mathematics

MDPI

Article

Decentralized News-Retrieval Architecture Using Blockchain Technology

Adrian Alexandrescu * and Cristian Nicolae Butincu *

Department of Computer Science and Engineering, Faculty of Automatic Control and Computer Engineering, Gheorghe Asachi Technical University of Iasi, 700050 Iasi, Romania
* Correspondence: adrian.alexandrescu@academic.tuiasi.ro (A.A.); cristian-nicolae.butincu@academic.tuiasi.ro (C.N.B.)

Abstract: Trust is a critical element when it comes to news articles, and an important problem is how to ensure trust in the published information on news websites. First, this paper describes the inner workings of a proposed news-retrieval and aggregation architecture employed by a blockchain-based solution for fighting disinformation; this includes a comparison between existing information retrieval solutions. The decentralized nature of the solution is achieved by separating the crawling (i.e., extracting the web page links) from the scraping (i.e., extracting the article information) and having third-party actors extract the data. A majority-rule mechanism is used to determine the correctness of the information, and the blockchain network is used for traceability. Second, the steps needed to deploy the distributed components in a cloud environment seamlessly are discussed in detail, with a special focus on the open-source OpenStack cloud solution. Lastly, novel methods for achieving a truly decentralized architecture based on community input and blockchain technology are presented, thus ensuring maximum trust and transparency in the system. The results obtained by testing the proposed news-retrieval system are presented, and the optimizations that can be made are discussed based on the crawling and scraping test results.

Keywords: blockchain; cloud computing; decentralized architecture; information retrieval; news aggregation; OpenStack; web crawler; web scraper

MSC: 68U35

check for updates

Citation: Alexandrescu, A.; Butincu, C.N. Decentralized News-Retrieval Architecture Using Blockchain Technology. *Mathematics* **2023**, *11*, 4542. https://doi.org/10.3390/math11214542

Academic Editors: Zaki Malik and M. Mustafa Rafique

Received: 28 September 2023
Revised: 31 October 2023
Accepted: 2 November 2023
Published: 3 November 2023

1. Introduction

Disinformation in the online environment is spread mostly by actors with malicious intent or with specific not-so-honest agendas. The fake news distributed all over the Internet can have significant consequences regardless of fields, e.g., political, social, business, and media [1]. Detecting fake news is an important subject in any global context. Specific dishonest pieces of information can be shared by a news publication, and from there, other news outlets can unknowingly spread misleading information, which can lead to serious effects. There is ample research when it comes to detecting fake news and rumors, as well as identifying various disinformation strategies [2] and countermeasures [3,4]. Techniques based on machine learning, deep learning, and natural language processing (NLP) are employed to solve this problem [5]. Some methods approach this as a classification problem, while others use data mining to make predictions or to assess the credibility of a piece of news. None of them provide maximum accuracy and, in many situations, not even pass an acceptable threshold.

In recent times, cognitive and informational warfare has become the preferred tool of action of rogue entities that are trying to destabilize societies. Trust in mainstream media is lower than ever, and the click-based ad revenue model that online media relies upon favors user engagement at the expense of thoroughly checking and vetting published information. This, in turn, generates a blend between real and fake news that affects the ability to distinguish the truth from disinformation and misinformation. A survey [6] conducted in 2019 revealed that only 19% of EU adults have high trust in the news media, while 40% had low or no trust at all.

A concept that, by its inherent nature, ensures trust, transparency, and traceability is blockchain technology. A blockchain is a decentralized structure based on distributed ledger technology (DLT) [7,8] that records and replicates data across many computing nodes. In general, it employs a peer-to-peer (P2P) network and consensus algorithms to guarantee that the data are the same on all computing nodes. In a blockchain, the recorded data are organized in a set of blocks linked together in a chain-like structure. These blocks can only be appended at the end of the chain and cannot be updated or deleted afterward. This implies that once recorded, the data cannot be altered, making the write operations append-only, thus providing traceability and transparency. The security is enforced using cryptographic hashes, keys, and digital signatures and guarantees the data's validity and integrity. Each node in the blockchain network replicates an identical copy of the data independently of other nodes. The consensus algorithms employed by the network ensure that the data preserved at each node is identical across the network. This implies that read operations of blockchain data exhibit high scalability, as any node in the network can be queried independently. However, the write operations need to go through the consensus algorithms that limit the maximum number of transactions per second. This metric varies between different DLT/blockchain implementations [9] and mainly depends on the consensus mechanisms [8,10]. The primary advantage of blockchain technology is the lack of central authority, and therefore, it does not expose a single point of failure. There are many applications of blockchain technology in areas like government, healthcare, the Internet of Things, transport systems, or cloud computing [11]. One of the areas with great potential for innovation is ensuring trust in the information published on the Internet.

At the moment, there are not enough scalable fact-checking platforms nor consistent standards for tracking, labeling, identifying, or responding to disinformation. In paper [12], the authors present a scalable fact-checking approach; however, the scalability comes from semantic aggregation, while the fact-checking is performed by humans. A computational fact-checking approach based on knowledge networks is given in [13]; however, in real-world fact-checking scenarios, the best performance for the AUC (area under the curve) of the ROC (Receiver Operating Characteristic) [14] curve was between 0.55 and 0.65, where 0.5 indicates random performance. Assuming a scalable system, important questions need to be answered, like: Who sets the standards? Who is responsible for validation? Who controls the system? Who manages potential disputes? The decentralized nature of the blockchain can address many of these concerns transparently, as it eliminates the need for a single, trusted institution to make critical decisions.

To have a fake news detection system, the first step is obtaining the said news articles. This implies using data extraction methods (i.e., crawlers and scrapers) to go through the web pages of a news website and extract the article information. In this sense, Figure 1 is a generic news-retrieval system diagram for extracting URLs and article information from a specific news website. The crawling process implies going through the website and extracting the URLs, while the scraping process consists of extracting the actual useful data (i.e., the article information). Usually, there is a list of news websites, which is used as a seed for the crawler, and the crawler stays in the confinements of those websites.

Regarding trust in the system, the main issue is that, now, there is only one actor (the one controlling the crawling/scraping processes), and that sole actor is not enough to ensure trust in the system.

In this direction, the FiDisD project [15] has been developed as an initiative to address the problem of misinformation and disinformation in online media. At its core, the project uses blockchain technology combined with both crowd intelligence and federated artificial intelligence. Its blockchain-based architecture creates a decentralized ecosystem that ensures scalability, transparency, and trust.

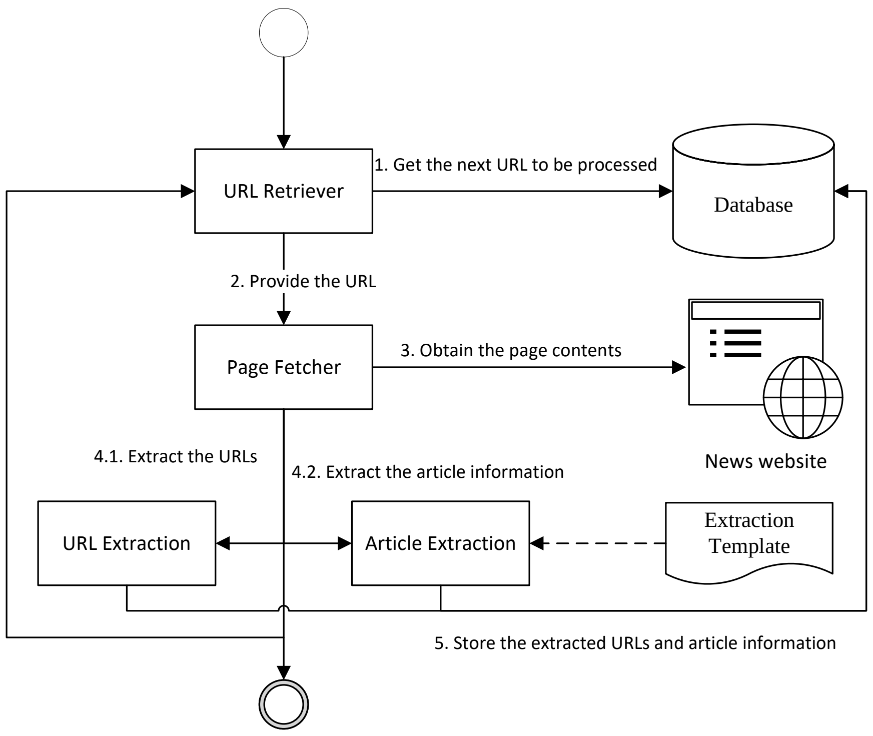
URL Retriever
1. Get the next URL to be processed
Database
2. Provide the URL
Page Fetcher
3. Obtain the page contents
News website
4.1. Extract the URLs
4.2. Extract the article information
URL Extraction
Article Extraction
Extraction Template
5. Store the extracted URLs and article information

Figure 1. Existing news-retrieval system diagram for extracting URLs and article information from a specific news website.

The research presented in this paper has, as its primary focus, the in-depth presentation of the news-retrieval system used by the FiDisD solution for fighting disinformation while also discussing potential architectural improvements. In [16], the authors presented only the overall architecture of the system and the communication between the news-retrieval component, blockchain, federated AI, crowd wisdom, web portal, and end users.

The most important and novel contribution that our paper brings is the method of achieving decentralization in news article retrieval. Other major contributions and novelty that the current paper showcases are:

– the proposed news-retrieval solution with its features: community involvement for decentralized URL and article extraction, distributed architecture, multiple web crawling and web scraping instances, component communication with RESTful APIs, interaction with the blockchain;
– the extraction template used for exact article details extraction, which overcomes the inconsistencies in the HTML web pages' structure;
– the provided solution for cloud deployment, especially on the open-source OpenStack cloud, with a focus on the services that have to be employed for increasing efficiency;
– the novel proposed solution for a community-based truly decentralized architecture.

First, this paper looks at existing content extraction and news-retrieval solutions and discusses the advantages and disadvantages compared to the proposed solution. The theoretical approach, the proposed system architecture, the components, and the information flow between the different services are then presented. Several solutions for deploying the proposed system are discussed, with the focus being mainly on the deployment of the open-source OpenStack cloud solution [17]. A discussion regarding having a community-based truly decentralized architecture follows, and possible solutions are presented. The proposed system is then evaluated by processing the articles from several news websites, and various optimizations are proposed and discussed. Finally, the conclusions of our research are highlighted.

2. Related Work

Part of the research presented in this paper is an in-depth presentation of the decentralized news-retrieval solution, which is one of the main components of the FiDisD project. The overall architecture of that project is described in [16] and consists of distributed crawlers/scrapers, an Offchain system, crowd wisdom, and artificial intelligence modules for detecting fake news, and the blockchain platform for ensuring trust and transparency. The crawling and scraping approaches are presented in that paper summarily without showing the underlying system and without any discussion regarding performance and efficiency.

In terms of existing news-retrieval research, the authors in [18] describe a crawler and extractor from news websites that require only the root URL. They used generic extractors that use heuristics to determine from where, on the web page, the information is to be extracted. Based on the authors' study, the Newspaper library (now evolved to Newspaper3k [19]) proved to obtain the best results. The authors from [20] use machine learning in extracting article-like information from multiple websites. The disadvantage of that proposed approach is the lack of perfect accuracy of the obtained results.

Our previous research regarding extracting specific information from websites containing products [21,22] showed that, for some websites, custom extraction methods must be employed. For example, on a few websites, the product information was available when the page loaded or when a specific HTML element finished loading. This required the use of a tool for automating web browsers and the execution of JavaScript code. Given the heterogeneity of websites in general and news websites in particular, generic extractors cannot provide 100% accuracy, which is needed when it comes to having a solution that is focused on providing trust.

Another example is in [23], where the authors propose a generic method for extracting news information, which is based on heuristics. The obtained results did not yield full success, and for 10% of the considered websites, the results were not accurate. This is unacceptable if the goal is to have a maximum success rate.

There are many generic web content extraction tools and libraries for extracting text from web pages, as is shown in [24]. In that paper, the authors present a web scraping library and compare various tools for text extraction. None of them provided optimum results and, therefore, cannot be employed by the fighting disinformation system considered in our paper.

The best information retrieval results are obtained using targeted extraction based on the structure of each website with a valuable payload. Even this approach proves challenging. The authors in [25] identify 13 issues regarding web scraping, e.g., data cleaning, web page structure, memory, and time consumption. They also propose an algorithm for extracting data, which auto-updates the parameters that are used to locate the target data on the web page. Basically, the extraction template defines the "start" and "end" unique identifiers for each attribute. Given the heterogeneity of the websites' structure, specifying the surrounding entities does not offer enough flexibility. For example, for some pages from the same website, the author's name is not present, there are multiple authors for the same article, or the author's name is surrounded by the same HTML entities as another piece of information. Another issue with their solution is the use of a headless browser for obtaining the data, which adds a significant overhead compared to simply obtaining the response from accessing the URL (Uniform Resource Name).

Other research focuses on dynamically determining the underlying web templates used by the websites [26]. The traditional approach involves clustering the web pages represented by the DOM (Document Object Model) tree. Even if the generic web template for a cluster of pages is determined, it would still require pinpointing the exact location of the article features (e.g., title, author, contents).

In terms of decentralized information retrieval and crawling, there is little to no published research. The authors from [27] present a peer-to-peer model for crawling, which uses geographical proximity and protocols based on distributed hash tables to exchange information between peers. This approach does not tackle the trust issue, but it is an interesting method for achieving full decentralization; more regarding this issue is discussed in Section 5. A paper related to decentralization, blockchain, and community involvement is [28], where the authors propose a method for constructing a knowledge graph based on crowdsourcing and smart contracts, which are used for recommendation systems. Involving the community is a significant step towards full decentralization in detecting fake news. Another issue is regarding traceability and a mechanism to determine how fake news spreads can increase the public's trust in online news media outlets. One such solution is described in [29], where a Python-based web crawler is employed to help with the traceability problem.

Existing distributed crawling solutions focus on performance and how each crawl process knows which URLs to handle. In [30], a hybrid peer-to-peer web crawler is deployed on the AWS (Amazon Web Services) cloud solution. Another example is a distributed news crawler in which the navigation between URLs is performed based on the URL's domain priority URL queues and by deploying it in the fog and cloud layers [31]. A summary of existing research related to distributed crawling is shown in [32]. Other distributed crawling approaches include crawling the hidden web [33], a web crawling solution deployed by a cloud service [34], and a crawler that extracts information only regarding certain topic by classifying the crawled articles [35].

The main novelty of our proposed solution compared to existing methods consists of the decentralized nature of the news extraction process, which involves multiple actors, a data allocation, and a majority-rule algorithm for ensuring trust in the extracted data and the separation of the actual crawling (extracting the URLs) and scraping (extracting the article information). In terms of the individual crawling and scraping processes, the algorithms are based on the traditional approach presented in Figure 1.

3. Proposed News-Retrieval System

The proposed news-retrieval architecture is critical for obtaining the article information needed to detect fake news. As previously mentioned, this proposed solution is an in-depth look at the data acquisition methods and components from the FiDisD project described in [16]. The main aspects discussed here are the theoretical approach to the proposed retrieval system, the overall retrieval system architecture with the reasoning behind the design decisions that were taken, the three distinct system components (i.e., OffchainCore, WebCrawler, and WebScraper), and the communication between them, the exposed API with authorization based on the user's role, the extraction template structure, and the method for data allocation for processing by the crawlers and scrapers.

The novelty of the news-retrieval system proposed in this paper encompasses two aspects: separating the URL extraction (performed by the crawler) from the article extraction (performed by the scraper) to increase scalability and, more importantly, allowing third-party actors to perform the crawling and scraping. The latter provides dynamic system distribution and makes the solution decentralized because multiple independent actors process the same URLs, and the majority decides on the correctly extracted information. Therefore, no bad actors can negatively influence the extraction process.

3.1. Theoretical Approach to the Proposed Decentralized Retrieval

Let $U = \{u_1, u_2, \ldots, u_n\}$, the set of URLs belonging to a particular website, and $A = \{a_1, a_2, \ldots, a_m\}$ a subset of U containing the URLs of pages identified as containing articles: $|U| = n, |A| = m, A \subseteq U, m \leq n$.

Let $C = \{c_1, c_2, \ldots, c_k\}$ and $S = \{s_1, s_2, \ldots, s_l\}$, the set of crawlers and scrapers, respectively, enrolled into the system: $|C| = k, |S| = l$. Although not a requirement, usually $k < l$, i.e., the number of scrapers exceeds the number of crawlers. This is because the number of potential article link URLs that need to be processed by scrapers is, in general, greater than the number of URLs processed by crawlers.

Let O be a random oracle used to generate uniformly distributed random values based on input URLs domain: $O : D \rightarrow b_{256}$, where D is the domain of URLs represented as ASCII strings and b_{256} is the domain of 256-bit values.

To determine the allocation of URLs to crawlers and scrapers, a simple single-valued modulo function applied on the oracle's output can be used: $svf : b_{256} \rightarrow b_{nb}$, where b_{nb} is the domain for crawler and scraper identifiers represented as a nb bit values, e.g., b_{32} represents the domain of 32-bit values. This modulo function uses k and l as modulo parameters for its second operand, for crawlers and scrapers, respectively. An efficient implementation is possible when the modulo's second operand is a power of two; in this case, the modulo can be obtained by simply applying a bit mask to the first operand: $v \bmod 2^t = v \,\&\, (2^t - 1) = v \,\&\, 0b\underbrace{11\cdots1}_{t\text{ times}}$, where & is the bitwise AND operator. However, this naive implementation is not fit for an in-production system as it does not account for the possibility that the selected crawler/scraper might be unavailable, in which case the overall system function would be impaired as the URLs remain unprocessed. To overcome this issue, our system uses a multi-valued modulo function $mvf : b_{256} \rightarrow b_{nb}$ that selects multiple crawler/scraper identifiers. This function outputs, for each input value, a set IDS of crawler/scraper ids with $|IDS| = sp$, where sp is the selective power of the function and is a configurable system parameter.

There are several ways to define such a multi-valued function. A simple approach would be to use a sliding bit window of size nb that sweeps over the b_{256} input to select the crawler/scraper identifiers. However, this approach increases the probability for clusters of crawlers/scrapers to be selected together and limits the selective power of the function to a maximum value of $256 - nb + 1$. Another approach would be to use a set of sp random oracles, $OS = \{O_1, O_2, \ldots, O_{sp}\}$, instead of just one, combined with the single-valued modulo function svf defined above. The end results are assembled to provide a multi-valued output for an input URL, u: $\{svf(O_1(u)), svf(O_2(u)), \ldots, svf(O_{sp}(u))\}$. This approach does not limit the selective power in any way and does not suffer from selection clustering. To increase the selective power, one would add more random oracles into the system. If managing a large number of random oracles is not desired, a third approach would be to use the random oracle O defined above, along with another random oracle $R : b_{256} \rightarrow b_{256}$ that can be applied recursively over its own outputs, and the single-valued module function svf, also defined above. As in the previous case, the end results are assembled to provide a multi-valued output for an input URL, u: $\{svf(R(O(u))), svf(R(R(O(u)))), \ldots, svf(\underbrace{R(R(\cdots R(}_{sp\text{ times}}O(u)\underbrace{))\cdots)}_{sp\text{ times}})\}$. As with the previous approach, there are no limits to selective power and no selection clustering. Increasing the selective power implies making more recursive calls to the R oracle. However, the downside of this approach, as opposed to the first two approaches, is that due to the recursive nature of the R oracle, it cannot be parallelized.

Independent of the selection scheme, the probability of a URL remaining unprocessed is inversely proportional to the selective power sp and represents the probability for all selected sp crawlers/scrapers to be unavailable. If we model the probability for a crawler/scraper to be available at a certain time as $P(A) = pa$, then the probability for it not to be available is $P(\overline{A}) = 1 - pa$. It follows that the probability for all sp crawlers/scrapers not to be available at a certain time is $P(\overline{A})^{sp} = (1 - pa)^{sp}$. Therefore, the probability for a URL to remain unprocessed drops exponentially as the selective power sp increases, and the probability to be processed by at least one crawler/scraper is $1 - P(\overline{A})^{sp} = 1 - (1 - pa)^{sp}$.

This approach to crawler/scraper selection solves two problems: first, it ensures that multiple crawlers/scrapers process the same URLs. This is necessary to detect and ban rogue actors that might try to inject invalid data into the system. The correct results are considered the ones that have the majority, and any crawler/scraper that deviates from this is penalized. By employing the aforementioned selection mechanism, a Sybil 51% majority attack is extremely improbable, and the probability for an attacker to control most selected crawlers/scrapers drops exponentially as the selective power sp increases. Second, it builds trust in the system, as the stored data are vetted by multiple randomly chosen actors.

A URL hacking attack is a technique employed by an attacker that alters the value of a URL (e.g., by adding invalid anchors and/or unused query parameters) while keeping its semantics in the hope that the selection function of the system selects crawlers/scrapers that are under his control. To prevent this type of attack, each URL is stored in the system using a canonical form.

3.2. General News-Retrieval Architecture

The news-retrieval components have the role of obtaining the news article information in a decentralized manner, storing the acquired data, and sending proof of what has been stored to the blockchain network. The overall architecture of the proposed news-retrieval system is presented in Figure 2. It encompasses the following components:

– OffchainCore—the main component, which contains the business logic for managing the crawlers and scrapers,
– WebCrawler—multiple instances that extract URLs from web pages and identify the relevant web pages that contain article information,
– WebScraper—multiple instances that extract the article information from article-containing web pages,
– ClientAPI—used by multiple actors to access the stored article information,
– Blockchain network—used to store the article hashes.

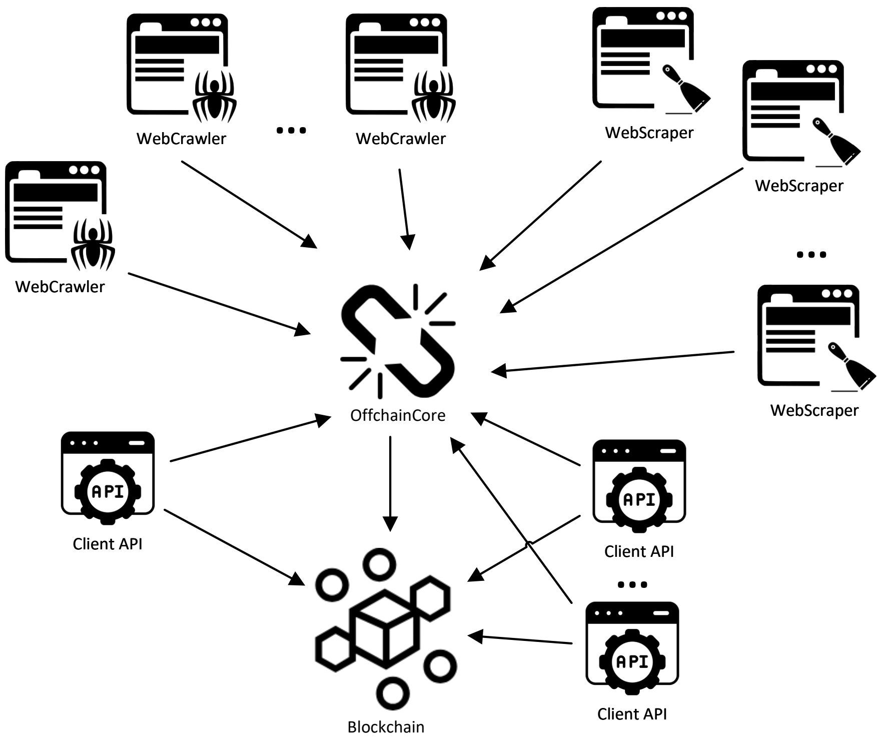

Figure 2. Proposed news-retrieval system architecture. The main components/applications are pictured, and the arrows show the direction of communication—the base of the arrow represents the entity initializing the communication.

Other components of the FiDisD project include Federated AI modules, which analyze the news arriving into the system and assign trust scores, Crowd Wisdom, which consists of human actors that also participate in news analysis and activity token rewards, and a public Frontend Web Portal, which serves as a news aggregator and a view to the OffchainCore database and the article trust scores. These components are beyond the scope of the current paper but are relevant to the overall view of the considered environment.

To have a distributed information retrieval system, multiple actors extract the data (i.e., URLs and article information from web pages) and send it to the OffchainCore component. Each actor processes specific websites, and the crawling is restricted to the URLs belonging to those websites. A first important decision was made to separate the URL extraction, performed by the WebCrawler, from the article information extraction, performed by the WebScraper. This way, a certain number of actors can obtain the URLs while a different number of actors can extract the article data. This is useful because each news website has its own particularities depending on the number of relevant web pages (the ones that contain article information) compared to the total number of pages and the different HTML page structures, which can influence the time it takes to extract information from a page, or even the article's featured image, which can take different amounts of time to download and process depending on the website.

However, having a distributed and separated crawler and scraper is not enough to achieve decentralization. The most important step in this direction is the decision to have independent persons/organizations perform the crawling and scraping. This raises the question of trust in those entities because entities can maliciously report that they extracted a specific article, which is altered with fake information introduced by that entity. Trust must be ensured at three levels: the news publisher, the crawlers, and the scrapers. For the former, the Federated AIs and the Crowd Wisdom ensure trust by verifying the published information. For the latter two, trust in the crawled and scraped data is obtained by majority rule. Therefore, each URL is processed by multiple crawlers, and each article URL is processed by multiple scrapers. Once a specific number of responses are received from processing a specific URL, the majority of the received responses are considered to be the trusted response. Because OffchainCore is the one that pseudo-randomly assigns the URLs to the crawlers and scrapers, no outside entity can intervene and negatively influence the majority decision. If one or more mal-intent scrapers send to the OffchainCore fake article information, then that information is ignored because it is different from the formed majority. Each crawler and scraper actor has an associated user in the OffchainCore. Therefore, if attempts to undermine the system by a specific actor are detected, then that actor can be banned from the system.

3.3. Data Allocation to Crawlers/Scrapers

Depending on the number of WebCrawler and WebScraper actors in the system, the same data will be processed by multiple actors. This is configurable at the OffchainCore component.

The crawling process depends on the number of considered news websites and the number of registered crawler actors. Ideally, each actor should process a limited number of sites, e.g., five sites, so as not to use too many resources from the computing instance the crawler runs on. Each WebCrawler handles a specific number of sites and has one thread/process running for each site. Every crawl request is succeeded by a small waiting period of between one and three seconds because, otherwise, the crawled website might ban the crawler's IP address. To obtain a majority regarding a URL that was extracted and identified as representing an article page, there must be enough WebCrawlers that obtain that URL so no mal-intent actors can take advantage of the system. Let us consider the scenario in which one site is assigned to be crawled by six random actors. In this case, if a URL is identified as containing an article and it is received by the OffchainCore from four of the actors, then we have a majority, and it can be marked as processable, subsequently, by the scrapers.

There is also the off chance that most actors processing a site are composed of bad actors. In this case, a flag would be raised because the minority of good actors did not extract that URL. At this point, the system would consider, based on probabilities, that the minority group is formed of bad actors and flag them accordingly. This can be partially overcome by constantly switching the sites that need to be crawled among the crawlers so no crawler handles a specific site for a long period of time. On the other hand, the paradigm must be updated so that when a crawler receives a new site, it should not process the already extracted URLs before the site allocation update. This can be dealt with by having OffchainCore provide a list of hashes on the URLs that have already been processed.

In terms of the scraping process, scrapers are not bound by extracting article information from a specific list of websites. When OffchainCore receives a list of article-containing URLs, it assigns each URL to a random subset of scraper actors in a similar manner to assigning sites to crawler actors. The same principle applies to ensuring trust in the system and to determining the bad actors. For a URL, each WebScraper extracts the article information and sends it to the OffchainCore together with the hash on the article contents. The OffchainCore then checks the integrity of the received information and stores it in the database and the article store (i.e., the file system). If there are different hashes for the article contents associated with the same URL, then the hash reported by most scrapers identifies the correct article content, and it is stored on the blockchain network.

3.4. Extracted Article Information and Extraction Template

The information extracted from each article page includes the article title, article contents, featured image, publishing date, and author. Only the first two are mandatory because some sites do not have a featured image, publishing data, or an author specified for every article. This information is extracted only from the pages identified by the crawler as having article data. The identification is made using the information provided in the `pageClassifier` property of the object describing the site that needs to be processed. The value of this property is another object containing two keys `containsList` and `containsNotList`, which describe the HTML code and text that have to be present or have to be missing in order for a page to be classified as containing an article. Basically, if certain strings are present in the web page contents and/or other strings are not present, that page is classified as having an extractable article.

To extract the article information, the scraper needs to know where the data are located inside the web page. This is why, for every website, there is an extraction template, which pinpoints the exact location in the HTML structure of the extractable features and filters unwanted data, like ads, scripts, or irrelevant elements. The location is given in the form of CSS selectors, which identify the targeted HTML elements. The extractor module used by the WebScraper is designed to be fast and accurate.

The extraction template is represented by a JSON object in which the keys are the features that need to be extracted, and the corresponding values are arrays describing the extraction process. A special key, `removeElements`, provides the means to target the HTML elements that must be removed before extraction. An example of a value for the `removeElements` key is `["script, style, div.ad-wrapper"]`. This removes all the `script` and `style` elements, and all the `div` elements that have the attribute `class="ad-wrapper"`. The array describing the extractable feature is composed of (1) a CSS selector, (2) a custom property, (3) a Boolean value stating if the extractable value can be missing or not, and (4) a regular expression. The custom property is used to further filter the HTML element identified by the CSS selector, and it can have one of the following values with the specified processing outcome:

– `text`—the contents of the HTML element as text (without any HTML tags),
– `html`—the contents of the HTML element as is (including the HTML tags),
– `attr:`*attributeName*—the value of attribute identified by the specified *attributeName* of the HTML element,
– `attr:src`—the value of the `src` attribute of the HTML element, which is resolved as an URL (normalized to an absolute URL),
– `attr:href`—same value as the previous one only for the `href` attribute.

The last possible element of the array is represented by a regular expression that further processes the value extracted using the CSS selector and the custom property. For example, the URL value corresponding to the featured image of an article is the value of the `srcset` attribute of multiple `source` elements belonging to a `picture` element. The goal is to obtain only the first URL and only that URL without any extra information. In the considered example, after the URL, there is a space character followed by the image dimension. We need only the URL string. Using the `([\^\\s]*)\\s?.*` regular expression as the fourth element of the extraction array achieves that.

3.5. Inter-Component Communication

If we look at the WebCrawler as a black box, its role is to extract URLs from the pages of websites provided by OffchainCore. The main communication between the OffchainCore and the WebCrawler actor is presented in Figure 3. Basically, a WebCrawler actor authenticates itself to the OffchainCore. Then, it has two communication workflow loops, which are handled in parallel. The first one is to obtain the list of sites to be crawled associated with that actor from the OffchainCore. This request is performed with a lower frequency, given the fact that the site list seldom changes. The second one is to send batches of URLs representing pages identified as articles to the OffchainCore. This request is sent only if a sufficient number of URLs are to be sent or if a specific time period has passed since the last sent information and there are URLs available.

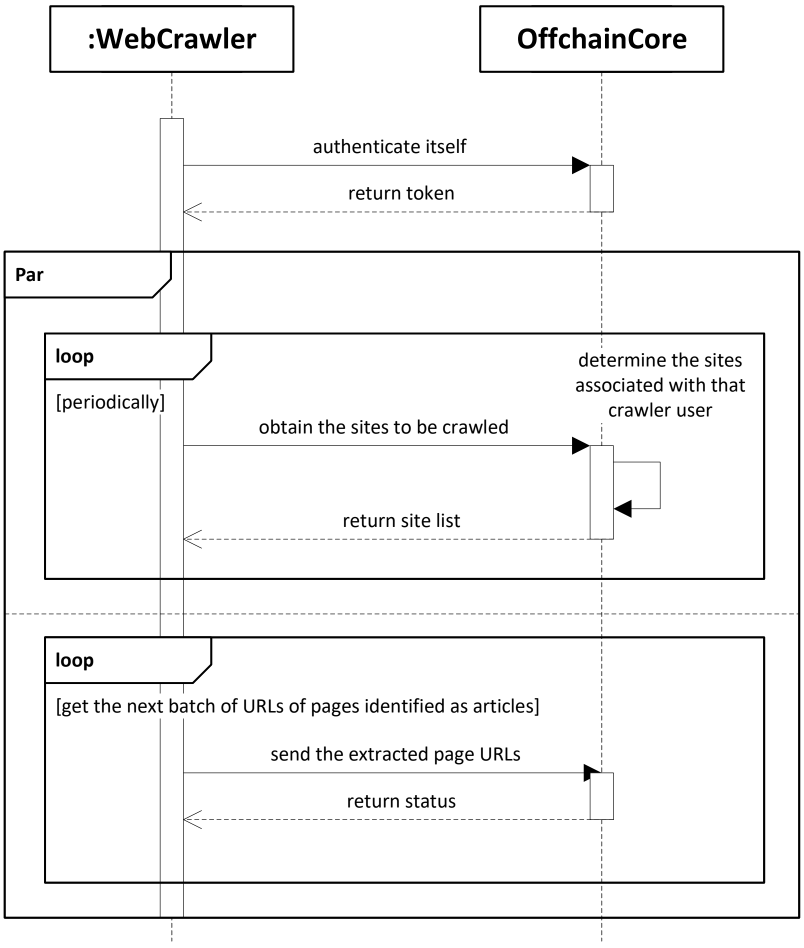
:WebCrawler
OffchainCore
authenticate itself
return token
Par
loop
[periodically]
determine the sites associated with that crawler user
obtain the sites to be crawled
return site list
loop
[get the next batch of URLs of pages identified as articles]
send the extracted page URLs
return status

Figure 3. UML Sequence diagram showing the main data flow between the WebCrawler instance and the OffchainCore component.

The WebScraper's role is to extract article information from the URLs provided by the OffchainCore. Three parallel loops handle the WebScraper–OffchainCore communication (Figure 4), as well as the authentication logic of the WebScraper actor to the OffchainCore. The first communication loop is to periodically obtain the extraction templates used to determine the exact location of the pieces of article information on the web page. This request can be performed occasionally because the page structure corresponding to a site rarely changes. The second communication situation is to obtain the URLs representing article pages from OffchainCore. Each request obtains the next batch of URLs, and the request is made, usually, when all the URLs received so far have been processed. Each WebScraper actor receives only the article URLs associated with that user. Lastly, as the article information is extracted, the third loop obtains the next batch of article data and sends it to OffchainCore. Similarly, to the WebCrawler, the request is made if there is enough data to create a batch or if enough time has passed since the last request. This is so the OffchainCore is not overwhelmed with too many requests from all the crawlers and scrapers.

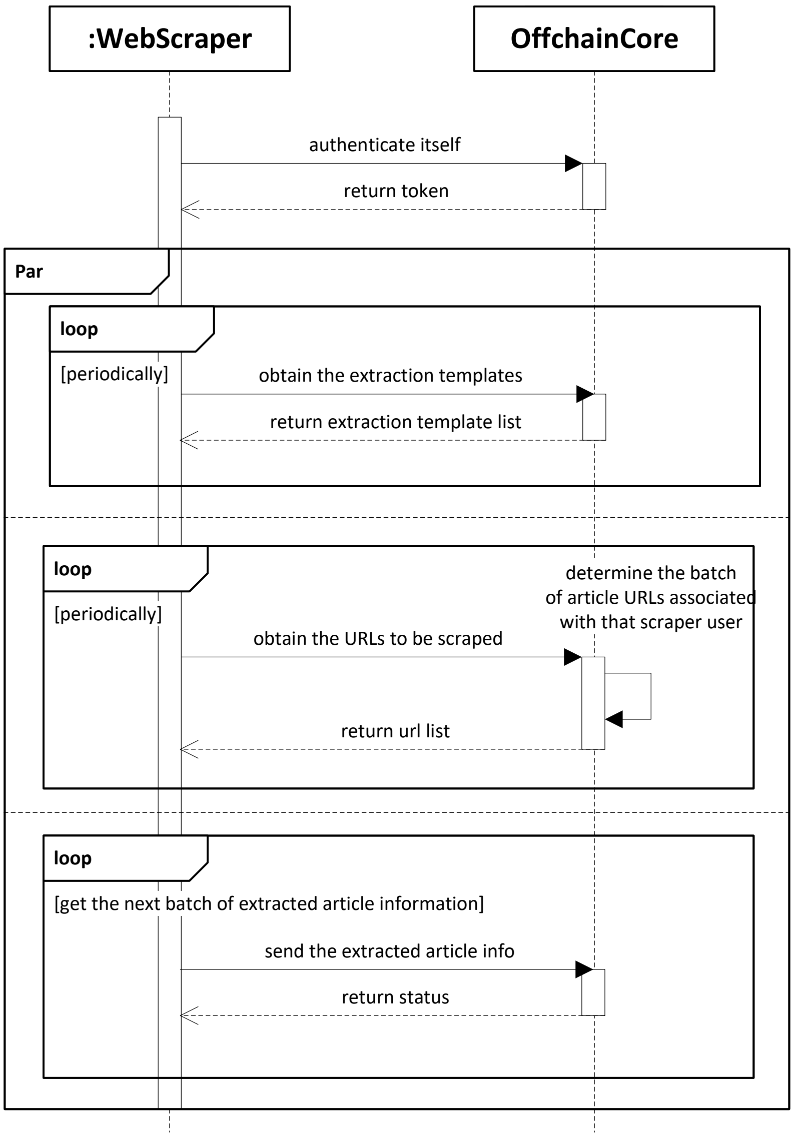
:WebScraper
OffchainCore
authenticate itself
return token
Par
loop
[periodically]
obtain the extraction templates
return extraction template list
loop
[periodically]
determine the batch of article URLs associated with that scraper user
obtain the URLs to be scraped
return url list
loop
[get the next batch of extracted article information]
send the extracted article info
return status

Figure 4. UML Sequence diagram showing the main data flow between the WebScraper instance and the OffchainCore component.

Even though both the crawler and the scraper require input from the OffchainCore, the communication between the OffchainCore and the WebCrawlers/WebScrapers is initiated by the latter two, so they work properly even if they are in a sub-network or are behind a firewall and are not accessible from outside their network. The OffchainCore is the only component that needs to be reachable from anywhere.

Another component that needs access to the OffchainCore is the ClientAPI. This component is used by third-party party applications to consume the public services provided by OffchainCore, i.e., the article information and the associated trust scores assigned by the Crowd Wisdom and the Federated AI modules.

The communication between the components is made using the RESTful (REpresentational State Transfer) APIs, which are an efficient means of using HTTP requests for communicating with server applications. These APIs are exposed by the OffchainCore component and consist of the following main URL paths, with the corresponding HTTP method in parenthesis:

– `/api/auth/signin` (POST)—authentication using username and password; returns a JWT (JSON Web Token) with the user roles,
– `/api/auth/signup` (POST)—creates a new user with the specified role,
– `/api/crawler/sites` (GET)—returns a list of sites (with the page type classifier) assigned to a specific crawler user,
– `/api/crawler/pages` (POST)—receives a list of page URLs representing articles,
– `/api/scraper/extractor-templates` (GET)—returns the site list with the extractor templates, which are used to extract article information,
– `/api/scraper/article-urls` (GET)—returns a paginated list of the article URLs assigned to the authenticated scraper user,
– `/api/scraper/articles` (POST)—receives a batch of article information with the corresponding URLs,
– `/api/public/articles` (GET)—returns a paginated and filtered list of articles based on the query params: pageNumber, pageSize, publication start date and end date, publisher, title, author, and order by,
– `/api/public/articles/article-id` (GET)—returns the article information for the specified article id—it includes the option to return the article metadata, the featured image, the contents with the stripped HTML tags and the full contents of the article.

The aforementioned API routes are accessible based on the role of the authenticated user.

The last communication scenario in the news-retrieval architecture is the communication with the blockchain network. This is made using a smart contract, which adds the computed hash for each article content stripped of HTML tags to the blockchain. The hashes are only computed for articles that have been validated by enough scrapers. The data are added to the blockchain network in batches to increase efficiency, i.e., send the hashes in fixed batches or send all the hashes that are available at that point in time if enough time has passed since the previous update. Storing only the content hashes on the blockchain network acts as proof of data and ensures that the extracted article information has not been tampered with. Therefore, storing the entire article's contents on the blockchain is not necessary and not even feasible due to the involved costs (i.e., the gas fee).

3.6. OffchainCore Component

The OffchainCore component represents the brains of the news-retrieval system. Its main components are showcased in Figure 5. It is comprised of various modules:

– `common`—contains useful methods for handling JSON strings, errors, and other functions,
– `config`—handles the application initialization,
– `security`—contains the model, configuration, and logic to ensure secure access to the OffchainCore,
– `auth`—contains the authentication and authorization logic,
– `persistence`—contains the data models, the services, and the repositories for ensuring data persistence,
– `api`—exposes three REST APIs, i.e., CrawlerAPI, ScraperAPI, and ClientAPI,
– `service`—contains various services for storing the article data in the file system, storing the article hash in the blockchain network, and other helper services for the API module.

There are three entry points in the OffchainCore component, which are handled by the three controllers that process incoming requests: one controller for communication with the WebCrawler, one for the WebScraper, and one for the ClientAPI. Each time a WebCrawler sends a batch of URLs identified as containing articles, the CrawlerController receives that information and calls the Site/Page Persistence module, which stores the information in the Offchain database. In this situation, a service that randomly allocates URLs to scrapers is triggered. The allocation information is stored in the database and is received upon request from the corresponding WebScraper. That request is handled by the ScrapperController module. When the WebScraper sends a batch of article information, that information is processed by the ScraperController, which calls on the ArticlePersistence module. As well as storing the article information in the Offchain DB and the Article Store, this module also updates the content hash count, which determines if an article information is considered to be valid. The Blockchain Article Store service periodically checks for newly determined valid articles by interrogating the Article Persistence module and storing the article hash on the blockchain network.

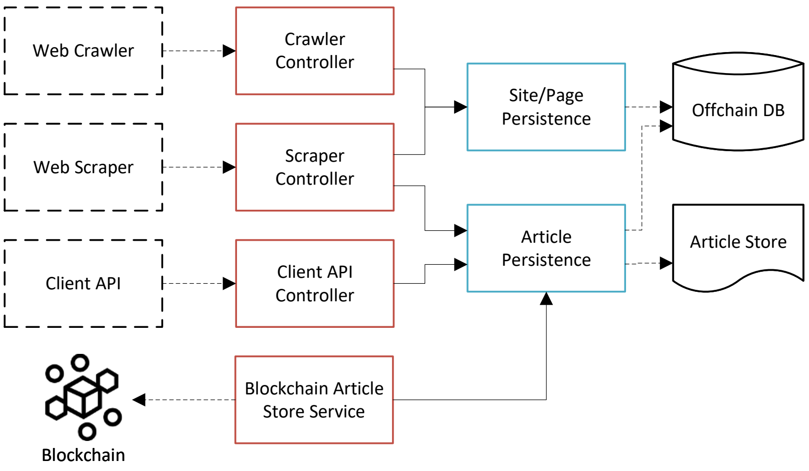

Figure 5. OffchainCore component main architecture, represented by the colored blocks. The red blocks represent services that run when a specific web service is called (for the three controllers) or run periodically (for the Blockchain Article Store service). The blue blocks represent the Data Persistence modules, and the arrows represent the module/component dependency direction.

3.7. WebCrawler Component

The WebCrawler component, whose main architectural blocks are presented in Figure 6, is executed by each crawler actor in the news-retrieval system. Three main services run periodically: the Site Retrieval Service and the Page Sender Service both use the OffchainCore API Handler module to communicate with the OffchainCore, while the Crawler Service manages the extraction of URLs from web pages and identifies which pages contain article information. All three services use the Data Persistence module to store and retrieve data from the crawler's database.

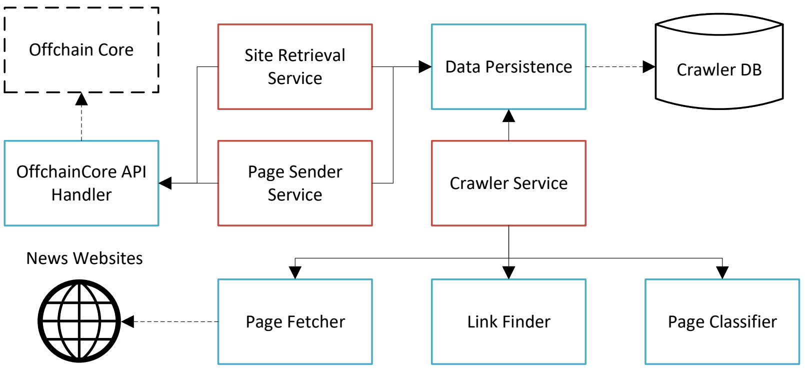

Figure 6. WebCrawler component main architecture, represented by the colored blocks. The red blocks represent services that run periodically. The blue blocks represent modules employed by the services, and the arrows represent the module/component dependency direction.

Periodically, the WebCrawler obtains the latest version of the site list to be crawled, including the page classifier for each site. A Crawler Service instance is created for each site that needs to be processed, and it is assigned its own thread; all the created instance threads run in parallel. Processing each site page is performed as follows:

1. Obtain the next unprocessed URL for that site from the database.
2. Obtain the page contents corresponding to that URL from that website.
3. Extract the URLs belonging to the crawled site by parsing the page contents.
4. Store each extracted URL in the crawler's database.
5. Use the page classifier to determine if the page contains an article and update the crawl status of the web page accordingly in the database.
6. Check if there still are unprocessed URLs. If there are, go to step 1. If not, then go to step 7.
7. Wait for a specified period of time (e.g., half an hour).
8. Reset the crawl status of the main page URL to unprocessed. Optionally, reset the crawl status of all page URLs if we want to support extracting news articles that update periodically.
9. Go to step 1.

The Page Sender Service periodically checks the database for processed URLs that have been identified as representing articles and that have not been sent to the OffchainCore. The URL list is sent in batches so as not to overload the OffchainCore with many requests.

3.8. WebScraper Component

The actual article information is extracted by the WebScraper component (Figure 7). There are a total of four services that run periodically, and all of them use the Data Persistence layer to access the database and/or the article store. Three of them involve communication with the OffchainCore, which is performed similarly to the WebCrawler component, using another OffchainCore API Handler module. The Extractor Templates Retrieval Service runs rarely, and it obtains updated versions of the extraction templates required to extract the article information. The Article URLs Retrieval Service obtains a batch of article URLs to be processed and is called after each batch is processed to obtain the next one. The Article Info Sender Service monitors the database and sends the article information from the Scraper DB and Scraper Store, also in batches. Finally, the core of the WebScraper component is represented by the Scraper Manager Service. This service constantly monitors the database for new article URLs and manages multiple Site Scrapers. There must be only one thread per site that handles the page contents retrieval in order not to have the instance's IP banned by the news website. A batch can contain URLs from different sites and multiple URLs from the same site. Therefore, a variable number of Site Scraper threads are used.

The information stored in the Scraper Store consists of a directory for each website in which there are two files saved for each article. The first is a JSON file containing an object with all the extracted information, i.e., URL, title, contents, contentsTextOnly, contentsTextOnlyHash, featuredImageUrl, featuredImageHash, publishDate, author, extractedDate. The other file is the featured image of the article in the format (e.g., png, jpg, gif, webp) downloaded using the Web Resource Retriever Service.

The steps taken by each Site Scraper are as follows:

1. Obtain the next unprocessed article URL.
2. Obtain the page contents corresponding to that URL from that website.
3. Remove the HTML elements specified in the extractor template and remove the comments.
4. Extract the article information specified in the extractor template.
5. Retrieve the featured image.
6. Compute the hashes for the extracted text-only content and the featured image.
7. Save the article information in the filesystem and update the scrap status of the corresponding article URL in the database.

8. Check if there still are unprocessed URLs. If there are, go to step 1. If not, then terminate (the scraper manager will update the list from the next batch).

Some other modules and functionalities make the OffchainCore, WebCrawler, and WebScraper function properly, but for the sake of brevity, these were omitted from this paper. Only the main components and main workflows are described.

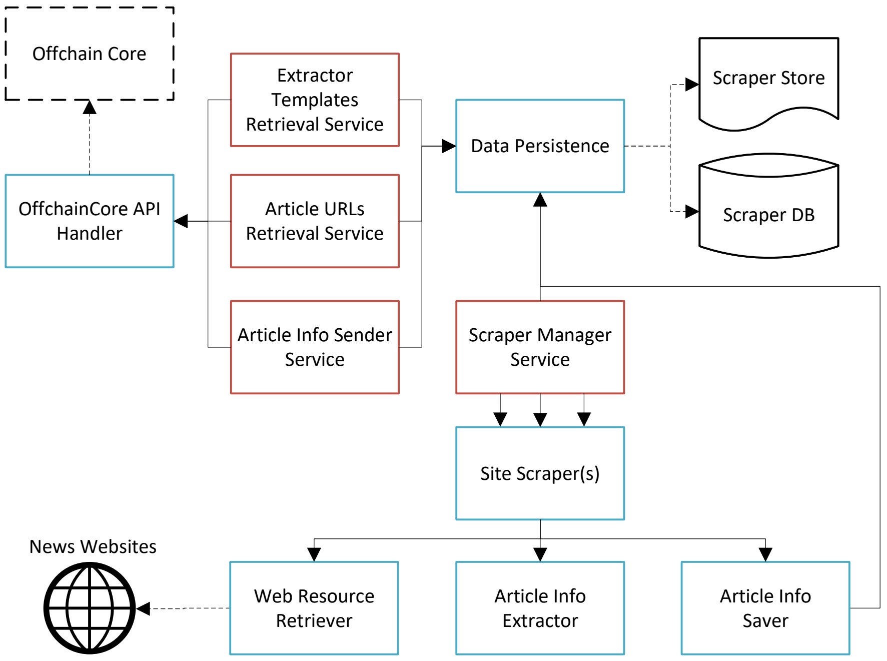

Figure 7. WebScraper component main architecture, represented by the colored blocks. The red blocks represent services that run periodically. The blue blocks represent modules employed by the services, and the arrows represent the module/component dependency direction.

4. Cloud Deployment

In this section, we identify the requirements of each system component and propose solutions for cloud deployment to four major cloud solutions: OpenStack, Amazon Web Services (AWS), Google Cloud Platform (GCP), and Microsoft Azure. OpenStack is an open-source cloud platform that is deployed on premises, while the other three are popular commercial cloud solutions. Deploying the components of the proposed system on a cloud solution provides the inherent advantages that the cloud brings, especially scalability and availability.

The proposed news-retrieval system must run in a decentralized way in which different actors run crawler and scraper instances. There is still a centralized component, i.e., the OffchainCore, in charge of coordinating the crawlers and scrapers. In terms of cloud deployment, we propose a solution for the OffchainCore and for the individual crawler and scraper instances for easy deployment and execution.

The minimum requirements for the OffchainCore component are a compute instance for the web server with the business logic, a database for handling all the data, and a storage solution for storing the featured image and the article JSON files. In order to make it more scalable, a load balancer can be used to manage multiple OffchainCore web server instances. The web server can be deployed on a virtual machine by installing first the necessary dependencies. On OpenStack, the Nova service can be used for managing virtual servers and Neutron for networking. On AWS, Elastic Compute Cloud (EC2) can be used. On GCP, there is Google Compute Engine, and on Microsoft Azure, we have Virtual Machine. A disadvantage of using OpenStack is that it does not provide an out-of-the-box solution for Platform-as-a-Service (PaaS), and the existing third-party solutions lack the proper support. A PaaS represents a service in which the cloud provider provisions everything that the developers need to manage applications. It is basically a compute instance (or more) with all the necessary application dependencies installed and with a seamless means to deploy the application in that environment. The web server can easily be deployed on AWS Lambda, Google App Engine, or Microsoft App Service, which also handles scaling automatically.

For database storage, each cloud provider takes advantage of a database cluster, and there are several solutions: OpenStack's Trove, AWS RDS, Google SQL, or Azure Database for MySQL. The OffchainCore also requires file storage, for which we can use OpenStack's Swift service, AWS Simple Storage Service (S3), Google Cloud Storage, or Microsoft Storage. The idea is to use a storage service to have the same storage for all the instances of OffchainCore so the stored data are consistent.

Regarding the crawler and scraper actors, the installation of the crawler/scraper application must be done without much interaction from the user. Both applications require a Java runtime environment. The crawler also needs a MySQL database server, whereas the scraper requires access to the file system for storing the SQLite database as a file and for storing the article information JSON files and the article featured images. The simplest method for cloud deployment is to have an instance for the crawler/scraper with everything installed. This is neither efficient nor scalable. Another issue is regarding configuration because each crawler/scraper must be configured with the proper credentials to communicate with the OffchainCore.

One solution is to use containerization and create a container with the application and one with the database server. There are two approaches here: run everything on a single machine or run it in a cluster of container daemons. There are cloud services that provide the means to work with containers: OpenStack's Zun and Magnum, AWS Elastic Container Service (ECS), Google Kubernetes Engine, and Azure Kubernetes Service. Using orchestration, the deployment of the crawler/scraper can be done quickly and with ease as long as the container solution is properly configured. The downside of this approach is the extra layer of containerization, which adds overheads.

Another solution is to use dedicated cloud services to deploy the crawler/scraper. Figure 8 shows an application architecture for the WebCrawler and WebScraper components for deployment on the OpenStack cloud solution. A similar approach can be achieved for the other aforementioned cloud solutions by employing the appropriate services.

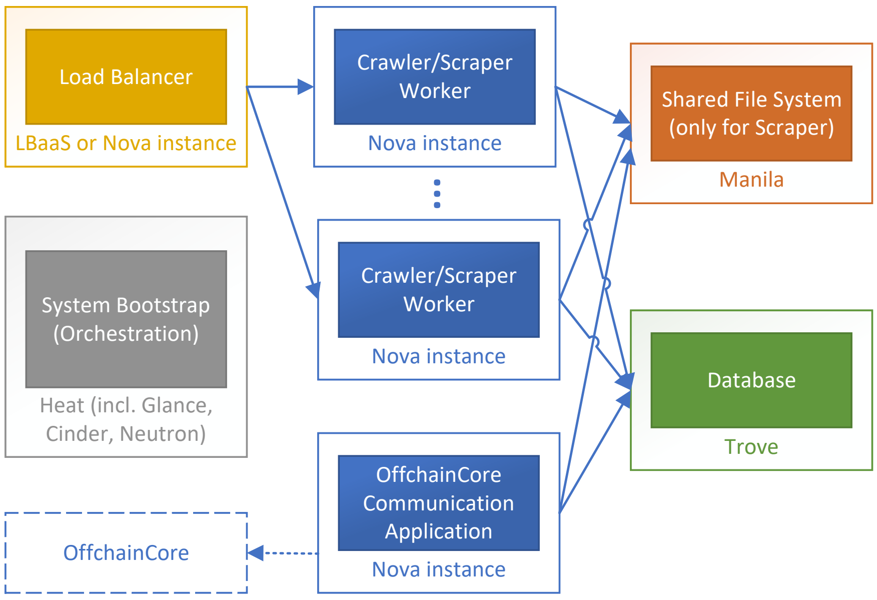

Figure 8. Application architecture for the WebCrawler and WebScraper components for deployment on the OpenStack Cloud.

The proposed architecture is highly scalable and allows multiple crawler/scraper worker instances to run in parallel and process the same sites. The trick is to allocate each instance a different floating IP (i.e., which allows the instance to have a real IP) so that a news website does not ban the application. Otherwise, many requests would have been sent from the same IP, and there would be a greater chance for that IP to be banned from crawling a specific website. There is a change that has to be made to the proposed news-retrieval architecture from Section 3. The logic that allows the communication between the crawler and the OffchainCore must be extracted from the system into another application so the two parts (the extraction logic and the OffchainCore communication logic) can scale independently. This is easily achievable thanks to the way the crawler was designed. The communication between the two parts is performed indirectly through the database server (i.e., the Trove service for the OpenStack deployment). The scraper instances require that the article information be saved on the file system. A good approach is to use the Manila service for a shared file system. This way, all the scrapers write in the same place, and the OffchainCore Communication Application can provide the OffchainCore with necessary article information and featured image resources. There are no concurrency issues when writing the data in the shared file system because each scraper creates, under its own unique ID path, two different files for each extracted article.

In terms of actual deployment, the Heat orchestration service from OpenStack can be employed. This service communicates with the Glance service to specify the details of the image used to create the instances, Cinder for the virtual disk drives, Neutron for networking, and the other aforementioned services required to run the crawler/scraper solutions. Using Heat Orchestration Template (HOT), we can configure the compute instances by specifying the image used to create the instance, the flavor (i.e., number of vCPUs, disk size, and RAM size of the instance), the instance IP/host, and the script that runs after the instance is created. The script includes the installation of all the application dependencies, obtaining the latest version from the application repository, and launching the application. The HOT YAML file also allows the configuration of the load balancing service (LBaaS) by defining the worker pools and the floating IPs that are used.

Having different crawler/scraper actors as part of the system is crucial to ensuring trust in the proposed solution. This is why it is important to have a solution for easy deployment and execution of the applications. Deploying on the cloud is not an easy feat, but it allows a significant increase in performance and efficiency.

5. Community-Based Truly Decentralized Architecture

A decentralized system means that it is controlled by multiple authorities rather than a single one. In the current state of the proposed solution, decentralization is achieved both at crawler and scraper levels, but the information is aggregated, and the majority rule is obtained at the OffchainCore component level.

One easy-to-implement improvement is to have the crawler instances put the hashes of the URLs identified as articles on the blockchain network through a smart contract. Similarly, the scraper instances can put themselves the hashes of the extracted article information and the featured image. This way, the hash is put on the blockchain network by each crawler/scraper rather than only by OffchainCore so that anyone can verify the veracity of the stored information. Now, when OffchainCore determines the majority rule regarding an article, it puts the hash on the blockchain network. Anyone can verify the fact that the uploaded hash adheres to the majority rule by looking at the hashes uploaded by each actor.

However, there are two main downsides. First, each written transaction on the blockchain costs gas, which represents the cost required to perform a transaction on the blockchain network. This can lead to unwanted costs for the crawler and scraper actors. An improvement can be made so that a transaction consists of multiple hashes. If the blockchain in question is a community blockchain solution, e.g., a private Ethereum network, then a custom gas price standard can be defined. Another downside is that the actors must have a valid crypto wallet for signing the transactions. As it stands, the OffchainCore is responsible for signing the transactions and paying the gas fee. The actors can be incentivized to be part of the network by obtaining certain benefits. For example, to have access to analytics regarding the information extracted by all actors or even to receive part of the revenue obtained by monetizing the proposed solution. In this way, the whole community of actors contributes to having a trusted news aggregator and the associated benefits.

Now, the OffchainCore component is a single point of failure and requires trust from the actors in the system. An alternative to the aforementioned improvement involving the actors uploading hash values on the blockchain network is to somehow decentralize the OffchainCore and its associated database. Custom majority-rule algorithms can be employed to ensure that multiple nodes handle the data and processing in a trustful manner, but this is a huge task, and it is not a feasible solution. A better approach is the use of a decentralized storage system, such as the InterPlanetary File System (IPFS), in order to have the OffchainCore as a decentralized web platform. The authors from [36] discuss the performance of having a decentralized web on IPFS.

A decentralized storage system consists of a peer-to-peer network of nodes holding parts of the overall data, creating a resilient file storage system. Such a system can be implemented on top of other decentralized architectures, such as a blockchain-based application or a dedicated peer-to-peer-based network. To analyze a decentralized storage solution, we have to look at the following aspects: persistence mechanism and incentivization schemes, data retention enforcement policy, the level of decentralization, and the majority-rule mechanisms. IPFS does not have a built-in incentive scheme; however, contract-based incentive solutions can be used to ensure longer-term persistence.

Another possible solution is using decentralized oracles [37]. Blockchain oracles are services that allow smart contracts to have access to information from the outside world, i.e., from external data sources. An example of a decentralized oracle-based mechanism for verification that can be used by our proposed solution is presented in [38]. These hybrid smart contracts can use the information provided by the crawler and scraper actors to determine the majority decision regarding specific URLs identified as containing articles and specific article information, respectively. Using oracles, each crawler and scraper can call upon a smart contract to send a hash string associated with processed information. The smart contract logic, based on the information already stored on the blockchain, can determine if a majority exists regarding that information and store the final decision on the blockchain network. The identity of each involved actor is also stored on the blockchain because each one must authenticate and sign the transaction. This approach eliminates OffchainCore's intermediary role of adding the hashes to the blockchain network and, consequently, improves the decentralization of the proposed system.

There are ways of having a truly trusted decentralized architecture using solutions like IPFS, which is an integral part of the Web 3.0 concept [39], and blockchain oracles, which can interact with Offchain resources. The aforementioned possible methods are the basis for extending the research presented in this paper and exploring new possibilities for ensuring trust, not just regarding news publications. As long as the Internet community is willing to contribute and participate as nodes in this decentralized environment, further developments, and enhancements can be made more easily.

6. Use-Case Scenario and Results

The proposed news-retrieval system was tested on seven news websites from Romania. The information published on those websites was written in Romanian, a fact that has no negative influence on running the presented system. Even more so, the system supports any language as the data are saved in UTF-8 character encoding. The details regarding the chosen use-case scenario, the testing environment, and the results are as follows.

6.1. Scenario Description

The seven news websites chosen are Adevarul, AgerPres, DCNews, Digi24, G4Media, Hotnews, and Stiripesurse. These are among the top news publishers in Romania and are considered to be reliable sources of information.

For each website, an extraction template was created manually by analyzing the HTML contents of the web pages containing articles. This can be done easily using the Inspect Element feature of the web browser and copying the selector associated with the selected element containing the extractable value. Listing 1 shows an example of the website input data, including the extraction template, for the Digi24 news website.

Listing 1. Example of an extraction template JSON string.

```
{
  "name": "Digi24",
  "urlBase": "https://www.digi24.ro",
  "logoUrl":"https://www.digi24.ro/static/theme-repo/bin/images/digi24-logo.png",
  "pageTypeClassifier": {
    "containsList": [
      "<main id=\"article-content\""
    ],
    "containsNotList": [
                  "<a href=\"/video\" title=\"Video\" class=\"breadcrumbs-item-link\"> Video </a> "
    ]
  },
  "extractorTemplate": {
    "removeElements": ["script, style, div.ad-wrapper"],
    "title": ["#article-content > article.article > div.container > div.flex.flex-center > div > h1", "text"],
    "contents": ["#article-content > article.article > div.container > div.flex.flex-end.flex-center-md.flex-stretch > div.col-8.col-md-9.col-sm-12 > div > div > div.entry.data-app-meta.data-app-meta-article", "html"],
    "featuredImage": ["#article-content > article.article > div.container > div.flex > div  > figure.article-thumb > img", "attr:src"],
    "publishDate": ["#article-content > article.article > div.container > div.flex.flex-center > div  > div.flex.flex-middle > div > div.author > div.author-meta > span:last-child > time", "attr:datetime"],
    "author": ["#article-content > article.article > div.container > div.flex.flex-end.flex-center-md.flex-stretch > div.col-8.col-md-9.col-sm-12 > div > div > div.entry.data-app-meta.data-app-meta-article a[href*=/autor/]", "attr:href", true]
  }
}
```

The OffchainCore process is initialized with a JSON file similar to the one from Listing 1 for each website. These files are used to initially populate the database with the site list and to allocate sites to web crawlers.

6.2. Testing Environment

For testing purposes, three OpenStack Nova instances were created, one for each of the OffchainCore, WebCrawler, and WebScraper, with the `m1.medium` flavor, which means that each instance has 2 vCPUs (virtual CPUs), 40 GB disk space and 4096 MB of RAM. Also, the three instances had allocated a floating IP so they could be publicly accessible for logging, debugging, and external access. Each Nova instance has installed the Rocky Linux operating system and the Java Virtual Machine for running the applications. The instances that run the OffchainCore and WebCrawler also had a MariaDB database server installed, while the WebScraper used SQLite. The decision to use different database solutions was made for performance reasons. The WebScraper does not need to store much information, and the database updates are few. Therefore, there was no need for a separate database server; instead, a simple in-process database was used.

In terms of the configuration of the crawler and scraper, the following parameters and corresponding values were used:

– minimum waiting time between requests made to the same site (in milliseconds): 1000,
– maximum waiting time between requests made to the same site (in milliseconds): 2500,
– re-crawl time interval (in HH:mm:ss format for time): 00:30:00,
– re-crawl time interval delta (in ISO 8601 format for the duration [40]); re-crawl will occur at the re-crawl time interval +/− a random value between 0 and re-crawl time interval delta).

The first two parameters are designed to generate requests to the same site with a random delay of between 1 and 2.5 s. This way, the target server is not overloaded with requests, and it is less likely that the server will have a problem with the crawler/scraper. The last two parameters refer to the situation in which a site is fully crawled. In this case, the crawler runs after a "re-crawl time interval" modified with a "delta" has passed and starts by re-crawling the main page of the site, which contains the latest news.

6.3. Results

All three processes, i.e., OffchainCore, WebCrawler, and WebScraper, were executed for a fixed period of time on the seven news websites. The statistics regarding the processing are presented in Table 1.

Table 1. Statistics regarding crawling and scraping from seven news websites.

		First Experiment [a]					Second Experiment [b]		
Site Name	**Site URL**	**Total Number of Extracted URLs**	**Number of Waiting to Be Processed Pages**	**Number of Article Pages**	**Number of Irrelevant Pages**	**Number of Invalid Pages**	**Avg. Time to Retrieve Contents (ms)**	**Avg. Time to Process a URL (ms)**	**Number of Processed URLs**
Adevarul	https://adevarul.ro (accessed on 4 August 2023)	35,041	30,526	3769	617	129	873	76,334	454
AgerPres	https://www.agerpres.ro (accessed on 4 August 2023)	14,145	840	2993	3696	6616	217	19,771	1673
DCNews	https://www.dcnews.ro (accessed on 4 August 2023)	26,156	22,730	2644	633	149	254	82,284	425
Digi24	https://www.digi24.ro (accessed on 4 August 2023)	60,977	56,043	3130	1803	1	334	193,100	181
G4Media	https://www.g4media.ro (accessed on 4 August 2023)	18,733	16,566	2076	81	10	364	88,129	392
Hotnews	https://www.hotnews.ro (accessed on 4 August 2023)	11,537	4749	6216	220	352	393	47,873	722
Stiripesurse	https://www.stiripesurse.ro (accessed on 4 August 2023)	45,661	40,625	2254	1720	1062	887	89,788	389
Total		212,250	172,079	23,082	8770	8319	475 [c]	85,325 [c]	4236

[a] over a longer period of time in which the crawler processed multiple times a smaller number of sites at the same time; [b] over a 4 h time period in which seven site crawler threads ran on the same instance at the same time (one crawler thread/site); [c] average values.

Two experiments were performed. The first one was over a longer period of time, and the crawler ran on subsets of the seven websites at the same time for performance reasons. Before the first test started, the main page URL of each of the seven sites was marked for a re-crawl. The second one was a shorter test over a four-hour period in which seven site crawler threads ran on the same instance at the same time (one crawler thread per site).

For the first experiment, the goal is to observe the percentage of useful web pages, i.e., the web pages that contain an article, compared to the total number of the website's pages. Table 1 is a good indicator of how the crawling progresses over a period of time. If we consider only the processed pages (the sum of the number of article pages, irrelevant pages, and invalid pages), the proportion that represents articles varies between 44% and 96%, with an exception. That exception is represented by AgerPres with a percentage of only 22% due to a large number of considered invalid pages of almost 50%. Looking at those invalid pages, these fall into three categories: URLs that represent, in fact, a phone number and contain the text `+tel`, URLs with fragments (e.g., `#mm-2`), and pages that were not available when the crawling ran (either because the web server was down or because the web crawler was temporarily blocked). Excluding those invalid pages, the article page/processed page percentage among the seven websites is between 45% and 97%. AgerPres also has a large number of irrelevant pages because they provide separate URLs to HTML pages that allow the user to view the photographs that appear on the page. Examples of such URLs are `/foto/watermark/11689895` and `/fotografia\_zilei/146/page/9`. The irrelevant pages of the Digi24 website consisted mostly of paginated views of multiple article snippets per page and URLs for each tag associated with the news articles. The same situation is for Stiripesurse, which also had many pages marked as irrelevant.

Regarding the second experiment, the main goal is to determine how many URLs can be processed by a single crawler, which processes all seven sites in parallel over four hours. The results from Table 1 have to be correlated with the data presented in Figure 9 to have a broader picture of the obtained time measurements.

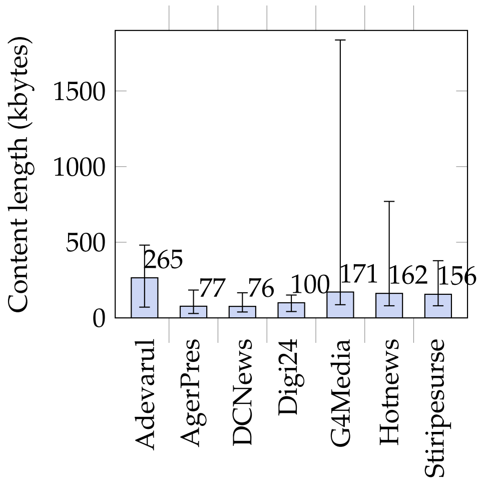

Figure 9. Average content length per page in kbytes. The minimum and maximum lengths are also highlighted.

Regarding the average time it takes to retrieve the web page contents, that time was between 217 ms for AgerPress and 887 ms for Stiripesurse. The time is directly related to the web page content length (the number of kilobytes received from the server) and the latency of the connection to the news web server. Looking at Figure 9, we see that G4Media's maximum page length is around 1840 KB, but this is the case for only three pages from that website out of the 392 that were processed. This is because one of those pages is the main page, which has multiple article snippets. Another consists of a list of article snippets that belong to a specific author, and the last one is a page with many article snippets.

During the four hours, the number of processed URLs varied from 181 for Digi24 to 1673 for AgerPres. The explanation for this discrepancy is directly related to the average time of processing a page. The fact that a page is marked as irrelevant still implies that the URLs are extracted and stored in the database. The average time of processing a URL varies between 19,771 ms and 193,100 ms. This is largely because of the extracted number of links that must be checked and written in the database. Figure 10 shows the average number of extracted links per page, while Figure 11 shows an average number of new links per page, i.e., links that have not appeared on the previously crawled pages. For G4Media, the maximum number of extracted links and new links was significantly higher due to the three web pages that contain many article snippets. Looking at the average values from all seven sites, a maximum of around 10% of links from a web page are new URLs that are not stored in the database.

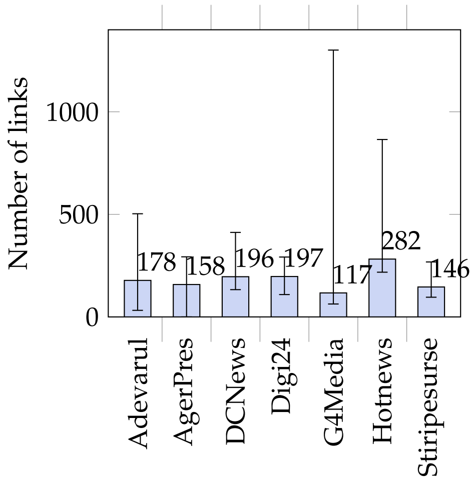

Figure 10. Average number of extracted links per page. The minimum and maximum number of links are also highlighted.

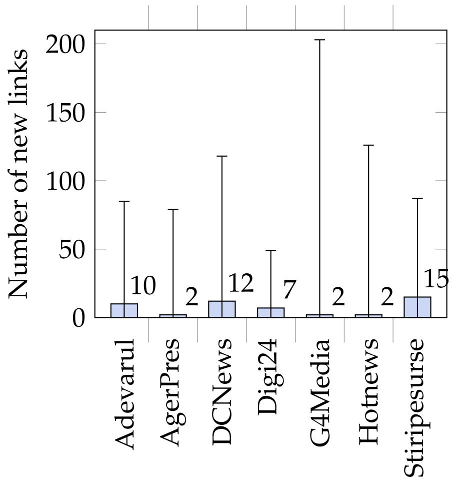

Figure 11. Average number of new links per page (links that have not appeared on the previously crawled pages). The minimum and maximum number of links are also highlighted.

In terms of the web scraper, which extracts the article information, accessing the page, extracting, and storing the data takes less, mainly because the interaction with the database comes down to obtaining the URL and updating twice the processing status (i.e., processing and processed). On the other hand, there is the overhead of writing the article information JSON file and the featured image in the local file system. The average time to entirely process a URL on the tested system was 722 ms. Each URL was processed one at a time, whereas for the second crawling experiment, there were seven threads in parallel. Even if we consider the scraping time seven times slower, the average URL processing time when crawling was around 16 times slower than scraping on the tested instances. The reasoning behind the approach and the results are discussed in the next section of this paper.

7. Discussion

Analyzing the results of the crawling and scraping processes, i.e., the obtained execution times for the various processing, the web page content length, the format of the URLs, and the execution environment, optimizations can be made to speed up the extraction.

The fact that, for some websites, there are many irrelevant and/or invalid page URLs adds more processing time to the crawling process. One improvement that can be made is to add an extra filter based on the URL string so that certain URLs are ignored. The filter must be customized for each site and can be established by analyzing the URLs of an article and non-article pages. The filter efficiency depends on the website's approach to establishing the URLs. Elements like a taxonomy or some other URL patterns can help in determining the filter.

Tests have shown that checking if a URL is in the database and writing the URL if it is not there takes a significant amount of time compared to the time it takes to retrieve the page from the web server and to parse it. Using certain data structures for fast look-up, e.g., PatriciaTrie [41], and adding a caching layer, e.g., using Redis [42], can significantly improve the crawler's performance.

Looking at the individual crawling and scraping methods, compared to existing solutions, those methods produce similar results. The proposed crawler and scraper were designed to be lightweight, and they can be substituted by an existing solution, provided that adaptations are made in order to adhere to the communication protocol with the OffchainCore.

The main focus of this paper was not the individual crawler and scraper but rather the method of achieving decentralization using the URL allocation algorithm and the majority-rule method. Section 3.1 presents a theoretical analysis of the efficiency of the proposed solution for ensuring trust in the decentralized news-retrieval solution. Even though the same pages are processed by a group of crawlers/scrapers, this allows the system to account for mal-intent actors that want to inject fake information and, using majority rule, to detect malicious attempts and to prevent this from happening.

Storing the news article hash on the blockchain network allows actors to verify the accuracy of the majority-rule decision and, implicitly, the veracity of the extracted information. One can say for certain that the article contents stored in the system are the same as the article on the news website from which it has been extracted. As for the veracity of the actual contents of the article, i.e., determining if the presented information is real and true, the method of determining this is based on crowd wisdom and AI methods, which are employed in the FiDisD system, although the exact details are beyond the scope of this paper.

The integration of blockchain technology in the proposed solution does not impact the end-user or the crawler and scraper actors. The OffchainCore component is the one interacting directly with the blockchain network. In terms of performance, the costliest operation is writing on the blockchain. Fortunately, only the hash string on the article contents decided by the majority rule is stored on the blockchain network. This significantly reduces the cost of using blockchain. Also, any blockchain solution can be employed by OffchainCore because the information stored on the blockchain network is public, so anyone can verify that the article contents saved locally by our system are accurate and it is the same as the one extracted by the scrapers.

In terms of the actual news content, usually, an article is picked up by multiple news agencies, which rephrase the piece of news and publish it on their websites. The current proposed system considers each news article from different websites as a distinct piece of news, even if certain passages of text are common to multiple article contents. An extension of our proposed system is to develop an artificial intelligence-based method for determining similar news articles from different websites. Based on the crawl timestamp, the news origin can be determined. This allows for tracing and determining the websites that spread misinformation and disinformation.

8. Conclusions

This paper presents the details in terms of architecture, implementation, and the obtained results of a proposed news-retrieval system, which is used for assessing trust and fighting disinformation when it comes to online news articles. Based on the tests conducted on seven news websites, various optimizations are proposed for improving the system by minimizing the processing times. Another important contribution that this paper brings is an extensive discussion referring to the system deployment in a cloud solution, specifically on the open-source OpenStack platform. Also, various improvements are proposed for having a community-based truly decentralized architecture. The advantages and disadvantages of each solution are discussed, and the potential for further research is emphasized.

Author Contributions: Conceptualization, A.A.; methodology, A.A. and C.N.B.; software, A.A. and C.N.B.; validation, A.A. and C.N.B.; formal analysis, A.A. and C.N.B.; investigation, A.A. and C.N.B.; data curation, A.A.; writing—original draft preparation, A.A.; writing—review and editing, A.A. and C.N.B.; supervision, A.A. All authors have read and agreed to the published version of the manuscript.

Funding: This research was supported by the project "Collaborative environment for developing OpenStack-based Cloud architectures with applications in RTI" SMIS 124998 from The European Regional Development Fund through the Competitiveness Operational Program 2014–2020, priority axis 1: Research, technological development and innovation (RTI)—the POC/398/1/1 program.

Data Availability Statement: Not applicable.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Wu, Y.; Ngai, E.W.; Wu, P.; Wu, C. Fake news on the internet: A literature review, synthesis and directions for future research. *Internet Res.* **2022**, *32*, 1662–1699. [CrossRef]
2. Chen, C.-H.; Ma, Y.; Lai, Y.H.; Chang, W.-T.; Yang, S.-C. Analyzing Disinformation with the Active Propagation Strategy. In Proceedings of the 24th International Conference on Advanced Communication Technology (ICACT)—Artificial Intelligence Technologies toward Cybersecurity, Pyeongchang, Republic of Korea, 13–16 February 2022; pp. 262–266.
3. Dowse, A.; Bachmann, S.D. Information warfare: Methods to counter disinformation. *Def. Secur. Anal.* **2022**, *38*, 453–469. [CrossRef]
4. Schneider, E.J.; Boman, C.D. Using Message Strategies to Attenuate the Effects of Disinformation on Credibility. *Commun. Stud.* **2023**, *74*, 393–411. [CrossRef]
5. Bondielli, A.; Marcelloni, F. A survey on fake news and rumour detection techniques. *Inf. Sci.* **2019**, *497*, 38–55. [CrossRef]
6. Guttmann, A. Survey: Index of Respondents' Trust towards Media in European Union (EU 28) Countries in 2019. Available online: https://www.statista.com/statistics/454409/europe-media-trust-index/ (accessed on 7 August 2023).
7. Gorbunova, M.; Masek, P.; Komarov, M.; Ometov, A. Distributed Ledger Technology: State-of-the-Art and Current Challenges. *Comput. Sci. Inf. Syst.* **2022**, *19*, 65–85. [CrossRef]
8. Soltani, R.; Zaman, M.; Joshi, R.; Sampalli, S. Distributed Ledger Technologies and Their Applications: A Review. *Appl. Sci.* **2022**, *12*, 7898. [CrossRef]
9. Antal, C.; Cioara, T.; Anghel, I.; Antal, M.; Salomie, I. Distributed Ledger Technology Review and Decentralized Applications Development Guidelines. *Future Internet* **2021**, *13*, 62. [CrossRef]
10. Chauhan, A.; Malviya, O.P.; Verma, M.; Mor, T.S. Blockchain and Scalability. In Proceedings of the 2018 IEEE International Conference on Software Quality, Reliability and Security Companion (QRS-C), Lisbon, Portugal, 16–20 July 2018; pp. 122–128. [CrossRef]
11. Mohanta, B.K.; Jena, D.; Panda, S.S.; Sobhanayak, S. Blockchain technology: A survey on applications and security privacy challenges. *Internet Things* **2019**, *8*, 100107. [CrossRef]
12. Yang, J.; Vega-Oliveros, D.; Seibt, T.; Rocha, A. Scalable Fact-checking with Human-in-the-Loop. In Proceedings of the 2021 IEEE International Workshop on Information Forensics and Security (WIFS), Montpellier, France, 7–10 December 2021; pp. 86–91. [CrossRef]
13. Ciampaglia, G.L.; Shiralkar, P.; Rocha, L.M.; Bollen, J.; Menczer, F.; Flammini, A. Computational Fact Checking from Knowledge Networks. *PLoS ONE* **2015**, *10*, e0141938. [CrossRef]
14. Classification: ROC Curve and AUC | Machine Learning. Available online: https://developers.google.com/machine-learning/crash-course/classification/roc-and-auc (accessed on 7 August 2023).

15. FiDisD—Fighting Disinformation Using Decentralized Actors Featuring AI and Blockchain Technologies. Available online: https://www.trublo.eu/fidisd/ (accessed on 4 July 2023).
16. Buțincu, C.N.; Alexandrescu, A. Blockchain-Based Platform to Fight Disinformation Using Crowd Wisdom and Artificial Intelligence. *Appl. Sci.* **2023**, *13*, 6088. [CrossRef]
17. OpenStack. Available online: https://www.openstack.org/ (accessed on 15 July 2023).
18. Hamborg, F.; Meuschke, N.; Breitinger, C.; Gipp, B. News-please: A Generic News Crawler and Extractor. In *Everything Changes, Everything Stays the Same: Understanding Information Spaces, Proceedings of the 15th International Symposium of Information Science (ISI 2017), Berlin, Germany, 13–15 March 2017*; Gäde, M., Ed.; Hülsbusch: Glückstadt, Germany, 2017; pp. 218–223.
19. Ou-Yang, L. Newspaper3k: Article Scraping & Curation. Available online: https://newspaper.readthedocs.io/en/latest/ (accessed on 4 July 2023).
20. Le Huy Hien, N.; Tien, T.Q.; Van Hieu, N. Web Crawler: Design and Implementation for Extracting Article-like Contents. *Cybern. Phys.* **2020**, *9*, 144–151. [CrossRef]
21. Alexandrescu, A. A distributed framework for information retrieval, processing and presentation of data. In Proceedings of the 2018 22nd International Conference on System Theory, Control and Computing (ICSTCC), Sinaia, Romania, 10–12 October 2018; pp. 267–272.
22. Alexandrescu, A. Optimization and security in information retrieval, extraction, processing, and presentation on a cloud platform. *Information* **2019**, *10*, 200. [CrossRef]
23. Dong, Y.; Li, Q.; Yan, Z.; Ding, Y. A generic Web news extraction approach. In Proceedings of the 2008 International Conference on Information and Automation, Changsha, China, 20–23 June 2008; pp. 179–183. [CrossRef]
24. Barbaresi, A. Trafilatura: A Web Scraping Library and Command-Line Tool for Text Discovery and Extraction. In Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing: System Demonstrations, Online, 1–6 August 2021; pp. 122–131. [CrossRef]
25. Qudus Khan, F.; Tsaramirsis, G.; Ullah, N.; Nazmudeen, M.; Jan, S.; Ahmad, A. Smart algorithmic based web crawling and scraping with template autoupdate capabilities. *Concurr. Comput. Pract. Exp.* **2021**, *33*, e6042. [CrossRef]
26. Gupta, G.; Chhabra, I. Optimized template detection and extraction algorithm for web scraping of dynamic web pages. *Glob. J. Pure Appl. Math.* **2017**, *13*, 719–732.
27. Singh, A.; Srivatsa, M.; Liu, L.; Miller, T. Apoidea: A Decentralized Peer-to-Peer Architecture for Crawling the World Wide Web. In *Distributed Multimedia Information Retrieval*; Callan, J., Crestani, F., Sanderson, M., Eds.; Springer: Berlin/Heidelberg, Germany, 2004; pp. 126–142.
28. Wang, S.; Huang, C.; Li, J.; Yuan, Y.; Wang, F.Y. Decentralized construction of knowledge graphs for deep recommender systems based on blockchain-powered smart contracts. *IEEE Access* **2019**, *7*, 136951–136961. [CrossRef]
29. Ye, H.; Lu, Y.; Qiu, G. Tracing Method of False News Based on Python Web Crawler Technology. In Proceedings of the International Conference on Advanced Hybrid Information Processing, Changsha, China, 29–30 September 2022; pp. 489–502.
30. Kim, Y.Y.; Kim, Y.K.; Kim, D.S.; Kim, M.H. Implementation of hybrid P2P networking distributed web crawler using AWS for smart work news big data. *Peer-Netw. Appl.* **2020**, *13*, 659–670. [CrossRef]
31. Prismana, I.G.L.P.E. Distributed News Crawler Using Fog Cloud Approach. In Proceedings of the International Joint Conference on Science and Engineering 2022 (IJCSE 2022), Surabaya, Indonesia, 10–11 September 2022; pp. 251–260.
32. Ren, X.; Wang, H.; Dai, D. A summary of research on web data acquisition methods based on distributed crawler. In Proceedings of the 2020 IEEE 6th International Conference on Computer and Communications (ICCC), Chengdu, China, 11–14 December 2020; pp. 1682–1688.
33. Kaur, S.; Geetha, G. SIMHAR-smart distributed web crawler for the hidden web using SIM+ hash and redis server. *IEEE Access* **2020**, *8*, 117582–117592. [CrossRef]
34. ElAraby, M.; Moftah, H.M.; Abuelenin, S.M.; Rashad, M. Elastic web crawler service-oriented architecture over cloud computing. *Arab. J. Sci. Eng.* **2018**, *43*, 8111–8126. [CrossRef]
35. Gunawan, D.; Amalia, A.; Najwan, A. Improving data collection on article clustering by using distributed focused crawler. *Data Sci. J. Comput. Appl. Inform.* **2017**, *1*, 1–12. [CrossRef]
36. Trautwein, D.; Raman, A.; Tyson, G.; Castro, I.; Scott, W.; Schubotz, M.; Gipp, B.; Psaras, Y. Design and evaluation of IPFS: A storage layer for the decentralized web. In Proceedings of the ACM SIGCOMM 2022 Conference, Amsterdam, The Netherlands, 22–26 August 2022; pp. 739–752.
37. Breidenbach, L.; Cachin, C.; Chan, B.; Coventry, A.; Ellis, S.; Juels, A.; Koushanfar, F.; Miller, A.; Magauran, B.; Moroz, D.; et al. Chainlink 2.0: Next steps in the evolution of decentralized oracle networks. *Chain. Labs* **2021**, *1*, 1–136.
38. Ma, L.; Kaneko, K.; Sharma, S.; Sakurai, K. Reliable decentralized oracle with mechanisms for verification and disputation. In Proceedings of the 2019 Seventh International Symposium on Computing and Networking Workshops (CANDARW), Nagasaki, Japan, 26–29 November 2019; pp. 346–352.
39. Alabdulwahhab, F.A. Web 3.0: The decentralized web blockchain networks and protocol innovation. In Proceedings of the 2018 1st International Conference on Computer Applications & Information Security (ICCAIS), Riyadh, Saudi Arabia, 4–6 April 2018; pp. 1–4.

40. W3C Date and Time Formats. Available online: https://www.w3.org/TR/NOTE-datetime (accessed on 15 July 2023).
41. Chatterjee, T.; Ruj, S.; Das Bit, S. Efficient Data Storage and Name Look-Up in Named Data Networking Using Connected Dominating Set and Patricia Trie. *Autom. Control Comput. Sci.* **2021**, *55*, 319–333. [CrossRef]
42. Su, Q.; Gao, X.; Zhang, X.; Wang, Z. A novel cache strategy leveraging Redis with filters to speed up queries. In Proceedings of the International Conference on High Performance Computing and Communication (HPCCE 2021), Haikou, China, 17–19 December 2021; Volume 12162, pp. 150–154.