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

Asadullah Al Galib, Md Humaion Kabir Mehedi, Ehsanur Rahman Rhythm, Annajiat Alim Rasel
Department of Computer Science and Engineering
BRAC University
Dhaka, Bangladesh
{asadullah.al.galib, humaion.kabir.mehedi, ehsanur.rahman.rhythm}@g.bracu.ac.bd, annajiat@bracu.ac.bd

***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, from finding cat videos to the deep mysteries of the universe, we tend to overlook the enormous complexities of today's search engines powered by the 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 will 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 will also explore distributed search engines and the way forward of finding information in the age of large language models like ChatGPT or Bard. Finally, we will 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. The initial design of Google was a centralized one that took into consideration the rate of growth of the internet and various scalability issues. According to the original PageRank paper [1], Google employed a distributed web crawler architecture to download content from hundreds of millions of web pages. At that time the internet could be considered to be in its nascent stage compared to today's explosion of data. The technology behind web crawlers grew ever more complex to accommodate billions of websites that exist today [2], in hundreds of languages. Everything has changed from fairly simple HTML web pages to complex web applications and animated sites that provide a gamified experience to users of the internet. 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 in terms of site structure, types of data, and many more.

Even though crawling and related technologies have gone through tremendous innovations in the past decade, the ever-expanding and fluid nature of the internet means crawling tools and techniques need to stay on top of the current trends of the web to collect and aggregate information. 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 complexities where simple link traversing from web pages' extracted content is no longer enough to meet the information demand of current internet users. 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 data. In order to take into consideration of emerging, prominent, and disruptive AI conversational agents like ChatGPT and Bard, and 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. This crawler can be used to crawl certain websites either once or at regular intervals to serve the specific data needs of any organization without relying on others.

The *Introduction* part describes the motivation and overall targets of what the paper will accomplish. 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 [4] on web crawling. Various crawling techniques such as 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 so that servers do not get overwhelmed by the parallel requests of different crawlers. Performance metrics of various web crawlers are also explored in great detail. Paper [5] 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. Traditional breadth-first crawling for forum sites struggles with spider-trap due to duplicated pages and noisy links for various user-actions. In their preferential or topical crawler, authors describe a structured approach to crawl all the post pages found in a forum site. A comprehensive study of deep web crawling techniques currently in use is explained in [3]. Authors also highlight the lack of standards and evaluation metrics to measure the performance of deep web crawlers against some common datasets. They provide a framework to compare different deep web crawling algorithms as well as features to infer entry points for various user-interface and search panels to initiate crawling. The authors of [6] describe the challenges and learnings of a high-performance web crawler, Mercator, that is distributed, scalable, and extensible. Some drawbacks of the implementation include, preventing duplicate URL visit by keeping the list of already visited URLs in memory, proportional growth of visited URL list as the crawler downloads even more pages, resolving DNS requests that is compatible to multi-threaded crawling modules. In [7], the authors propose a distributed search engine based on cooperative model, where local search engines reduce the update interval time as compared to a centralized search engines. Each local search engine maintains a local index by accessing files locally. Meta search engines utilize these local indices to provide results against search queries. The update time interval of files is given priority over search performance. A hadoop based distributed search engine is proposed in [8]. Due to the ever-evolving and constantly growing size of the internet, a distributed search engine is proposed to reduce query time. Hadoop is consisted of two dimensions; the storage dimension and processing dimension. For the storage, the proposed system uses Hadoop Distributed File System or HDFS, and for indexing and processing of users' requests MapReduce is used. This personalized distributed search engine present an end-to-end model, where crawling, extracting, indexing, and fetching results for users' queries are implemented in Hadoop. The performance test highlights the efficacy of multi-node system as the file size grows. A detailed explanation of various crawlers as well as a comparison among crawlers in terms of applicability, usability, scalability, and crawling and refreshing policies can be found in [10]. The authors used precision and recall as the performance metrics.

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, 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 [4]. Using the raw content, downstream tasks index the data as per the requirements at hand. In this section, we will focus on the recent advancements in the field of crawling along with the challenges that arise as the size of the data grows.

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 [1], when they first started, Google now contains a search index of hundreds of billions of web pages that powers Google Search [9]. Initially, the crawler was one of the most challenging components in the entire search engine system architecture. Google used a distributed approach where they used 3 or 4 crawlers, each with hundreds of open connections to download web pages for indexing in the next step. In order to reduce the DNS lookup time for each crawler, the initial design of Google used a caching mechanism for each crawler to lookup IP addresses from the local cache before using a DNS lookup service. Using the vast amount of crawled data from the internet along with the PageRank algorithm, anchor text, and proximity information, Google soon became one of the most reliable search engines, and today, it is synonymous with searching for anything on the internet. Their initial design has gone through continuous optimization steps over the past couple of decades.

A. Types of Crawlers

While the basic idea of crawling remains the same, which is to discover new links by visiting a web page, and then visiting the newly discovered links of web pages to extract new links. This is an ongoing process where new pages are discovered as well as old pages are updated. Based on the ever-growing

and evolving nature of the Web, different types of crawlers have emerged over the years, whether it is consumer data for some companies or survey reports based on information that is shared on various social sites and forums. 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 [4].

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. It does not focus on any particular domain. 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. Topical crawlers are part of a broader category, called preferential crawlers [4]. These focused crawlers are used for tasks that require data from certain domains, rather than an exhaustive collection of web pages. Forum crawlers are a type of preferential crawler that collects information from online forums of different kinds. These crawlers start at the home pages of the forum sites and gradually discover all the boards and posts found in that forum [4]. the data from these crawled forums can be used to identify how social interaction and discourse take place online.

As the size of the internet grows rapidly from one day to the next, the vast majority of the world's accumulated information can be found hidden behind millions of databases that is only accessible through a search panel or form. Users query these data after logging in to these sites and using the interactive search pages. In order to collect data from these hidden parts of the internet, that cannot simply be found by crawling links from one website to another, 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 database and storage systems, typically accessible through an authentication system.

While the aforementioned crawlers are good at crawling from scratch and storing the crawled data for downstream tasks, stale data is of little use in terms of providing search results to users or running analysis. For the data to be most useful, it needs to stay updated as time passes. this is done by incremental crawlers. The sole purpose of these types of crawlers is not to visit new web pages, but rather to update existing and already crawled sites and remove various redundancies in order to increase storage efficiencies [10]. To match up with the current growth rate of the internet, any significant crawling task is done using a parallel and distributed architecture [1][6]. 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 [11].

B. Techniques

We will explore emerging technologies and approaches for distributed and hidden web crawlers in this section. As we have explained in the previous section that modern crawlers need to have a distributed design to tackle the enormity and rapid growth of data on the internet.

In a stand-alone crawler module, also known as centralized crawling, a single instance performs all required steps of crawling, such as DNS lookup, downloading page content, parsing, extracting links to visit, and storing crawled data in relational, non-relational, or file-based storage systems. While this approach has its advantages, e.g. easy to implement, deploy and maintain, easy to debug in case of errors, and predictable load on network bandwidth; given the sheer scale of data, this simple approach is only suitable for rudimentary use cases [13]. The idea of distributed crawlers has emerged to overcome this challenge.

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. Distributed crawling is slightly different from parallel crawling. In parallel crawling, individual crawler instances reside in the same LAN, whereas crawler instances in the distributed architecture are distributed across different geographical locations [13].

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. However, the centralized URL manager introduces single-point failure for the whole system [13].

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 [11]. Each crawler keeps a local DNS cache to minimize the DNS lookup which is a major source of bottleneck for crawlers [1]. Since the master node has a global view of the working load of each crawler instance, it 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. Moreover, with this approach, each crawler instance needs to update a shared *visited URL* list so that other instances avoid processing duplicate pages and wasting computing and storage.

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, where users only need to provide the scraping rules to crawl and parse the web content. 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. Scrapy-Redis distributed component can be used to efficiently crawl sites with semi-structured information. The introduction of Redis enables the crawler module to store and fetch content with exceptional speed [11][14].

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 which is a popular distributed event streaming platform is used as the medium of communication for the crawling instances. [11]

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 which is highly efficient in batch processing large datasets [12].

Apache Spark: Apache Spark is a highly scalable data processing framework that can process large datasets efficiently and quickly, whether it is batch processing or stream data manipulation. Spark can distribute data processing tasks to multiple nodes configured in a cluster [16]. This ability to orchestrate data processing on extremely large datasets makes it a good candidate for large-scale crawling. A distributed crawling system can be designed using Spark based on the master-worker architecture described in a previous section. A control node manages child crawling nodes which perform the actual crawling tasks given seed URLs, distribute crawling tasks to nodes, keeps the crawled information up-to-date to prevent redundant crawling by child nodes [15].

In some cases, a simple focused or topical crawler may collect web pages that seemingly fall under a domain or topic, but that do not contain any specific information that can be used for particular tasks. To further narrow down the crawling of sites that not only contain information of certain domains but contain very specific facts that can be readily applicable in downstream tasks. To achieve this goal, a traditional focused crawler that relies upon machine learning techniques to identify sites of interest can be enhanced by introducing a language model component in the architecture so that the crawler only processes sites that contain actionable information for a given domain [13]. First, the machine learning classifier determines the category of the site, and then a language model is used on the content of the site to rank its usability for the task at hand. Classifiers like SVM can be trained on positive and negative examples to filter out web pages that do not belong to a certain domain [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 [17]. It is stated that, datasets from March-April, 2023 archive contains around 3 billion web pages from 34 million domains [18]. 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. The list of potential targets is sorted before being sent to distributed crawling instances. It uses a back-off algorithm to prevent overwhelming the servers with crawling requests [17]. Given the enormity and quality of the data, datasets crawled by Common Crawl have been used for training GPT-3 large language model [19].

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. In a world where data is everything, crawlers are the tools to aggregate high-quality and relevant data from the internet. Even before the rapid growth of the internet, distributed crawling systems were dealing with various scaling and policy issues. 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. Discovering relevant and good-quality pages is a major challenge for crawling. Another challenge for distributed crawlers is overcoming various bot-prevention and rate-limiting constraints.

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.

Dark web crawling: For discovering sites on the dark web, focused crawlers need to use TOR proxies to discover new links. A set of onion links can be provided to the crawler as seed URLs. This adds to the already complex nature of distributed web crawlers. These crawlers sometimes also require manual login initially to get access to links on these sites [13]. This makes crawling websites on the dark web a very complex problem.

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. Web crawlers can simply ignore the robots.txt file, which basically tells crawlers which pages to crawl next and which pages to avoid. 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. Furthermore, advanced NLP techniques and language models can be used to prioritize crawling candidates as well as the relevancy of web pages for certain topics or domains. A stricter crawling standard can be introduced so that crawlers abide by certain rules and policies to better server resource utilization. Emerging cloud services could be used to make distributed crawling more efficient and cost-effective. New tools can be integrated with crawling modules to better handle unstructured and multimedia data.

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 [22].

A. Architectures

In 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. When using a search engine, users are primarily concerned about 3 things, quality of the ranking of search results, low latency response, and considerable amount of web pages being available in the index which powers the search results [24]. For distributed search engines to be useful in the real world, these constraints must be met in an efficient, scalable, and cost-effective way.

A P2P system such as Gnutella, provides a simple keyword search mechanism that broadcast 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 ID [23].

Traditionally P2P systems utilize the computing and storage resources of regular Internet users. However, the complexity of various search components and the sheer scale of available data on the internet, makes this approach even more challenging when it comes to search engines. 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 [25]. YaCy (https://yacy.net/) provides the implementation of a peer-to-peer distributed search engine where individual nodes take part in crawling and updating DHT. Each node keeps a local copy of indexed data by that node. While searching based on user keywords, YaCy queries the peers containing relevant indexes.

B. Open-source Search Engines

While the most popular search engines like Google, Bing, or Baidu are commercial projects, there are some powerful open-source indexing and search engine frameworks that are used quite extensively to build search applications. Here, we will discuss two such search engines, Apache Lucene and Elasticsearch.

Apache Lucene: Lucene is a high-performance indexing and search engine library. 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 engine. It indexes text documents and then upon query, generates ranked search results from the indexed content [20]. For the data structure, Lucene uses inverted index. The basic workflow of Lucene is as follows - when a new text document arrives, it is sent through an analyzer that tokenize the text and creates an inverted index by storing both the term frequency as well as document ID. This is why finding documents while searching by text is extremely fast and efficient in Lucene. To provide the auto-complete feature, Lucene uses n-gram tokenizer. It also provides various query types to create complex queries that can be used to search indexed documents by keywords.

Elasticsearch: Elasticsearch is another open-source distributed analytics and search engine built on top of Lucene [21]. It provides REST APIs to index documents and to search 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. Users can choose to provide the document schema for an index during creation time, however, Elasticsearch has the ability to infer schema type of fields from documents during indexing. It uses JSON as its document storage type [21].

C. Challenges & Prospects

Even though the idea of an open, and censor-free web is very promising, there are major technical and practical challenges while running a highly scalable, distributed, and fault-tolerant search engine at the internet scale. 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 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 [23]. 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 [22].

Recent advancements in blockchain technologies can be used to optimize different aspects of a distributed search engine, such as crawling, indexing, and storage [22]. Extensive

research is also needed to prevent attacks on the P2P system from adversarial players.

V. AI Conversational Agents

The emergence of ChatGPT has taken the internet by storm. Unlike previous advancements in AI and machine learning, where only the experts in the field were aware of the advancements, ChatGPT is now being used by laymen as well as professionals in specialized fields of study; from generating songs, and recipes to debugging complex code blocks. 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. ChatGPT is published by OpenAI, to interact with users in a conversational manner by generating responses based on provided instructions [26].

ChatGPT is based on GPT-3, a large language model [19] trained on petabytes of data to produce human-like text. It can be used for a variety of tasks, such as conversational agent, summarizing text, correcting grammar, explaining technical topics, generating functional code, and many more. Everyday social networking sites are inundated with numerous generated content using prompt engineering from ChatGPT. It can even be used to generate datasets that can be used to train other machine learning models.

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 and engaging. It is easily noticeable if search engines cannot find lists of web pages matching users' queries, but it is very difficult to distinguish factual content from made-up text using these language models. This poses a significant risk. We need to keep in mind that the purpose of search engines is to parse users' queries, find the links of pages or documents that are relevant to the keywords in the search queries, and finally rank the search results to provide high-quality responses that are beneficial to the users. Whereas large language models simply generate a sequence of words given a starting prompt. 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. Many examples can be found online where ChatGPT provides fictional text as facts and accompanying arguments for its validity.

The appeal of these conversational agents in finding information on the web stems from the fact that search engines still provide only a list of URLs that include the data users are looking for. They cannot combine information from multiple sources and then aggregate to produce a coherent and factually correct answer. Whereas these agents provide answers to questions like another human expert would do [27].

Using search engines, we can learn more about a previously unknown topic from reliable sources and be certain of its authenticity. 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

The world is already plagued by propaganda and disinformation abundant on social media sites. The possibility of generating human-like text will send shockwaves across many sectors. 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. Any task that is about generating text on a given topic now needs to be re-examined and re-evaluated. Various professional roles in the domain of content generation, whether article writers or programmers, will be transformed significantly. Creating products that took months before integrating these AI tools into the workflow, will take days now. 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 of all sorts 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. There should be a governing standard on how to use these models for the benefit of humanity by introducing policies against the harmful use of these technologies.

VI. Domain-Agnostic Asynchronous Crawler

In the previous sections, we have described different types of crawlers, such as universal and focused crawlers. But these traditional approaches of crawling fetch raw page data and index them for future use during users' searching queries. Based on the existence and relevance of keywords in the crawled documents, search results are generated by ranking a list of URLs to present high-scoring documents on top. 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 and no uniform extraction tools can be applied to all. There are some commercial large-scale crawling tools, such as DiffBot (https://www.diffbot.com/) that can crawl sites of the same content type but with different architectures.

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. Given some basic extraction rules, this crawler can fetch raw content and produce structured data from sites of any domain.

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. Asynchronous segments of each component are denoted using fork-join (two horizontal bars with arrows in between) (see Figures 1, 2, 3, and 4). Item is any entity that has attributes describing some properties of that entity, e.g. when crawling an e-commerce site, item is the product.

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. This list can be loaded from the local file system as well as from a web API. After loading the seed URLs, the orchestrator dynamically imports site handlers from the local directory. Each site has a handler associated with it. 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, orchestrator logs the total duration and exits the program.

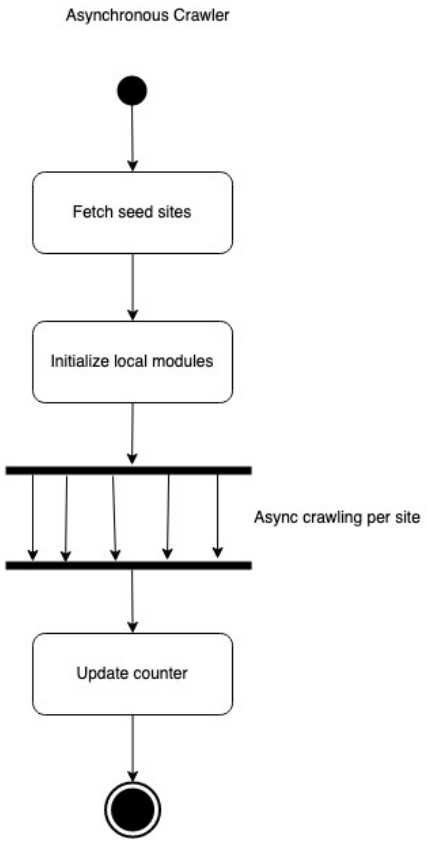

Fig. 1. Asynchronous Crawler - Orchestrator

BaseCrawler: BaseCrawler is the root class that defines the structure of every SiteHandler. It contains both concrete and abstract methods. Concrete methods define the common functionalities that every site crawler needs to perform. The starting point of every site crawler is the asynchronous "crawl" method. When the orchestrator initiates asynchronous crawling for each site, this is the method that gets called. Here the handler gets initialized first and then site-specific pagination is extracted. Since each site will have its own way of handling pagination, SiteHandler instances implement an abstract method that returns pagination counts depending on the site. 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. This is an abstract method that each SiteCrawler implements according to the structure of its corresponding site. The page handler then creates an asynchronous task for each item and initiates them 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 4). Finally, the item is stored in 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 such as extracting pagination count, extracting items from the list-view page, extracting item attributes from the details-view page, and others. SiteHandler instances are child classes of BaseCrawler, and as such they inherit all properties and methods from BaseCrawler. Each SiteHandler contains two components - item schema for items in this site and implementation of abstract methods from BaseCrawler. In this way, no matter how structurally different a site is, as long as it has a structure of list-view and details-view, the custom crawler will be able to crawl those sites.

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 did this to measure the performance improvement of the asynchronous crawler. We used

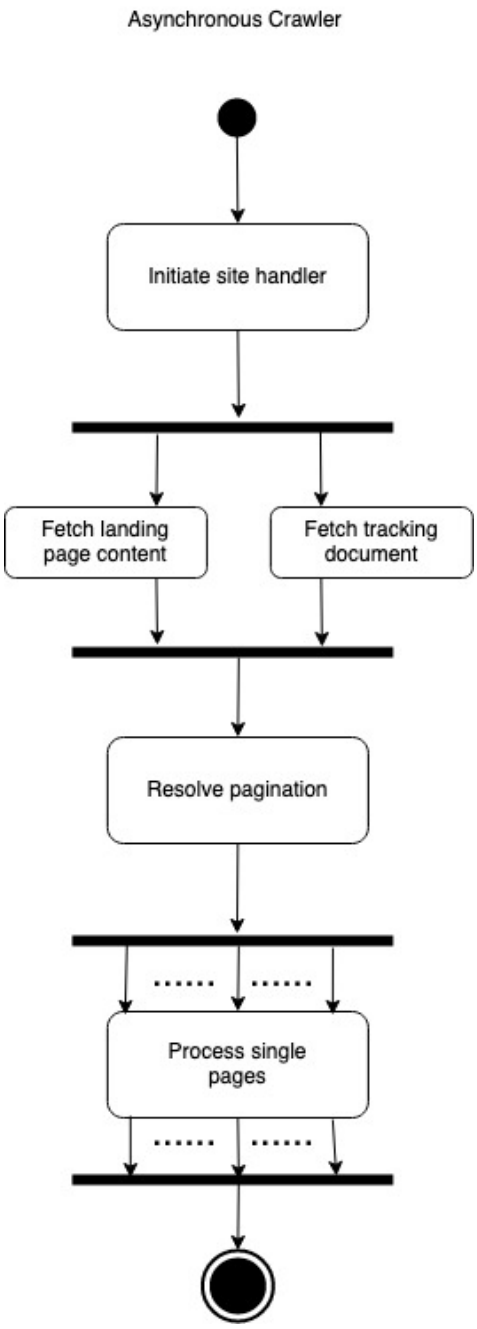

Fig. 2. Asynchronous Crawler - Single Site Handler

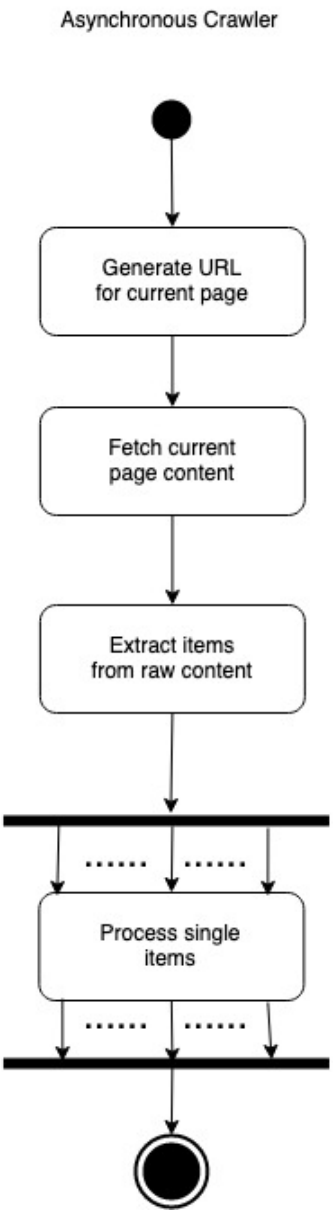

Fig. 3. Asynchronous Crawler - Single Page Handler

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.

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 custom SiteHandler and generate structured datasets wihtin hours rather than days. Such crawling tasks may include, creating product information datasets from an e-commerce site, generating bids or request for proposals from government sites and many more.

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

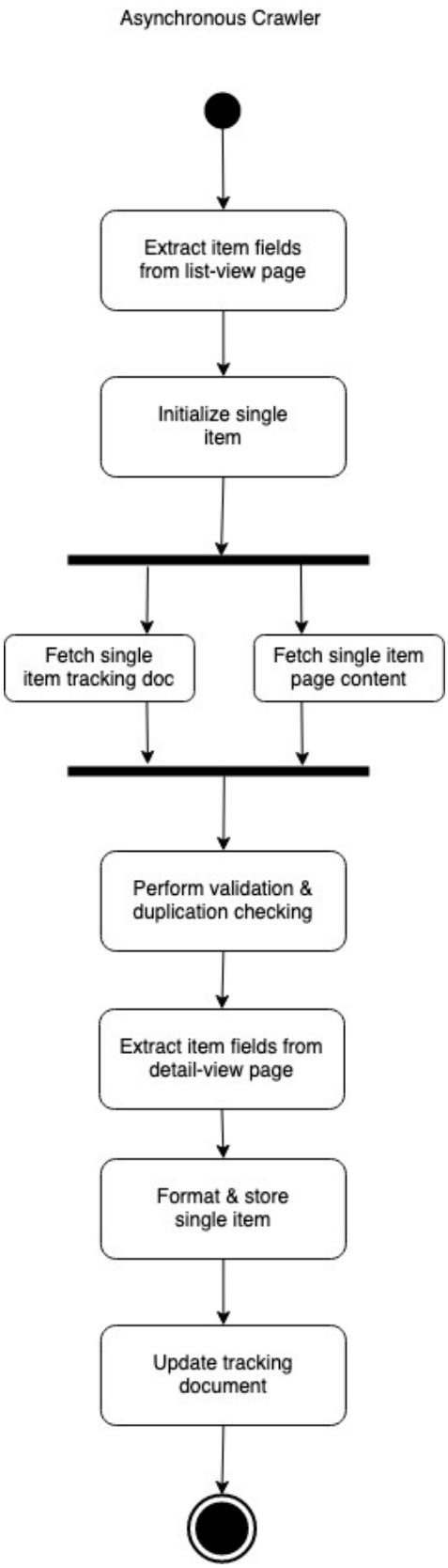

Fig. 4. Asynchronous Crawler - Single Item Handler

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 while requesting content for web pages. We would also like to include a mechanism to fetch content from dynamically loaded web pages that require interaction with the site through client-side scripting

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 data. Common crawl is a public alternative to commercial crawling data. Then, we looked at the current status of distributed or peer-to-peer search engines as well as some open-source versions such as Apache Lucene and Elasticsearch. We also highlighted the scope of improvement for distributed searching. With the ubiquity of large language model based applications, users have started to use these models for collecting information from the web. We explored the impact of these models on traditional search engines and their impacts on various societal issues. 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.

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] https://www.statista.com/chart/19058/number-of-websites-online/
[3] Hernández, I., Rivero, C. R., & Ruiz, D. (2019). Deep Web crawling: a survey. World Wide Web, 22, 1577-1610.
[4] 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.
[5] 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.
[6] Najork, M., & Heydon, A. (2002). High-performance web crawling (pp. 25-45). Springer US.
[7] 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.
[8] 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.
[9] https://www.google.com/search/howsearchworks/how-search-works/organizing-information/

[10] 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.
[11] 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.
[12] https://nutch.apache.org/
[13] 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.
[14] 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.
[15] 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.
[16] https://spark.apache.org/
[17] https://commoncrawl.org/
[18] https://commoncrawl.org/2023/04/mar-apr-2023-crawl-archive-now-available/
[19] 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.
[20] Białecki, A., Muir, R., Ingersoll, G., & Imagination, L. (2012, August). Apache lucene 4. In SIGIR 2012 workshop on open source information retrieval (p. 17).
[21] https://www.elastic.co/what-is/elasticsearch
[22] https://rorur.com/kcoin2.pdf
[23] 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.
[24] 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.
[25] 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.
[26] https://openai.com/blog/chatgpt
[27] https://www.technologyreview.com/2021/05/14/1024918/language-models-gpt3-search-engine-google/