Large Scale Web Crawling and Distributed Search Engines: Techniques, Challenges, Current Trends, and Future Prospects

***Abstract*—The heart of any substantial search engine is a crawler. A crawler is simply a program that collects web pages by following links from one web page to the next. Due to our complete dependence on search engines for finding information and insights into every aspect of human endeavors, we tend to overlook the enormous complexities of today's search engines powered by web crawlers to index and aggregate everything found on the internet. The sheer scale and technological innovation that enabled the vast body of knowledge on the internet to be indexed and easily accessible upon queries is constantly evolving. In this paper, we look at the current state of the massive apparatus of crawling the internet, specifically focusing on deep web crawling given the explosion of information behind an interface that cannot be simply extracted from raw text. We also explore distributed search engines and the way forward for finding information in the age of large language models like ChatGPT or Bard. Finally, we present the design of a new asynchronous crawler that can be used to extract information from any domain into a structured format.**

***Index Terms*—web crawling, crawler, distributed systems, search engines, deep web crawling, common crawl, large language models, asynchronous crawler**

I. Introduction

The technology behind web crawling has come a long way since Google was first introduced in the late 1990s. According to the original PageRank paper [1], Google employed a distributed web crawler architecture to download content from hundreds of millions of web pages. Since then the technology behind web crawlers has grown ever more complex. The underlying crawlers, whether collecting raw text from web pages for any well-known and established search engines or aggregating structured information for domain-specific tasks, now have to deal with the enormous variety of the web. The regular web, part of the internet that is accessible through various search engines, is what most of us are familiar with in our day-to-day life. But due to the sudden explosion of various web technologies, most of the data on the internet now sits behind various search forms that can only be accessed using search queries. For crawlers, this adds another layer of complexity where simple link traversing is no longer enough. Along with the current technological advancements in the field of large-scale web crawling, we will also focus on deep web crawling where crawlers need to interact with various web forms to access information. We will look at the current status of distributed search engines and the challenges it faces in the era of big data. In order to take into consideration the disruptive AI conversational agents like ChatGPT and Bard, and to understand their implications on how people search for information on the web, we will explore how these agents can upend the status quo of current search engines. Finally, we are proposing a new asynchronous web crawling technique that can be employed by small to medium organizations for domain-agnostic crawling needs that require data to be extracted from web pages with similar content architectures and stored in specific formats.

The *Introduction* part describes the motivation and overall targets of the paper. In the *Related Work* section, we will explore previous survey works related to web crawling techniques and various architectures of distributed search engines. In the next section, *Crawling*, we will take a detailed look at various crawling techniques currently in use as well as different scaling issues that crawlers need to deal with. In the *Distributed Search Engine* section, we will go through various architectural challenges and prospects of distributed search engines. We will explore open-source search engines such as Apache Lucene, and ElasticSearch. In the section, *AI Conversational Agents*, we will discuss the implications of ChatGPT and other conversational AI agents for the domain of search engines and how we acquire information from the web. The section, *Domain-Agnostic Asynchronous Crawler* proposes a new asynchronous crawler suitable for crawling sites of any domain, all of which share a common architecture. Finally, we will conclude with the utility and prospects of crawling in the age of AI and generated content.

II. Related Work

A systematic literature review of over 1488 articles is conducted in [3] on web crawling. Various crawling techniques such as a universal crawler, topical crawler, and hidden or deep web crawler are described here. The authors highlight the challenges of large-scale web crawling and a set of policies that web crawlers should adhere to. Paper [4] discusses ways of efficiently crawling board sites and how they perform against traditional graph crawling techniques. The authors introduce a type of preferential crawler, Board Forum Crawling, specifically designed for crawling forum sites efficiently. A comprehensive study of deep web crawling techniques currently in use is explained in [2]. The authors also highlight the lack of standards and evaluation metrics to measure the performance of deep web crawlers against some common datasets. The authors of [5], describe the challenges and learnings of a high-performance web crawler, Mercator,

that is distributed, scalable, and extensible. In [6], the authors propose a distributed search engine based on a cooperative model, where local search engines reduce the update interval time as compared to a centralized search engine. A Hadoop-based distributed search engine is proposed in [7]. Due to the ever-evolving and constantly growing size of the internet, a distributed search engine is proposed to reduce query time. A detailed explanation of various crawlers as well as a comparison among crawlers in terms of applicability, usability, and scalability can be found in [9].

III. Crawling

The basic components of a web crawler are - a list of URLs to start crawling with, an aggregator that collects web pages using the URLs from the initial list, and a parser that parses the content of web pages and adds new URLs in the initial list. The crawler keeps repeating this process until the crawling list is empty or some other thresholds are achieved [3]. In their seminal paper [1], introducing the Google search engine and ground-breaking PageRank algorithm, Brin and Page described the powerful crawling component that enabled the algorithm to generate astonishingly accurate and relevant results against users' queries. From hundreds of millions of web pages, when they first started, Google now contains a search index of hundreds of billions of web pages that powers Google Search [8].

A. Types of Crawlers

Based on the scope and type of information collected during crawling, the crawlers can be divided into multiple categories, such as universal crawlers, topical crawlers, forum crawlers, and hidden or deep web crawlers [3]. A universal or broad crawler, as the name suggests, crawls everything. It visits every web page, extracts links from the page, and visits those pages in turn. Topical crawlers can be thought of as domain or topic-specific crawlers. These types of crawlers crawl only the web pages of certain domains or topics. In order to determine the type of web page that it encounters, topical crawlers use various machine learning algorithms to classify web pages of interest [3]. In order to collect data from the hidden parts of the internet, a new type of crawler has emerged, called hidden or deep web crawlers. These crawlers need to generate queries for the search interfaces in order to acquire information buried in various databases and storage systems. Running multiple instances of the crawler in a distributed architecture to minimize traffic load and scale horizontally is a required attribute of modern web crawlers [10].

B. Techniques

In a stand-alone crawler module, also known as centralized crawling, a single instance performs all required steps of crawling. While this approach is easy to implement and maintain; given the sheer scale of data, this approach is only suitable for simple use cases [11]. In a distributed system, each instance of the crawler is a complete crawling module equipped with the necessary tools to perform the end-to-end crawling task. Depending on how these individual modules are managed and run, distributed crawling can be divided into two categories - master-worker and peer-to-peer crawling. The hybrid architecture of crawling tries to combine the simplicity of centralized crawling with the scalability of distributed crawling. In this approach, URL queue management is centralized, and content fetching from the URLs is distributed across many instances [11]. In the master-worker mode of distributed crawling, a master node performs the job of an orchestrator, where crawling tasks are managed and assigned to worker nodes to perform the actual crawling tasks. The master node manages the global URL list of pages to visit and assigns URLs to each crawler instance [10]. Each crawler keeps a local DNS cache to minimize the DNS lookup which is a major source of bottleneck for crawlers [1]. The master node can perform load balancing to avoid overwhelming crawler instances. In a peer-to-peer architecture, crawler instances independently discover and visit web pages without a master node. Some disadvantages of this approach include - a lack of load balancing where one crawler may be downloading a huge number of pages, whereas other crawlers may not have a sufficient number of URLs to crawl data from.

C. Distributed Crawling Architectures

Some recent implementations of distributed crawling use various open-source crawling frameworks and combine those with the power of cloud services to build robust and scalable systems. Some of these approaches are described below:

Scrapy and Redis: Scrapy is an open-source crawling and scraping framework. Redis is an efficient, in-memory, and key-value data store that is used to manage the message queues used to assign tasks to crawler instances. The Scrapy-Redis distributed component can be used to efficiently crawl sites with semi-structured information. [10] [12].

Container clustering: This approach uses a cluster of docker containers that host the crawler instances. Kubernetes is used to orchestrate the clusters of distributed containers. Apache Kafka is used as the medium of communication for the crawling instances [10].

Apache Nutch: Apache Nutch is an open-source, powerful, highly scalable, and configurable web crawler that can be customized in various ways to handle all sorts of web pages found on the internet. It uses Hadoop for data processing.

Apache Spark: Apache Spark is a highly scalable data processing framework that can process large datasets efficiently and quickly. The ability of Spark to orchestrate data processing on extremely large datasets makes it a good candidate for large-scale crawling. A control node manages child crawling nodes, distributes crawling tasks to nodes, and keeps the crawled information up-to-date to prevent redundant crawling by child nodes [13].

D. Common Crawl

Common Crawl is an initiative where monthly crawled data is made accessible for the public to conduct research and analytical tasks. It publishes the crawled data using Amazon

S3. It is stated that datasets from March-April, 2023 archive contains around 3 billion web pages from 34 million domains [14]. Common Crawl uses Apache Nutch to run its web crawler in a distributed system. It uses MapReduce to find potential crawling targets from the collected data. Datasets crawled by Common Crawl have been used for training the GPT-3 large language model [15].

E. Challenges & Prospects

As the size of the internet keeps growing and web technologies are transforming at a rapid pace, web crawlers which are at the heart of information gathering must adapt to new challenges. Some of the challenges facing modern web crawlers include,

Scale and resource utilization: As the number of web pages is increasing at a rapid speed, crawlers need to balance between visiting new and relevant web pages and updating the index for existing crawled pages.

Hidden web crawling: As more and more websites adopt dynamic and user-friendly content that relies heavily on client-side scripting, it presents a new challenge for traditional crawlers which are used to follow hyperlinks and fetch the content from those links. Identifying search panels and interfaces that can be used to crawl the deep web is a major challenge in deep web crawling.

Lack of standards: Due to the lack of standards and agreed-upon policies, servers containing the web pages that are being crawled are at the mercy of the crawlers. Aggressive crawling on a server can lead to poor performance for the real users of that particular server.

Despite these major challenges, many optimization techniques can be applied to increase the efficiency of the crawlers. To identify more effective search panels or user interfaces for deep web crawling, advanced machine learning models can be applied to infer the usability of search interfaces or forms. In deep web crawling, AI can be used to generate better queries to extract more data with few queries. Emerging cloud services could be used to make distributed crawling more efficient and cost-effective.

IV. Distributed Search Engine

In a centralized search engine architecture, every component of the search engine, starting from the crawling, indexing, storage, and all the way to generating results based on users' queries is controlled by a central server. The search engine providers control the indexes of the web as well as the ranking algorithms that present users with the most relevant results per their algorithms. To democratize the web and create a community-built search engine, the concept of distributed or peer-to-peer search engines has emerged [18].

A. Architectures

In a distributed or decentralized search engine, there is no single node that controls the different components of the search engine, namely crawling, indexing, storing data, and ranking search results. The responsibility of all these components is shared among multiple nodes. For distributed search engines to be useful in the real world, these constraints must be met in an efficient, scalable, and cost-effective way - the quality of the ranking of search results, low latency response, and a considerable amount of web pages being available in the index which powers the search results [20]

A P2P system such as Gnutella, provides a simple keyword search mechanism that broadcasts the users' search queries over the entire overlay network. This approach is quite slow even with a limited document size. To solve this issue, Distributed hash Table or DHT is introduced to store document links against some IDs [19]. To avoid unpredictable and unstable computing resources of end-users, a network of web servers can be used to index local documents and store network address cache to reduce routing overhead [21]. YaCy provides the implementation of a peer-to-peer distributed search engine where individual nodes take part in crawling and updating DHT.

B. Open-source Search Engines

Apache Lucene: Lucene is a high-performance indexing and search engine library that uses an inverted index for searching. Its features include document indexing, full-text searching, and ranked searching where the most relevant results appear on top. Many websites use Lucene to implement their own search engines. It indexes text documents and then upon query, generates ranked search results from the indexed content [16].
Elasticsearch: Elasticsearch is another open-source distributed analytics and search engine built on top of Lucene [17]. It provides REST APIs to index documents and then searches relevant documents using highly configurable and advanced queries. Due to its distributed architecture using clustering of nodes, Elasticsearch can scale horizontally and rebalance indexes as necessary. It also provides features for automatic node recovery in case of failure. Elasticsearch has the ability to infer the schema type of fields from documents during indexing [17].

C. Challenges & Prospects

Current implementations of P2P text-based searching using DHT works efficiently for a fraction of the actual size of the web. Two main issues regarding the decentralized search engines are, available storage on individual nodes considering the ever-expanding nature of the internet, and constraints on network bandwidth used during full-text searching on a P2P network [19]. Another challenge is to reduce the response time of search queries, given that multiple nodes with indexed data need to be queried before a result can be sent to the users. Since there is no central server controlling the addition of new nodes, some adversarial entities can manipulate the crawled index and ranking of search results [18]. Recent advancements in blockchain technologies can be used to optimize different aspects of a distributed search engine, such as crawling, indexing, and storage [18]. Extensive research is also needed to prevent attacks on the P2P system from adversarial players.

V. AI Conversational Agents

Since it was published in November 2022, ChatGPT has become one of the most popular sites on the web in just a couple of months [22]. ChatGPT is based on GPT-3, a large language model [15] trained on petabytes of data to produce human-like text. It can be used for a variety of tasks, such as conversational agents, summarizing text, correcting grammar, explaining technical topics, generating functional code, and many more.

A. Challenges to Traditional Search Engines

People have been using ChatGPT to not only generate text but as an interactive search engine to find out information. While search engines simply return a list of web pages, these AI conversational agents provide information in a way that humans are more comfortable with. Moreover, the models do not have access to the most recent data since their training and also cannot provide all sources that played a role in generating certain content. The appeal of these conversational agents in finding information on the web stems from the fact that search engines cannot combine information from multiple sources and then aggregate it to produce a coherent and factually correct answer, whereas these agents provide answers to questions like another human expert would do [23]. Due to the lack of reference materials for generated content, it is more difficult to ascertain the authenticity of generated content by these models.

B. Societal & Ethical Impacts

Since these models are trained on human-generated text in the first place, they may have inherent biases due to the training datasets. Producing disinformation will be much easier with human-like text and this will exacerbate the already fragile social and political divides across the world. Various professional roles in the domain of content generation, whether article writers or programmers, will be transformed significantly. Prompt engineering, in other words, providing the correct starting sequence of words to generate the best possible output will be a key skill in the coming days. Exams and evaluation criteria need to be rethought in light of the ubiquity of these language models.

C. Future Prospects

A key area of research that needs to take place is to retain the sources of information generated by the language models and provide them as references to the users. Since people will be using these AI agents for finding information on the web, further improvements can be introduced to incorporate recent events in the generated content along with references. Much more attention should be given to de-bias the training datasets as well as shielding the models from being tricked into generating harmful and dangerous content.

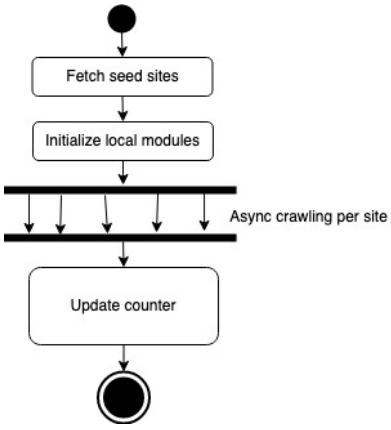

Fig. 1. Orchestrator activity diagram

VI. Domain-Agnostic Asynchronous Crawler

Traditional approaches of crawling fetch raw page data and index them for future use during users' searching queries. But what if the target of crawling is to generate structured data from unstructured raw content, sourced not only from a handful of similar sites rather each site will have a completely different architecture. In order to create an open-source crawling engine that can handle a multitude of sites, all having different content types and architectures, we have designed a domain-agnostic asynchronous crawler.

A. Architecture

Here we will describe the architecture of the asynchronous crawler in detail. The crawler is written entirely in Python. For parsing HTML content, we have used beautifulsoup4. Asynchronous API calls have been implemented using asyncio and aiohttp. We have used Amazon DynamoDB as our storage service. Communications with AWS services from the crawler are performed through the boto3 library. The crawler has four major components: Orchestrator, BaseCrawler, SiteHandler, and Utility. Asynchronous crawling is implemented at four different levels, namely at the root level, site level, page level, and item level. *Orchestrator*: Orchestrator is the starting point of the crawling engine. First, the list of seed URLs, which describes the starting point for every site that needs to be crawled. After loading the seed URLs, the orchestrator dynamically imports site handlers from the local directory. Each handler is imported as a separate module and gets mapped to the corresponding site. Finally, the orchestrator creates separate asynchronous crawling tasks for each site and initiates them all at once (see Figure 1). Once all the crawling tasks have finished, the orchestrator logs the total duration and exits the program.

BaseCrawler: BaseCrawler is the root class that defines the structure of every SiteHandler. For each page, the site crawler creates an asynchronous task and initiates them all at once (see Figure 2). Inside each page handler, all the items that are available on that page are extracted first. The page handler then creates an asynchronous task for each item and initiates them

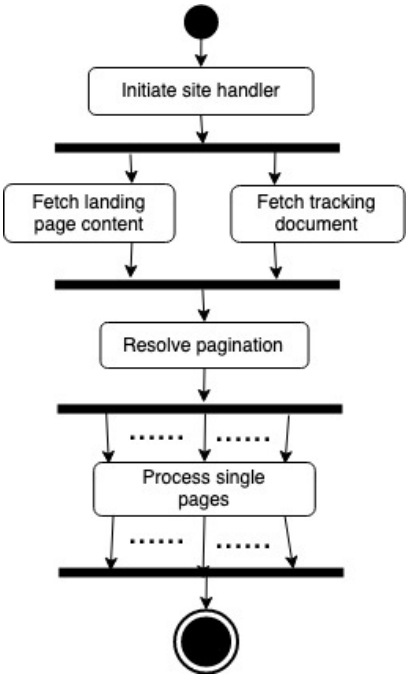

Fig. 2. Site handler activity diagram

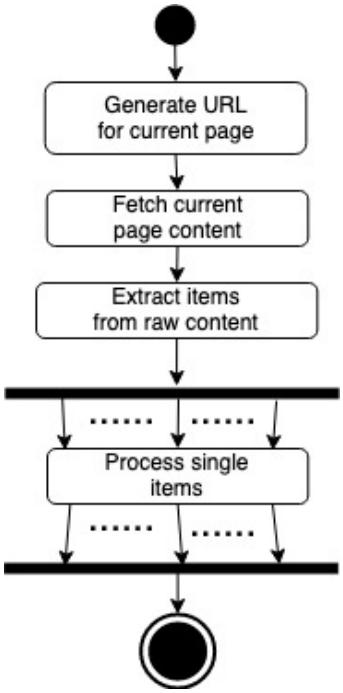

Fig. 3. Page handler activity diagram

all at once (see Figure 3). Each item handler initializes the item depending on the site. Then all the required information is collected in an asynchronous process (see Figure **??**). Finally, the item is stored on Amazon DynamoDB. Once the processing of all the items on a page is done, each page handler returns to the site handler. Similarly, once the processing of all the pages in a site is done, each site handler returns to the orchestrator.

SiteHandler: For every site in the seed URL list, there is a corresponding SiteHandler module created to provide site-specific functionalities. Each SiteHandler contains two components - item schema for items in this site and implementation of abstract methods from BaseCrawler.

Utility: This module contains all the utility functions and services such as the base model for all custom item models, methods to fetch and upload documents to DynamoDB, functions to process raw page content, and generic functions for asynchronous I/O.

B. Experiment

We ran the crawler in two modes - once in synchronous mode and the other in asynchronous mode in a single thread using asynchronous I/O. We used two seed URLs. Each URL was from a popular e-commerce site. One of the sites had 167 pages and the other had 33 pages. For the synchronous version of the crawling, it took around 382 minutes to complete the crawling of all 200 pages with dozens of items on each page. Whereas for the asynchronous crawler, the elapsed time was 79 minutes.

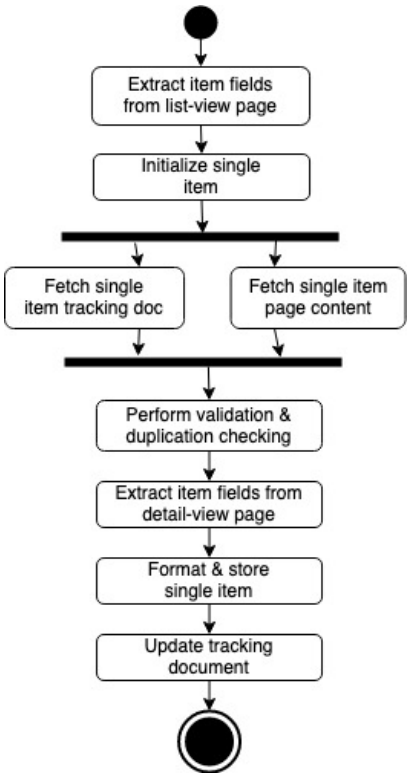

Fig. 4. Item handler activity diagram

C. Usage, Scopes, and Limitations

This asynchronous crawler can be used to crawl data from sites of any domain that contain data in a list-view and details-view structure. Even more, using the custom SiteHandler instances, the same asynchronous crawler can create documents of different schemas without needing to change anything in the base crawler module. For small and medium crawling tasks that require structured datasets generation from sites of a particular domain, this asynchronous crawler can set up a custom SiteHandler and generate structured datasets within hours rather than days. There are some limitations of the current version of the crawler that we plan to improve in future versions. Currently, the crawler does not keep extensive log records for API calls and crawling errors. Instead of manually checking and running the crawler for failed sites or items, a better approach would be to include an auto-healing mechanism where the crawler uses exponential backoff to retry crawling problematic sites before raising an alarm. The current version of the crawler does not include any mechanism to handle rate-limiting from servers.

VII. Conclusion

In this paper, we have described techniques for large-scale crawling that are currently being used in various crawling applications. We have explored different distributed crawling architectures and the challenges and future directions of large-scale crawling in the age of big data. Then, we looked at the current status of distributed or peer-to-peer search engines. We also highlighted the scope of improvement for distributed searching. We explored the impact of large language models on traditional search engines. Finally, we described the proposed domain-agnostic asynchronous crawler in detail and showed the performance improvement of a simple single-threaded asynchronous crawler over its synchronous version by crawling 200 pages from two popular e-commerce sites. Crawling will always be a part of our web experience. Without crawling, we will not be able to find the right content in the ocean of raw data. In order to overcome the upcoming challenges due to the vastness of the internet, crawlers need to constantly invent new ways of aggregating information from the ever-changing web.

References

[1] Brin, S., & Page, L. (1998). The anatomy of a large-scale hypertextual web search engine. Computer networks and ISDN systems, 30(1-7), 107-117.

[2] Hernández, I., Rivero, C. R., & Ruiz, D. (2019). Deep Web crawling: a survey. World Wide Web, 22, 1577-1610.

[3] Kumar, M., Bhatia, R., & Rattan, D. (2017). A survey of Web crawlers for information retrieval. Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery, 7(6), e1218.

[4] Guo, Y., Li, K., Zhang, K., & Zhang, G. (2006, December). Board forum crawling: a Web crawling method for Web forum. In 2006 IEEE/WIC/ACM International Conference on Web Intelligence (WI 2006 Main Conference Proceedings)(WI'06) (pp. 745-748). IEEE.

[5] Najork, M., & Heydon, A. (2002). High-performance web crawling (pp. 25-45). Springer US.

[6] Sato, N., Uehara, M., Sakai, Y., & Mori, H. (2001, April). Distributed information retrieval by using cooperative meta search engines. In Proceedings 21st International Conference on Distributed Computing Systems Workshops (pp. 345-350). IEEE.

[7] Ling, L., Fu, Y., Ma, X., Zhang, H., & Zhang, Y. (2013, August). The realization of the distributed search engine on cloud platform. In Proceedings of 2013 2nd International Conference on Measurement, Information and Control (Vol. 1, pp. 691-695). IEEE.

[8] Google Search. (2023). How Google Search organizes information. Retrieved May 10, 2023, from https://www.google.com/search/howsearchworks/how-search-works/organizing-information/

[9] Deshmukh, S., & Vishwakarma, K. (2021, May). A Survey on Crawlers used in developing Search Engine. In 2021 5th International Conference on Intelligent Computing and Control Systems (ICICCS) (pp. 1446-1452). IEEE.

[10] Ren, X., Wang, H., & Dai, D. (2020, December). A Summary of Research on Web Data Acquisition Methods Based on Distributed Crawler. In 2020 IEEE 6th International Conference on Computer and Communications (ICCC) (pp. 1682-1688). IEEE.

[11] Koloveas, P., Chantzios, T., Tryfonopoulos, C., & Skiadopoulos, S. (2019, July). A crawler architecture for harvesting the clear, social, and dark web for IoT-related cyber-threat intelligence. In 2019 IEEE World Congress on Services (SERVICES) (Vol. 2642, pp. 3-8). IEEE.

[12] Yin, F., He, X., & Liu, Z. (2018, December). Research on scrapy-based distributed crawler system for crawling semi-structure information at high speed. In 2018 IEEE 4th International Conference on Computer and Communications (ICCC) (pp. 1356-1359). IEEE.

[13] Liu, F., & Xin, W. (2020, May). Implementation of Distributed Crawler System Based on Spark for Massive Data Mining. In 2020 5th International Conference on Computer and Communication Systems (ICCCS) (pp. 482-485). IEEE.

[14] Nagel, S. (2023, April 6). March/April 2023 crawl archive now available. Common Crawl. Retrieved May 10, 2023, from https://commoncrawl.org/2023/04/mar-apr-2023-crawl-archive-now-available/

[15] Brown, T., Mann, B., Ryder, N., Subbiah, M., Kaplan, J. D., Dhariwal, P., ... & Amodei, D. (2020). Language models are few-shot learners. Advances in neural information processing systems, 33, 1877-1901.

[16] Białecki, A., Muir, R., Ingersoll, G., & Imagination, L. (2012, August). Apache lucene 4. In SIGIR 2012 workshop on open source information retrieval (p. 17).

[17] Elasticsearch. (2023). What is Elasticsearch?. Retrieved May 10, 2023, from https://www.elastic.co/what-is/elasticsearch

[18] Srednyak, S. (2020). Knowledge Coin. Retrieved May 10, 2023, from https://rorur.com/kcoin2.pdf

[19] Li, J., Loo, B. T., Hellerstein, J. M., Kaashoek, M. F., Karger, D. R., & Morris, R. (2003). On the feasibility of peer-to-peer web indexing and search. In Peer-to-Peer Systems II: Second International Workshop, IPTPS 2003, Berkeley, CA, USA, February 21-22, 2003. Revised Papers 2 (pp. 207-215). Springer Berlin Heidelberg.

[20] Baeza-Yates, R. (2010). Towards a distributed search engine. In Algorithms and Complexity: 7th International Conference, CIAC 2010, Rome, Italy, May 26-28, 2010. Proceedings 7 (pp. 1-5). Springer Berlin Heidelberg.

[21] Ahmed, R., Bari, M. F., Haque, R., Boutaba, R., & Mathieu, B. (2014, November). DEWS: A decentralized engine for Web search. In 10th International Conference on Network and Service Management (CNSM) and Workshop (pp. 254-259). IEEE.

[22] OpenAI. (2023). Introducing ChatGPT. Retrieved May 10, 2023, from https://openai.com/blog/chatgpt

[23] Heaven, W. D. (2021, May 14). Language models like GPT-3 could herald a new type of search engine. MIT Technology Review. Retrieved May 10, 2023, from https://www.technologyreview.com/2021/05/14/1024918/language-models-gpt3-search-engine-google/