Efficient Academic Retrieval System Based on Aggregated Sources

Virginia Niculescu[a], Horea Greblă[b], Adrian Sterca[c] and Darius Bufnea[d]
Computer Science Department "Babeş-Bolyai" University, 1. M. Kogălniceanu, Cluj-Napoca, Romania

Keywords: Data Engineering, Retrieval Systems, Recommender Systems, Research Paper Databases, Academic Sources, Big-Data Processing, NLP, Apache Spark, Graph-Databases.

Abstract: On account of the extreme expansion of the scientific research paper databases, the usage of searching and recommender systems in this area increased, as they can help researchers find appropriate papers by searching in enormous indexed datasets. Depending on where the papers are published, there might be stricter policies that force the author to also add the needed metadata, but still there are other for which these metadata are not complete. As a result, many of the current solutions for searching and recommending papers are usually biased to a certain database.

This paper proposes a retrieval system that can overcome these problems by aggregating data from different databases in a dynamic and efficient way. Extracting data from different sources dynamically and not only statically, based on a certain database, is important for assuring a complete interrogation, but in the same time incur complex operations that may affect the performance of the system. The performance could be maintained by using carefully designed architecture that relies on tools that allow high level of parallelization.

The main original characteristic of the system is represented by the hybrid interrogation of static data (stored in databases) and dynamic data (obtained through real-time web interrogations).

1 INTRODUCTION

Recommender systems have a large number of applications in many fields including economic, education, and scientific research. On the other hand, *data engineering* is the practice of designing and building systems for collecting, storing, and analyzing data at scale. These systems collect, manage, and convert raw data into usable information for data scientists and business analysts to interpret. The goal of these systems is to make data accessible, and so they enable subsequent data analysis (Reis and Housley, 2022).

In academic research it is essential to be able to easily find the papers that treat specific themes in order to have a complete and correct view over the previous work. In addition, it is very useful to find experts and possible collaborators in specific domains, based on their published work.

Paper recommender systems aim to help researchers mitigate information overload and find relevant papers for their research, while author recommender systems can provide collaborators suggestions for researchers and help them find specialists in certain domains.

Nowadays, there are different scientific databases that index scientific papers such as Scopus, Web of Science, DBLP, Crossref, ACM Digital Library, IEEE Xplore, Semantic Scholar, Google Scholar, etc. Also, researchers may share their research findings and publications via digital platforms (e.g. arXiv, ResearchGate) for free for knowledge exchange (Sun et al., 2014). A complete search implies looking in many (if not all) these sources in order to identify all needed information. Many of the current solutions (e.g.: ConnectedPapers (Alex Tarnavsky et al., 2020), OpenCitations (Peroni and Shotton, 2020), SciGraph (SciGraph, 2022)) that index scientific papers are usually biased to a certain scientific database.

Given that the number of published papers grows exponentially, it's hard to keep track of the latest findings in a field of interest. For this reason, the metadata (i.e. – information that describes the resource: name, creator, description, date of creation, keywords) associated with them is crucial in a modern information system, to easily find the corresponding resources. Depending on where the papers are published, there might be strict policies that force the author to add

[a] https://orcid.org/0000-0002-9981-0139
[b] https://orcid.org/0000-0002-8529-5797
[c] https://orcid.org/0000-0002-5911-0269
[d] https://orcid.org/0000-0003-0935-3243

Niculescu, V., Greblă, H., Sterca, A. and Bufnea, D.
Efficient Academic Retrieval System Based on Aggregated Sources.
DOI: 10.5220/0011850600003464
In *Proceedings of the 18th International Conference on Evaluation of Novel Approaches to Software Engineering (ENASE 2023)*, pages 436-443
ISBN: 978-989-758-647-7; ISSN: 2184-4895

the needed metadata or not so strict.

Even though the metadata is missing, they are still a key part of an information retrieval system, and it would be useful if they could be automatically extracted if necessary.

There are ranked databases that allow publishing the work even in the manuscript form without even requiring important meta-fields, such as the subject, the abstract, the affiliations of the authors. Some of those can be automatically generated (subject, abstract), while others might be filled in through a process called *metadata harvesting* which aggregates the same resource from multiple sources, which might have those fields filled in, or have some additional information (Gill et al., 2008).

For any recommender or retrieval system it is essential to have fast access to a complete and up-to-date set of data from which it can extract the most appropriate solutions.

In addition, nowadays systematic literature reviews are more and more important for a solid research. A system that could easily extract all the relevant results of a particular domain is extremely useful.

Objectives and Contribution. The main objective of this research is investigate how to efficiently collect and aggregate academic papers from multiple sources, in order to obtain a *complete academic base* for academic recommendation. Through a complete academic base we understand an interrogation space (that may have static but also dynamic parts) of scientific papers with complete metadata information.

- A pure dynamic approach would imply dynamic searching (using web crawlers) on different sources, aggregation of the results and only temporary store these results.
- A pure static approach would imply to prior collect all the information from the remote sources and periodically update the locally stored global database. These implies not only very high initial costs, but also continuous costs required by the periodical updating.

We propose here a hybrid variant that stores internally data from one source (up to a certain date) and then uses a dynamic approach to obtain information from other sources. In order to demonstrate the proposed approach we developed a tool called ARRS (Aggregator Research Retrieval System) that allows the user to investigate a field of interest: see what papers have already been published on that topic, the authors that worked on a specific paper and all the other authors they collaborated with.

We assert that a graph database would be appropriate to be used for the system storage due to its characteristics that allow establishing the natural connections between the data (papers and authors). In addition, the storage is updated and supplemented through any usage of the system. The users of the system will be those that accomplish the maintenance.

Performance is another important issue that could be achieved by employing parallel computing for the dynamic search and database updating. For this reason, the metadata normalization process is done on an Apache Spark cluster (Haines, 2022; Apache Software Foundation, 2022).

Most of the open-source solutions (e.g. (Alex Tarnavsky et al., 2020), (Corporation for Digital Scholarship, 2006), (Peroni and Shotton, 2020), (SciGraph, 2022)) that allow a user to query papers on a specific topic have behind the scenes an indexed dataset which is used to deliver a response in real-time. Even though this aspect greatly increases responsiveness, this also means that the newest papers might not be present in this dataset. It is also the case for the tools that rely entirely on an external data source (e.g.: VOSviewer (Centre for Science and Technology Studies,)), which allows the user to build a network of papers dynamically, using API calls.

In contrast, the proposed approach besides using an indexed dataset stored in a graph database, it also interrogates multiple academic data sources to update the dataset while each user searches for the topics of interest. Thus, this hybrid approach allows responsiveness through the use of an indexed dataset, brings newly published papers using APIs for the platforms that provide such services, such as IEEE, Scopus and Crossref, and crawlers to also gather data from sources that don't have an available API.

In terms of efficiency, the ARRS was designed as a scalable solution, by using multiple parallelization techniques.

In the same time, the system represents a proof of concept for a general hybrid big data system that interrogates static data (stored in databases) and dynamic data (obtained through web interrogations).

Paper Structure. After this first section that gives a short introduction into the context, objectives and contribution, section 2 analyzes the related work in the field of academic recommender and search systems, with a special interest on freely web accessible tools. In section III the design of the proposed solution is described: the system requirements and functionalities, the architecture and some implementation challenges. The evaluation is done based on three criteria: usability, accuracy, and performance, and these are reported in section IV. Finally, the conclusions are drawn, and the future research directions that could be taken.

2 RELATED WORK

Recommender systems (also known as recommendation engines) are tools that offer useful item suggestions based on the user input or profile. Ideally, a recommender system provides recommendations automatically by inferring the needs from the user's item interactions. Alternatively, the recommender system asks users to specify their needs by providing a list of keywords or through some other methods. Even if by introducing keywords the system becomes more similar to a search engine or a retrieval system, it is estimated that about 80% of the recommender systems requires users to either explicitly provide keywords or to provide text snippets (Beel et al., 2016).

In this regard, recommender and retrieval systems are alike, meaning they both search for relevant items/documents to the user's query. Still, a recommender system can also provide a ranked list of suggestions by checking the importance of each resource found (Ricci et al., 2022) or additional information that could be discovered from the initial suggestion list.

The surveys (Beel et al., 2016; Bai et al., 2020) emphasize the following recommendation techniques as being the most appropriate approaches in the field of research-paper recommender systems: Stereotyping, Content-based Filtering, Collaborative Filtering, Co-Occurrence, Graph-based, Global Relevance, or Hybrid.

From the same surveys it may be noticed that more than half of the recommendation approaches applied content-based filtering (Pazzani and Billsus, 2007; Caragea et al., 2014), while collaborative filtering and graph-based (Gori and Pucci, 2006; Xia et al., 2016; Zhou et al., 2014) recommendations were applied each in around 15% of the approaches. In addition, most approaches neglected the user-modeling process and did not infer information automatically, but let users provide keywords, text snippets or a single paper as input. In this regard, they are much similar to retrieval systems. During development of our system, we have considered: content based, graph-based, global relevance and hybridization(Burke, 2007).

Following an analysis of the open-source solutions that tackle the problem of academic papers retrieval based on some keywords or specific queries, the next tools have been identified as representatives.
CitNetExplorer: is a tool developed at the Leiden University that can be used to visualize and analyze citation patterns of scientific publications. It uses various algorithms to detect the connected components, most relevant papers, shortest and longest paths. It also allows indirect citation relations to be visible on the graph, and supports direct import from Web of Science. Its main advantage is that it can handle large citation networks (millions of papers and ten times more relations) (van Eck and Waltman, 2014).
VOSviewer: is another tool from the Leiden University that is used to build and visualize bibliometric networks. The networks can contain many types of publications and the relations between the nodes can use the citation, bibliographic coupling or co-authorship. This tool also allows to build and visualize networks that are related to a specific query. It can download data from Web of Science, Scopus, Dimensions, Lens and PubMed. Through the APIs provided by Crossref, Semantic Scholar, OpenCitations and WikiData, it can build co-authorship and co-occurrence networks (Centre for Science and Technology Studies,). Even though the VOSviewer tool might have full subscriptions to all the APIs above, depending on the required size of the network, querying those databases through the API might not be very suitable, due to the high latency.
ConnectedPapers: is a visual tool, very easy to use, that provides a graph overview of the academic landscape on a specific topic. It's mainly used to discover relevant prior work on the subject of interest and to create bibliographies for research papers.
Unlike the previous solutions, ConnectedPapers doesn't use a citation tree. It builds a graph in which the papers are organized depending on how similar they are. As a result, even if the papers don't necessarily cite one another, they can still be highly related and put into the graph close to each other. The metric used for determining the similarity uses co-citations and bibliographic coupling, which means that if two papers have similar references, they're most likely related. When building the graph, ConnectedPapers searches through approximately 50,000 papers, groups similar papers in clusters, while highlighting the popular papers (highly cited) with larger nodes, and recent papers with darker colors (Alex Tarnavsky et al., 2020). The only limitation is the fact that the queried papers are retrieved from a single database, namely the Semantic Scholar Paper Corpus. Besides having this single point of failure, the papers might not be relevant for criteria such as geographic location.
Zotero: was first proposed in 2006; this is an open-source tool which is used to gather, arrange, and analyze research papers. The user can manage a personal library, as well as generating bibliographies, citations, and reports (Corporation for Digital Scholarship, 2006). Zotero can be seen more like a research paper manager rather than a recommender system, because although the user can supply RSS feeds and

other data sources, this implies that the user has to introduce them manually.

OpenCitations: is a tool for open scholarship that tackles the publication of open citation data in the form of Linked Open Data. This means that it builds an open source citation index. It currently contains information about approximately 14 million citations. The main goal of this tool is to make scholarly bibliographic and citation data largely available to any user (Peroni and Shotton, 2020). Though it provides a huge index of research papers, this tool is more like a database, rather than a recommender system, but it can be well used as an input for such a software tool.

SciGraph: is a knowledge graph from Springer, which offers linked open data from the following aggregates sources: Springer Nature, Digital Science and Unsilo. This aggregated data generates a rich semantic abstract of how various pieces of information are related. It currently can hold between 1.5 and 2 billion relations between various types of papers, such as journals, articles, books, and conferences (SciGraph, 2022). SciGraph's dataset is highly biased on the indexes provided by their partners. They plan to extend their dataset to bring more highly qualitative data.

3 ARRS –FUNCTIONALITY AND DESIGN

The proposed system – ARRS – uses aggregated academic sources in order to extract the relevant papers on a topic specified through a keyword based interrogation. By choosing a resulted paper, its authors together with their collaborators are extracted and shown using connected graphs that emphasize also the degree of connectivity.

3.1 System Functionality

Papers query.

The keyword interrogation can be given in a simple implicit way as an enumeration of words (when we assume an OR operator between them), but also in more complex way using operators AND, OR, and NOT.

An example of interrogation is:

"software bot"
OR robotic AND software
OR "robotic process automation"
OR "software robot" NOT "cognitive automation"

The result of such query is a graph of papers that is also graphically represented on the interface. The edge between two nodes in this papers graph represents one or more common keywords from the query, the more common keywords, the wider the edge (each edge has an associated weight).

Authors Network. After the relevant papers are shown, the user may select a specific paper, and then a graph of the corresponding authors and also all their collaborators will be shown (a collaborator is an author that is co-author to at least one paper). In the authors' graph, an edge means co-authorship on at least one paper.

Author's Papers. On the authors' graph, if the user selects any author node (which can be an author corresponding to the initial selected paper, or a collaborator), the system will display the papers published by that author, as well as their metadata. This allows us to inspect on what fields those researchers worked on, in general, not only the fields described by the initial query.

3.2 Data Normalisation

Behind the scene, there is a very important process: data normalization. The collected data come from different sources with various representation schemas and they should be transformed in order to have a common representation of the associated metadata (e.g.: title, authors, authors' affiliations, subject, publisher, publish year, abstract, DOI, type of article). If for example the keywords are missing, ARRS will try to extract keywords from the abstract (if available), or from the title. The collected data are then used to update the database, either by adding missing information for certain papers or just adding new ones.

The system uses NLP (Natural Language Processing) in the searching process (the roots of the keywords are extracted using a stemming algorithm) and in the normalisation process, too. For example, in order to extract keywords from an abstract or from a title, NLP algorithms have been enrolled.

3.3 Architecture, Design and Implementation Challenges

Since we propose the creation of a system that can gather data from multiple sources, normalize them on a cluster, and aggregate them into a graph database, this implies a very high level of complexity, and so a hierarchical decomposition through modularisation is needed.

In order to provide a high level of scalability, ARRS was designed as a scalable solution, by using multiple parallelization techniques.

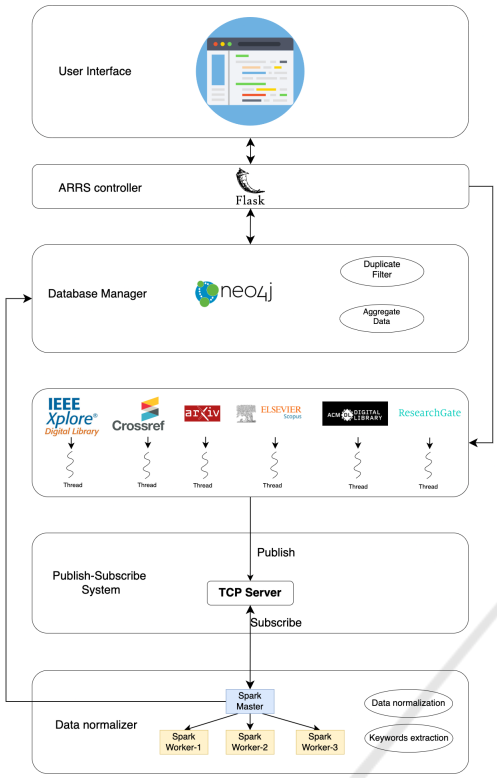

Figure 1: Architecture of the ARSS Retrieval System.

We have considered Apache Spark[1] in order to improve the performance of the normalization process – Spark has been chosen as being appropriate for efficient streaming computation. (Many other recommender systems use the machine learning component – Apache ML[2] for generating recommendations; this could be added in a further development for AI recommendations.) The Sparck cluster uses a load balancer to distribute the workload to the worker nodes.

The architecture of the system is depicted in Figure 1.

For each emphasised component there were specific implementation challenges, and we emphasise the most important of them in what it follows. Many of these challenges were surmounted by using appropriate specialized frameworks, but putting all these to work together represented also a challenge.

Spark Streaming Component. The Spark component handles the data normalization. Once the data normalization and keywords extraction is done, the result is inserted into the graph database.

The Spark component accepts data as a socket text stream and first has to deserialize the data and apply a flatMap operation to ensure the result is a single collection, instead of a collection of collections (each operation that processes the data is executed on the Spark worker nodes). Then, after obtaining the deserialized items, a transform operation is triggered on all the RDDs (Resilient Distributed Datasets) which is then chained with a map operation per RDD; this map operation calls the function that normalizes the items. After the processing is done, foreach operations are used to call the database manager function that inserts or updates the items.

Publish-Subscribe System. Due to the fact that the results obtained from the crawlers are yielded at different times, there is a need to transform them into a stream, such that they can be processed when they are available. To achieve this, a custom publish-subscribe system based on TCP sockets was created that allows the Web Sources Aggregator to send its data, as soon as it becomes available, to the TCP server, which will then forward it to its subscribers, meaning the Spark system that builds a stream from the data received through the TCP socket.

Web Sources Aggregator. This component has the responsibility of retrieving papers from a large variety of data sources, such as Crossref, IEEE, ACM, Scopus, ResearchGate, arXiv, in an efficient way, using thread pool executors (employing thread pool executors was essential in order to assure a good performance). This component also allows the use of Web crawlers that simulate the user interaction with a browser to get the data, and a specialised framework – Selenium[3] – was used for this purpose.

The Web crawling process is triggered each time an user enters a new query. Still, even parallelised, crawling on the spot still takes tens of seconds, which hinders the responsiveness of the system. To overcome this, priorities for each dataset were assigned: highest priority for the database, medium priority for API calls (since it's near instantaneously), and lowest priority for the crawling. These priorities determine what data will be first shown on the graph, which will be regenerated when a new dataset is available.

Graph Database Manager. Considering the many-to-many relationship between the papers and their authors, as well as the complex join operations that are in the dataset, the chosen graph database was neo4j[4]. Initially, this database is populated with data from Crossref that contains around 120 million papers. Still, this contains only the work published until 2021, which means that the system also has to query more recent papers by crawling multiple databases, such as IEEE, ACM, Scopus, arXiv, and ResearchGate. Each query produces an update of the database by updat-

[1]https://spark.apache.org
[2]https://spark.apache.org/mllib/
[3]https://www.selenium.dev
[4]https://neo4j.com

ing some missing metadata of some papers (if they were found on the web sources) and by adding the new papers that were found. This way the database is dynamically and implicitly maintained through the usage of the system by the users.

User Interface. The entries selected from the graph database based on the user's query are displayed on the interface, as lists of papers and authors, as well as graphical representations of the created graphs. When the user enters a query, ARRS will first match the papers from the database based on three meta-fields: the title, the associated keywords, and the abstract. The result of an interrogation is updated after the web sources interrogation is finalized.

In addition to these main components, we emphasize the following two modules:

Network Analysis Module. The network analysis module was introduced as a part of ARRS controller and uses the framework networkx in order to compute degree, betweenness, closeness and eigenvector centralities, as well as some metrics such as the diameter, the average shortest path length and the density of the graph. It is important to note that all the graphs in the ARRS use weighted edges, to better highlight what the most important nodes are. Their weight is computed as the number of common keywords between two nodes in the papers graph, and the number of collaborations between nodes in the authors graph (in this context, a collaboration means that two authors wrote a paper together).

NLP Processing Module. During normalisation process, the extraction of keywords from the abstract or the title was based on NLP by using RAKE[5] – an algorithm that can extract keywords from individual documents by splitting the text into a vector depending on a list of delimiters. This vector is then divided into continuous sequences of words using the stop words as delimiters. The roots of the words are computed using a stemming function based on the Porter algorithm (Porter, 2006) to get to the root of the word (e.g.: 'running' becomes 'run'). This was used to allow the same word to match multiple documents which might have otherwise been excluded because they used another form of the word.

Normalization functions were created for each data source.

NLP processing is also used for achieving a uniform style for the DOI (e.g.: only the ID, not the full URL) and build the item following a chosen schema.

Cluster of Docker Containers. The system uses a cluster of Docker containers[6]. The container images used are similar and we wrote an image specification for each of them. Since there are multiple containers that need to be started in a specific order, and they require a custom configuration, docker-compose was used to handle the creation of the containers. Figure 2 shows the cluster of Docker containers and how they interact.

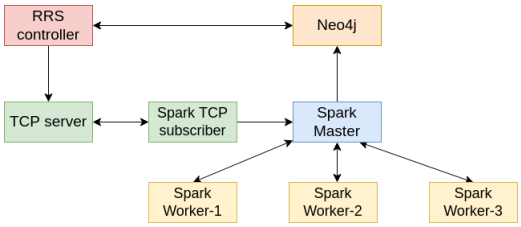

Figure 2: Docker containers cluster.

Heuristic Optimization. Since the crawling process is a slow one, depending on the data source and whether it needed to simulate a browser interaction with Selenium or not, the process can take anywhere between half a minute and a few hours. For this reason, the number of entries brought back by the crawlers is limited based on the factors above, to some parameters that were heuristically estimated. Furthermore, each data source first provides the most relevant papers to the user's query. This ensures that the impact of this limitation is minimal, and guarantees an acceptable response time of the ARRS.

4 ARRS – EVALUATION

The evaluation of the proposed solution is addressed from the software quality attributes, with a special interest on the following three perspectives: usability, accuracy, and performance. For evaluation a main test case has been considered for the following simple query: *"machine learning text summarization"*

4.1 Usability

The user interface was inspired from the one provided by ConnectedPapers(Alex Tarnavsky et al., 2020), providing a base layout with a search bar on the top, two side columns that show information about the papers, and a center section that displays an interactive graph.

The authors graph is shown when a node in the papers graph is selected (double click). The graph contains the authors of the selected papers, as well as their peers (other researchers with whom they collaborated). The user can further inspect the papers published by each author, and see the metadata corresponding to the each resulted papers. In the papers' graph, there are edges that are wider than the rest. The

[5]RAKE (Rapid Automatic Keyword Extractor)
[6]https://www.docker.com/

weight of an edge that links two papers depends on the number of matches between their keywords.

The responsiveness of the system is assured by obtaining the results in steps, based on some priorities (as described in the previous section), and by caching them in memory.

4.2 Accuracy

Even though ARRS is similar to ConnectedPapers tool in terms of UI, the way the network of papers is built is entirely different. ConnectedPapers starts, for example, from a user query, then it asks the user to select the most relevant papers, based on which it will build the graph, whereas the ARRS uses the user's query to directly build the graph. Overall, this allows a better overview of the research published for the topic of interest, given that another important purpose of the system is to allow the user to check the authors graph to look for possible collaborators.

When comparing the results obtained from the ConnectedPapers and ARRS, the first one yielded 40 papers, while the proposed solution provided over 100. In terms of papers matching between the two results, this greatly varies depending on the paper that the ConnectedPaper user chose to build the graph. As a consequence, they cannot be directly compared, as in most cases, ARRS will yield an entirely different dataset. However, in terms of searching for future collaborators, ARRS proved to offer a better overview of the literature, which gives the user the chance to inspect the most relevant papers to the query, based on which edge is wider (meaning it has more keywords matching with the ones in the interrogation).

4.3 Performance

The parallelisation implemented through the thread pool executor for crawling and using Spark for normalisation, significantly improves the obtained performance. Several experiments have been conducted on to emphasize the importance of parallel computation introduced in the system.

In terms of hardware specification, the system was tested on two virtual machines, each with 16 CPU cores and 64GB RAM. The storage used was 1TB.

In terms of performance obtained for crawling, for the main test query the parallel approach yielded the results in 29.678 seconds, while the sequential one (without the executor thread pool) provided the results in 6 minutes and 9.018 seconds. So, the speedup ($speedup = sequentialTime/parallelTime$) is for this process 12.434.

Regarding the performance of the normalization process, the sequential execution took 32.304 seconds, while the execution on the Spark cluster only took 2.461 seconds, resulting in a speedup of 13.126.

Table 1: ARRS Performance Analysis. T_s denotes the sequential execution time in seconds, and T_p denotes the parallel execution time in seconds.

Query	T_s	T_p	Speedup
abstractive text summarization	222.788	33.624	6.626
abstractive AND text AND summarization NOT extractive	252.619	26.792	9.429
state-of-the-art nlp techniques for data cleaning	207.006	33.966	6.095
GAN networks	245.739	32.214	7.628
human face generation using GAN networks	243.652	31.967	7.622
text to image translation	253.977	31.681	8.016
clothing AND translation NOT "GAN network"	215.248	28.517	7.548
automatic metadata generator	248.436	31.389	7.915
"data encryption" AND "data decryption"	255.079	30.650	8.322
blockchain technology in the banking industry	246.831	30.251	8.159

In order to evaluate the performance in more detail, we have done other several experiments besides that main test case – for different queries, and the results are shown in Table 1. In this table, T_s denotes the sequential execution time in seconds, and T_p denotes the parallel execution time in seconds, and the speedup obtained through parallelization is emphasized. From these, an analysis of average performance could be extracted:

- Average sequential time: 239.127 sec.
- Average parallel time: 30.215 sec.
- Average speedup: 7.736.

5 CONCLUSIONS

We proposed an Aggregator Researcher Retrieval System - ARRS, as a hybrid system that retrieves data from a large variety of data sources, such as

Crossref, IEEE, ACM, Scopus, ResearchGate, and arXiv, through crawlers and APIs. The system aggregates the duplicate papers by merging and fills in their missing metadata when possible (keywords extraction, metadata harvesting).

Through parallelization using thread pool executors, the crawling process has been highly improved, which greatly improved the responsiveness of the application. The normalization process was run on a Spark cluster deployed on Docker containers. ARRS system provides a solid base which can be further improved by scaling and aggregating data from more data sources.

The system we proposed could represent the data engineering component of a more complex recommender system that adds an AI data analysis component that extracts the best recommendations that fit the user profile. This is the reason of using the term 'retrieval system', even if besides the papers extraction and aggregation, the system also provides the possibility to extract the network of authors and their collaborators.

On a larger view, this system could be considered a proof of concept for a general efficient approach in building and updating big databases from different web sources. In many cases, creating and managing a database that aggregates information from different sources is difficult and implies considerable cost. This dynamic approach of updating and supplementing a database by using a tool that offers the possibility to search for the information stored into the database and in associated sources, could be an efficient and productive solution.

ACKNOWLEDGEMENTS

The implementation of the ARRS system started in 2021 with the dissertation thesis of the student O. Oprisan, from the master specialization *High Performance Computing and Big Data Analytics* of "Babeş-Bolyai" University. The thesis was under the coordination of dr. Virginia Niculescu.

REFERENCES

Alex Tarnavsky, E., Eddie, S., Itay Knaan, H., and Sahar, P. (2020). Connected Papers - Find and explore academic papers. https://www.connectedpapers.com/. Accessed 10.05.2022.

Apache Software Foundation (2022). Apache Spark - Unified Engine for large-scale data analytics. https://spark.apache.org. Accessed 5.05.2022.

Bai, X., Wang, M., Lee, I., Yang, Z., Kong, X., and Xia, F. (2020). Scientific paper recommendation: A survey.

Beel, J., Gipp, B., Langer, S., and Breitinger, C. (2016). Research-paper recommender systems: A literature survey. *Int. J. Digit. Libr.*, 17(4):305–338.

Burke, R. (2007). *Hybrid Web Recommender Systems*, pages 377–408. Springer Berlin Heidelberg.

Caragea, C., Bulgarov, F. A., Godea, A., and Das Gollapalli, S. (2014). Citation-enhanced keyphrase extraction from research papers: A supervised approach. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1435–1446. Assoc. for Comp. Linguistics.

Centre for Science and Technology Studies (). VOSviewer - Visualizing scientific landscapes. https://www.vosviewer.com. Accessed 10.05.2022.

Corporation for Digital Scholarship (2006). Zotero - Your personal research assistant. https://www.zotero.org. Accessed 10.05.2022.

Gill, T., Gilliland, A. J., Whalen, M., and Woodley, M. S. (2008). *Introduction to Metadata*. Getty Publications.

Gori, M. and Pucci, A. (2006). Research paper recommender systems: A random-walk based approach. In *2006 IEEE/WIC/ACM International Conference on Web Intelligence (WI 2006 Main Conference Proceedings)(WI'06)*, pages 778–781.

Haines, S. (2022). *Modern Data Engineering with Apache Spark: A Hands-On Guide for Building Mission-Critical Streaming Applications*. Apress.

Pazzani, M. J. and Billsus, D. (2007). *Content-Based Recommendation Systems*, pages 325–341. Springer Berlin Heidelberg, Berlin, Heidelberg.

Peroni, S. and Shotton, D. (2020). OpenCitations, an infrastructure organization for open scholarship. *Quantitative Science Studies*, 1(1):428–444.

Porter, M. (2006). The Porter stemming algorithm. https://tartarus.org/martin/PorterStemmer/.

Reis, J. and Housley, M. (2022). *Fundamentals of Data Engineering*. O'Reilly Media.

Ricci, F., Rokach, L., and Shapira, B. (2022). *Recommender Systems Handbook*. Springer, 3rd ed. 2022 edition.

SciGraph (2022). SciGraph - A Linked Open Data platform for the scholarly domain. https://www.springernature.com/gp/researchers/scigraph. Accessed 10.05.2022.

Sun, J., Ma, J., Liu, Z., and Miao, Y. (2014). Leveraging Content and Connections for Scientific Article Recommendation in Social Computing Contexts. *The Computer Journal*, 57(9):1331–1342.

van Eck, N. J. and Waltman, L. (2014). CitNetExplorer: A new software tool for analyzing and visualizing citation networks. *Journal of Informetrics*, 8(4):802–823.

Xia, F., Liu, H., Lee, I., and Cao, L. (2016). Scientific article recommendation: Exploiting common author relations and historical preferences. *IEEE Transactions on Big Data*, 2(2):101–112.

Zhou, Q., Chen, X., and Chen, C. (2014). Authoritative scholarly paper recommendation based on paper communities. In *2014 IEEE 17th International Conference on Computational Science and Engineering*, pages 1536–1540.