be strongly co-cited with each other. On contrast, if a third publication is cited by two publications then they are said to be bibliographically coupled and the strength of the relation between them is defined by how commonly the third publication is referred to in both the publications.

Apart from analysing how the publications are interconnected we also discuss about the journals these publications are grouped into and their citations.

2.5 Citation Networks of Journals

Research activities have developed over the years and have been spread among different expert groups representing new fields of scientific endeavour. Thus, each community created their own specialist journals. Such journals, however, also often include articles from other research fields outside of their domain, which at a given time might be of interest to any reader of a specific journal.This addition of literature outside of a core research field suggests that publications in one journal should cite journal publications from contiguous fields more often than those from fields or disciplines further away. The citation flows in a network of journals are influenced by three main factors which are: thematic contiguity, the size of a journal, and the reputation of a journal. Apart from these, there is another key influencing variable which is the usual number of cited sources per article for the particular area of speciality in question. The review publications with long references are cited more frequently than original articles publishing research results. Thus, journal citation networks, constructed with such a normalization, will just mirror the actual thematic relationships of those periodicals and their respective reputations[See 2, p8].

Since journal networks contain fewer nodes when compared to publication networks they are more transparent and also more easier for numeric analysis. The citation numbers required to build a journal network are presented in aggregated form[See 2, p9]. As part of this thesis, we implement this aspect by using graph theory in "GraphX" on Big data. In the next section, we shall discuss the similarity metrics of the documents and what are systems were defined in the past are available and how we can tweak them to fit our purpose.

2.6 Similarity Metrics

Semantic similarity is a metric defined over a set of documents or terms, where the In a broad sense, the similarity between documents can be partitioned into three categories: first, string-based(character-based and term-based) secondly, corpus-based and finally knowledge-based (similarity, relatedness). The similarity between documents is determined by the shared features relevant to the nature of the comparison. The frequency of observation of such shared features is used to determine the degree of similarity[See 6]. In the linguistics literature, it is common to classify features of text documents into three broad feature classes, namely syntactic, semantic, and lexical[2]. Thus, a pair of documents can be similar or dissimilar based on the purpose of the similarity. We may represent the organisation of the various types of document similarity and their relation of the classes and properties with the below figure.

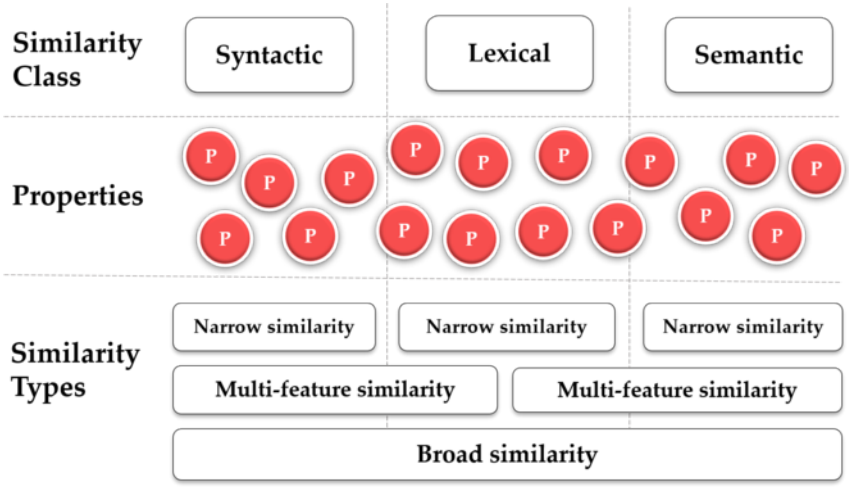

FIGURE 2.3: Classes, Properties and Types of Document Similarities[See 6, p26]

Furthermore, certain tokens representing document properties have been shown overlapping the boundaries between similarity classes. For instance, we consider the presence or absence of particular terms in a document to be a syntactic feature, yet the presence of particular terms in a document to be a syntactic feature, yet the presence of groups of terms holding some lexical relation to be a lexical feature.

[2]Certain authors also add to this set of feature classes the class of pragmatic features, under which we consider how words and expressions contained in a document are used. These are, without a doubt, important features when determining the meaning of text, but for that reason precisely I would consider such features to be semantic

In this thesis, our focus is about searching for scientific publications and measuring the similarity between the resulting publications based on their thematic topics. Thus, what we are trying to achieve can be either syntactic similarity or semantic similarity based on whether we are checking for the similarity between query text and corpus documents or just paraphrasing a paragraph or summarizing documents. This again produces a new publication that is semantically similar to the original document[See 6].

2.7 Document similarity using NLP

One of the advanced approaches to compare documents is finding how similar their words are. There are two methods to do this. One by using the Term Frequency-inverse document frequency (TF-idf) technique where we look at words that appear in both pieces of text and scores them based on how often they appear. This, however, is a useful method if we expect the same words to appear in both pieces of text, but some words are more important than others. The second method is by calculating the semantic similarity score by borrowing techniques from natural language processing (NLP), such as word embeddings. This is useful if the word overlap between texts is limited.

2.8 Citation based similarity measures

The next approach is to find the similarities between publications which can be determined by the citation linkages or the co-occurrence similarities. Citation linkages include direct citation links, co-citation links, bibiliographic coupling, or a combination of two or all of the above. The citation similarities are derived directly from the citation databases. The nature of the citation relations depends on the hierarchy of the provided database[See 7]. However, determining the strength and the weakness of the relationship is challenging due to the lack of extensive experiments and standards for comparison on this research issue. Co-occurrence similarities include co-author, co-citation, or keywords co-occurrence. We use this method of citation-based similarity measures to build a graph and get all these metrics needed. Before we proceed further, we need to understand about the graph theory and how it is applied.

2.9 Graph Theory

As discussed in the above sections, there is an urgent need to efficiently deal with the emerging data every nano-second and extract knowledge out of it. One way to solve this problem is to look at this data as a network connected with links representing relationships between the objects or systems. One can effectively address many challenges if we understand the "connectedness" of these complex systems. The major aspect of this connectedness can be understood by Graph Theory which is the study of the network structure - who is connected to whom. A small example of “connectedness” is the interaction between search engines and the authors of web pages. [See

8] Before we dive deep into the Graph Theory we need to understand the difference between a network, a graph, and how they can be used interchangeably.

A network is a terminology that non-mathematicians use to refer to real systems. For example, a social network is a real-world system of people connected with each other and the relationship between people is friendship. When dealing with a network we talk about nodes and links. In the previous example, people are represented as nodes and the friendships are related as links.

On the other hand, a graph is a terminology that mathematicians use when they discuss the mathematical representation of the networks. A social graph is constructed for the social network for the mathematician to work with. Here, we represent people as vertices and friendships as edges. This distinction between networks and graphs is rarely used, so these two terminologies are often synonyms of each other, and we shall use them interchangeably.

The organization of interactions in a community is best represented as a network. Graph theory is a field of mathematics developed to analyse the structure of such systems. Every community can be abstracted by a graph, which is a representation of the system components and their arrangement. These components are called nodes and linked together by edges[See 9, p3]. There is a lot of research going on using the graph theory in the recent times. In this paper published in 2019[See 9], the author makes use of graph theory for important scientific discoveries, such as improved understanding of the breakdown of electricity distribution systems or the propagation of infections in social networks and to characterize the complexity of ecological networks. Graph Theory can be applied to absolutely anything. It can be used as simple as social networking and as complex as geomorphology for characterizing geomorphic systems and incorporating information from a multitude of information sources[See 10]. Graph theory when used along with Network analysis can facilitate the characterization of spatial relationships between landforms, sediment sources and sink, transport vectors, and may serve as a basis for process-regime mapping(Bishop et al., 2018). Taking this a step forward by traversing our discussion to the technical aspects of graph theory implementation in our work we talk about the Graph X framework in the next section.

2.10 Graph X framework

Applying traditional graph algorithms to extremely large graphs requires distributed processing as well as out-of-core computation. Therefore, several parallel graph frameworks such as "GraphX" have been developed on top of data-parallel systems, such as Apache Spark. To take advantage of such distributed graph processing frameworks, we need to design new map-reduce network embedding algorithms. A common assumption underlying existing methods and our new algorithm is that we expect that the embedding of a vertex is more similar to the embeddings of its neighbours rather than to the embedding of a random vertex outside of its neighbourhood[See 11].

Handling graphs in a data-parallel computation system is thus more complex than map-reduce operations when the vertices are to be processed in the context of their neighbours. To address that, "GraphX" introduced edge triplets which join

the structure of vertices and edge RDDs. Each triplet carriesan edge attribute and the attributes of vertices incident to that edge. Therefore, by grouping the triples on the id of the source or destination vertex, one can access the value of all the neighbours of each vertex. Moreover, since the triplets are distributed, if the neighbours of a vertex are located on different machines, then Spark workers have to communicate with each other to construct the result. Therefore, different strategies for distributing graphs over partitions result in significant differences in communication and storage overheads. "GraphX" supports both edge-cut and vertex-cut graph partitioning strate-gies[See 11].

Apart from these, "GraphX" also provides a vertex-centric programming model for developing distributed graph algorithms. Vertex-centric programming models such as Pregel are widely used for implementing sequential algorithms in graph-parallel frame-works such as Apache Giraph or "GraphX". In a vertex-centric programming model, we develop an algorithm from a vertex point of view, which in general includes three different steps: gathering messages from its parallel framework iteratively executes these steps in one super-steps until no more messages are produced by any vertex. "GraphX" implements all this functionality using map-reduce operations over edge triplets[See 11].

Big graphs have a wide range of applications. These graphs help us identify the relationships between users, products, and ideas thereby allowing us to identify communities, target advertising, and decipher the meaning of documents. Nowadays, the graph data have increased drastically and caused the implementation of large-scale distributed graph parallel frameworks. While there are many frameworks that address the challenges of graph computation, they do not take into consideration the data ETL (preprocessing and construction) or the process of interpreting and applying the results of the computation. However, we have certain data-parallel systems like MapReduce and spark that deal with scalable data processing and graph construction (ETL). However, doing this leads to complex joins and excessive data movement and does not exploit the graph structure.

The data-parallel computation uses a record centric view of data, while the graph- parallel computation uses a vertex and edge centric view of computation. Furthermore, the program logic in data-parallel systems is done by functional transformations on collections. On the other hand, existing graph-parallel systems restrict program logic to the level of vertex processes. These issues can be tackled with the help of "GraphX" which runs in the Spark data-parallel framework. "GraphX" extends the Spark's Resilient Distributed Dataset (RDD) to introduce the Resilient Distributed Graph (RDG). In "GraphX" each transformation yields a new immutable graph.

Chapter 3

Analysis/Design

3.1 Introduction

Several scholarly repositories such as Google Scholar, Microsoft Academic Search, Semantic Scholar play a vital role in scientific information propagation and to the discovery of related work. The evolution of citation networks have been under investigation for the past six decades. Many fascinating theories have been proposed to explain the intrinsic forces driving citation and to enable newer models to mimic more and more salient properties observed in real networks. In one of the publications[See 12], a different model for copying both the references made from (out-neighbours), as well as the citations made to(in-neighbours) a paper.

With the growing number of published scientific papers world-wide, the need to evaluate and quality assessment methods for research papers is increasing.

Another publication deals with GraphX implementation in Big data but concentrates on Network embedding as an important step in many different computations based on graph data[See 11]. Yet another publication deals with RDF data and the use of "SPARKQL" queries for graph processing systems based on Apache Spark GraphX[See 13]. This chapter is composed of the following sections: Analysis of Citation Networks, System Design, System specifications, System Architecture.

3.2 Analysis of Citation Networks

The analysis of citation networks is done by assigning weights for each unit/arc. This determines the importance of that unit/arc. These weights can later be used to determine the essential substructures in the network too. Hummon and Doreian (1989) [14, 15] proposed methods to assign weights to arcs. We consider arcs here instead of edges because arcs represent directed edges which are how citations are interpreted in a graph or citation network. The following equation describes this analysis of citation networks where w represents the weights of arcs and R represents the relationship between any two arcs:

$$w : R \rightarrow R_0^+$$

There are three different methods that are proposed by Hummon and Doreian to achieve this task which are:

- Node Pair Projection Count (NPPC) method:
  - This method considers all paths between all node pairs an edge occurs on.

$$w_d(u,v) = |R^{inv*}(u)|.|R^*(v)|$$

- Search Path Link Count (SPLC) method:
  - This method considers the number of "all possible search paths through the network emanating from an origin node through the arc $(\boldsymbol{u}, \mathbf{v}) \in \mathbf{R}$", [2, p. 50]. This is represented as:

$$w_l(u,v)$$

- Search Path Node Pair (SPNP) method:
  - This method "accounts for all connected vertex pairs along the paths through the arc $(\boldsymbol{u}, \boldsymbol{v}) \in \boldsymbol{R}$", [2, p. 51]. This is represented as:

$$w_p(u,v)$$

All these three methods not only help with assigning the weights but to determine the main paths to identify information pathways that are characterized by high traversal counts in search paths and low latency. As part of this project, we construct graphs with different weights to the vertices. Furthermore, the weights assigned are going to be different for publications and journals based on the proposed algorithms which we shall discuss as part of the chapter 5.

While dealing with Citation Networks of various publications there are several baseline properties that are same across the citation networks which are:

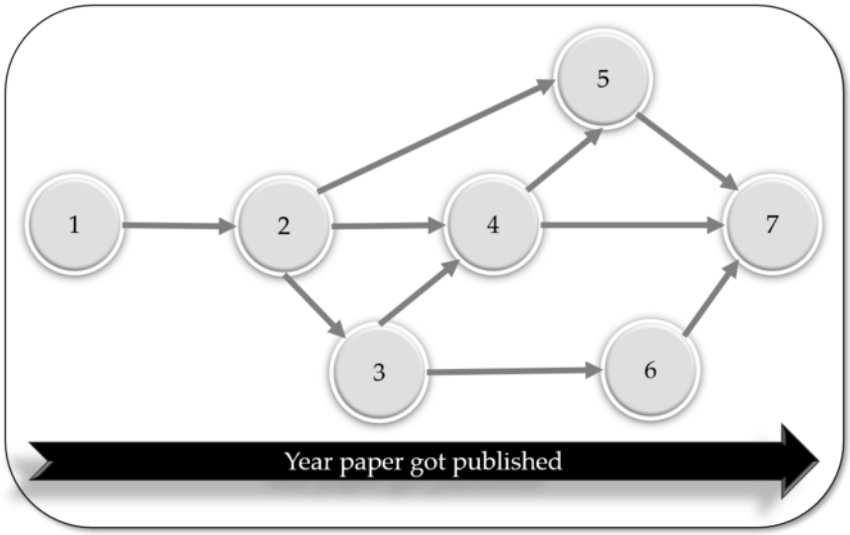

FIGURE 3.1: Citation Network Sample

- Citation Networks are acyclic because every graph can access the graph that is published at that time.

- Secondly, citation networks are always directed graphs. The edges always go from one publication to the next as shown above.
- Thirdly, all edges in the network point backward in time.
- The existing citation network never changes but the future citations keep getting added to the existing ones. Thus, the vertices and edges are permanent.
- Furthermore, as more papers get published for the journals, the weights between the journals keep on increasing over time.

In the sections following this, we shall discuss the system design and specifications.

3.3 System Design

In this section, we shall decide on the blue print of the system design and the various components involved in it. The main features and significant functionalities are defined here which gives an overview of our system. At a very high level the system-design consists of three crucial components including the input data stage, main system components stage and output results stage as shown in Figure 3.2.

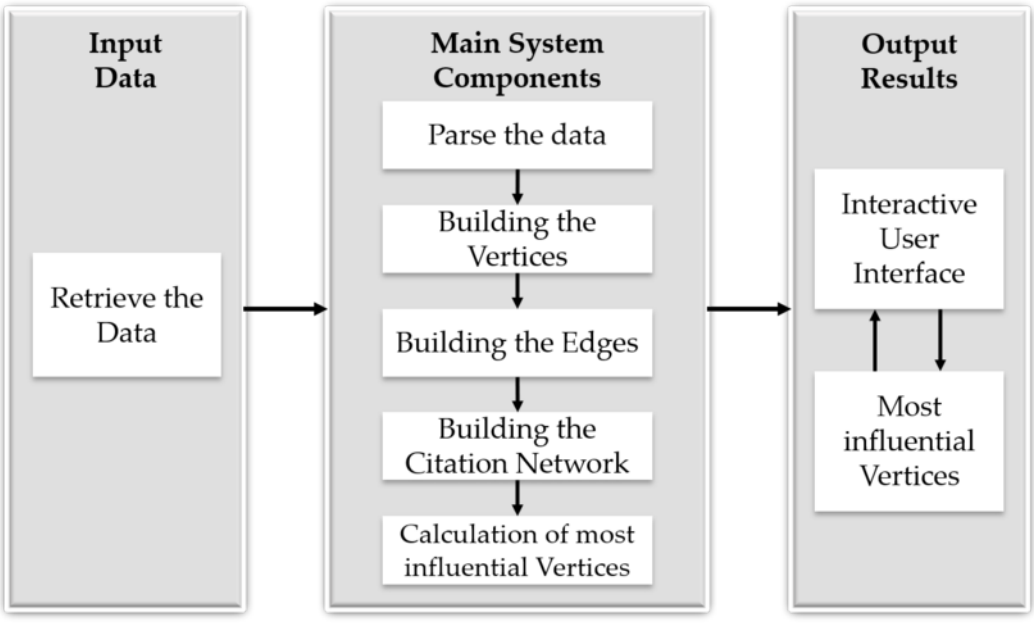

FIGURE 3.2: High Level System Design

The various components of this system include:

- **"Input Data"** - This component includes reading and retrieving the data from the ICHEC's system.

- **"Main System Components"** - This is the second component in our architecture that processes the input data, the entire business logic lies here. There are several sub-components including parsing of the data followed by the proposed method of "graphX" implementation by building the vertices, edges depending on when the user wants to search with a journal name or publication. Finally, we build a citation network graph of either journal or publication. The final sub-component involves the logic to find the most influential vertices.
- **"Output Results"** - This is the last component of the system which deals with an interactive user interface with the ability to take input and retrieve the most influential journals or publications. This simplifies the search process for researchers & students and improves the cognitivism of related publications and journals.

After understanding the high-level outline of the system design, it is now essential to answer the question "How we design this system?" which comprises of the "Software technological decisions" made for the project.

3.4 Big Data Technologies

To start with, the software technological decisions are made keeping in mind that we are dealing with "Big Data" here. Big data refers to the large, diverse sets of information that grow at ever-increasing rates. It encompasses the volume of information, the velocity or speed at which it is created and collected, and the variety or scope of the data points being covered. *Big data* often comes from multiple sources and arrives in multiple formats[1]. *Hadoop* is one of the tools designed to handle big data.

3.4.1 Hadoop Systems

Hadoop is an open-source framework that allows us to store and process big data in a distributed environment across clusters of computers using the MapReduce paradigm. According to the Apache Hadoop website, Yahoo! Has more than 100,000 CPUs in over 40,000 servers running Hadoop, with its biggest Hadoop cluster running 4,500 nodes. Yahoo! Stores 455 petabytes of data in Hadoop.

The core components of the Hadoop Ecosystem are[2]:

- **"Hadoop Common"** - It refers to the collection of common utilities and libraries that support other Hadoop modules. Hadoop Common is the core of the framework as it provides the necessary Java Archive (JAR) files and scripts required to start Hadoop. Furthermore, it provides the source code and documentation, as well as a contribution section that includes different projects from the Hadoop Community.

[1] Segal, T. *The Deal With Big Data* Investopedia. https://www.investopedia.com/terms/b/big-data.asp (2020).

[2] *What is Hadoop?* Bernard Marr. https://www.bernardmarr.com/default.asp?contentID=1080 (2020).

- **"Hadoop Distributed File System (HDFS)"** - The most important two are the Distributed File System, which allows data to be stored in an easily accessible format, across a large number of linked storage devices, and the MapReduce - which provides the basic tools for poking around in the data. Hadoop system uses its file system which sits above the file system of the host computer - meaning it can be accessed using any computer running any supported OS. The default map-reduce operation can be shown as in the Figure 3.3.
- **"MapReduce"** – It is named after the two basic operations this module carries out - reading data from the database, putting it into a format suitable for analysis (map), and performing mathematical operations i.e counting the number of males aged 30+ in a customer database (reduce).
- **"YARN"** - The final module is YARN, which manages resources of the systems storing the data and running the analysis.

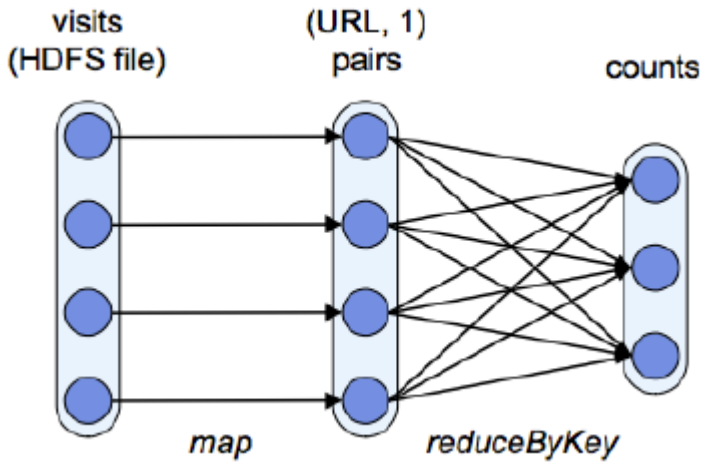

FIGURE 3.3: Lineage graph for the RDDs. The oblong ovals represent RDDs, while circles show partitions within a dataset.

The Data Access components of Hadoop Ecosystem are[3]:

- **"Pig"** - It is a platform that is used to analyse large dataset which consists of a high-level language to express data analysis programs, along with the infrastructure to evaluate these programs. One of the most significant features of Pig is that its structure is responsive to significant parallelization.
- **"Hive"** - On the other hand, Apache Hive is a data warehouse system that is often used with an open-source analytics platform like Hadoop.

The Data Integration Components of Hadoop Ecosystem are:

- **"Sqoop"** - Apache Sqoop "SQL to Hadoop" is a Java-based, console-mode application designed for transferring bulk data between Apache Hadoop and non-Hadoop datastores, such as relational databases, NoSQL databases, and data warehouses.

[3] *Techopedia – IT Dictionary for Computer Terms and Tech Definitions* https://www.techopedia.com/dictionary (2020).

- **"Flume"** - Flume is a distributed, reliable, and available service for efficiently collecting, aggregating, and moving large amounts of log data. It has a simple and flexible architecture based on streaming data flows. It is robust and fault-tolerant with tuneable reliability mechanisms and many failover and recovery mechanisms. It uses a simple extensible data model that allows for online analytic application[4].

The Data Storage Components of Hadoop Ecosystem are[5]:

- **"Hbase"** - Apache HBase is a specific kind of database tool written in Java and used with elements of the Apache software foundation's Hadoop suite of big data analysis tools. Apache HBase is an open-source product, like other elements of Apache Hadoop. It represents one of several database tools for the input and output of large data sets that are crunched by Hadoop and its various utilities and resources.

The Monitoring, Management, and Orchestration Components of Hadoop Ecosystem are[6]:

- **"Oozie"** - Apache Oozie is a Java web application that is used with Apache Hadoop systems. In these big data systems, Apache Oozie is a kind of job handling tool that works in the general Hadoop environment with other individual tools like YARN as well as MapReduce and Pig.
- **"Zookeeper"** - Apache ZooKeeper is an open-source Apache project that is meant to allow clusters to distribute information such as configuration, naming, and group services over large clusters. The project was originally a sub-project of Apache Hadoop but was made a major project in its own right managed by the Apache Software Foundation.

Even though Hadoop has many advantages there are few shortcomings of it as well. One of which is the MapReduce component which has a reputation for being slow for real-time data analysis. Enter Apache Spark, a Hadoop-based processing engine designed for both and streaming workloads and outfitted with features that exemplify what kinds of work Hadoop is being pushed to include[7].

3.4.2 Apache Spark

Spark is an open-source, distributed computing engine used for processing and analyzing a large amount of data that runs on top of existing Hadoop clusters to provide enhanced and additional functionality. It works with the system to distribute data

[4] *Welcome to Apache Flume — Apache Flume* https://flume.apache.org/ (2020).

[5] *Techopedia – IT Dictionary for Computer Terms and Tech Definitions* https://www.techopedia.com/dictionary (2020).

[6] *Techopedia – IT Dictionary for Computer Terms and Tech Definitions* https://www.techopedia.com/dictionary (2020).

[7] *Apache Spark with Hadoop – Why it Matters? | Edureka.co* Edureka. Section: Big Data. https://www.edureka.co/blog/apache-spark-with-hadoop-why-it-matters/ (2020).

across the cluster and process the data in parallel. In addition, Spark is 100 times faster than Hadoop MapReduce in memory or 10 times faster on disk. The Figure 3.4[8] shows how Spark and Hadoop can work together.

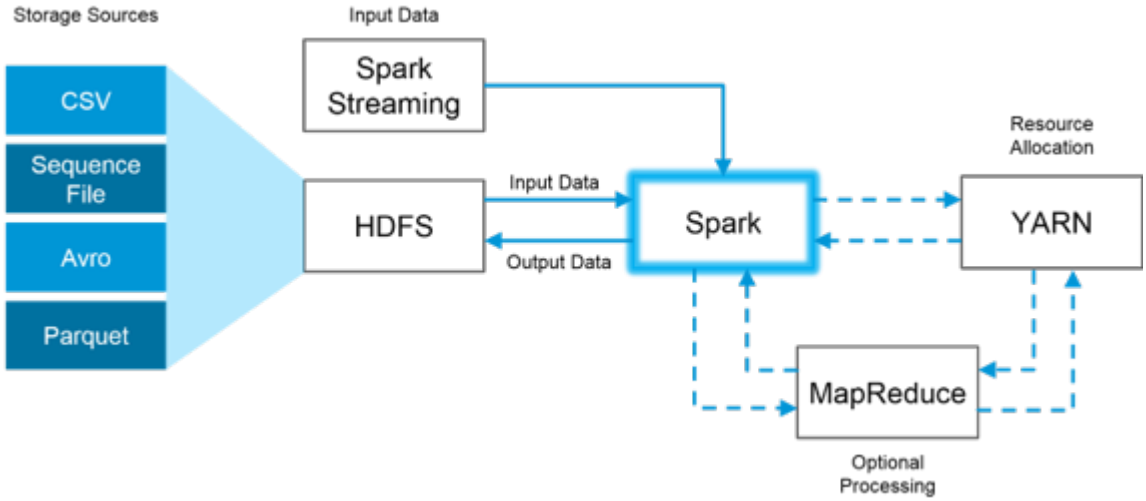

FIGURE 3.4: Spark Components.

Before we understand the Apache Spark in detail, we need to know what its basic entities are and how it is different from map-reduce. In a map-reduce system, a master process exists which coordinates with a set of workers. Several partitions are created and are assigned to the workers by the master process. After which each worker starts applying user-defined map functions onto the key-value pairs which then results in intermediate key-value pairs that are stored on local disks of workers. On the other hand, Apache Spark defines the map-reduce model in terms of operation over distribution collection objects called resilient distributed datasets (RDDs)[See 22] which are immutable collections of objects that are partitioned across different Spark nodes in the network.

There are several features of Spark which are:

- Spark creates a graph called a directed acyclic graph or DAG engine which finds the optimum path to solve the request or question.
- The most fundamental data structure of Apache Spark is Resilient Distributed Dataset(RDD).
- The advantage of using RDDs over other alternatives are:
  - In-Memory Computation – stores results in RAM.
  - Lazy Evaluation – transformations are applied only when called.
  - Fault Tolerance – any lost partition of an RDD can be rolled back by applying the simple transformations onto the lost partition of the lineage.
  - Immutability – only read-only access, to modify you can only apply a transformation to the existing RDD.

[8]Kovachev, D. *A Beginner's Guide to Apache Spark* Medium. https://towardsdatascience.com/a-beginners-guide-to-apache-spark-ff301cb4cd92 (2020).

- Partitioning – the fundamental unit of parallelism in Spark RDD.
- Persistence – users can apply certain transformations on an RDD and save its state for reuse by storing them in-memory or on Disk.
- Coarse-Grained Operation – is applied to ALL elements in datasets through maps or filter or group by operations.
- Location Stickiness – partitions are placed in such a way that task is close to data to speed up computation.

An RDD can be created using any of the three methods:

- Parallelized collections
- From External Storage – like HDFS, HBase, Hive, etc.
- From existing RDDs - using persistence

For example, given an RDD of (visitID, URL) pairs for visits to a website, we might compute an RDD of (URL, count) pairs by applying a map operator to turn each event into an (URL, 1) pair, and then a reduce to add the counts by URL[See 23, p2]. We make use of the "GraphX" framework in our case.

As already noted in section 2.10, "GraphX" is a new component in Spark for graphs and graph-parallel computation. At a high level, "GraphX" extends the Spark RDD by introducing a new Graph abstraction: a directed multi-graph with properties attached to each vertex and edge. To support graph computation, "GraphX" exposes a set of fundamental operations(eg., *subgraph*, *joinVertices*, and *aggregateMessages*) as well as an optimized variant of the Pregel API. Besides, "GraphX" includes a growing collection of graph algorithms and builders to simplify graph analytic tasks[9]. We shall discuss in detail of the construction and implementation of the graphs using "GraphX" in chapter 5 and will get to the system specifications in the following section.

3.5 System specifications

The most difficult problem that needs to be solved to handle big data effectively is storage, it is not easy to deal with large quantities and varieties of data[See 25]. This forms the beginning of our hardware architectural decisions.

For our project, we make use of the "ICHEC"(Irish Centre for High-End Computing) systems. ICHEC delivers complex compute solutions to Irish Higher Education Institutions (HEIs), Enterprises, and the Public Sector on behalf of the State. ICHEC's National HPC Service (NS)[10] offers academic researchers access to HPC computing resources. ICHEC initially used the *Fionn* supercomputers as their primary computing devices. However, they were soon replaced by *Kay*.

[9] *GraphX - Spark 3.0.0 Documentation* https://spark.apache.org/docs/latest/graphx-programming-guide.html.

[10] *National HPC Service* ICHEC. Library Catalog: www.ichec.ie. https://www.ichec.ie/academic/national-hpc.

Kay(kay.ichec.ie) is ICHEC's primary supercomputer and Ireland's national supercomputer for academic researchers[11]. This system comprises of several components as below:

- **"Cluster"** - A cluster of 336 nodes where each node has 2x 20-core 2.4 GHz Intel Xeon Gold 6148 (Skylake) processors, 192 GiB of RAM, a 400 GiB local SSD for scratch space and a 100Gbit OmniPath network adaptor. This partition has a total of 13,440 cores and 63 TiB of distributed memory.
- **"GPU"** - A partition of 16 nodes with the same specification as above, plus 2x NVIDIA Tesla V100 16GB PCIe (Volta architecture) GPUs on each node. Each GPU has 5,120 CUDA cores and 640 Tensor Cores.
- **"Phi"** - A partition of 16 nodes, each containing 1x self-hosted Intel Xeon Phi Processor 7210 (Knights Landing or KNL architecture) with 64 cores @ 1.3 GHz, 192 GiB RAM and a 400 GiB local SSD for scratch space.
- **"High Memory"** - A set of 6 nodes each containing 1.5 TiB of RAM, 2x 20-core 2.4 GHz Intel Xeon Gold 6148 (Skylake) processors, and 1 TiB of dedicated local SSD for scratch storage.
- **"Service & Storage"** - A set of service and administrative nodes to provide user login, batch scheduling, management, networking, etc. Storage is provided via Lustre file systems on a high-performance DDN SFA14k system with 1 PiB of capacity.

The machine is also capable of running heterogeneous work-flows that require large computing power and large amounts of memory either during the pre- or post-processing phases of researcher's work.

The Kay systems are highly secured and we connect to these using the SSH Authentication with highly secure "4096" bit secure SSH key. Initially, a public key and a private key are generated and the public hey is configured on this and we use the private key to connect to the system along with a secure password.

3.6 System Architecture

Even though the computational power of the system is quite incredible there is no Resource Manager or Cluster Manager setup on the ICHEC machine. With the absence of the "YARN" cluster setup, we need to come up with a solution to build the clusters and an application master that triggers the program. In addition to running on the Mesos or YARN cluster managers, Spark also provides a simple standalone deploy mode. We chose this mechanism to build the architecture of our system. There are several ways we can launch a standalone cluster which includes, either manually, or by starting a master and workers by hand or use the launch scripts provided. It is possible to run these daemons on a single compute node or many.

[11] *Kay* ICHEC. Library Catalog: www.ichec.ie. https://www.ichec.ie/about/infrastructure/kay.

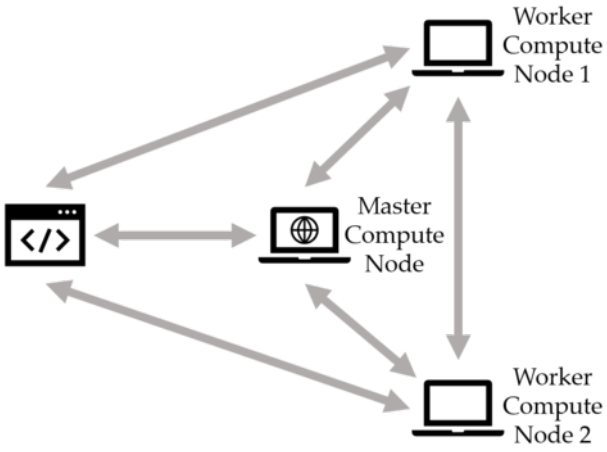

FIGURE 3.5: Spark Standalone Cluster

If we chose a single node, the resources of the node are divided to build the masters and executors of the clusters. On the other hand, if we use multiple nodes, the combined resources of all these nodes are used to build the masters and executors. We chose the second approach to build our cluster. More specifically, we chose 3 *ProdQ* nodes with each containing 192 GB RAM and 40 cores. The setup of our system looks as shown in Figure 3.5 with a master computer node and two other worker compute nodes together forming our 3-node cluster. We make use of a third node from which we trigger the driver program onto the master node. Here we are talking about just the physical nodes. However, in a logical sense, we create many other executor instances. The Spark Configuration that we are using to build the clusters can be shown in Table 3.1. We have also built two separate shell script files one to achieve this task of cluster creation which is specified in Lst. A.8 and the second to submit a batch job to trigger spark-submit onto the master node which is specified in Lst. A.9. In this way, we build our cluster and now we move forward to understand the dataset that we are going to use and its sources.

Entity	Description
Uncited Physical Nodes	3
Physical Cores per Node	40
RAM per Node in GB	192
Total Cores	110
Cores per Executor	5
GB per Core	5
Executor Instances	22
GB per Executor	26

TABLE 3.1: Spark Configuration

Chapter 4

Methodology

In this chapter, we describe the dataset being used and their sources along with the software and programming languages used in the project. The data description involves basic exploratory data analysis. Exploratory Data Analysis(EDA) is an approach to analysing datasets to summarize their main characteristics, often with visual methods. EDA is used for seeing what the data can tell us before the modelling task. It is not easy to look at a column of numbers or a whole spreadsheet and determine important characteristics of the data. It may be tedious, boring, or overwhelming to derive insights by looking at plain numbers. Exploratory data analysis techniques have been devised as an aid in this situation.

4.1 Data Description

The data of research publication listings were gathered from the Semantic Scholar Open Research Corpus(S2 ORC) which contains records for research papers published in all fields provided as an easy-to-use JSON archive[28].

The papers are provided as JSON objects, one per line. Archives are partitioned in batches and shared as a collection of gzipped files. We have downloaded the manifest via http, and used it to download all archive files via http.

```
1 wget https://s3-us-west-2.amazonaws.com/ai2-s2-research-
    public/open-corpus/2020-05-27/manifest.txt
2 wget -B https://s3-us-west-2.amazonaws.com/ai2-s2-research-
    public/open-corpus/2020-05-27/ -i manifest.txt
```

LISTING 4.1: Data Retrieval

Each of these papers contains several attributes that help us identify it. The structural entities of the publications are shown in Table 4.1.

Few of the most important attributes of the publication are

- id
- title
- outCitations

Attribute	Description
entities	Extracted list of relevant entities or topics
journalVolume	The volume of the journal where this paper was published.
journalPages	The pages of the journal where this paper was published.
pmid	A Unique identifier used by PubMed.
year	Year this paper was published as an integer.
outCitations	List of S2 paper IDs which this paper cited.
s2Url	URL to S2 research paper details page.
s2PdfUrl	URL to S2 research paper pdf link
id	S2 generated research paper ID
authors	List of authors with an S2 generated author ID and name.
journalName	Name of the journal that published this paper.
paperAbstract	Extracted abstract of the paper
inCitations	List of S2 paper IDs that cited this paper.
pdfUrls	URLs related to this PDF scraped from the web.
title	Research paper title.
doi	Digital Object Identifier registered at doi.org.
sources	Identifies papers sources
doiUrl	DOI link for registered objects.
venue	Extracted publication venue for this paper.

TABLE 4.1: Publication Attributes

- inCitations
- journalName

These attributes are our main points of interest because they define the connectedness of the various publications. *outCitations* tell us how many other publications this paper is citing to and *inCitations* tell us how many other publications are citing to this paper. Thus, these help us with the analysis of the data. An *ID* attribute is a unique identifier for a publication. The *title* attribute defines the name of the publication while *journalName* attribute defines the name of the journal.

We shall understand the dataset by analysing the below:

- Temporal information - which can be understood by looking at the trend across the years.
- Journal importance - which can be understood by looking at:
  - Publication count - the number of publications per journal
  - Citation count - the number of citations per journal

As an initial step, we plot the publications based on the year they got published in. The *year* attribute helps us with this aspect. Figure 4.1 shows the trend of these publications count per year. In this figure, we notice that the highest number of papers got published in the year 2016 from which we see a decreasing trend till 2000 in a consistent manner. By looking at this plot we can say that there is consistency in the growth of the publications per year. We do not see any sudden increase or decrease. However, from the year 2014 till 2016 we do not see a huge difference and are almost the same. Based on this data we can predict the percentage of growth that can be expected for the coming years.

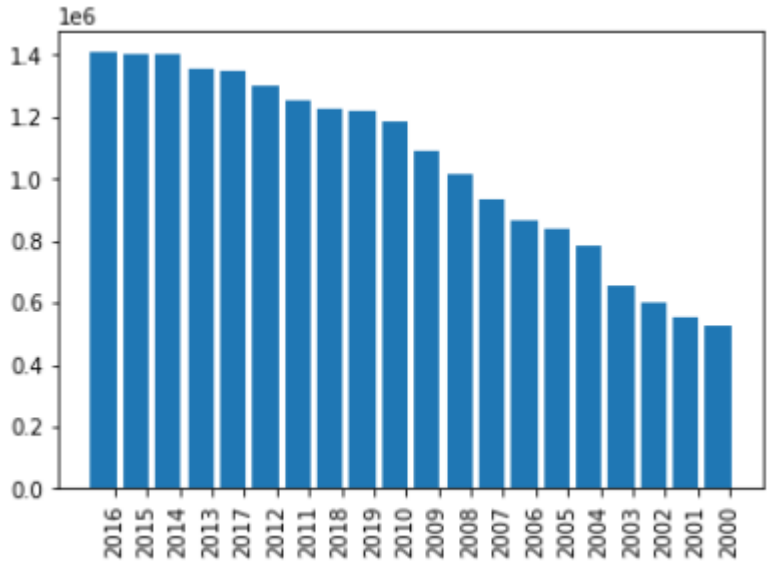

FIGURE 4.1: Plot for Top publications per year

The total number of publications in the entire dataset is "30852649". However, out of this total number "23802544" publications are uncategorised into any journal. Thus, the percentage of publications with journal name is "22.85%". We now perform the exploratory data analysis to find the top 20 journals with the most publications. These can be depicted in Table 4.2. Based on the statistics of this table we can see that the journal with the highest number of publications is for "Nature" in which we see "67369" publications. On the other hand, the second-highest number of publications is recorded for the journal "PLoS ONE" in which we see "31041" publications. By comparison, we can say that the publications in the top journal are more than twice the number of publications for the second top journal. Following these, the count follows a trend from "28748" till "11107".

The statistics in this table can be depicted in a bar plot between the top journals based on the publications and the count of publications as shown in Figure 4.2. This plot gives a much better picture of the information in the previous table because this figure shows that the journal "Nature" is different from the other. After all, while the others follow a gradually increasing trend this has a sudden increase in the publications.

After we understand how publications are spread across it is necessary to verify if the journal with the highest publications is indeed the most important or not. This can be identified by exploring the citations of the publications. There are two

Journal	No. of Publications
Uncategorised journals	23802544
Nature	67369
PLoS ONE	31041
Science	28748
The Journal of biological chemistry	27826
ArXiv	27522
Advanced Materials Research	23451
Proceedings of the National Academy of Sciences of the United States of America	23245
The Lancet	21022
Applied Mechanics and Materials	18546
British medical journal	18264
Biochimica et biophysica acta	16327
Biochemical and biophysical research communications	14772
Physical review letters	14593
The New England journal of medicine	12977
BMJ	12894
JAMA	12540
bioRxiv	12495
Scientific Reports	11421
Nursing standard(Royal College of Nursing(Great Britain):1987)	11107

TABLE 4.2: Top 20 journals with the highest publications

types of citations as discussed earlier, the inCitations and outCitations. Here we take into consideration of both. Even though a particular journal has the most publications it doesn't need to have the highest citation count because it only depends on how the research of the papers was carried out and what papers they acknowledged. We can see the statistics of the top journals based on citation counts in Table 4.3.

When we look at the citation plot it can be visualized as Table 4.3. The journal "PLoS ONE" has the highest number of citations with a count of "1288850". Following this, the second-highest number of citations for the journal "ArXiv" are recorded with a count of "668912". From the plot, we can say that the citations are high for the top four and the rest of the journals are comparatively low, and the trend remains the same. Following this, we shall understand the soft wares and programming languages used in the next section.

4.2 Software and programming languages used

The various software and programming languages used in the project are as below:

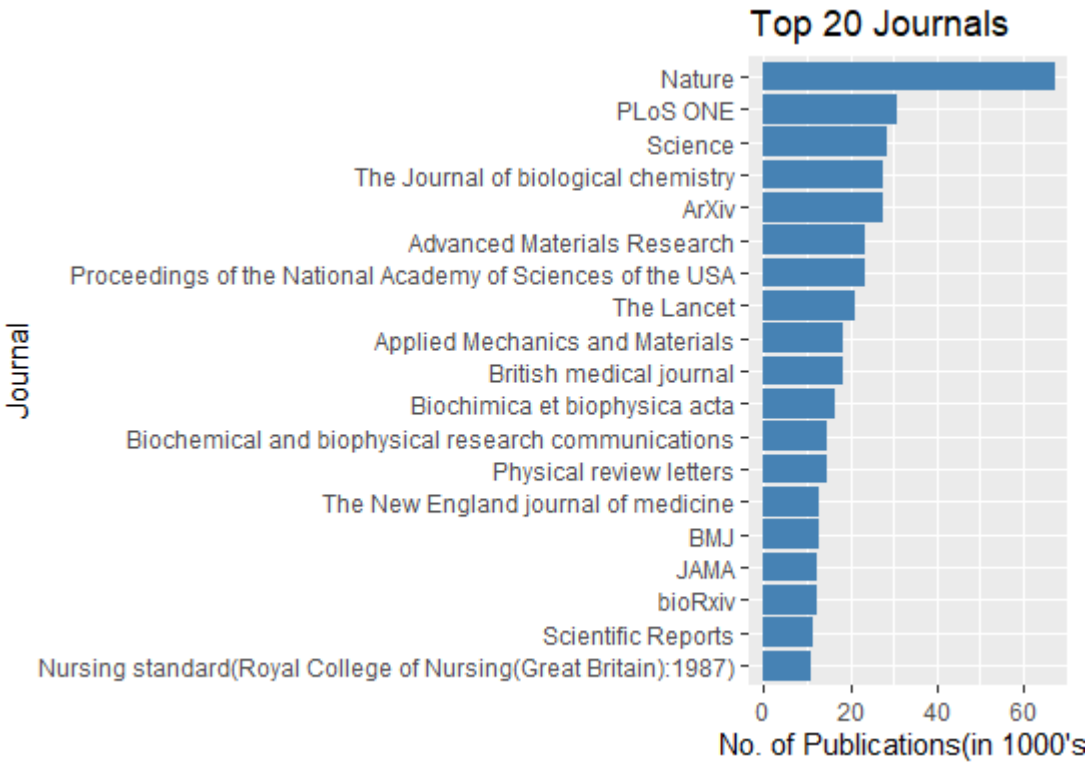

FIGURE 4.2: Top 20 journals with the highest publications

- **"Scala"** - "Scala", also referred to as the Scalable Language is a pure object-oriented high-level programming language. We have used Scala version "2.11.12" for our project along with spark version "2.4.3". Below are few of the most prominent features of Scala:
  - One of the key features of Scala is its functional programming methodology of coding. This allows the user to write the code for a program in the least number of lines but with the best efficiency.
  - The next feature is its ability to compile and execute its code on Java Virtual Machine(JVM) thus making it platform independent just like java.
  - One another feature of Scala is it's able to work with big-data technologies like Hadoop and Spark.
- **"IntelliJ IDEA"** - This is a client-side software which is useful for the users for integrated development. An integrated development environment(IDE) is a tool that lets any developer build, compile, run, and execute the program at one place. This is written in java. In our project, we use this tool to achieve the client side environment configuration. We used IntelliJ IDEA version "11.0.7" for the project with java version "JavaSE8 update 251".
- **"SBT build tool"** - An SBT tool is the most used build tool for Spark. A build tool helps the developer by adding the required dependencies and thus leaving the trouble for worrying about the task of loading jars and their versions into the project. A few of the prominent features include continuous compilation, testing, deployment, incremental testing, and compilation.

Journal	No. of Citations
PLoS ONE	1288850
ArXiv	668912
bioRxiv	536213
Scientific Reports	485544
The Journal of neuroscience: the official journal of the Society for Neuroscience	256692
Proceedings of the National Academy of Sciences of the United States of America	253344
Blood	203387
Cancer research	179792
IEEE Access	171115
International Jounal of Molecular Sciences	160814
Circulation	143262
Journal of virology	135455
Environmental Science and Pollution Research	127488
NeuroImage	126195
ACS applied materials & Interfaces	120066
The Journal of Cell Biology	108528
Applied Microbiology and Biotechnology	108241
Nature Communications	107520
Biochimica et biophysica acta	103127

TABLE 4.3: Top 20 journals with the highest citations

- **"Linux"** - Linux is an open-source operating system which is quite similar to Unix. The infrastructure of "ICHEC" is hosted on Linux machine connected to the cluster of nodes with different computational power. Every compute node of this infrastructure is a Linux machine that forms several clusters.

- **"Bash Scripting"** - Bash or Bourne Again Shell is an interpreter that is capable of processing the shell commands. A shell commands are of fixed keywords that kernel can interpret and perform an action. For example, the "ls" command lists all the files and directories in the current working directory. When we combine a few numbers of sequential shell commands that perform a common action then it is called shell scripting. Bash, however, is an improved version of Bourne shell which can also be used to perform shell scripting. In this project we use bash scripting at multiple cases for spark cluster creation, submission of jar file with spark-submit using a bash script.

- **"MobaXterm"** - MobaXterm is a client toolbox that can be used for remote computing. It is quite powerful and provides multiple functions for the users to work based on whether they are programmers or IT administrators, etc. We use MobaXterm to connect to the Linux machines of the "ICHEC" infrastructure using SSH encryption.

- **"R Language"** - "R" is a language used for statistics and generate meaningful graphical plots out of the data. It provides a variety of techniques to evaluate

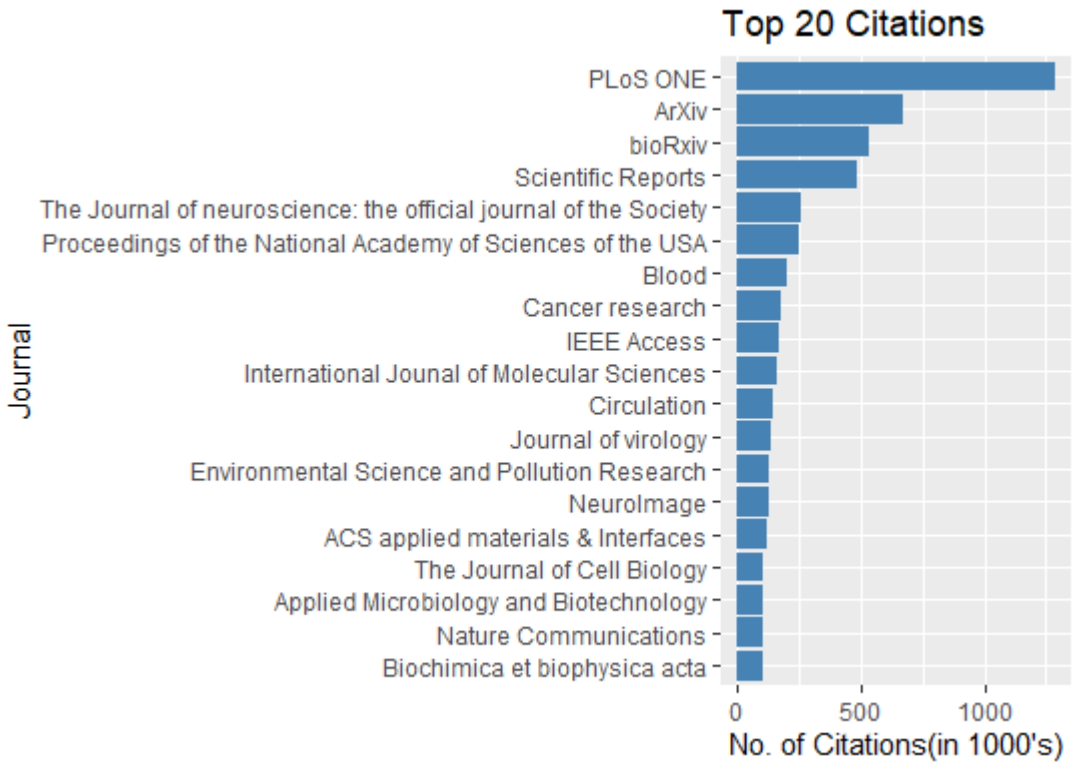

FIGURE 4.3: Top 20 journals with the highest citations

the data. Several plots generated from "R" are very useful to get a clear picture of the data structure and know about the outliers(unusual data) that helps the user in deciding the actions of applying a model for making future predictions. In our work, we use R for data exploration analysis.

- **"GitHub"** - Git is a version control tool and GitHub is the repository hosting service used for software development. A version control tool tracks the various changes in the code by versioning it. Each version of code is a stable version of the code that is saved in a distributed fashion. We say it as a distributed version control system as many users can access it and make changes simultaneously. This makes it a great tool to increase productivity and also helps the user with disaster recovery to get back to a stable version when required. In our project, we use this tool for the version controlling of our code.

The next section of this thesis deals with the implementation of the project.

Chapter 5

Implementation

In this chapter, we explain in detail the implementation process throughout each phase to achieve our objective. First, our objective in the implementation process is to extract the data from the JSON files by parsing it to get individual publication information. Following that, we focus on building a spark context to handle the big data. After this, we build a code to construct vertices and nodes from the publications. This step also includes the construction of vertices and nodes specific to journals if a journal graph is required. Once, this is done we build a citation network graph representation such that the nodes represent the publication while the links represent the citation/reference relations. For a journal citation network graph, the nodes represent the journals while the links represent the aggregated citation/reference relations. We then build a method to take input from the user and get the closest nodes to the searched journal/publication and process the most influential journals or publications and give back to the user.

5.1 Reading the Data

We have chosen Semantic Scholar to retrieve the publication information in the form of *JSON* files. This data is currently stored in the ICHEC system from which we will be reading and processing the data to achieve our objectives of building a citation network. The next step includes parsing the publications. We shall discuss the data exploration in section 4.1. The retrieved *JSON* format is in a hierarchical tree structure as shown in Lst. 5.1.

```
{
  "entities":[
        "..."
        "..."
        ],
  "journalVolume":"...",
  "journalPages":"...",
  "pmid":"...",
  "year": ...,
  "outCitations":[
          ...
          ...
          ],
```

```
{
  "s2Url":"https://semanticscholar.org/...",
  "s2PdfUrl":"...",
  "id":"...",
  "authors":[
        {
        "name":"...",
        "ids":[
              "..."
              "..."
              ]
        },
        ...
        ],
  "journalName":"...",
  "paperAbstract":"...",
  "inCitations":[
          ...
          ...
          ],
  "pdfUrls":[...],
  "title":"...",
  "doi":"...",
  "sources":[...],
  "doiUrl":"https://doi.org/...",
  "venue":"..."
}
```

LISTING 5.1: JSON data format

This *JSON* structure gives details of an individual publication and its citation/reference relationship to other publications. The *inCitations* field talks about all the parent publications this publication cited from. This tells us that our publication has added research objectives and findings to these parent publications in the corresponding field or journal. The *outCitations* field, on the other hand gives us information about the child publications which took the research findings of these publications and added more on top of it. In this scenario, the *inCitations* are published before the current publication and the *outCitations* are published after the current publication. The *inCitations* are always fixed while the *outCitations* cannot change but can have more publications to be added to the current as more publications make use of the current research of the publication. The Figure 5.1 clearly shows these properties.

For instance, the publication "P5" makes use of the research findings of "P1","P2","P3","P4" as parent publications and develop its research. Similarly, the publication "P6", "P7", "P8" make use of the research findings of "P5". This entire information is stored as a single record in a *JSON* file. Since we are using *Apache Spark* to implement our project, we shall read the data in the form of an "RDD". Reading the data involves parsing the *JSON* data which we shall discuss in detail in section 5.2.

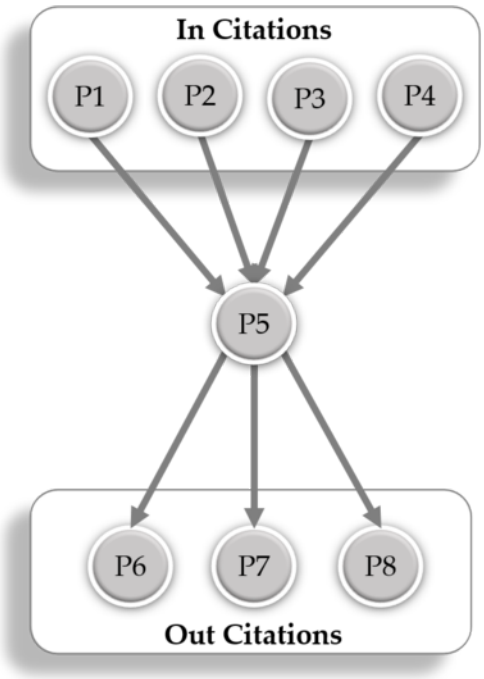

FIGURE 5.1: Citations of Publications

5.2 Parsing the Publications

To parse is to break up a sentence or group of words into separate components, including the definition of each part's function or form[1]. The pre-processing phase in this scenario is to parse the data from the *JSON* format into individual publications. Before we go to detail we shall first understand what a *JSON* is.

JSON(JavaScript Object Notation) is a lightweight data-interchange format that is easy for humans to read and write while at the same time easy for machines to parse and generate. JSON is a subset of Javascript programming language. It is in text format that is completely language independent but uses conventions that are familiar to programmers. These properties make JSON an ideal data-interchange language[2].

A *JSON* is built on two structures:

- A collection of name/value pairs.
- An ordered list of values.

Each file of the data retrieved from *Semantic Scholars* is in the raw *JSON* data format. Each *JSON* file contains the hierarchy tree structure required for identifying the citation relations between publications. To parse the publications from the JSON files, we use the Lst. 5.2.

```
1 for each file in the list of JSON files
```

[1] *What is Parse? - Definition from Techopedia* https://www.techopedia.com/definition/3853/parse.
[2] *JSON* https://www.json.org/json-en.html.

```
{
  Read the file into an RDD
  for each file in RDD{
    Read the file
    for each line in the file{
      Read the line as a JSON value
      Parse this JSON value into individual words
      Create a publication item from the parsed JSON words
      Add this publication to a publication RDD
    }
  }
}
```

LISTING 5.2: Reading the data

5.3 Proposed Algorithm to build a Graph

For the construction of the citation network, we use the Apache Spark cluster computing framework. The fundamental programming abstraction is *RDD (Resilient Distributed Datasets)*, which provides a logical view for the collection of data partitioned across multiple machines. RDDs can be also created by referencing datasets in external storage systems or applying a coarse-grained transformation (e.g. map, filter, reduce, join) on existing *RDDs*.

For graph computations, we utilize "GraphX" APIs provided by "Apache Spark". "GraphX" unifies "ETL", exploratory analysis, and iterative graph computation within a single system. we can view the same data as both graphs and collections, transform and join graphs with *RDDs* efficiently, and write custom iterative graph algorithms using the "Scala API".

To build a citation network, we need to convert the list of publications from *JSON* format to a connected directed graph G. It contains a number of vertices, V representing the publications and number of edges, E representing the citation/reference relations between the publications such that:

$G = (V, E)$; V : set of Vertices, E : set of Edges.

In our scenario,

$V = v_1, v_2, ...v_k$ are the set of publications or journals. $E = e_1, e_2, ...e_k$ are the set of citations/references between publications or journals v_i and v_j.

There are three steps to build the citation network graph in "graphX" which involves:

- Building the Vertices
- Building the Edges
- Building the Graph

In our proposed solution, we shall construct two graphs - one for *Publications* and the second for *Journals*.

5.4 Construction of Publication Graph

The construction of the publication graph is pretty straight forward and involves the below steps:

- Building the publication vertices where each vertex represents an individual publication.
- Building the publication edges where each edge represents a citation between any two publications.
- Building the publication graphs.

The built publication graph looks as shown in Figure 5.2

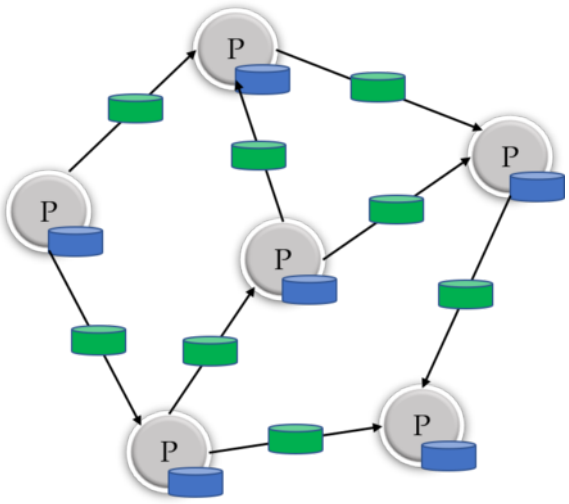

FIGURE 5.2: Sample Publication Graph

5.4.1 Algorithm to build the Publication Vertices

For every publication from the publications *RDD* generated from Lst. 5.2 we construct a publication vertex *RDD* with publication ID and publication Title. The complete algorithm to achieve this is defined in Lst. 5.3.

```
1
2 Read the publications RDD
3 for each publication in the list of publications RDD
4 {
5   Read the publication
6   Extract the id of the publication
```

```
7   Extract the publication title of the publication
8   Create a publication vertex with these two values, id and
     publication title as properties
9   Add or update this publication vertex to an RDD of
     publication vertices.
10 }
```

LISTING 5.3: Building the Publication Vertices

This algorithm can be depicted using the Figure 5.3.

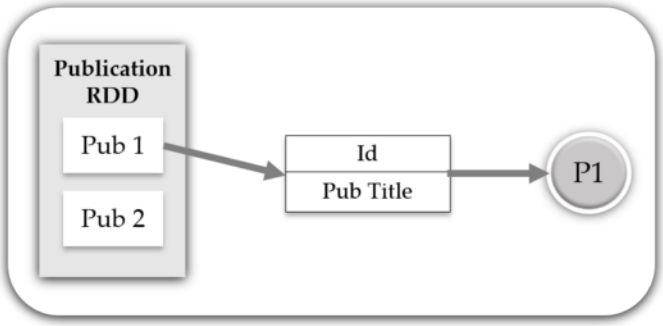

FIGURE 5.3: Building the Publication Vertices

5.4.2 Algorithm to build the Publication Edges

For building the citation edges of the publications we need to parse the out citations for each publication and construct a scala tuple (Long, Long, int) representing the publication id of source publication along with publication id of the destination publication and finally the weight of the citation link. This form of citation link is construction because we are building a directed graph of citations and "GraphX" API would require it in this format.

To build the Citation edges of the publication we use Lst. 5.4 by making use of the publications *RDD* generated from Lst. 5.2.

```
1
2 Read the publications RDD
3 for each publication in the list of publications RDD
4 {
5   Read the publication
6   Extract the id of the publication
7   Extract the list of out citations for each publication
8   Create a new RDD publicationCitations, with id
     representing each publication along with the list of out
     citations which contains all the child publications that
     are using this publication.
9   for each publicationCitation in the list of
     publicationCitations RDD
10  {
11    Read a single publication with all its citations
```

```
12      for each citation from the list of out citations
13      {
14        Create a publication citation edge with the id of the
      publication and this specific citation
15        Add this citation edge created in the above to an RDD
      of citation edges.
16      }
17    }
18  }
```

LISTING 5.4: Building the Publication Edges

The algorithm can be depicted in the Figure 5.4.

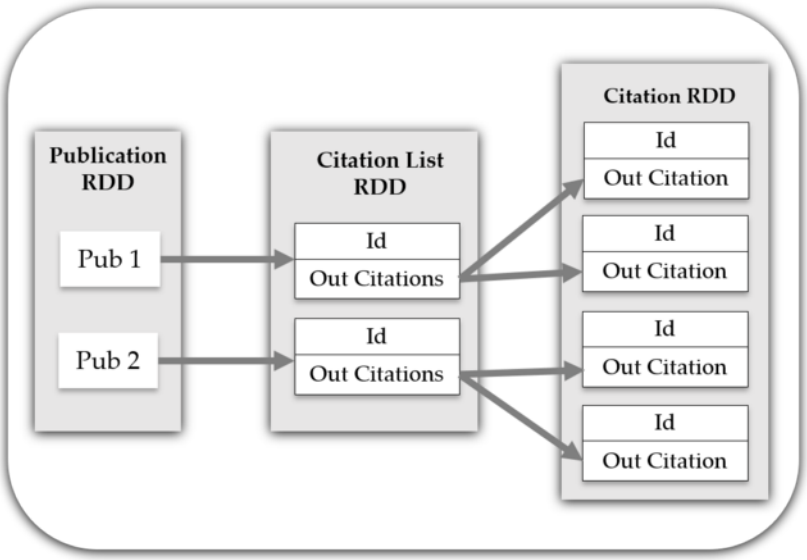

FIGURE 5.4: Building the Publication Citation Edges

As a final step, we can use the publication vertices and edges created in the above steps to create a graph triplet object in "graphX" which can later be used to perform graph operations and features extraction. The creation of a graph can be depicted using Figure 5.5[3].

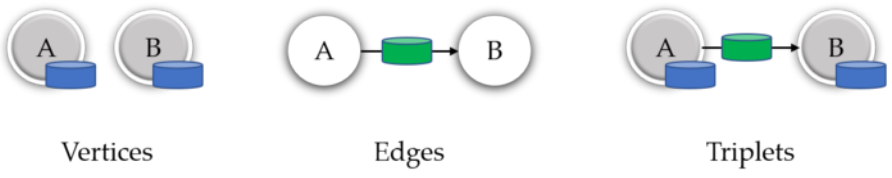

FIGURE 5.5: Building the Graph

[3] *GraphX - Spark 3.0.0 Documentation* https://spark.apache.org/docs/latest/graphx-programming-guide.html.

5.5 Construction of the Journal Graph

The construction of the journal graph involves various steps and is a bit different than creating the publication graph. This is due to the fact that the data is structured in such a way that retrieval of the journal citations is not a one-step process like in the case of publications. The high-level process of journal graph creation involves the below steps:

- Building the journal vertices where each vertex represents an individual journal which comprises of a list of publications with the same journal name.
- Building the journal edges where each edge represents a citation between any two journals.
- Building the journal graph.

5.5.1 Algorithm to build the Journal Vertices

For every publication from the publications *RDD* generated from Lst. 5.2 we construct a journal vertex *RDD* by aggregating and extracting the journal ID and journal name. The complete algorithm to achieve this is defined in Lst. 5.5.

```
1
2 Read the publications RDD
3
4 Group the publications together based on journal name
5
6 for each publication list from the group of publication list
7 {
8   Read the publication list
9   Generate an index for the publication list, this index
     represents the journal ID corresponding to the journal
     name this publication list is part of
10  Extract the journal ID(the index) along with the journal
     which contains a publication list
11  Extract the journal ID along with the journal name
12  Create a journal vertex with these two values, id and
     journal name as properties
13  Add or update this journal vertex to an RDD of journal
     vertices.
14 }
```

LISTING 5.5: Building the Journal Vertices

The above algorithm can be depicted using Figure 5.6.

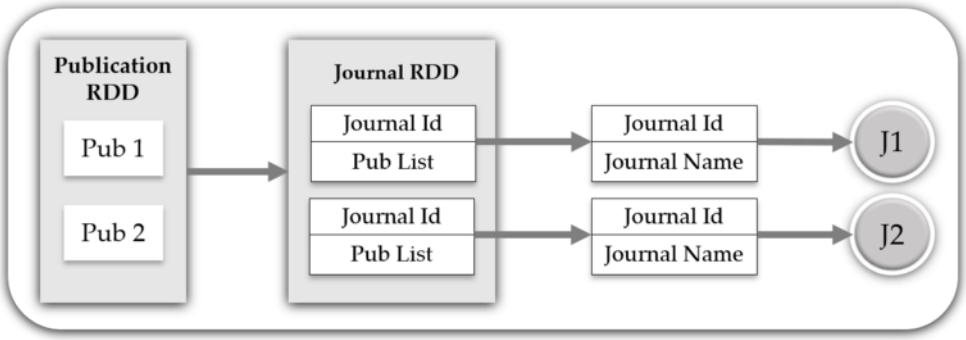

FIGURE 5.6: Building the Journal Vertices

5.5.2 Algorithm to build the Journal Edges

To build the citation edges of the journals, we need to group the publications first and then parse the out citations for each publication and construct a scala tuple(Long, Long, int) representing the journal id of the source journal along with the destination journal and finally combined with the weight of the citation link which depends on the total number of out citations and in citations between the publication of the two journals.

To understand this better we can look at Figure 5.7 which shows how the publications are grouped into journals and how the publications are interconnected. If we consider any two journals we see multiple out citations and multiple in citations between them each carrying a weight of 1. However, what we are trying to achieve a consolidated citation with the weight of each citation representing the total weight of out or in citations between any two journals.

To build these journal citations, we do not consider any citations that are connected to publications from the same journals. Hence, breaking down this scenario into considering the critical publications that connect to other publications from different journals can be depicted in Figure 5.8.

Finally, what we are trying to extract is just the citations with the weights representing the aggregate weights of citations between journals. Henceforth, this can be depicted with Figure 5.9. We now proceed further at looking at the algorithms of achieving this.

The first step to creating journal citation edges is to create a dictionary of journal publications containing publication id and journal id. This dictionary can be used to retrieve the journal when we query with a publication. This dictionary can be built using the following algorithm.

```
1
2 Read the journals RDD containing journal index and journal
     containing a list of publications
3
4 for each journal from the journals RDD
5 {
6   Read the journal index and journal
```

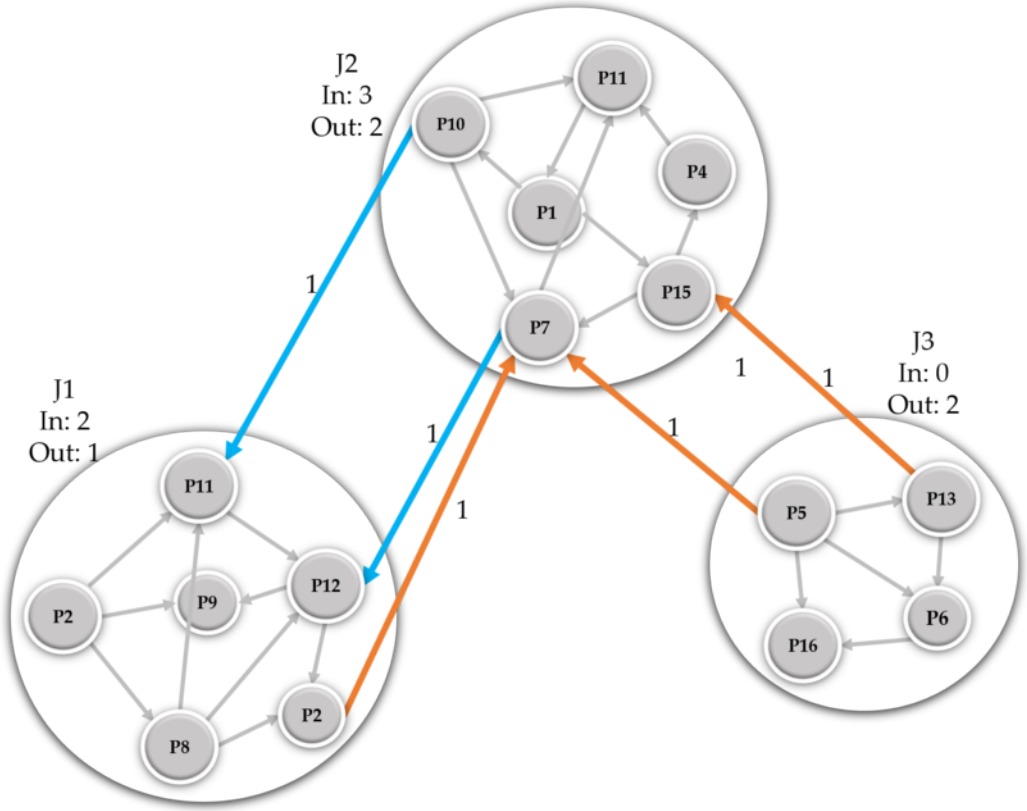

FIGURE 5.7: Proposed solution to build Journal Citations

```
7    Extract the publications list of the read journal
8    for each publication from the publications list
9    {
10     Extract the publication ID of this publication
11     Add or update this publication ID along with the journal
       index to an RDD representing the journal publication
     dictionary.
12   }
13 }
```

LISTING 5.6: Building the Journal Publications dictionary

This algorithm can be depicted in Figure 5.10.

To build the citation edges of the journal we use Lst. 5.8 by making use of the journals *RDD* generated in Lst. 5.5.

```
1
2 Read the journals RDD containing journal index and journal
     containing a list of publications
3
4 for each journal from the journals RDD
5 {
6    Read the journal index and journal
7    Extract the publications list of the read journal
```

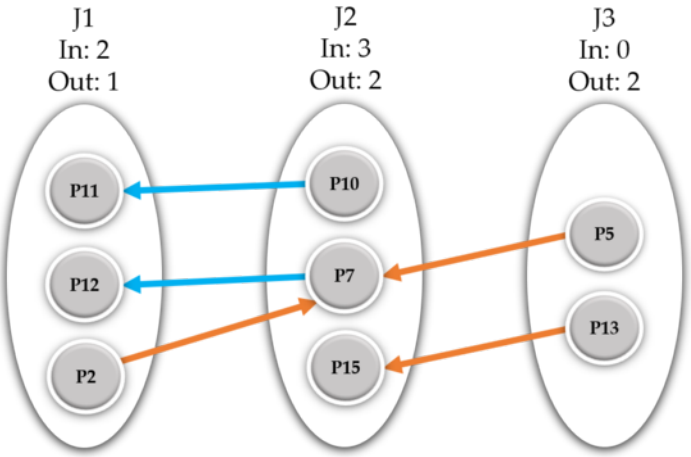

FIGURE 5.8: Aggregated Journal Citations

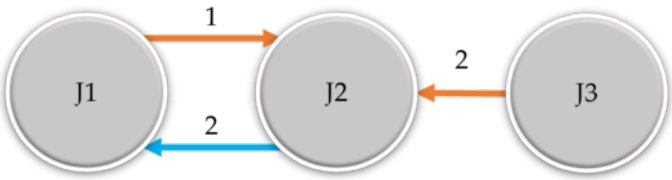

FIGURE 5.9: Aggregated Journal Citations

```
8   for each publication from the publications list
9   {
10    Extract the out citations list of this publication
11    for each out citation from the citation list
12    {
13      Read the out citation which is an ID representing any
     publication this item is citing to
14      Extract the journal ID this out citation ID is part of
     using the Journal Publication dictionary created in \
     autoref{lst:BuldgJrnlPblctnDctnry}
15      Extract the already read journal index and combine it
     with the journal id created for the out citation to
     create a citation edge with an additional parameter
     representing the weight/strength of 1
16      Add or update the journal edge created above to an
     intermediate RDD called test edges. (The test edges
     include all the individual citations between publications
      of one journal to publications of another journal. Thus,
      we can see multiple out citations and in citations
     between a pair of journals. However, since we need the
     aggregated citations we see it in the below steps)
17    }
18  }
19 }
```

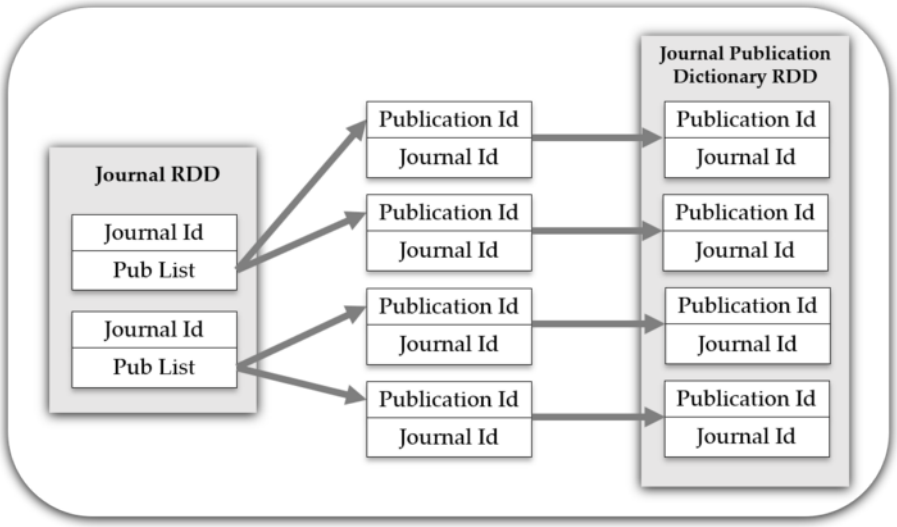

FIGURE 5.10: Journal publications dictionary

LISTING 5.7: Building the Journal Edges

```
1
2 Read the test edges RDD created above and remove all the
    invalid values
3
4 for each citation from the test edges RDD
5 {
6   Remove all the citations inside the same journal because
     we do not need them to find the citations inter-
     connecting journals
7   create or update the results of the test edges RDD
8 }
9
10 for each citation from the new test edges RDD generated
    above
11 {
12   Read the citation
13   Extract the source journal id and destination journal id
     of the citation
14   Group the citations with both the source and destination
     journal ids and add the weights
15   The result citation will be an aggregated citation between
      any two journals and not publications
16   Add or update the result citation to a journal edges RDD
     which is our final result.
17 }
```

LISTING 5.8: Building the Journal Edges

The algorithm can be depicted in Figure 5.11.

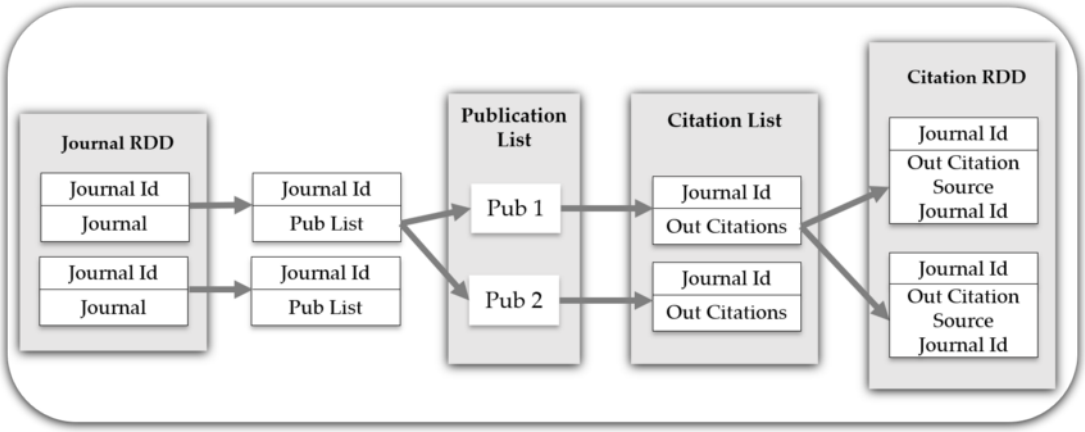

FIGURE 5.11: Aggregated Journal Citations

The final step to create a journal object is quite similar to how we build the publication object it is just the pre-processing of the data to build the journal vertices and edges which is a bit tricky. As shown in Figure 5.5[4] the journal object can be created in "GraphX".

5.6 Calculation of the most influential Publications or Journals

The publication and journal graphs created above shows a researcher for any given publication or journal what is the next most important publication or journal they need to look into. In other words, the graphs represent a weighted network of publications or journals where each edge weight of a publication or journal corresponds to how many times it got referenced from another publication or journal. We set rank to each vertex of a publication or journal representing this number which signifies how important it is to any given journal. We use page-rank value as one of the graph-theoretic features for calculating the rank of a vertex.

5.6.1 Page Rank Algorithm

Page Rank(PR) is a link analysis algorithm and assigns a numerical weight to each element of a hyper-linked set of documents, such as the world wide web, to measure its relative importance within the set. The algorithm may be applied to any collection of entities with reciprocal quotations and references. The numerical weight that it assigns to any given element E is referred to as the Page Rank of E[5]. A sample page rank can be displayed in Figure 5.12[6].

[4] *GraphX - Spark 3.0.0 Documentation* https://spark.apache.org/docs/latest/graphx-programming-guide.html.

[5] *PageRank* Page Version ID: 965942931. https://en.wikipedia.org/w/index.php?title=PageRank&oldid=965942931 (2020).

[6] User:Stannered, e. *Numeric examples of PageRank values in a small graph with a damping factor of 0.85. The exact solution is:* https://commons.wikimedia.org/wiki/File:PageRanks-Example.svg (2020).

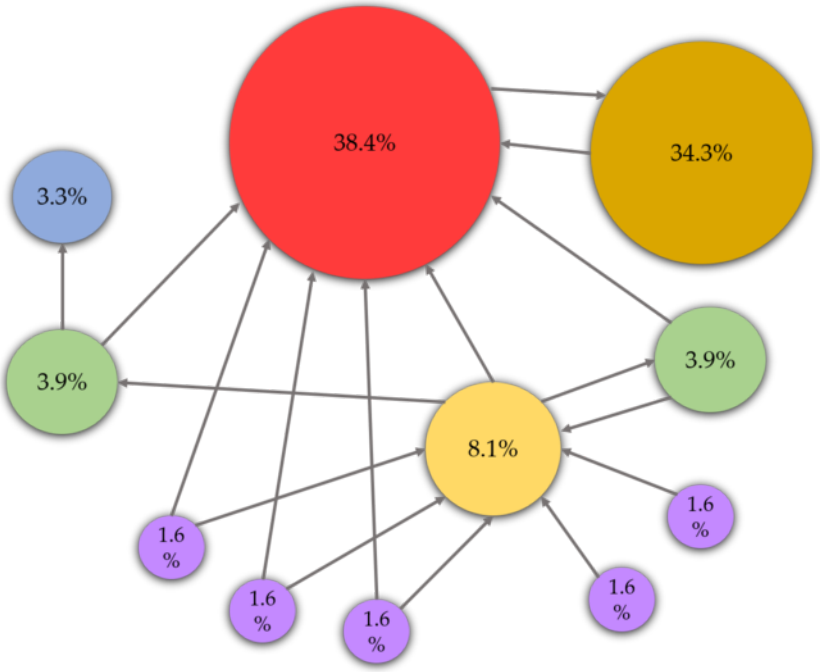

FIGURE 5.12: Page Rank Algorithm

5.6.2 User Interface

In the user interface, we consider the simplicity in the design, easy to learn, and use by users, integrated all the parts in one user interface. There are two basic operations the user is capable of performing which includes searching for a publication or searching for a journal.

Based on the user selection, the user enters the corresponding loop of either publication or journal search. Now the user enters the search string which is read into the system. For a publication search, the publication graph is constructed by building the publication vertices and publication citations from the raw data. After the graph is created, the algorithm first finds the publication being searched and then it retrieves the adjacent vertices and displays them to the user with the highest page rank which is the most influential publication connected to the publication being searched followed by all other results in the order of their importance. This can be displayed in the 5.13.

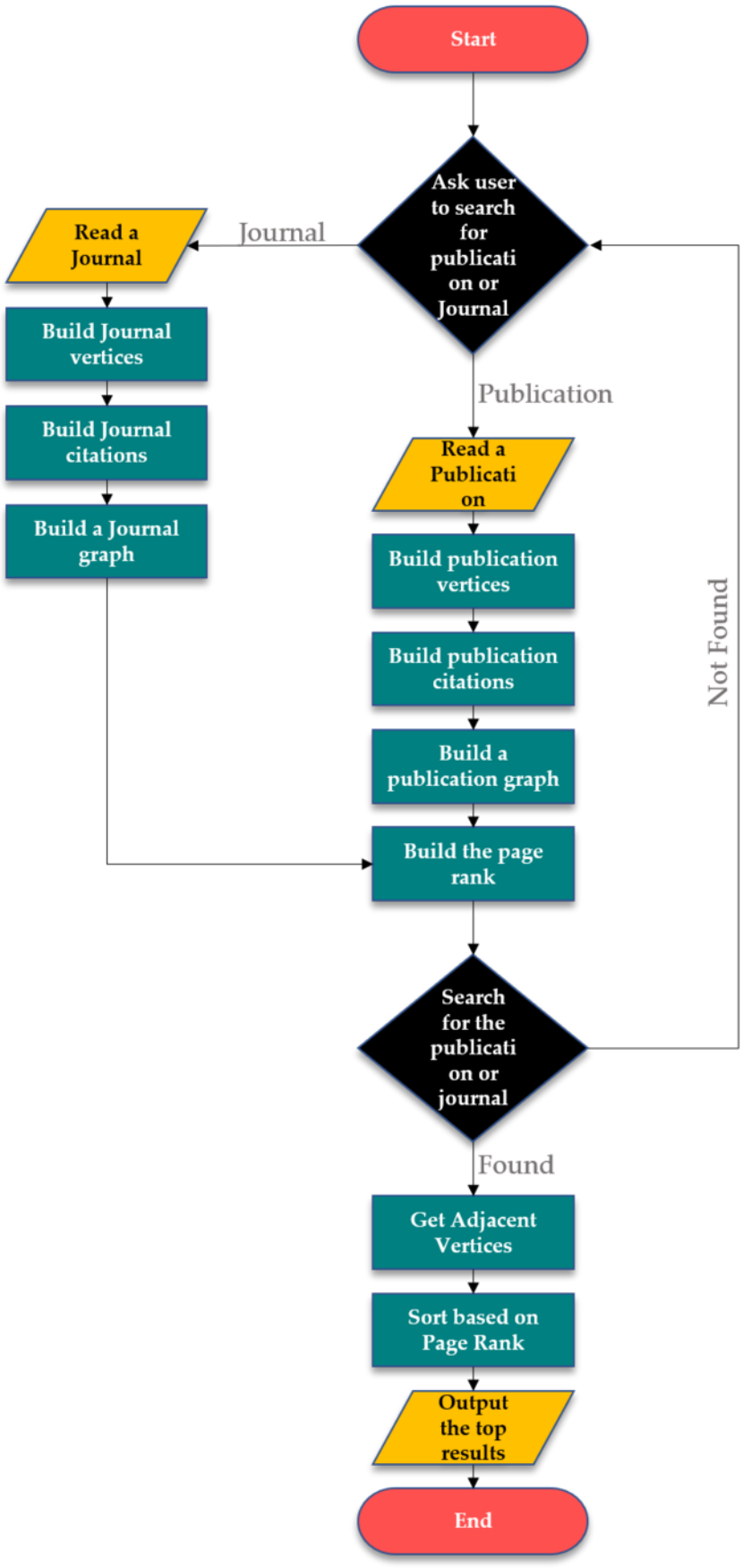

FIGURE 5.13: User Interface

Chapter 6

Evaluation

6.1 Evaluation Setup and Procedure

Program testing and analysis are the most practised means of verifying that a program possesses the featured required by its specification. Testing is a dynamic approach to verification in which code is executed with test data to assess the presence (or absence) of required features[See 33, p1]. There are innumerable tests that can be performed. However, we shall focus on assessing system proficiency. For this, we made an experiment that aims to determine the performance and accuracy which involves the following testing methods:

- Unit Testing - Testing the functionality of the application and its accuracy.
- Performance testing - Testing the response time and stability of the application.

6.2 Unit and Functionality Testing

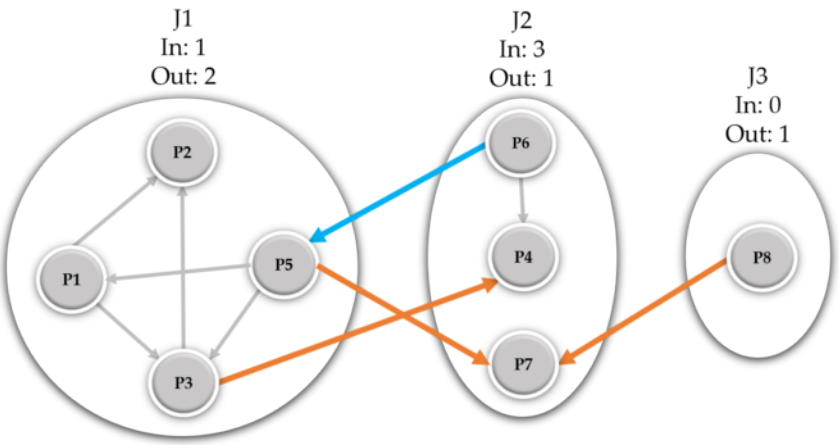

FIGURE 6.1: Unit Test Data

Unit testing is a level of software testing where individual units/components of software are tested. The major purpose of a unit test is to validate that each unit performs as designed. A unit is the smallest testable part of software which usually has one or a few inputs and usually a single output. In procedural programming, a unit may be an individual program, function, procedure. On the other hand, in object-oriented programming, the smallest unit is a method, which may belong to a base/ super class, abstract class or derived/child class. Some consider and treat a module of an application as an unit[1].

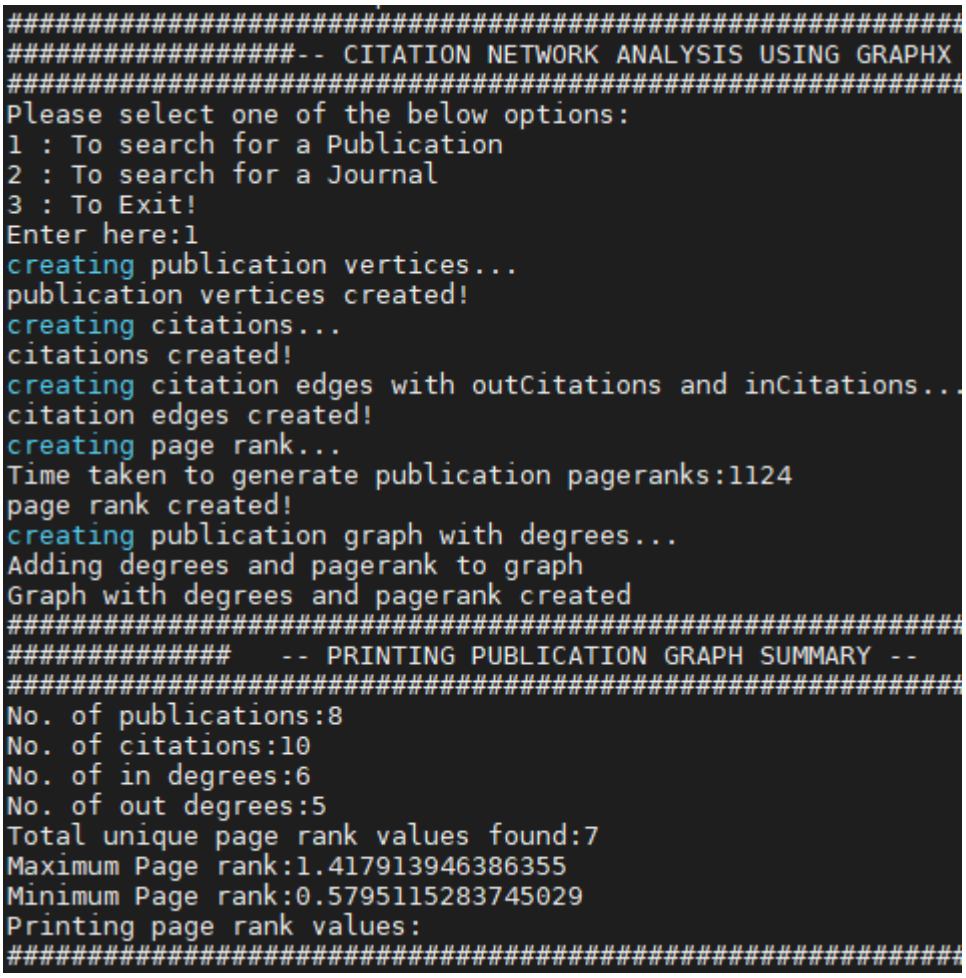

FIGURE 6.2: Publication Interactive Module

To perform the unit-testing, we create a simulated dataset with publications and journals as depicted in Figure 6.1 which helps us better understand the results. To start with,we have built an interactive module in the system. This interactive module takes input from the user processes the data into a citation network and allows the user to query on this graph. The system then results out with the most influential publications and journals as per the input publication/journal string. The simulated dataset contains publications named from "P1" to "P8" which are grouped into three

[1] *Unit Testing* Software Testing Fundamentals. Library Catalog: softwaretestingfundamentals.com. http://softwaretestingfundamentals.com/unit-testing/ (2020).

journals from "J1" to "J3" as shown in Figure 6.1. We shall proceed by testing the below:

- Testing the interactive module which contain the choice of search article.
- Test individual results from the citation networks for publications and journals and compare them with our understanding from the simulated data.

As an initial step, we start the publication interactive module as shown in Figure 6.2. The publication graph is built as expected and the summary of the graph clearly shows that 8 publications vertices are created with 10 total edges as per our sample dataset design. Furthermore, we can also see that the vertices that contain in-degrees are 6 and the vertices count that contain out-degrees are 5 which is also accurate to our sample dataset.

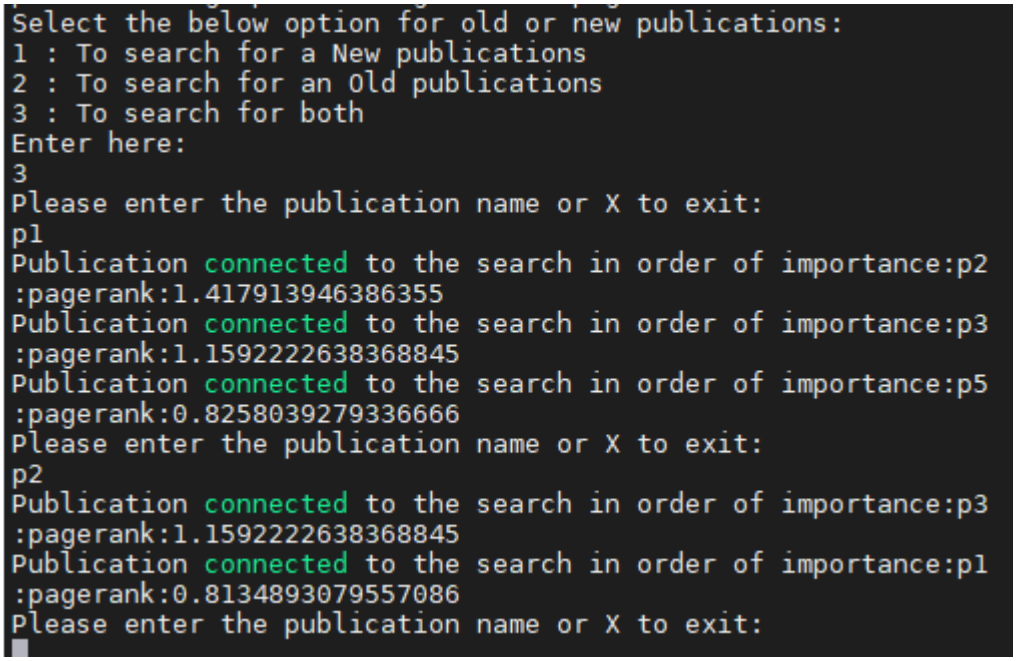

FIGURE 6.3: Publication Interactive Module Results

Our application has another option that allows you to get the publications that the current paper's research is based on or get the publications that used the current paper's research. Instead, we may use both of these together to get the papers of interest. In Figure 6.3, we include all the publications connected to our current search and try to get the results. For each of the publications searched by the user, we get all the influential publications in the order of their importance. For instance, when searched with publication "P1" we get publication "P2" as the most important publication that this publication is connected with. For someone who is performing his research would expect this result which is as expected. We see that the *pagerank* value is highest for "P2" followed by other publications "P3" and "P5" with decreasing *pagerank* values. Henceforth, the unit testing for publications is successful. Followed by publications we shall perform unit testing for journals.

As a second step, we start the journal interactive module as shown in Figure 6.4. The graph summary shows that the journal vertices created are 3 and the total number of citations are 3 as well. This matches exactly to our dataset as shown in Figure 6.1. We also see that the vertices that contain in-degrees are 2 and the vertices that contain out-degrees are 3.

```
Please select one of the below options:
1 : To search for a Publication
2 : To search for a Journal
3 : To Exit!
Enter here:2
creating journal vertices...
journal vertices created!
creating journal edges...
journal edges created!
Time taken to generate journal pageranks:8407
creating journal graph with degrees...
##########################################################
##############     -- PRINTING JOURNAL GRAPH SUMMARY
##########################################################
No. of journals:3
No. of citations:3
No. of in degrees:2
No. of out degrees:3
Total unique page rank values found:3
Maximum Page rank:1.459401633867416
Minimum Page rank:0.15005348867263996
Printing page rank values:
1.3905448774599436
1.459401633867416
0.15005348867263996
##########################################################
```

FIGURE 6.4: Journal Interactive Module

Once the journal graph is built as expected the next step is to test the functionality of retrieving the influential journals given a search journal name. For this test, we shall now search for only previous research papers connected with our journals. For the search query of the first journal "J1", we get only "J2" as the most important journal. When compared with Figure 6.5 we see that this result is as expected. Similarly, the journal searches for "J2" and "J3" yields successful results as well. In our next section, we shall discuss the performance testing.

6.3 Performance testing

Performance testing is the process of determining how a system performs in terms of responsiveness and stability under a particular workload. Furthermore, it can also

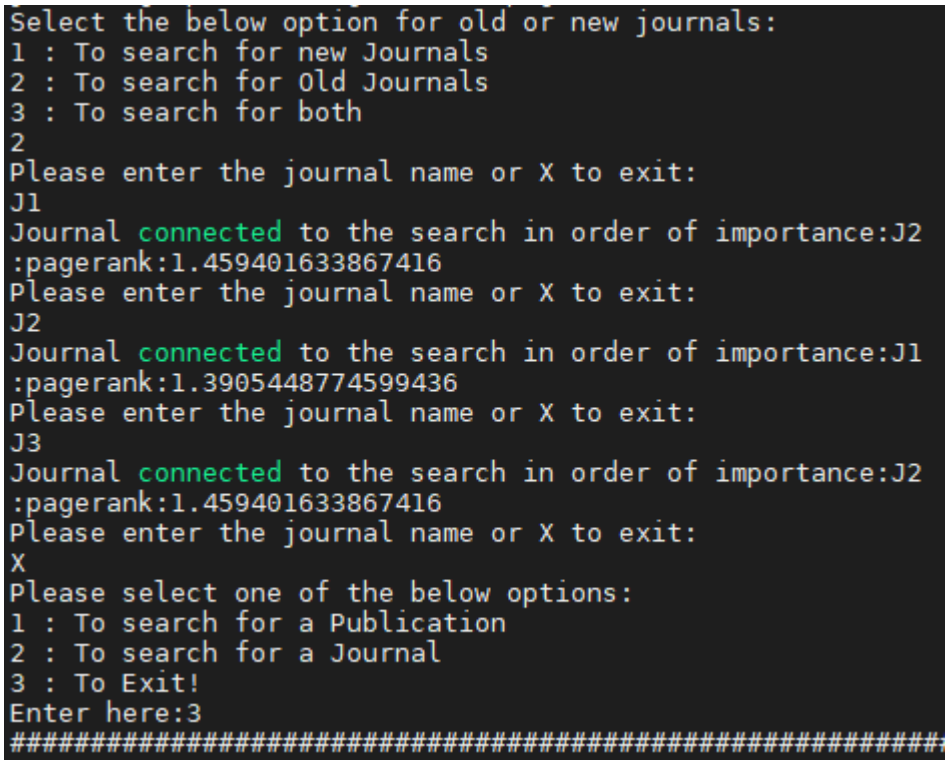

FIGURE 6.5: Journal Interactive Module Results

server to investigate, measure, validate, or verify other quality attributes of the system such as scalability, reliability, and resource usage[2]. Performance testing can serve different purposes:

- It can demonstrate that the system meets performance criteria
- It can compare two systems to find which performs better
- It can measure which parts of the system or workload cause the system to perform badly.

The performance goals will differ depending on the system's technology and purpose, but for this thesis, we include the following:

- Application Response time
- CPU Performance and Stability

6.3.1 Application Response time

Response time testing is a type of testing that measures the time taken for one system node to respond to the request of another. It is time a system or a system unit

[2] *Software performance testing* Page Version ID: 943165615. https://en.wikipedia.org/w/index.php?title=Software_performance_testing&oldid=943165615 (2020).

takes to react to a specific input until the process is over[3]. Response time measures the server response of every single transaction or query. Response time starts when a user sends a request and ends at the time that the application states that the request has completed. It is important to measure the response time for a request because the response time varies slightly depending on various factors. These factors can include the way metrics are calculated and gathered simulated load and captured speed, additional items recorded while monitoring the system, computing metrics gathered due to high resource consumption, the architecture of the system.

Experimental Setup

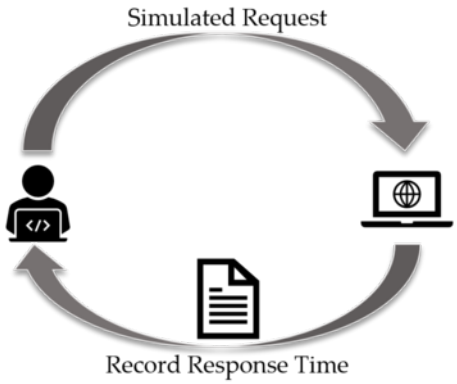

FIGURE 6.6: Simulation of Response time testing

The response time testing starts with developing a test that sends a request to the application and simulates this process for a defined number of iterations as shown in Figure 6.6. We perform this test for 500 times each for publications and journals and record the metrics.

The flowchart for the evaluation setup for publications can be depicted in Figure 6.7. As part of this evaluation setup, we first load the data into the system. For the next step, we get 500 publications from this data. However, while doing so we do not retrieve any random values instead pick the top most important publications in the entire data sample. Once, we have the top most important publications we send a request to each of these publications to the system to record the response time and various other metrics which we shall discuss more in the below sections. This process repeats for various dataset sizes including “5 GB”, “10 GB”, “15 GB”, “20 GB”, “25 GB”.

Similarly, the flowchart for the evaluation setup for journals can be depicted in Figure 6.8. As part of the journal evaluation setup, the publication data is loaded

[3]team, Q. P. *Response Time Testing* QA Platforms. Library Catalog: qa-platforms.com Section: QA articles. https://qa-platforms.com/response-time-testing/ (2020).

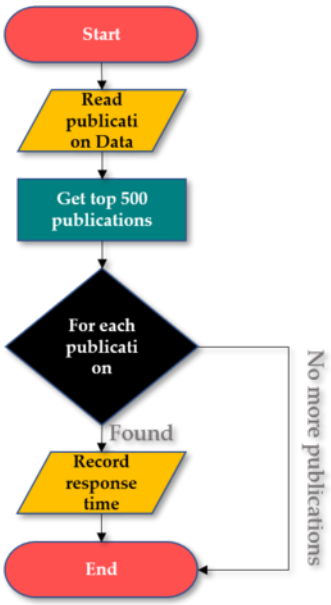

FIGURE 6.7: Publication Response time flow chart

similarly. However, the publications are grouped according to the journals, and journal data is generated. After this step, a set of 500 journals is picked not at random but instead the top most important journals are picked. As a final step, the journals are now processed one by one, and response time is recorded for each. This process is repeated for the same dataset sizes used for publication testing including "5 GB", "10 GB", "15 GB", "20 GB", "25 GB".

Apart from just the response time various other test metrics are recorded for each iteration which is as below:

- Response time to get the most important publications/journals for the specific publication/journal searching for.
- The count of the number of publications
- The count of the number of publication citations
- The count of the number of publication in-degrees
- The count of the number of publication out-degrees
- The page rank execution time for each iteration of publication
- The total time taken for the iterations of publications
- The total execution runtime for all our iterations of publications
- The total execution CPU time for all our iterations of publications
- The count of the number of journals
- The count of the number of journal citations
- The count of the number of journal in-degrees

- The count of the number of journal out-degrees
- The page rank execution time for each iteration of journal
- The total time taken for the iterations of journals
- The total execution runtime for all our iterations of journals
- The total execution CPU time for all our iterations of journals

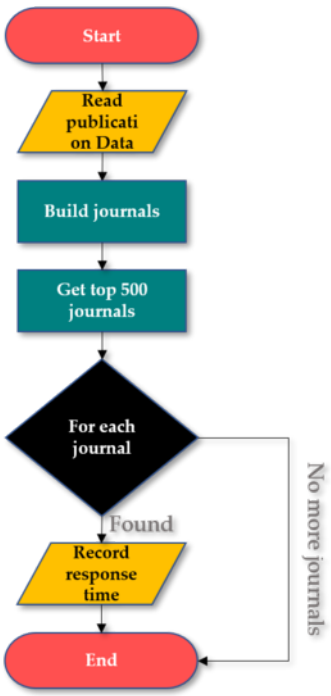

FIGURE 6.8: Journal Response time flow chart

Experimental Evaluation Results

For Publications:

As an initial step, let us understand the volume of the test data. We record the total publication count for each test dataset along with the citation count. From the Table 6.1 we can say that the rate at which citations increase is quite larger than the rate at which the publications increase. This is because with the increase in the dataset there will be more publications to connect with and this will cause fewer missing links. The *in degrees* and *out degrees* count also increases gradually as the data set size increases. However, the most important aspect to notice here is that the count of these two values are quite near all the time which makes these dataset sample ideal to compare the response time. We say this because if one of these datasets has the values *in degrees* and *out degrees* very far and another dataset has these values very close then the response

time will be very different for both of these and it would not be an accurate test in that case.

Publications count	Citations count	In degrees	Out degrees	Datset size
3086708	214017	136262	147398	5 GB
6171196	855245	435683	459474	10 GB
9253003	1897606	838254	841261	15 GB
12954467	3714183	1422987	1356578	20 GB
16037555	5679980	1974690	1811644	25 GB

TABLE 6.1: Publication Test Attributes

FIGURE 6.9: Publication Performance Statistics

In the next step, we look at the *pagerank* statistics to identify the number of unique *pagerank* values, the maximum *pagerank* for each dataset, the minimum *pagerank* for each dataset, and the total execution time to calculate the *pagerank* values. The test metrics obtained can be shown in Table 6.2 which says that the number of unique *pagerank* values increases exponentially from "5 GB" dataset until "25 GB" dataset. Similarly, the maximum *pagerank* values increase gradually from "191.2" to "5556.9". This suggests that as the dataset size keep increasing, we can measure more important publications more accurately. On the other hand, there is not much difference in the minimum *pagerank* values which is very small for all the datasets.

The time taken to record the test for all the 500 publication iterations is

Unique pageranks	Max Pagerank	Min Pagerank	PR Execution time	Datset size
9702	191.1982022	0.947947362	66599	5 GB
62711	2454.766657	0.919822884	154359	10 GB
177223	1253.493731	0.901585842	114948	15 GB
393809	5091.557431	0.885971417	161083	20 GB
634175	5556.893462	0.876395538	356973	25 GB

TABLE 6.2: Publication Page rank Attributes

recorded and tabulated in Table 6.3 which shows the total test duration(in milliseconds) taken for each of the datasets for all the iterations. The second entity is the total spark execution run time taken for each of the datasets. Following this is the third entity which records the total spark CPU run time. In Figure 6.9, we plot all the statistics together to get a better understanding of the test metrics and Figure 6.10 helps us understand the *pagerank* variation better.

Total Test Duration	Total Execution Run time	Total CPU Run time	Datset size
1256312	1138167	747756	5 GB
2657490	564647	389868	10 GB
4534177	698971	587440	15 GB
8495879	1043193	967593	20 GB
14363901	12457383	11926442	25 GB

TABLE 6.3: Publication Test Time Attributes

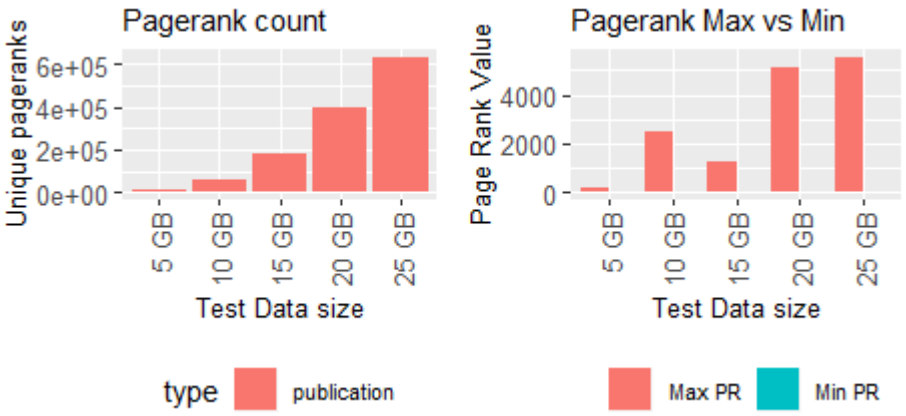

FIGURE 6.10: Publication Pagerank Statistics

One key phenomena to be noted here is that the spark execution run time metrics and the CPU run time metrics derived are pretty high for 25 GB for the publications. Even though the results for all the 500 publications are retrieved as shown in Figure 6.11 which we shall discuss in more detail in the next part of the thesis. From this we can conclude that the executors are just awaiting even after the tasks are completed. One of the reasons this happens is due to the irregular partitioning of the RDD done by spark which creates several empty partitions that awaits for the iterations to complete and return it's control to the Application Master. To fix this we need to find the right repartitioning value.

The response time results obtained are plotted into a box plot as shown in Figure 6.11. As the dataset increases the graph generated gets bigger and bigger each time and thus the response time increases gradually too. We can interpret the plot as below:

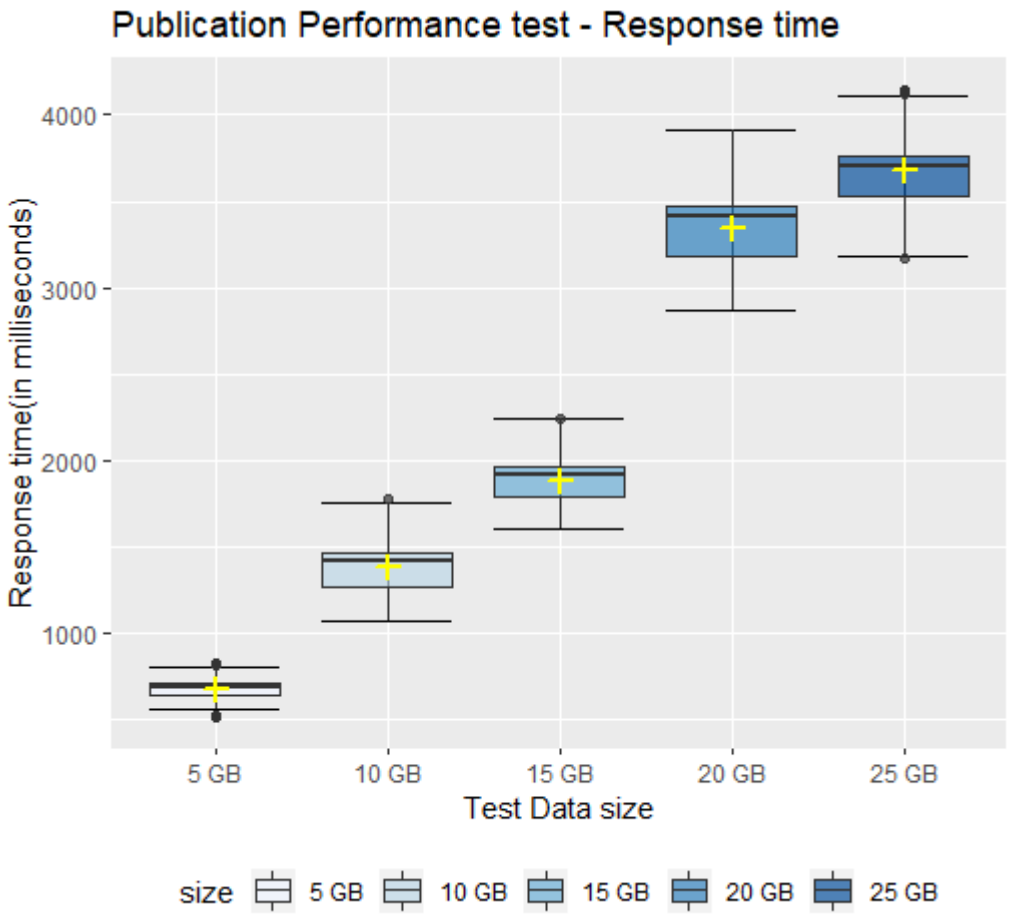

FIGURE 6.11: Publication Performance Response time 5 GB to 25 GB

- The response time for the "5 GB" dataset is the lowest and gradually increases till "25 GB".
- There is a small jump in the response time from "15 GB" to that of "20 GB".
- The interquartile ranges for all the boxplots are narrow and thus we can say that the response time generated is not dispersed and the system gives almost uniform results.
- Comparatively, the "20 GB" dataset has response little widely spread than the others.
- From the boxplots, for all the different dataset sizes we see one thing in common which being they are asymmetric. The size of the upper whisker is larger than that of lower whisker which suggests that it is left-skewed(negative skewed).

For Journals:

After the publications, let us now understand the test metrics of journals and the number of journals generated for the publication test data. From Table 6.4 we can see that the journal count is increasing but this not as exponentially increasing as the publication because the new publications getting appended to our test data are getting grouped inside the existing journals. However, the citations increase exponentially in this case because the count of publications increases the citations connecting the journals too. And finally, for the comparison between the journal "in degrees" and "out degrees" we can say that the distance is quite far from the "in degrees" to "out degrees" values of the publications. However, the difference of these two values remains constant across the various dataset sizes including "5 GB", "10 GB", "15 GB", "20 GB", "25 GB" which makes it these samples ideal for the test.

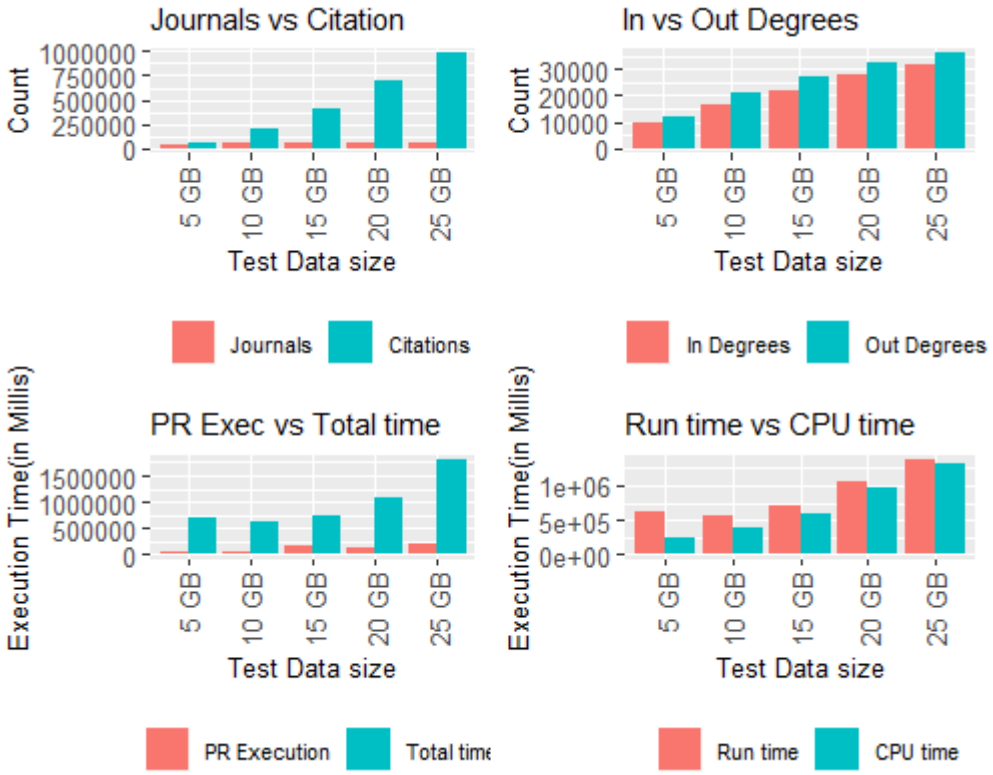

FIGURE 6.12: Journal Performance Statistics

Journals count	Citations count	In degrees	Out degrees	Datset size
41291	62736	9635	11976	5 GB
49144	209918	16627	20712	10 GB
52820	408519	21980	27059	15 GB
55938	701165	27876	32671	20 GB
57736	976336	31522	35831	25 GB

TABLE 6.4: Journal Test Attributes

While analysing the *pagerank* test metrics for journals we can notice that the unique *pagerank* values are increasing with the size but when compared to the publications these are still very less. The maximum *pagerank* values are quite less when compared to the test results of publications for the respective data sizes with the maximum among the samples being "308.1" and minimum among the samples being "155.28". The minimum *pagerank* values are less than one for all the samples. These test metrics for *pagerank* can be shown in the Table 6.5.

Unique pageranks	Max Pagerank	Min Pagerank	PR Execution time	Datset size
6051	155.2860853	0.70884747	43174	5 GB
12425	214.9800044	0.517758313	54981	10 GB
17906	308.1061748	0.330377613	150202	15 GB
23455	216.8584179	0.397901044	142497	20 GB
27355	237.9339538	0.346411447	191199	25 GB

TABLE 6.5: Journal Page rank Attributes

Total Test Duration	Total Execution Run time	Total CPU Run time	Datset size
706400	621701	249070	5 GB
610699	564647	389868	10 GB
721947	698971	587440	15 GB
1067040	1043193	967593	20 GB
1807728	1377584	1314531	25 GB

TABLE 6.6: Journal Test Time Attributes

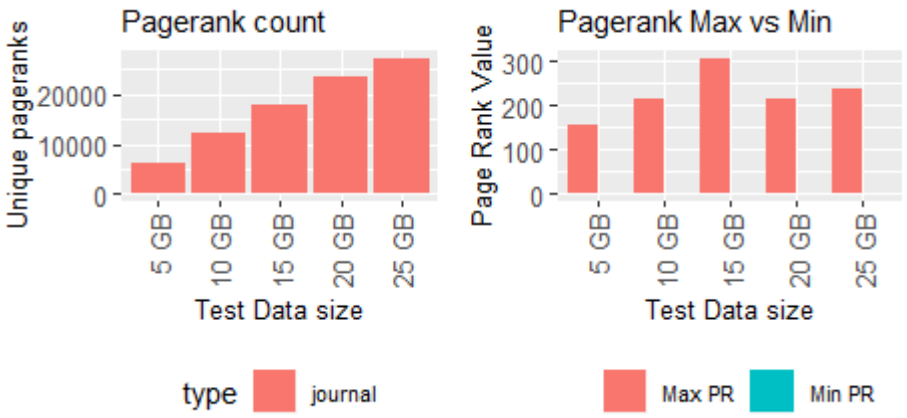

FIGURE 6.13: Journal Pagerank Statistics

Lastly, the time taken to record the test for all the 500 journal iterations is recorded and tabulated in Table 6.6 which shows the total test duration(in milliseconds) taken for each of the datasets for all the iterations. The second entity is the total spark execution run time taken for each of the datasets. Following this is the third entity which records the total spark CPU run time. In Figure 6.12, we plot all these statistics together to get a much better understanding of the test metrics and Figure 6.13 helps us understand the *pagerank* variations better. The key aspect to notice here is that the total time taken to generate the journal test results are very much lower than the time taken to generate the publication results. The same applies to the total execution run time and total CPU run time.

We record the response time for the journals separately just like publications and plot into a boxplot as shown in the Figure 6.14. When compared to the publication search the journals are retrieved much faster because the number of nodes for journals graph are less than the publications graph.

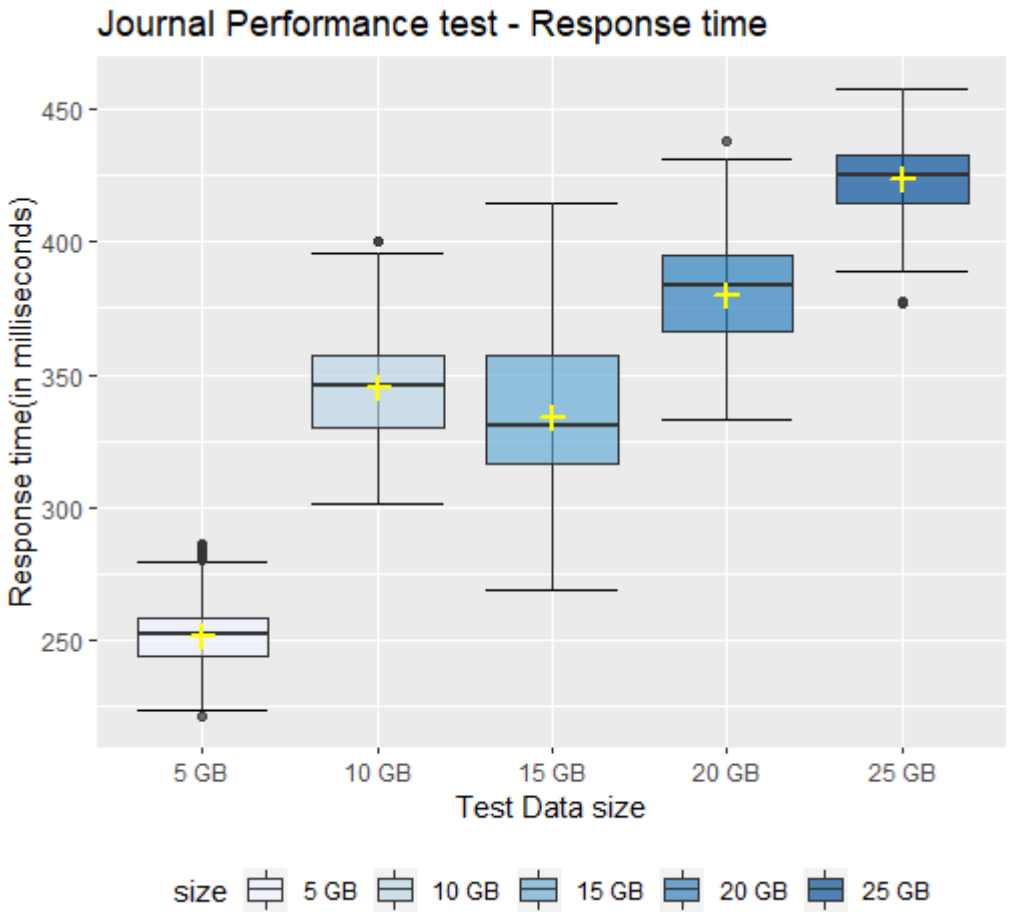

FIGURE 6.14: Journal Performance Response time

We can interpret the journal response time *boxplot* as below:

- The response time for the "5 GB" dataset is the lowest and increases gradually until "25 GB".
- From the interquartile ranges, the response time dispersion is small for "5 GB" and "25 GB" by "10 GB" and "20 GB". And finally followed by "15 GB" which has the highest dispersion of the response time.
- The overall spread of the data can be shown by the extreme values at the end of two whiskers. A wider distribution is noticed for "15 GB" followed by "10 GB" and "20 GB" followed by "25 GB" and "5 GB".
- Similar to the publications the journals show left skewness as well but "15 GB" shows right skewness.

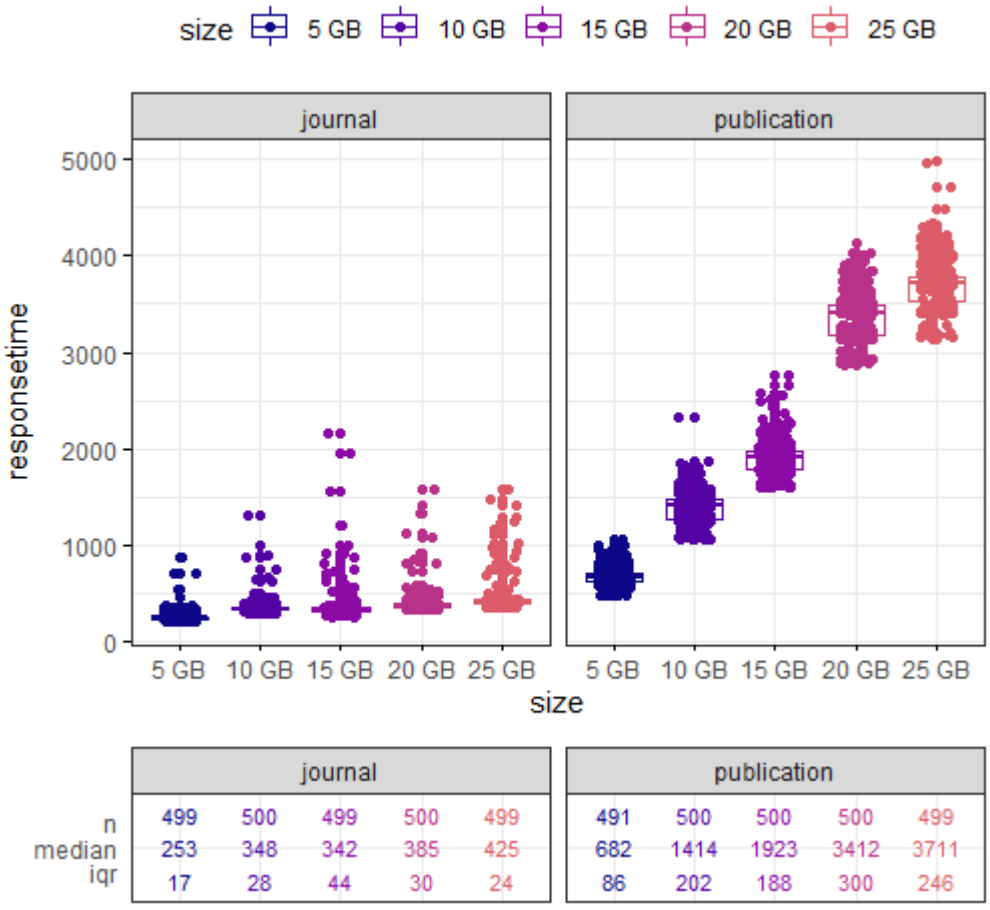

FIGURE 6.15: Journal and Publication Response Time Comparison

In Figure 6.15, we compare side by side the response time dispersion between various data sets for journals vs publications. There is an exponential increase of publications while there is consistency in the response time for the journals. The median values range from "250" to "420" milliseconds for journals which is the average amount of time it takes to get the most influential journals for any search. On the other hand, the median values range from "682" to "3700" milliseconds for publications which is the average amount of time it takes to get the most influential publications for any search. The dispersion of the response time is also very close for journals and publications as well which is between "17" to "44" for journals and between "86" to "300" for publications.

6.3.2 CPU Performance and Stability

As a final step of our evaluation, we compare the CPU response time for the top 10 tasks for each of the datasets to the execution run time for these tasks. We first calculate this for a smaller dataset from "500 MB" to "2.5 GB". The plot Figure 6.16 represents

these results for publications for all these datasets. The histogram in the plot represents the execution runtime for the top 10 tasks in the sorted order while the red line plot represents the CPU performance. From the figure, we can notice that the execution time keeps increasing gradually from "500 MB" to "2.5 GB". However, it is to be noted that there is a spike of CPU response time at certain parts of the graph while at other parts the CPU response time stays uniform.

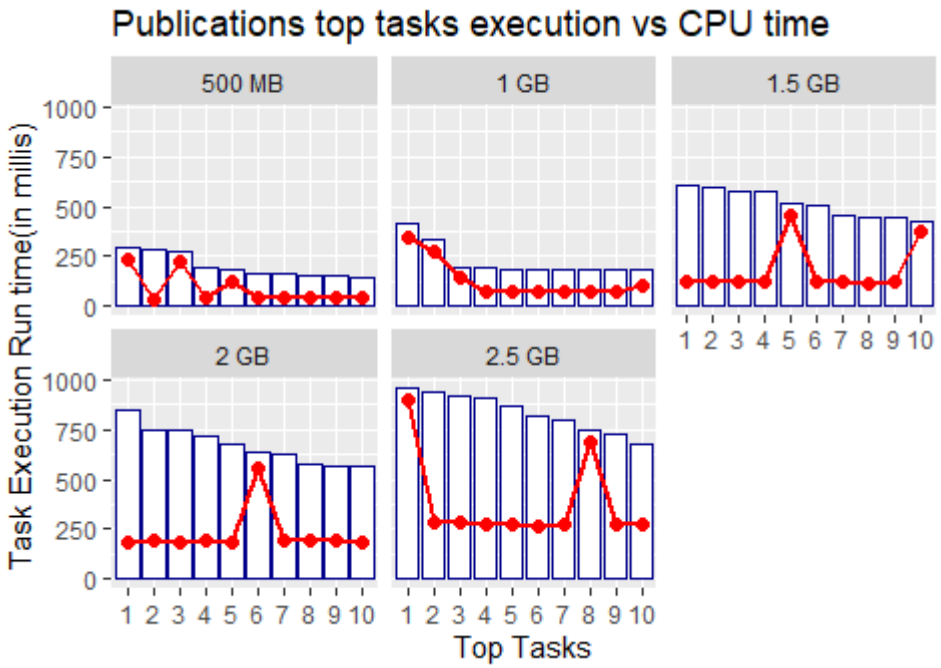

FIGURE 6.16: Publications Top Tasks Execution vs CPU time from 500 MB to 2.5 GB

When dealing with journals for the same dataset as shown in Figure 6.17 represents this plot between execution time and CPU performance. Here we notice that the execution response time for the tasks is almost uniform except for the top highest task which took little more time. The CPU response time shows some uniformity but also shows a slight variation which is different from that of the publication plots.

We now plot these same plots for the much bigger and broader datasets from "5 GB" to "25 GB". The plots again show the information about the execution response time and CPU response time for the top 10 tasks. The histogram in the plot represents the execution runtime for the top 10 tasks in the sorted order while the red line plot represents the CPU performance.

Figure 6.18 represents this plot for the publications. From the figure, we can notice that the execution time increases gradually from "5 GB" to "25 GB" in a uniform fashion. On the other hand, the CPU response time also increases from one dataset to another. However, it seems to be uniform across all the tasks.

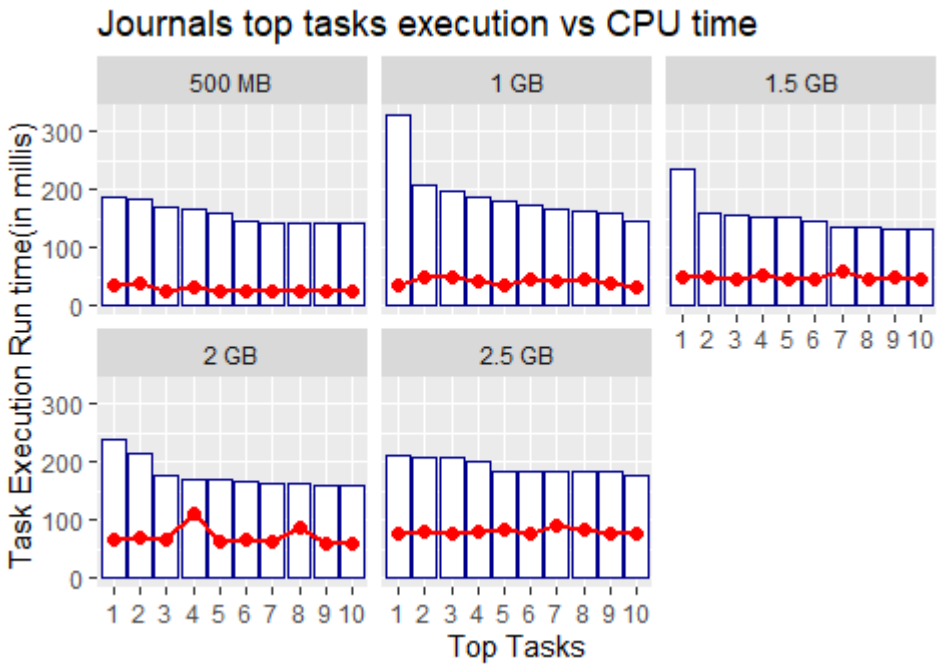

FIGURE 6.17: Journals Top Tasks Execution vs CPU time from 500 MB to 2.5 GB

Figure 6.19 represents a similar plot for the journals. From the figure, we can notice that the execution time does not show as much change as compared to publications and we see uniformity. On the other hand, the CPU response time also remains the same across datasets and performing steadily among the tasks. When comparing the publication and journal pair of plots between bigger datasets to much lower datasets we notice that the CPU performance is much stable when looked at a broader perspective. We conclude our evaluation here and proceed with the conclusion in our next and final chapter.

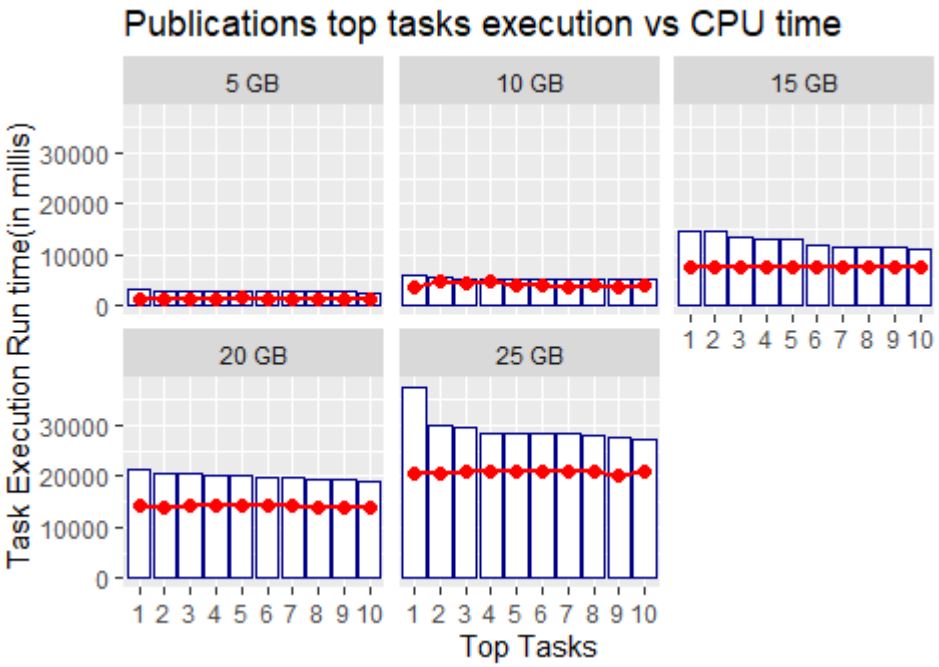

FIGURE 6.18: Publications Top Tasks Execution vs CPU time from 5 GB to 25 GB

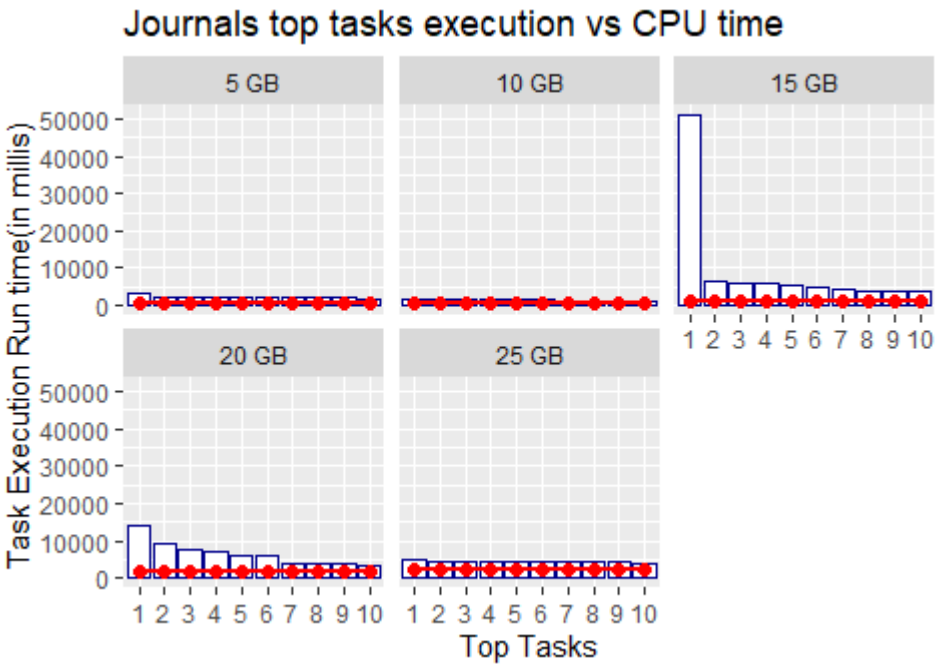

FIGURE 6.19: Journals Top Tasks Execution vs CPU time from 5 GB to 25 GB

Chapter 7

Conclusions

In this dissertation, we have implemented a *Citation Network Analysis* engine for big data using Apache Spark and Graph X framework. This system successfully identifies the prominence of a publication from a collection. Furthermore, the results of this thesis suggest how advantageous it is to implement Apache Spark for distributed, big-data processing.

In this concluding chapter, we outline our main contributions and recommend future directions for the community to build upon our work.

7.1 Author's Contribution

This dissertation makes the following main contributions:

- An Interactive IR System that allows researchers to retrieve scientific publications or journals. While doing so, the researchers can retrieve the publications or journals that are relevant to their field of research. This saves them a lot of time and effort and put them in the right direction for their research. Apart from this, there are many other practical applications such as tracing prior art, citation of early patents, calculating citation indexes and referring back to judgements made in the previous cases by judges and much more. When we compare this to the other existing systems, they work based on the search mechanisms like pattern matching or textual similarities that depend on the user queries which are quite subjective to human understanding. Thus, they result in a lot of irrelevant documents and the documents actually matching user's needs are stashed in the rest of the results. And it becomes even more challenging when we had to deal with the ever-increasing number of documents. Our system overcomes this issue by making use of more concrete citation-based similarity metrics to generate the results.
- A citation network graph is generated that can handle large amounts of data as Apache Spark is implemented. Apache Spark is capable of handling and processing iterative and interactive algorithms like *pagerank* and is much faster than the traditional map-reduce paradigm. The reason behind this is due to the use of distributed memory abstraction called Resilient Distributed Datasets. The use of RDD's helps a lot in building this graph since it helps reduce disk writes by promoting in-memory data processing.

- Creation of multiple graphs for multiple search entities to improve performance. Our algorithm proposes to create multiple citation network graphs to retrieve information in the most efficient way. For instance, a publication search and journal search looks like a small and similar task but when queried on large data it is essential to come up with much more sophisticated and efficient ways. To understand the difference between these two requests it is essential to know that the size of the journal graph that gets generated is much smaller than the publication graph since the journals are summarised by a group of publications and it makes no sense in using the bigger publication graph for this request of journal search. To overcome this our system constructs an optimal graph for the type of request in order to retrieve the results in the quickest time possible with the least load on the system.
- We also provide insights in determining the speed of graph algorithms and how this system performs for various datasets by calculating the response time, execution run time and CPU run time.

7.2 Future Works

Even though this dissertation provides pretty good results in terms of accuracy, stability and performance of the proposed system, there exist several research directions one can explore. Below are certain points that open opportunities for future work as an addition to the accomplishments of this thesis:

- We can extend improvements to this system by implementing additional features such as co-citation analysis as brought in section 2.8. By implementing this feature one can query for the most prominent publications or journals based on the co-cited author information. Furthermore, this feature becomes even more powerful since we combine it with big data and Apache Spark. We can extend the functionality of the existing system as well by implementing a search mechanism based on depth. This facilitates a user to search for a publication or journal and specify the depth along with it which increases the radius of the adjacent publications or journal to query upon.
- We can explore to understand if any other graph algorithm can be used to achieve our objectives or to increase the efficiency of our system. Few of the algorithms that can be considered are MOZ, Majestic, Ahrefs, OnCrawl and TrustRank.
- We can study further optimizations on the understanding and improvement of the graph algorithm performance. This can be achieved by performing various experiments of distribution on spark and effect of disk and network I/O. We can see how network latency and disk I/O latency affect the graph performance on Apache Spark.
- Incremental Graph Computations is one of the topics always spoken about. In a real-world the data is often streaming continuously from various sources and the graphs need recomputation of entire data pipeline. If Incremental Graph computations can be researched it will be quite helpful in the future.

Bibliography

1. Abrishami, A. & Aliakbary, S. Predicting citation counts based on deep neural network learning techniques. *Journal of Informetrics* **13,** 485–499 (May 2019).
2. Havemann, F. & Scharnhorst, A. Bibliometric Networks (Dec. 2012).
3. He, Q. *et al.* Detecting topic evolution in scientific literature: How can citations help? *International Conference on Information and Knowledge Management, Proceedings*, 957–966 (Jan. 2009).
4. *Digital library - New World Encyclopedia* https://www.newworldencyclopedia.org/entry/digital_library#Types_of_digital_libraries.
5. Batagelj, V. Efficient Algorithms for Citation Network Analysis. *Preprint Series* **41** (Oct. 2003).
6. Grefenstette, E. *Analysing Document Similarity Measures* in (2009).
7. Gipp, B., Meuschke, N. & Lipinski, M. *CITREC: An Evaluation Framework for Citation-Based Similarity Measures based on TREC Genomics and PubMed Central* in (Feb. 2015).
8. Easley, D. & Kleinberg, J. *Networks, Crowds, and Markets: Reasoning About A Highly Connected World* ISBN: 978-0-521-19533-1 (Jan. 2010).
9. Besson, M., Delmas, E., Poisot, T. & Gravel, D. in (Jan. 2018). ISBN: 9780124095489.
10. Bishop, M., Young, B., Huo, D. & Chi, Z. in (Jan. 2020). ISBN: 9780124095489.
11. Riazi, S. & Norris, B. Distributed-Memory Vertex-Centric Network Embedding for Large-Scale Graphs. arXiv: 2006.04236 [cs.DC] (2020).
12. Pandey, P., Singh, M., Goyal, P., Mukherjee, A. & Chakrabarti, S. Analysis of Reference and Citation Copying in Evolving Bibliographic Networks (Dec. 2019).
13. Kassaie, B. *SPARQL over GraphX* 2017. arXiv: 1701.03091 [cs.DB].
14. Hummon, N. & Doreian, P. Connectivity in a Citation Network: The Development of DNA Theory. *Social Networks - SOC NETWORKS* **11,** 39–63 (Mar. 1989).
15. Hummon, N. & Doreian, P. Computational methods for social network analysis. *Social Networks* **12,** 273–288 (Dec. 1990).
16. Segal, T. *The Deal With Big Data* Investopedia. https://www.investopedia.com/terms/b/big-data.asp (2020).
17. *What is Hadoop?* Bernard Marr. https://www.bernardmarr.com/default.asp?contentID=1080 (2020).
18. *Techopedia – IT Dictionary for Computer Terms and Tech Definitions* https://www.techopedia.com/dictionary (2020).
19. *Welcome to Apache Flume — Apache Flume* https://flume.apache.org/ (2020).

20. *Apache Spark with Hadoop – Why it Matters? | Edureka.co* Edureka. Section: Big Data. https://www.edureka.co/blog/apache-spark-with-hadoop-why-it-matters/ (2020).
21. Kovachev, D. *A Beginner's Guide to Apache Spark* Medium. https://towardsdatascience.com/a-beginners-guide-to-apache-spark-ff301cb4cd92 (2020).
22. Zaharia, M. *et al. Resilient Distributed Datasets: A Fault-Tolerant Abstraction for In-Memory Cluster Computing* in *Presented as part of the 9th USENIX Symposium on Networked Systems Design and Implementation (NSDI 12)* (USENIX, San Jose, CA, 2012), 15–28. ISBN: 978-931971-92-8. https://www.usenix.org/conference/nsdi12/technical-sessions/presentation/zaharia.
23. Xin, R., Gonzalez, J., Franklin, M. & Stoica, I. *GraphX: a resilient distributed graph system on Spark* in (June 2013).
24. *GraphX - Spark 3.0.0 Documentation* https://spark.apache.org/docs/latest/graphx-programming-guide.html.
25. Elgendy, N. & Elragal, A. *Big Data Analytics: A Literature Review Paper* in. **8557** (Aug. 2014), 214–227.
26. *National HPC Service* ICHEC. Library Catalog: www.ichec.ie. https://www.ichec.ie/academic/national-hpc.
27. *Kay* ICHEC. Library Catalog: www.ichec.ie. https://www.ichec.ie/about/infrastructure/kay.
28. Ammar, W. *et al. Construction of the Literature Graph in Semantic Scholar* in *NAACL-HLT* (2018).
29. *What is Parse? - Definition from Techopedia* https://www.techopedia.com/definition/3853/parse.
30. *JSON* https://www.json.org/json-en.html.
31. *PageRank* Page Version ID: 965942931. https://en.wikipedia.org/w/index.php?title=PageRank&oldid=965942931 (2020).
32. User:Stannered, e. *Numeric examples of PageRank values in a small graph with a damping factor of 0.85. The exact solution is:* https://commons.wikimedia.org/wiki/File:PageRanks-Example.svg (2020).
33. Morell, L. Unit Testing and Analysis, 41 (Apr. 1989).
34. *Unit Testing* Software Testing Fundamentals. Library Catalog: softwaretesting-fundamentals.com. http://softwaretestingfundamentals.com/unit-testing/ (2020).
35. *Software performance testing* Page Version ID: 943165615. https://en.wikipedia.org/w/index.php?title=Software_performance_testing&oldid=943165615 (2020).
36. team, Q. P. *Response Time Testing* QA Platforms. Library Catalog: qa-platforms.com Section: QA articles. https://qa-platforms.com/response-time-testing/ (2020).

Appendix A

Code

A.1 Execution Class

```
import net.liftweb.json.{DefaultFormats, _}
import org.apache.spark.graphx._
import org.apache.spark.rdd.RDD
import org.apache.spark.sql.SparkSession

object CitationParser {

  def main(args: Array[String]): Unit = {

    //    Getting the properties for the environment
    val prop = Utils.getProperties()
    //    checking for test flags
    val testMode = prop.getProperty("test.mode")

    //    Creating a spark configuration
    //    val conf = new SparkConf()
    //    conf.setAppName("Citation")
    //    Creating a spark context driver and setting log level to error
    //    val sc = new SparkContext(spark)//spark context code

    val env = prop.getProperty("env")
    val spark = if (env.equals("local")) {
      SparkSession
        .builder()
        .master("local[*]")
        .config("spark.executor.extraJavaOptions", "-XX:+UseG1GC -XX:+
    UnlockDiagnosticVMOptions -XX:+G1SummarizeConcMark -XX:
    InitiatingHeapOccupancyPercent=35 -verbose:gc -XX:+PrintGCDetails -XX
    :+PrintGCDateStamps -XX:OnOutOfMemoryError='kill -9 %p'")
        .config("spark.driver.extraJavaOptions", "-XX:+UseG1GC -XX:+
    UnlockDiagnosticVMOptions -XX:+G1SummarizeConcMark -XX:
    InitiatingHeapOccupancyPercent=35 -verbose:gc -XX:+PrintGCDetails -XX
    :+PrintGCDateStamps -XX:OnOutOfMemoryError='kill -9 %p'")
        .appName("Citation")
        .getOrCreate()
    } else {
      SparkSession
        .builder()
        .config("spark.executor.extraJavaOptions", "-XX:+UseG1GC -XX:+
    UnlockDiagnosticVMOptions -XX:+G1SummarizeConcMark -XX:
    InitiatingHeapOccupancyPercent=35 -verbose:gc -XX:+PrintGCDetails -XX
    :+PrintGCDateStamps -XX:OnOutOfMemoryError='kill -9 %p'")
```

```
34        .config("spark.driver.extraJavaOptions", "-XX:+UseG1GC -XX:+
    UnlockDiagnosticVMOptions -XX:+G1SummarizeConcMark -XX:
    InitiatingHeapOccupancyPercent=35 -verbose:gc -XX:+PrintGCDetails -XX
    :+PrintGCDateStamps -XX:OnOutOfMemoryError='kill -9 %p'")
35        .appName("Citation")
36        .getOrCreate()
37    }
38    val sc = spark.sparkContext
39    sc.setLogLevel("ERROR")
40
41    //    Reading file to RDD
42    println("Reading Input...")
43    val lines_orig = sc.textFile(prop.getProperty("file.path")) //spark
    context code
44    //    val lines_orig = spark.read.text(prop.getProperty("file.path")
    )
45
46    println("Initial paritition size:" + lines_orig.partitions.size)
47    val lines_sample = lines_orig.sample(false, prop.getProperty("sample
    .size").toDouble, 2)
48    println("Input data loaded!")
49
50    println("Sample paritition size:" + lines_sample.partitions.size)
51
52    val repart_size = prop.getProperty("repartition.size").toInt
53    val lines = lines_sample.repartition(repart_size)
54
55    //    printing the number of records
56    println(s"Number of entries in input data is ${lines.count()}")
57
58    //    extracting the data using lift json parser
59    //    println("Extracting the data using lift json parser...")
60    val publicationsRdd: RDD[Publication] = lines.map(line => {
61      implicit val formats: DefaultFormats.type = DefaultFormats;
62      parse(line).extract[Publication]
63    }).cache()
64    //    println("publicationRdd created!")
65
66    if (testMode.equals("true")) {
67      val tstEntyPblctn = prop.getProperty("test.entity.publication")
68      val tstEntyJrnl = prop.getProperty("test.entity.journal")
69      val testPrintMode = prop.getProperty("test.print.results")
70      val taskMetrics = ch.cern.sparkmeasure.TaskMetrics(spark)
71
72      if (tstEntyPblctn.equals("true")) {
73
74        Utils.prntHdngLne("TESTING PUBLICATIONS")
75        //        this retrieves the task metrics of this code snippet
76        Publication.prfmChkPblctns(prop, sc, publicationsRdd,
    testPrintMode, taskMetrics)
77        Utils.prntGrphTstMtrcs(spark)
78        Utils.prntHdngLne("TESTING PUBLICATIONS COMPLETED")
79      }
80      if (tstEntyJrnl.equals("true")) {
81        Utils.prntHdngLne("TESTING JOURNALS")
82        //        this retrieves the task metrics of this code snippet
83        Journal.prfmChkJrnls(prop, sc, publicationsRdd, testPrintMode,
    taskMetrics)
84        Utils.prntGrphTstMtrcs(spark)
85        Utils.prntHdngLne("TESTING JOURNALS COMPLETED")
86      }
87    } else {
```

```
88      Utils.prntHdngLne("CITATION NETWORK ANALYSIS USING GRAPHX AND
    APACHE SPARK")
89      println("Please select one of the below options:")
90      println("1 : To search for a Publication")
91      println("2 : To search for a Journal")
92      println("3 : To Exit!")
93      print("Enter here:")
94
95      Iterator.continually(scala.io.StdIn.readInt)
96        .takeWhile(_ != 3)
97        .foreach {
98          input =>
99            input match {
100             case 1 => {
101               //  Searching for publications
102               val pblctnDgrGrph = Publication.getPublicationGraph(sc,
    publicationsRdd)
103
104               println("Select the below option for old or new
    publications:")
105               println("1 : To search for a New publications")
106               println("2 : To search for an Old publications")
107               println("3 : To search for both")
108               println("Enter here:")
109
110               val searchMode = scala.io.StdIn.readInt match {
111                 case 1 => EdgeDirection.In
112                 case 2 => EdgeDirection.Out
113                 case 3 => EdgeDirection.Either
114               }
115
116               println("Please enter the publication name or X to exit:
    ")
117               Iterator.continually(scala.io.StdIn.readLine)
118                 .takeWhile(_ != "X")
119                 .foreach {
120                   searchPublication =>
121                     pblctnDgrGrph.collectNeighbors(searchMode).lookup
    ((pblctnDgrGrph.vertices.filter {
122                       journal => (journal._2.publicationName.equals(
    searchPublication))
123                     }.first)._1).map(publication => publication.
    sortWith(_._2.pr > _._2.pr).foreach(publication => println("
    Publication connected to the search in order of importance:"+
    publication._2.publicationName+"\n:pagerank:"+publication._2.pr)))
124                     println("Please enter the publication name or X to
     exit:")
125                 }
126               pblctnDgrGrph.unpersist()
127             }
128             case 2 => {
129               //  Searching for journal
130               val jrnlDgrGrph = Journal.getJournalGraph(sc,
    publicationsRdd)
131
132               println("Select the below option for old or new journals
    :")
133               println("1 : To search for new Journals")
134               println("2 : To search for Old Journals")
135               println("3 : To search for both")
136
137               val searchMode = scala.io.StdIn.readInt match {
138                 case 1 => EdgeDirection.In
```

```
                    case 2 => EdgeDirection.Out
                    case 3 => EdgeDirection.Either
                  }

                  println("Please enter the journal name or X to exit:")

                  Iterator.continually(scala.io.StdIn.readLine)
                    .takeWhile(_ != "X")
                    .foreach {
                      searchJournal =>
                        jrnlDgrGrph.collectNeighbors(searchMode).lookup((jrnlDgrGrph.vertices.filter {
                          journal => (journal._2.journalName.equals(searchJournal))
                        }.first)._1).map(journal => journal.sortWith(_._2.pr > _._2.pr).foreach(journal => println("Journal connected to the search in order of importance:"+journal._2.journalName+"\n:pagerank:"+journal._2.pr)))
                        println("Please enter the journal name or X to exit:")
                    }
                  jrnlDgrGrph.unpersist()
                }
                case _ => {
                  println("Invalid Input!")
                }
              }
              println("Please select one of the below options:")
              println("1 : To search for a Publication")
              println("2 : To search for a Journal")
              println("3 : To Exit!")
              print("Enter here:")
          }
      }

      //closing cached data
      publicationsRdd.unpersist()
      //    sc.stop()
      spark.close()
      Utils.prntHdngLne("CNA APP CLOSED")
    }
}
```

LISTING A.1: Execution Class

A.2 Journal Graph processing code

```
import java.util.Properties

import org.apache.spark.SparkContext
import org.apache.spark.graphx.{Edge, EdgeDirection, Graph, VertexId, VertexRDD}
import org.apache.spark.rdd.RDD

//  define the journal schema
case class Journal(
                    journalName: String,
                    publications: List[Publication]
                  )

```

```
13 //  define the journal graph schema with degrees
14 case class JournalWithDegrees(
15                                jid: VertexId,
16                                journalName: String,
17                                inDeg: Int,
18                                outDeg: Int,
19                                pr: Double
20                              )
21
22 object Journal {
23
24   def prfmChkJrnls(prop: Properties, sc: SparkContext, publicationsRdd:
     RDD[Publication], testPrintMode: String, taskMetrics: ch.cern.
     sparkmeasure.TaskMetrics): Unit = {
25
26     val jrnlDgrGrph = getJournalGraph(sc, publicationsRdd)
27     val tstJrnlSze = prop.getProperty("test.journal.size").toInt
28     val journalSamples = jrnlDgrGrph.vertices.sortBy(_._2.pr, false).
     take(tstJrnlSze)
29
30     //     println("printing the sample journal names:")
31     //     journalSamples.foreach(println)
32
33     println("Querying for top " + tstJrnlSze + " journals with highest
     pagerank:")
34     taskMetrics.runAndMeasure(if (testPrintMode.equals("true")) {
35       journalSamples.foreach {
36         searchJournal =>
37           val timedResult = Utils.time {
38             println("Retrieving influential journals for: " +
     searchJournal._2.journalName)
39             jrnlDgrGrph.collectNeighbors(EdgeDirection.In).lookup((
     jrnlDgrGrph.vertices.filter {
40               journal => (journal._2.journalName.equals(searchJournal._2
     .journalName))
41             }.first)._1).map(journal => journal.sortWith(_._2.pr > _._2.
     pr).foreach(journal => println(journal._2)))
42           }
43           //        println("Time taken :" +{timedResult.
     durationInNanoSeconds})
44           println(timedResult.durationInNanoSeconds.toMillis + ",")
45       }
46     } else {
47       journalSamples.foreach {
48         searchJournal =>
49           val timedResult = Utils.time {
50             jrnlDgrGrph.collectNeighbors(EdgeDirection.In).lookup((
     jrnlDgrGrph.vertices.filter {
51               journal => (journal._2.journalName.equals(searchJournal._2
     .journalName))
52             }.first)._1).map(journal => journal.sortWith(_._2.pr > _._2.
     pr))
53           }
54           //        println("Time taken :" +{timedResult.
     durationInNanoSeconds})
55           println(timedResult.durationInNanoSeconds.toMillis + ",")
56       }
57     })
58   }
59
60   def getJournalGraph(sc: SparkContext, publicationsRdd: RDD[Publication
     ]) = {
61
```

```
62      //    create journal RDD vertices with publications
63      val publicationsRdd_nonempty = publicationsRdd.filter(publication =>
         publication.journalName != "")
64      val publicationGroups = publicationsRdd_nonempty.groupBy(publication
         => publication.journalName)
65      val journalRDD: RDD[Journal] = publicationGroups.map(publication =>
        Journal(publication._1, publication._2.toList)).distinct
66
67      println("creating journal vertices...")
68      val journalWithIndex = journalRDD.zipWithIndex()
69      val journalVertices = journalWithIndex.map { case (k, v) => (v, k) }
70
71      val journalVertices2 = journalVertices.map {
72        case (k, v) => (k, v.journalName)
73      }
74
75      val journalPublicDict = sc.broadcast(journalVertices.flatMap {
76        case (jid, journal) =>
77          journal.publications.map(p => (p.id, jid))
78      }.collectAsMap())
79
80      println("journal vertices created!")
81
82      val nocitation = "nocitation"
83
84      val jrnlEdgsWthDplcts1 = journalVertices.flatMap {
85        case (jid, journal) =>
86          journal.publications.flatMap(
87            publication => publication.outCitations.map(
88              outCitation => (jid,
89                journalPublicDict.value.getOrElse(outCitation, -1.toLong)
90                , 1)))
91      }
92
93      val jrnlEdgsWthDplcts2 = jrnlEdgsWthDplcts1.filter(dupEdgs1 =>
        dupEdgs1._1 != dupEdgs1._2).filter(dupEdgs1 => dupEdgs1._2 != -1)
94
95      val zeroVal = 0
96      val addToCounts = (acc: Int, ele: Int) => (acc + ele)
97      val sumPartitionCounts = (acc1: Int, acc2: Int) => (acc1 + acc2)
98
99      println("creating journal edges...")
100     val journalEdges = jrnlEdgsWthDplcts2.map(dupEdgs2 => ((dupEdgs2._1,
         dupEdgs2._2), dupEdgs2._3)).aggregateByKey(0)(addToCounts,
        sumPartitionCounts).map(unqEdgs => Edge(unqEdgs._1._1, unqEdgs._1._2,
         unqEdgs._2))
101     println("journal edges created!")
102
103     val jrnlGrphWthotDgrs = Graph(journalVertices2, journalEdges,
        nocitation)
104
105     val inDegrees = jrnlGrphWthotDgrs.inDegrees
106     val outDegrees = jrnlGrphWthotDgrs.outDegrees
107     val pageRankTimed = Utils.time {
108       jrnlGrphWthotDgrs.pageRank(0.0001).vertices
109     }
110     println("Time taken to generate journal pageranks:"+pageRankTimed.
        durationInNanoSeconds.toMillis)
111     val pageRank = pageRankTimed.result
112
113     println("creating journal graph with degrees...")
114     //    creating journal graph with degrees
115     val JournalGraph = jrnlGrphWthotDgrs.mapVertices {
```

```
116       case (jid, jname) =>
117         JournalWithDegrees(jid, jname, 0, 0, 0.0)
118     }
119
120     val jrnlDgrGrph: Graph[JournalWithDegrees, Int] = JournalGraph.
    outerJoinVertices(inDegrees) {
121       (jid, j, inDegOpt) => JournalWithDegrees(jid, j.journalName,
    inDegOpt.getOrElse(0), j.outDeg, j.pr)
122     }.outerJoinVertices(outDegrees) {
123       (jid, j, outDegOpt) => JournalWithDegrees(jid, j.journalName, j.
    inDeg, outDegOpt.getOrElse(0), j.pr)
124     }.outerJoinVertices(pageRank) {
125       (jid, j, prOpt) => JournalWithDegrees(jid, j.journalName, j.inDeg,
     j.outDeg, prOpt.getOrElse(0))
126     }
127
128     prntJtnlGrphSmry(jrnlGrphWthotDgrs,inDegrees,outDegrees,pageRank)
129
130     println("journal graph with degrees and page rank added")
131     jrnlDgrGrph.cache()
132   }
133
134   //  method to print journal graph summary
135   def prntJtnlGrphSmry(jrnlGrphWthotDgrs: Graph[String, Int], inDegrees:
     VertexRDD[Int], outDegrees: VertexRDD[Int], pageRank: VertexRDD[
    Double]): Unit = {
136
137     val vrtcsCnt = jrnlGrphWthotDgrs.vertices.count
138     val edgsCnt = jrnlGrphWthotDgrs.edges.count
139     //     val inDegrees = pblctnGrphWthotDgr.inDegrees
140     //     val outDegrees = pblctnGrphWthotDgr.outDegrees
141     //     val maxInDegree = inDegrees.reduce((a, b) => (a._1, a._2 max b
    ._2))
142     //     val maxOutDegree = outDegrees.reduce(Utils.max)
143     //     val maxDegrees = pblctnDgrGrph.degrees.reduce(Utils.max)
144     //     val pageRank = pblctnGrphWthotDgr.pageRank(0.0001).vertices.
    distinct
145     val pageRankList = pageRank.map(_._2).distinct
146
147     Utils.prntSbHdngLne("Printing Journal Graph Summary")
148     println("No. of journals:" + vrtcsCnt)
149     println("No. of citations:" + edgsCnt)
150     println("No. of in degrees:" + inDegrees.count)
151     println("No. of out degrees:" + outDegrees.count)
152     //     println("Highest in degree vertex:" + maxInDegree)
153     //     println("Highest out degree vertex:" + maxOutDegree)
154     //     println("Highest degree vertex:" + maxDegrees)
155     println("Total unique page rank values found:" + pageRankList.count)
156     println("Maximum Page rank:" + pageRankList.max)
157     println("Minimum Page rank:" + pageRankList.min)
158     println("Printing page rank values:")
159     pageRankList.foreach(println)
160     Utils.prntSbHdngEndLne()
161   }
162 }
```

LISTING A.2: Journal Graph processing code

A.3 Publication Graph Processing code

```
1
2  import java.util.Properties
3
4  import org.apache.spark.SparkContext
5  import org.apache.spark.graphx._
6  import org.apache.spark.rdd.RDD
7
8  //  define the publication schema
9  case class Publication(
10                          entities: List[String],
11                          journalVolume: Option[String],
12                          journalPages: String,
13                          pmid: String,
14                          year: Option[Int],
15                          outCitations: List[String],
16                          s2Url: String,
17                          s2PdfUrl: String,
18                          id: String,
19                          authors: List[Authors],
20                          journalName: String,
21                          paperAbstract: String,
22                          inCitations: List[String],
23                          pdfUrls: List[String],
24                          title: String,
25                          doi: String,
26                          sources: List[String],
27                          doiUrl: String,
28                          venue: String)
29
30 //  define the publication graph schema with degrees
31 case class PublicationWithDegrees(
32                                     id: VertexId,
33                                     publicationName: String,
34                                     inDeg: Int,
35                                     outDeg: Int,
36                                     pr: Double
37                                   )
38
39 object Publication {
40
41   //define a method for performance check of publications
42   def prfmChkPblctns(prop: Properties, sc: SparkContext, publicationsRdd
       : RDD[Publication], testPrintMode: String, taskMetrics: ch.cern.
     sparkmeasure.TaskMetrics): Unit = {
43     val pblctnDgrGrph = getPublicationGraph(sc, publicationsRdd)
44     val tstPblctnSze = prop.getProperty("test.publication.size").toInt
45     val pblctnSmpls = pblctnDgrGrph.vertices.sortBy(_._2.pr, false).take
     (tstPblctnSze)
46
47     //    println("printing the sample publication names:")
48     //    journalSamples.foreach(println)
49
50     println("Querying for top " + tstPblctnSze + " publications with
     highest pagerank:")
51     taskMetrics.runAndMeasure(
52       if (testPrintMode.equals("true")) {
53         pblctnSmpls.foreach {
54           srchPblctn =>
55             val timedResult = Utils.time {
56               println("Retrieving influential publications for: " +
     srchPblctn._2.publicationName)
57               pblctnDgrGrph.collectNeighbors(EdgeDirection.In).lookup((
     pblctnDgrGrph.vertices.filter {
```

```
58                publication => (publication._2.publicationName.equals(
   srchPblctn._2.publicationName))
59              }.first)._1).map(publication => publication.sortWith(_._2.
   pr > _._2.pr).foreach(publication => println(publication._2)))
60            }
61            //        println("Time taken :" +{timedResult.
   durationInNanoSeconds})
62            println(timedResult.durationInNanoSeconds.toMillis + ",")
63        }
64      } else {
65        pblctnSmpls.foreach {
66          srchPblctn =>
67            val timedResult = Utils.time {
68              //            println("Retrieving influential journals for
   : " + srchPblctn._2.publicationName)
69              pblctnDgrGrph.collectNeighbors(EdgeDirection.In).lookup((
   pblctnDgrGrph.vertices.filter {
70                publication => (publication._2.publicationName.equals(
   srchPblctn._2.publicationName))
71              }.first)._1).map(publication => publication.sortWith(_._2.
   pr > _._2.pr))
72            }
73            //        println("Time taken :" +{timedResult.
   durationInNanoSeconds})
74            println(timedResult.durationInNanoSeconds.toMillis + ",")
75        }
76      }
77    )
78  }
79
80  //  define the method to get publication graph
81  def getPublicationGraph(sc: SparkContext, publicationsRdd: RDD[
    Publication]): Graph[PublicationWithDegrees, Int] = {
82
83    val pblctnsWthIndxRdd = publicationsRdd.sortBy(_.id).zipWithIndex().
    map { case (k, v) => (v, k) }
84    val pblctnIdDict = sc.broadcast(pblctnsWthIndxRdd.map(p => (p._2.id,
     p._1)).collectAsMap())
85
86    //    create publication RDD vertices with ID and Name
87    println("creating publication vertices...")
88    val publicationVertices: RDD[(Long, String)] = publicationsRdd.map(
    publication => (pblctnIdDict.value.getOrElse(publication.id, -1.
    toLong), publication.title)).distinct
89    println("publication vertices created!")
90
91    //     Defining a default vertex called nocitation
92    val nocitation = "nocitation"
93
94    println("creating citations...")
95    val ctnEdgsWthInvldEntrs = publicationsRdd.map(publication => ((
    pblctnIdDict.value.getOrElse(publication.id, -1.toLong), publication.
    outCitations), 1)).distinct
96    val citations = ctnEdgsWthInvldEntrs.filter(_._1._2.length != 0)
97    println("citations created!")
98
99    println("creating citation edges with outCitations and inCitations
    ...")
100   //    creating citation edges with outCitations and inCitations
101   val ctnEdgsExpnd = citations.flatMap {
102     case ((id, outCitations), num) =>
103       outCitations.map(outCitation => Edge(id, pblctnIdDict.value.
    getOrElse(outCitation, -1.toLong), num))
```

```
104     }
105     val citationEdges = ctnEdgsExpnd.filter(edge => edge.dstId != -1)
106     println("citation edges created!")
107
108     val pblctnGrphWthotDgr = Graph(publicationVertices, citationEdges,
        nocitation)
109
110     val inDegrees = pblctnGrphWthotDgr.inDegrees
111     val outDegrees = pblctnGrphWthotDgr.outDegrees
112
113     println("creating page rank...")
114     val pageRankTimed = Utils.time {
115       pblctnGrphWthotDgr.pageRank(0.0001).vertices
116     }
117     println("Time taken to generate publication pageranks:"+
        pageRankTimed.durationInNanoSeconds.toMillis)
118     val pageRank = pageRankTimed.result
119
120     println("page rank created!")
121
122     println("creating publication graph with degrees...")
123     //     creating publication graph with degrees
124     val publicationGraph = pblctnGrphWthotDgr.mapVertices {
125       case (pid, pname) =>
126         PublicationWithDegrees(pid, pname, 0, 0, 0.0)
127     }
128
129     println("Adding degrees and pagerank to graph")
130     val pblctnDgrGrph: Graph[PublicationWithDegrees, Int] =
        publicationGraph.outerJoinVertices(inDegrees) {
131       (pid, p, inDegOpt) => PublicationWithDegrees(pid, p.
        publicationName, inDegOpt.getOrElse(0), p.outDeg, p.pr)
132     }.outerJoinVertices(outDegrees) {
133       (pid, p, outDegOpt) => PublicationWithDegrees(pid, p.
        publicationName, p.inDeg, outDegOpt.getOrElse(0), p.pr)
134     }.outerJoinVertices(pageRank) {
135       (pid, p, prOpt) => PublicationWithDegrees(pid, p.publicationName,
        p.inDeg, p.outDeg, prOpt.getOrElse(0))
136     }.cache()
137     println("Graph with degrees and pagerank created")
138
139     //     get publication graph summary
140     prntPblctnGrphSmry(pblctnGrphWthotDgr, inDegrees, outDegrees,
        pageRank)
141
142     println("publication graph with degrees and page rank added")
143     pblctnDgrGrph
144   }
145
146   //  method to print publication graph summary
147   def prntPblctnGrphSmry(pblctnGrphWthotDgr: Graph[String, Int],
        inDegrees: VertexRDD[Int], outDegrees: VertexRDD[Int], pageRankRDD:
        RDD[(VertexId,Double)]): Unit = {
148
149     val vrtcsCnt = pblctnGrphWthotDgr.vertices.count
150     val edgsCnt = pblctnGrphWthotDgr.edges.count
151     //     val inDegrees = pblctnGrphWthotDgr.inDegrees
152     //     val outDegrees = pblctnGrphWthotDgr.outDegrees
153     //     val maxInDegree = inDegrees.reduce((a, b) => (a._1, a._2 max b
        ._2))
154     //     val maxOutDegree = outDegrees.reduce(Utils.max)
155     //     val maxDegrees = pblctnDgrGrph.degrees.reduce(Utils.max)
```

```
156     //     val pageRank = pblctnGrphWthotDgr.pageRank(0.0001).vertices.
        distinct
157
158     Utils.prntSbHdngLne("Printing Publication Graph Summary")
159     println("No. of publications:" + vrtcsCnt)
160     println("No. of citations:" + edgsCnt)
161     println("No. of in degrees:" + inDegrees.count)
162     println("No. of out degrees:" + outDegrees.count)
163     //     println("Highest in degree vertex:" + maxInDegree)
164     //     println("Highest out degree vertex:" + maxOutDegree)
165     //     println("Highest degree vertex:" + maxDegrees)
166
167     val pageRank = pageRankRDD.map(_._2).distinct
168
169     println("Total unique page rank values found:" + pageRank.count)
170     println("Maximum Page rank:" + pageRank.max)
171     println("Minimum Page rank:" + pageRank.min)
172     println("Printing page rank values:")
173     Utils.prntSbHdngEndLne()
174   }
175 }
```

LISTING A.3: Publication Graph processing code

A.4 Utils Code

```
1
2 import java.io.FileInputStream
3 import java.util.Properties
4
5 import org.apache.spark.graphx.{Graph, VertexId}
6 import org.apache.spark.sql.SparkSession
7
8 import scala.concurrent.duration._
9 import scala.reflect.ClassTag
10
11 case class TimedResult[R](result: R, durationInNanoSeconds:
     FiniteDuration)
12
13 object Utils {
14
15   def getProperties(): Properties = {
16     try {
17       val prop = new Properties()
18       if (System.getenv("PATH").contains("Windows")) {
19         prop.load(new FileInputStream("src/main/resources/application-
     local.properties"))
20       } else if (System.getenv("PATH").contains("ichec")) {
21         prop.load(new FileInputStream("src/main/resources/application-
     ichec.properties"))
22       } else {
23         println("Issue identifying the environment, PATH is:", System.
     getenv("PATH"))
24       }
25       prop
26     } catch {
27       case e: Exception =>
28         e.printStackTrace()
29         sys.exit(1)
30     }
```

```
31  }
32
33  def hex2dec(hex: String): BigInt = {
34    hex.toLowerCase().toList.map(
35      "0123456789abcdef".indexOf(_)).map(
36      BigInt(_)).reduceLeft( _ * 16 + _)
37  }
38  //  method to print task metrics
39  def prntGrphTstMtrcs(spark: SparkSession): Unit = {
40    println("Looking for metrics tables...")
41    spark.sql("show tables").show
42    println("Printing metrics table...")
43    spark.sql("select * from PerfTaskMetrics").show
44    println("Printing Top tasks based on Execution time and CPU time..."
      )
45    val perfTaskMetrics = spark.sql("select stageId, count(*) as count,
      sum(executorRunTime) as sum_exctrRnTme, sum(executorCpuTime) as
      sum_exctrCpuTme " +
46      "from PerfTaskMetrics " +
47      "group by stageId " +
48      "order by sum_exctrRnTme desc, sum_exctrCpuTme desc limit 1000")
49    perfTaskMetrics.show
50    //        perfTaskMetrics.repartition(1).write.format("com.
      databricks.spark.csv").save(prop.getProperty("test.task.metrics")) //
      commenting code to save the results to file
51  }
52
53  def prntHdngLne(heading: String): Unit = {
54
55    val headingUpper = heading.toUpperCase
56    val wordSize = headingUpper.length
57    val preSpace = " " * ((40 - wordSize - 6) / 2)
58    val postSpace = if (wordSize % 2 == 0) {
59      " " * ((40 - wordSize - 6) / 2)
60    } else {
61      " " * (((40 - wordSize - 6) / 2) + 1)
62    }
63    println("##############################################################
      ##################################################")
64    if (wordSize > 50) {
65      print("####################")
66    } else {
67      print("####################################")
68    }
69    print(preSpace)
70    print(s"-- $headingUpper --")
71    print(postSpace)
72    if (wordSize > 35) {
73      println("####################")
74    } else {
75      println("####################################")
76    }
77    println("##############################################################
      ##################################################")
78  }
79
80  def prntHdngEndLne(): Unit = {
81    println("##############################################################
      ##################################################")
82    println()
83  }
84
85  def prntSbHdngLne(subHeading: String): Unit = {
```

```
86
87      val headingUpper = subHeading.toUpperCase
88      val wordSize = headingUpper.length
89      val preSpace = " " * ((46 - wordSize - 6) / 2)
90      val postSpace = if (wordSize % 2 == 0) {
91        " " * ((46 - wordSize - 6) / 2)
92      } else {
93        " " * (((46 - wordSize - 6) / 2) + 1)
94      }
95      println("#################################################################
        ################")
96      if (wordSize > 40) {
97        print("###########")
98      } else {
99        print("###############")
100     }
101     print(preSpace)
102     print(s"-- $headingUpper --")
103     print(postSpace)
104     if (wordSize > 35) {
105       println("###########")
106     } else {
107       println("###############")
108     }
109     println("#################################################################
        ################")
110   }
111
112   def prntSbHdngEndLne(): Unit = {
113     println("#################################################################
        ################")
114     println()
115   }
116   // method to get highest degree vertex in graphs
117   def max(a: (VertexId, Int), b: (VertexId, Int)): (VertexId, Int) = {
118     if (a._2 > b._2) a else b
119   }
120
121   //  function to identify the time taken
122   def time[R](block: => R): TimedResult[R] = {
123     val t0 = System.nanoTime()
124     val result = block
125     val t1 = System.nanoTime()
126     val duration = t1 - t0
127     TimedResult(result, duration nanoseconds)
128   }
129
130   def drawGraph[VD: ClassTag, ED: ClassTag](g: Graph[VD, ED]) = {
131
132     val u = java.util.UUID.randomUUID
133     val v = g.vertices.collect.map(_._1)
134     println(
135       """%html
136 <div id='a""" + u +
137         """' style='width:960px; height:500px'></div>
138 <style>
139 .node circle { fill: gray; }
140 .node text { font: 10px sans-serif;
141      text-anchor: middle;
142      fill: white; }
143 line.link { stroke: gray;
144     stroke-width: 1.5px; }
145 </style>
```

```
<script src="//d3js.org/d3.v3.min.js"></script>
<script>
.var width = 960, height = 500;
var svg = d3.select("#a""" + u +
        """").append("svg")
.attr("width", width).attr("height", height);
var nodes = [""" + v.map("{id:" + _ + "}").mkString(",") +
        """];
var links = [""" + g.edges.collect.map(
        e => "{source:nodes[" + v.indexWhere(_ == e.srcId) +
          "],target:nodes[" +
          v.indexWhere(_ == e.dstId) + "]}").mkString(",") +
        """];
var link = svg.selectAll(".link").data(links);
link.enter().insert("line", ".node").attr("class", "link");
var node = svg.selectAll(".node").data(nodes);
var nodeEnter = node.enter().append("g").attr("class", "node")
nodeEnter.append("circle").attr("r", 8);
nodeEnter.append("text").attr("dy", "0.35em")
 .text(function(d) { return d.id; });
d3.layout.force().linkDistance(50).charge(-200).chargeDistance(300)
.friction(0.95).linkStrength(0.5).size([width, height])
.on("tick", function() {
link.attr("x1", function(d) { return d.source.x; })
  .attr("y1", function(d) { return d.source.y; })
  .attr("x2", function(d) { return d.target.x; })
  .attr("y2", function(d) { return d.target.y; });
node.attr("transform", function(d) {
return "translate(" + d.x + "," + d.y + ")";
});
}).nodes(nodes).links(links).start();
</script>
 """)
  }
}
```

LISTING A.4: Utility Code

A.5 Application Properties for Prod Environment

```

################################################################################################################################################
################################################LOCAL HOST#####################################################
################################################################################################################################################
### Environment details
env = prod

### File Input details
file.path = file:///ichec/home/users/rameshs999/PubCiteNetAnalysis/s2-corpus-*
sample.size = 1
repartition.size=4

### Test Mode
test.mode = true
test.entity.publication = true
test.entity.journal = true
```

```
17
18 ### Test Size
19 test.publication.size=500
20 test.journal.size = 500
21
22 ### Test Output
23 test.print.results = false
24 #test.task.metrics = file:///ichec/home/users/rameshs999/
       PubCiteNetAnalysis/s2-corpus-000/test.task.csv
25
26 ##########################################################################
       ################################################
```

LISTING A.5: Application Properties for Prod Environment

A.6 Application Properties for Dev Environment

```
1
2 ##########################################################################
       ################################################
3 ##########################################################LOCAL HOST#########
       ################################################
4 ##########################################################################
       ################################################
5 #### Environment details
6 #env = local
7 ##
8 #### File Input details
9 #file.path = file:///All_Items_Offline/Sem2/CS648_Project/full_data/s2-
       corpus-000
10 #sample.size = 0.1
11 #repartition.size = 4
12 #
13 #### Test Mode
14 #test.mode = true
15 #test.entity.publication = true
16 #test.entity.journal = true
17 #
18 #### Test Size
19 #test.publication.size = 500
20 #test.journal.size = 500
21 #
22 #### Test Output
23 #test.print.results = true
24 ##test.task.metrics = file:///All_Items_Offline/Sem2/CS648_Project/
       sample_data/s2-corpus-00/test.task.csv
25 ##test.stage.metrics = file:///All_Items_Offline/Sem2/CS648_Project/
       sample_data/s2-corpus-00/test.stage.csv
26
27 ##########################################################################
       ################################################
28 ##########################################################Testing#########
       ################################################
29 ##########################################################################
       ################################################
30 ### Environment details
31 env = local
32
33 ## Test Input details
```

```
file.path = file:///All_Items_Offline/Sem2/CS648_Project/sample_data/s2-corpus-00/unittestdata
sample.size = 1
repartition.size = 2

## Test Mode
test.mode = false
test.entity.publication = false
test.entity.journal = false

## Test Size
test.publication.size = 500
test.journal.size = 500

## Test Output
test.print.results = false

################################################################################################################
```

LISTING A.6: Application Properties for Dev Environment

A.7 SBT Build file

```

import sbt.util

name := "CNA"

version := "0.1"

scalaVersion := "2.11.12"

val sparkVersion = "2.4.3"

logLevel := util.Level.Debug

libraryDependencies ++= Seq(
  "org.apache.spark" %% "spark-core" % sparkVersion,
  "org.apache.spark" %% "spark-sql" % sparkVersion,
  "org.apache.spark" %% "spark-graphx" % sparkVersion,
  "net.liftweb" %% "lift-json" % "3.4.1"
)

libraryDependencies += "ch.cern.sparkmeasure" %% "spark-measure" % "0.16"

libraryDependencies += "org.apache.logging.log4j" %% "log4j-api-scala" % "11.0"

//libraryDependencies ++= Seq(
//  "org.scala-js" %%% "scalajs-dom" % "0.9.1",
//  "com.lihaoyi" %%% "scalatags" % "0.6.1"
//)

assemblyMergeStrategy in assembly := {
 case PathList("META-INF", xs @ _*) => MergeStrategy.discard
 case x => MergeStrategy.first
}
```

LISTING A.7: SBT build file

A.8 Code to Build the Spark Standalone Cluster

```
#!/bin/bash
#SBATCH --nodes=3
#  ntasks per node MUST be one, because multiple slaves per work doesn't
#  work well with slurm + spark in this script (they would need increasing
#  ports among other things)
#SBATCH --ntasks-per-node=1
#SBATCH --cpus-per-task=5
#SBATCH --mem-per-cpu=5G
#  Beware! $HOME will not be expanded and invalid paths will result Slurm jobs
#  hanging indefinitely with status CG (completing) when calling scancel!
#SBATCH --time=15:00:00
#SBATCH -A mucom001c
#SBATCH -p ProdQ

# This section will be run when started by sbatch
if [ "$1" != 'srunning' ]; then
    this=$0
    # I experienced problems with some nodes not finding the script:
    #   slurmstepd: execve(): /var/spool/slurm/job123/slurm_script:
    #   No such file or directory
    # that's why this script is being copied to a shared location to which
    # all nodes have access to:
    script=/ichec/work/mucom001c/${SLURM_JOBID}_$( basename -- "$0" )
        echo $script
    cp "$this" "$script"

    module load spark java

    export sparkLogs=/ichec/work/mucom001c/myspark/spark/logs
    export sparkTmp=/ichec/work/mucom001c/myspark/spark/tmp

    mkdir -p -- "$sparkLogs" "$sparkTmp"

    export SPARK_HOME=/ichec/packages/spark/2.3.3/kay/spark-2.3.3
    export SPARK_ROOT='/ichec/home/users/rameshs999/spark-2.3.3-bin-kay-spark'
    export SPARK_WORKER_DIR=$sparkLogs
    export SPARK_LOCAL_DIRS=$sparkLogs

    export SPARK_MASTER_PORT=7077
    export SPARK_MASTER_WEBUI_PORT=8080
    export SPARK_WORKER_CORES=$SLURM_CPUS_PER_TASK
    export SPARK_DAEMON_MEMORY=$(( $SLURM_MEM_PER_CPU * $SLURM_CPUS_PER_TASK / 2 ))m
    export SPARK_MEM=$SPARK_DAEMON_MEMORY

    srun "$script" 'srunning'
        # If run by srun, then decide by $SLURM_PROCID whether we are master or worker
else
    source "$SPARK_ROOT/sbin/spark-config.sh"
    source "$SPARK_ROOT/bin/load-spark-env.sh"
    if [ "$SLURM_PROCID" -eq 0 ]; then
        export SPARK_MASTER_HOST=$( hostname -I | awk '{print $1}' )
        export SPARK_LOCAL_IP=$( hostname -I | awk '{print $1}' )
```

```
53          MASTER_NODE=$( scontrol show hostname $SLURM_NODELIST | head -n
    1 )
54
55          # The saved IP address + port is necessary alter for submitting
    jobs
56          echo "spark://$SPARK_MASTER_HOST:$SPARK_MASTER_PORT" > "
    $sparkLogs/${SLURM_JOBID}_spark_master"
57          "$SPARK_ROOT/bin/spark-class" org.apache.spark.deploy.master.
    Master
58      else
59          # $(scontrol show hostname) is used to convert e.g. host20
    [39-40]
60          # to host2039 this step assumes that SLURM_PROCID=0 corresponds
    to
61          # the first node in SLURM_NODELIST !
62          MASTER_NODE=spark://$( scontrol show hostname $SLURM_NODELIST |
    head -n 1 ):7077
63          "$SPARK_ROOT/bin/spark-class" org.apache.spark.deploy.worker.
    Worker $MASTER_NODE
64      fi
65
66  fi
```

LISTING A.8: Code to Build the Spark Standalone Cluster

A.9 Code to trigger Spark Submit on the cluster

```
1  #!/bin/sh
2
3  #SBATCH --time=02:00:00
4  #SBATCH --nodes=1
5  #SBATCH -A mucom001c
6  #SBATCH -p ProdQ
7
8  cd CNA9/CNA
9  #sbt clean
10 #sbt update
11 #sbt assembly
12
13 /ichec/home/users/rameshs999/spark-2.3.3-bin-kay-spark/bin/spark-submit
       --driver-memory 140G --executor-memory 26G --total-executor-cores 110
        --num-executors 22 --executor-cores 5 --class "CitationParser" --
       master "spark://10.54.$1:7077" /ichec/home/users/rameshs999/CNA/CNA9/
       CNA/target/scala-2.11/CNA-assembly-0.1.jar
```

LISTING A.9: Code to trigger Spark Submit on the cluster