



Big Data Management

April 2024

Assignment P1

Rana İşlek

Berat Furkan Koçak

Maria Camila Salazar

Universitat Politècnica de Catalunya

Big Data Management and Analytics

1. What is scholarIA?

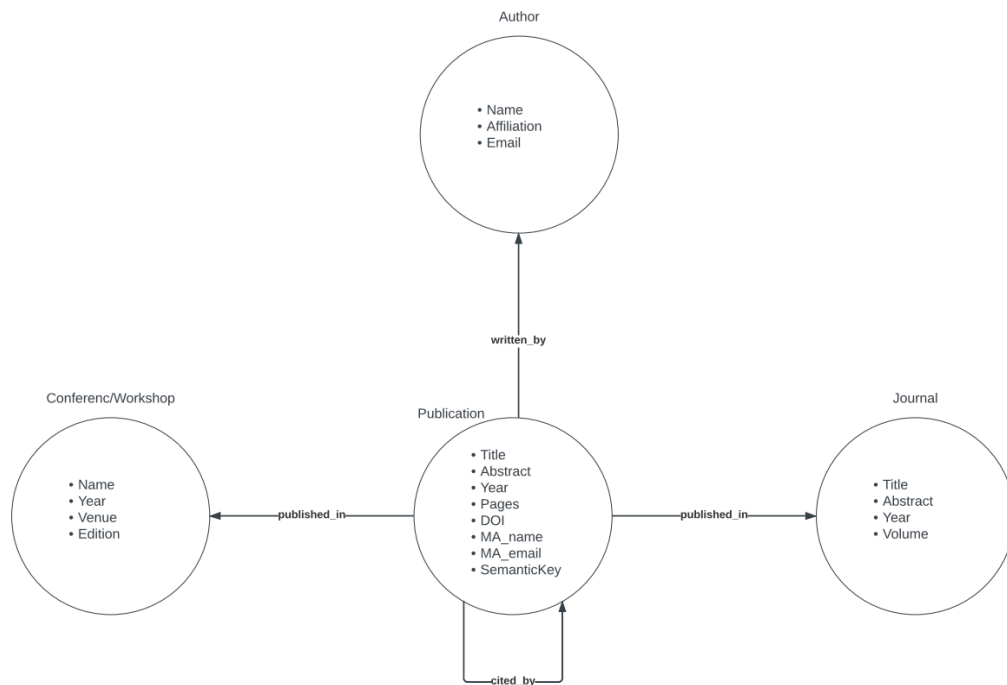
ScholarIA is a web-based intelligent application supported by artificial intelligence that can revolutionize the research process. From the customer 's perspective, they simply upload all of their research papers, and the application will sort them in the most organized and efficient way. It will assist you in the reading process, track where you left off in each paper, and provide automated categorizations, such as by author, field, or journal. You will also be able to see summaries of papers while scrolling down to each paper and easily reach other related papers in your field of interest thanks to a recommendation system. This can save you time and effort, help you stay organized and on top of your reading, and easily find the information you need.

The main purpose of this is helping academic people and entities, like students and researchers, universities, research laboratories and training institutes to simplify the research journey providing the best experience through the *ScholarIA* ecosystem.

2. Modeling approach and technology

2.1. Graph Model

As our project requires categorization and recommendation systems based on highly connected academic paper data, we approach this problem with a graph database design. This enables us to efficiently represent relationships between each academic entity and traversal queries. Below is the graph schema of the provisional database.

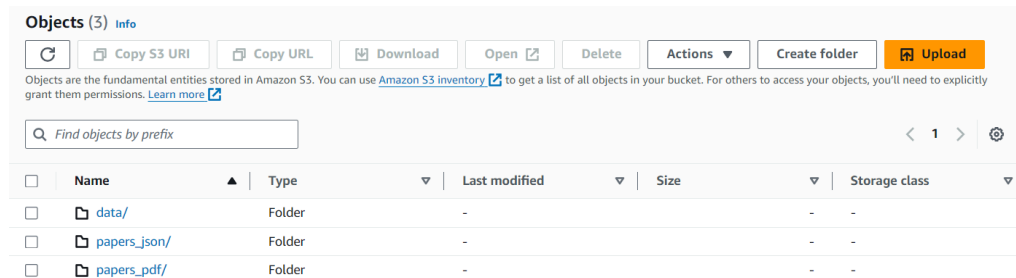


2.2. Technology Stack

2.2.1 AWS S3 - Data Lake

S3 is a cloud storage solution provided by major cloud service *AWS*. Datalakes require additional features such as catalog management on top of a storage place. *AWS* manages such advancements through its other services such as *AWS Glue*. However, our previous experiences combined with our research showed that *AWS Glue* can lead to uncontrollable pricing [1]. In that regard for the first iteration of the project, we will keep S3 as is; however, we provide our action plan to convert it to a *lakehouse* for scalability purposes in the section 2.2.3. We chose to utilize *AWS S3* compared to other storage options because of the following reasons:

- **Scalability:** As we want our platform to respond to all user requests in a fast and efficient manner, we need both a scalable and accessible storage. In this regard, choosing the most known cloud storage is the best way to go.
- **Cost effectiveness:** *AWS S3* provides 5GB free usage for 12 months as well as comes with a pricing equal to other cloud storage services with 0.023 \$ / GB.
- **Integration:** *AWS S3* seamlessly integrates with both other *AWS* services as well as 3rd party open-source data lakehouse solutions such as Starburst and Dremip.



Objects (3) Info

Copy S3 URI Copy URL Download Open Delete Actions Create folder Upload

Objects are the fundamental entities stored in Amazon S3. You can use [Amazon S3 inventory](#) to get a list of all objects in your bucket. For others to access your objects, you'll need to explicitly grant them permissions. [Learn more](#)

Find objects by prefix

<input type="checkbox"/>	Name	Type	Last modified	Size	Storage class
<input type="checkbox"/>	data/	Folder	-	-	-
<input type="checkbox"/>	papers_json/	Folder	-	-	-
<input type="checkbox"/>	papers_pdf/	Folder	-	-	-

2.2.2 Neo4j - Graph DBMS

Neo4j is a leading graph database management system renowned for its ability to efficiently store and query highly connected data. We opted for *Neo4j* as our graph DBMS for the following reasons:

- **Graph Data Model:** By leveraging *Neo4j's* graph capabilities, we can develop sophisticated recommendations that leverage the inherent structure of our dataset.
- **Scalability and Performance:** Our previous experience with graph database benchmarks showed us that *Neo4j* both scales well with increasing scale factor as well as providing the best user experience compared to other graph databases [2].
- **Flexibility and Extensibility:** *Neo4j's* flexible data model and schema-less architecture provide the agility we need to adapt to evolving project requirements. It also enables us to easily inject representations from csv files we produce.

2.2.3 Additions

Starburst: As we previously mentioned, for scalability and performance purposes, we plan to convert our *S3* data lake into a lakehouse using open-source solution *Starburst*.

Apache Airflow: For the incremental updates, we will utilize *Airflow* to create automated data pipelines that will retrieve the new paper data from *Semantic Scholar API* and directly upload it to our data lake.

2.3. Data sources

2.3.1 Semantic Scholar

Semantic Scholar's *Graph API* provides a free and on-demand source of data about authors, papers, citations, venues, and more utilizing its AI-powered academic search engine with nearly 200 million papers covering all disciplines [3]. The choice of *Semantic Scholar* as our main data source for this project can be justified with several reasons:

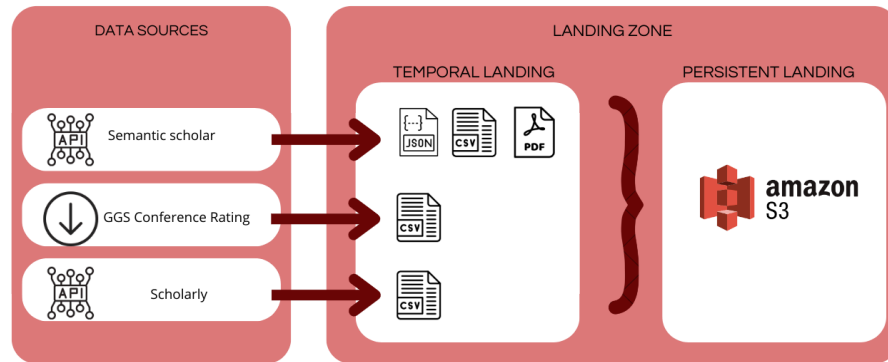
- **Semantic Embeddings:** One of the most important features of *Semantic Scholar* is that they are providing *SPECTER2 embeddings* of papers, enabling us to provide advanced paper recommendations given the semantic intersection of uploaded papers.
- **Comprehensive Coverage:** *Semantic Scholar* provides a rich and diverse dataset for our project, especially matching the expectations arising from our graph database design.
- **Structured Data:** *Semantic Scholar* provides structured metadata for academic papers, including information such as authors, affiliations, citations, abstracts, and keywords.
- **API Accessibility:** *Semantic Scholar* provides both historical and incremental based api call, enabling automated data retrieval and integration into our data lake infrastructure which aligns well especially with our aim of providing notifications with new papers.

2.3.2 Scholarly

For our secondary data source, we decided to use *Scholarly*, a script-based data collection model, leveraging the scholarly Python library to programmatically access and retrieve academic information from Google Scholar [4]. This model supports ScholarIA's aim to streamline the research process by automating the organization and categorization of academic papers.

Similarly to *Semantic Scholar*, from this API we retrieved different features of authors and papers such as author's name, affiliation, email, interests, citations and indexes; publication's name, published year, citations. But, the strongest advantage of this datasource is its querying capabilities. Since *Semantic Scholar API*'s paper search by keyword parameter does not work stable enough and throws errors on server side, we preferred to use *Scholarly* instead and could query by author, keyword or title successfully.

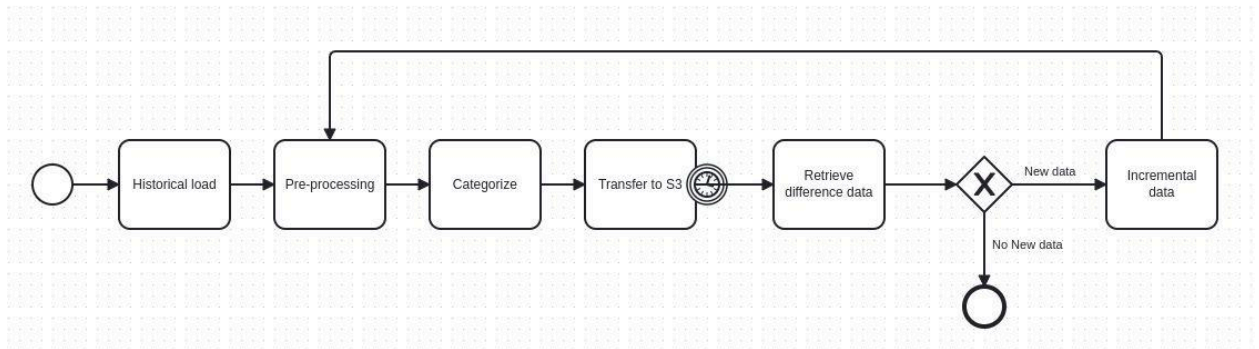
3. Data analysis backbone



- Scholarly API** At this phase, the data analysis backbone of *ScholarIA* is primarily structured around data collection and basic processing. The use of Python, alongside the *Scholarly* library, forms the core of this backbone, enabling efficient retrieval of academic papers and author information.
- Semantic Scholar API:** *Semantic Scholar* is providing most of the core attributes needed by our graph database through its Python API.
- GGS Conference Rating:** It is important for researchers to find papers that are published in respected conferences. We use this organization's report in order to enrich our dataset.

4. Data process

The data process involves sequential steps to establish a comprehensive database for *ScholarIA*.



4.1 Data Collection

So far we used *Scholarly API* and *Semantic Scholar* as data sources. We are collecting data for 2 top fields under 5 different work fields such as: Computer science (Machine learning, Artificial intelligence), Chemistry (Organic, Physical), Biology (Genetics, Ecology), Business (Management, Marketing) and Law (Criminal, International).

SCHOLARIA

Your long term academic assistant.

- Our *Scholarly API* script queries Google Scholar for information on authors and their publications across selected fields, aiming to retrieve data for up to 10 authors per field.
- On the *Semantic Scholar* side, we utilize preprocessing scripts to both retrieve and convert the raw data into semantically structured files.

4.2 Data Structuring

Semantic Scholar data is structured to 3 file formats:

- **CSV files:** These contain node and edge information of graph model in relational sense.
- **JSON files:** These contain all the paper id-title information we are using.
- **PDF files:** Retrieved through http requests for papers with open access.

Scholarly data is structured into CSV files for each field, detailing authors and their top publications. This step lays the groundwork for creating a searchable, organized database.

4.3 Data Selection Design Decisions

As the S3 offers only 5GB of free storage in its free tier, we had to make some decisions on the amount of data we are collecting. Not only storage but also computational power and api request limitations were effective in the following decisions. For example, only collecting the author data via authors.py preprocessing script took almost 7 hours to complete.

- We are collecting 100 paper data on each of the 10 top fields.
- We are collecting each graph entity in a separate csv file.
- We are collecting the pdfs of all papers that are available.
- We are only collecting papers having at least 10 citations. This is to ensure graph representation will not be too scarce. As there is no statistics on distribution of citations for each study field, it is hard to come up with a specific number of min citations.

Github link: <https://github.com/furkanbk/BDM-P1>

References

- [1] <https://github.com/furkanbk/TPC-DI>
- [2] <https://github.com/furkanbk/AdvancedDatabases-GraphDB>
- [3] <https://github.com/danielnsilva/semanticscholar/tree/master/semanticscholar>
- [4] <https://github.com/scholarly-python-package/scholarly>