Grenze International Journal of Engineering and Technology, June Issue

Fair Play: A Web Scraping Tool

Paras Mittal[1], Ajay kumar[2], Arpita Gupta[3] and Yuvraj Tomar[4]
[1-4]Meerut Institute of Engineering and Technology
Email: paras.mittal.csit.2020@miet.ac.in, ajay.kumar@miet.ac.in, arpita.gupta.csit.2020@miet.ac.in, yuvraj.tomar.csit.2020@miet.ac.in

***Abstract*—In the era of advancing technologies, the proliferation of internet users has led to a substantial surge in the generation of unstructured data on the web. Unearthing valuable insights from this vast pool of information often relies on scraping as a primary method for *data extraction. However, when confronted with the task of extracting a large volume of data, several challenges come to the forefront. These challenges include the encounter with* captchas, issues related to storage for substantial datasets, the necessity for significant computational capacity, and the paramount importance of ensuring the reliability of the data extraction process.**

This paper delves into an in-depth analysis of the scraping process when applied to the extraction of massive amounts of data from the internet. In response to the identified challenges, we propose a cloud-based web scraping architecture that adeptly manages storage and computing resources with elastic scalability on demand. Our chosen platform for implementation is Amazon Web Services, leveraging Elastic Compute Cloud (EC2) and DynamoDB. This solution aims to provide a comprehensive approach that addresses both the intricacies of the scraping process and the practicality for applications in industries reliant on large-scale data.

Selenium emerges as a key tool in our web scraping endeavors, primarily due to its support for various web drivers, enabling the simulation of user interactions with a browser. The paper includes an examination of the scalability and performance metrics of our proposed cloud-based scraper, outlining the advantages it offers over alternative cloud-based scraping solutions. By presenting a unified architecture tailored for data-centric industries, our approach aims to enhance the efficiency and effectiveness of web scraping processes while seamlessly integrating with the dynamic and elastic capabilities of cloud computing.

***Index Terms*— Cloud Computing, Web Scraping, cloud-based web scrapper.**

I. INTRODUCTION

The World Wide Web's wealth of data represents a doubleedged sword, offering both opportunities The World Wide Web's abundance of data presents both opportunities and challenges. While it serves as a vast source of valuable information, acquiring this data poses a significant challenge. Most information in the web is represented as as HTML document, a sketch to accord information to humans, not for computers and such data are Unstructured data [1]. A substantial portion of web information is formatted as HTML documents, intended for human users rather than computers. This type of data is commonly known as unstructured data. The annual surge in internet data is remarkable, and a predominant portion of this data is characterized by its unstructured nature.

Grenze ID: 01.GIJET.10.2.671_1

II. Background and Related Work

Cloud Computing, renowned for its ability to handle intensive computing tasks and offer elastic resources through a pay-per-use model, emerges as the optimal solution for small businesses looking to initiate web scraping operations with limited resources. Web scraping, commonly referred to as web harvesting or web data scraping, is a method of extracting information from websites. Numerous
Prominent cloud-based tools and software options are available for web scraping, encompassing both free trial and paid versions. Below, we provide a brief introduction to some of these cloud-based scraping tools.

A. Import io

Import io [8] stands out as a web-centric platform tailored for non-coders to effortlessly scrape data from diverse websites. This platform adopts a Software as a Service (SaaS) model within the cloud, storing all extracted data on its dedicated cloud server. Users can conveniently download the gathered data in various formats such as CSV, Excel, Google Sheets, JSON, or access it through an API. Import.io not only facilitates seamless scraping but also allows integration with third-party analytics and visualization tools via API.
Noteworthy features encompass automatic data and image extraction, auto page linking, monitoring, scheduling, all achieved without the necessity for coding skills.

B. Webscraper.io

There is also another Webscraper.io[4] , Web Scraping tool which offers both extensions based and cloud-based web scraping service. Its extension for Google Chrome can be used freely, whereas, for cloud-based web scraper, it comes with a price depending on the number of URLs to scrape. With the extension provided by webscrapper.io, a user can create a plan implying how and what to scrape.

C. Scrapy

Scrapy [17] emerges as a collaborative, open-source framework designed for web scraping, offering the added functionality of extracting data through APIs. This versatile tool caters to both web scraping and web crawling needs and is implemented using the Python programming language. Users can leverage Scrapy Cloud to scale their operations on demand. Notable features encompass a choice between visual or code-based interaction, intelligent resource management, and complete API access.
Existing solutions predominantly focus on the initial stage of big data, involving scraping or data acquisition. However, these solutions often fall short as the data collected through scraping must undergo subsequent processes not considered in the aforementioned systems. We propose the development of a cloud-based distributed scraping model that addresses the comprehensive workflow of big data. This envisioned system would seamlessly handle both the scraping process and subsequent data processing on the cloud, utilizing parallel machines for enhanced efficiency.

III. Methodology

A. Scraper Background

Figure 1: The typical scraping process involves the client supplying URLs and configuration details to the scrape-hub, a Python-based program responsible for initiating, managing, and monitoring all tasks within the system. The scrape-hub program then launches the scraping engine, which, in turn, initializes a web page, automates it to the desired state using the Selenium library, and parses it using the Requests library. It's important to note that different scrapers may employ various libraries for parsing and automating web pages, with Selenium and Requests being highlighted here. The collected data is stored in a database.

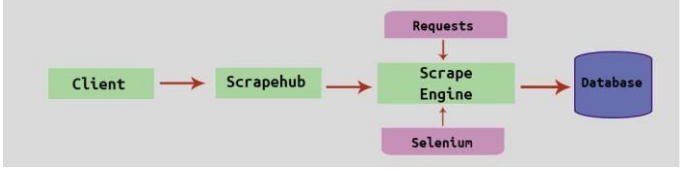

Fig. 1: Normal Scrapper

This conventional sequential method operates by scraping one site at a time, leading to a limitation in the amount of data extracted within a given timeframe. While this limitation can be mitigated by allocating additional

computing and storage resources, such an approach can become expensive and challenging to maintain as demand increases. Consequently, this solution lacks flexibility.

B. Cloud-based Scraper

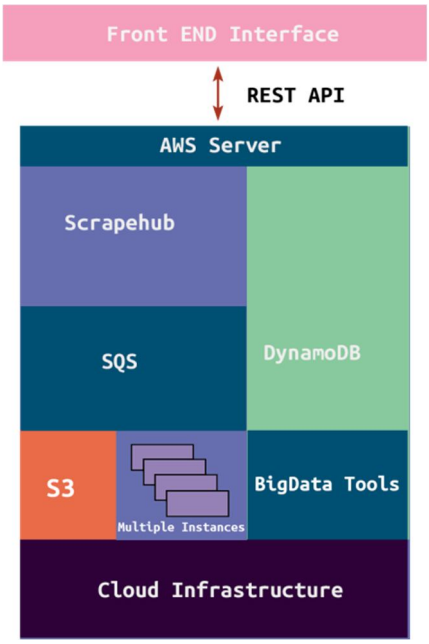

Fig. 2: Scrapper Ecosystem

Our solution tackles the aforementioned challenges by leveraging Amazon Elastic Compute Cloud (EC2) [18] services along with Simple Queue Service (SQS) [14] and Simple Storage Service (S3) [14]. This approach allows us to utilize precisely the required amount of resources and seamlessly scale based on demand. Amazon S3 is employed for storing Amazon Machine Image, which facilitates the creation of instances of EC2 virtual machines. Through EC2, we can dynamically generate multiple instances of virtual computing machines as needed. The cloud-based model offers flexibility with elastic resources and proves to be cost-effective,given Amazon's pay-per-use charging model, which is more economical than establishing and maintaining private resources.

The architecture of the proposed web scraper is depicted in the figure. The web browser serves as the front-end platform connected to the Amazon Web Server in the cloud via REST API. The cloud-residing scrape-hub manages all client requests for scraping, in addition to handling and monitoring events. It also determines the requisite number of computing instances based on URL chunks. Scrape-hub utilizes Amazon SQS to maintain a list of URLs slated for scraping, managing two queues—one for incoming requests from the scraper hub and another for responses. It serves as an interface in cases where the scrape engine cannot process every request.

Scrape-hub segments URLs into chunks and stores them in an S3 bucket, awaiting retrieval by a scrape engine instance. The scrape engine, a program receiving a URL as input, performs the scraping of the page's content residing in an EC2 instance. Each EC2 instance serves as a virtual machine running the scraping engine, and the number of instances to be created is determined by scrape-hub, allocating a machine instance for a specified number of URLs as per user configuration.

DynamoDB [12], an open-source cross-platform documentoriented NoSQL database program, serves as the storage solution for the scraped data. When a user initiates a scraping request on the cloud-based platform, the request is directed to scrape-hub, containing essential information: the URLs to be scraped and the scraper configuration. Initially, scrape- hub generates machine instances based on the provided URLs, divides the URLs into multiple chunks, and dispatches them to the S3 bucket. Concurrently, SQS request and response queues are established, each assigned a unique name or ID. The scrape-hub then initiates a service request to EC2 machines utilizing the SQS request queue, processing requests in a First-In-First-Out (FIFO) scheduling manner. Each task in the request queue holds a specific ID, and scrape-hub maintains metadata about the generated tasks and their respective statuses.

Throughout the scraping process, scrape-hub continually monitors progress through the response queue, where machines communicate the results of the scraping. CloudWatch [5] can also be employed for monitoring various Amazon services. The final results are stored in DynamoDB, and users have the option to instantly download the outcomes in CSV, TSV, or JSON formats.

Figure 3 illustrates the nucleus of the Scrape Engine. Users initiate a scraping service request through the UI, directing it to scrape-hub, wherein they provide the URLs and their corresponding configurations. These configurations encompass Selenium configurations for the respective URL, specifying mouse events and keyboard events, along with Parser configurations such as XPath, CSS selector, and Regex.

The URLs and configurations are stored in the database in

JSON format, and scrape-hub generates a request in the SQS the page. When the required page is reached, python requests library is used to extract

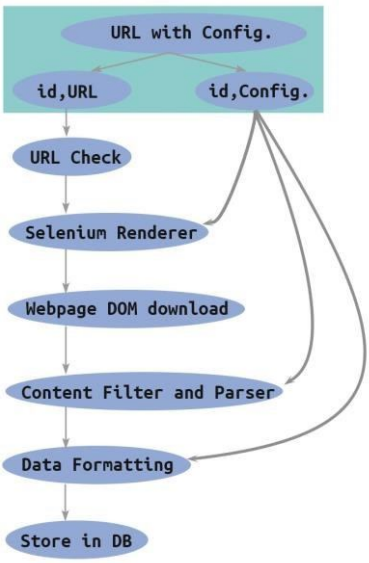

Fig. 3: Scrape Engine

The Scrape Engine reads the URLs and configurations from the JSON file, subjecting the URLs to a validity check in the URL Check component. Invalid URLs are promptly reported to users, while valid ones proceed to the Selenium Renderer. The Selenium Renderer employs a web driver to render the specified URL, utilizing Selenium Configuration to execute crucial steps like page automation and reaching the desired state of the page. Once the requisite page is attained, the Python Requests library comes into play, extracting the Document Object Model (DOM), which is subsequently stored in the database.

For parsing, various element selectors are applied using the HTML parser library to extract the necessary content from the DOM. The extracted content is then formatted in accordance with the configuration, filtered as per specifications, and ultimately stored in the database.

Scrape-hub dynamically generates multiple instances of EC2 machines based on the load of URLs to be scraped, typically grouping URLs in predefined batches, such as 40, 50, or 60 per instance. These machines are strategically distributed across different geographical locations, enhancing the effectiveness of the scraping process. The geographical distribution serves a dual purpose: circumventing captcha challenges and overcoming geographical restrictions imposed by certain websites. Figure 4 illustrates the creation of multiple machines and their communication within the system. Scrape-hub places tasks in the request queue, and each task is then assigned to an instance of the machine.

Our architecture exhibits scalability and versatility, accommodating not only scraping tasks but also serving as a platform for big data applications. Integration within the same cloud architecture or as a dedicated scraping tool is possible. Additional applications, such as data cleaning or transformation, can be seamlessly executed by integrating tools into scrape-hub. For computation, the same EC2 machines are utilized, and DynamoDB is employed for storage. The architecture readily supports the implementation of MapReduce [11], a distributed computation technique, using EC2 machines. Both MapReduce and data cleaning applications are seamlessly

integrated into the architecture, accessible through scrapehub. The extracted data can then be utilized by big data applications for visualization and analysis.
The creation of EC2 machines is tailored to the specific analytical requirements or the volume of data to be analyzed. Exploring additional applications for organizational benefits represents an area of potential growth and development.
Upon completion of the entire scraping process, users have the option to download the results in various formats, including CSV, TSV, plain text, XML, or HTML. The outcome is stored in DynamoDB, providing a convenient and accessible repository for the scraped data.

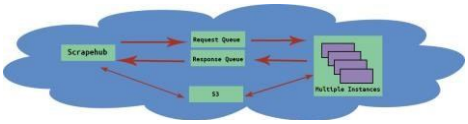

Fig. 4: Instances of Scrape Engine in Cloud

IV. Experimental Setup

In this experimental setup, we deployed multiple instances as parallel workers, where each instance represents an AWS (EC2) machine. All nodes collaboratively utilize AWS DynamoDB, S3, and SQS as shared resources. These machines receive inputs from the request queue, containing split URLs. The nodes work concurrently to scrape the provided URLs, and the resulting data is stored in DynamoDB. The web interface facilitates client interaction, allowing them to input a list of URLs and create virtual machine instances within AWS. Two crucial constraints influencing the architecture's performance include the total number of machine instances that can be created and the allocated URLs per machine.
Our performance analysis involved evaluating the system's efficiency as the number of URLs to be scraped increased.
Test scenarios included various URLs from platforms like
Amazon,
Google Shopping, and other e-commerce sites. File chunks containing 50 URLs each were utilized, enabling each machine to scrape 50 websites. Selenium served as the web renderer, and the request library was employed for downloading website content. To mimic human behavior and avoid captchas, a specific delay was introduced in Selenium, causing a slight slowdown in the process.
The architecture underwent testing on Amazon's cloud services, encompassing experiments with a single local node, 20 machines, and 30 machines utilizing Amazon's EC2 general-purpose T2 instances. Image, text, and URLs were extracted from the specified websites using XPath for the required items. The t2.nano [6] instance type, the smallest provided by EC2, was employed for this experiment. This instance type features 1 High-Frequency Intel Xeon Processor vCPU and 0.5 GiB memory, offering both cost-efficiency and burstable performance capabilities when necessary.

V. Result and Analysis

The graph depicted in Figure 5 illustrates the time required by the scraper to complete the scraping process. On the xaxis, we have the number of URLs, while the y-axis represents the time taken by the scraper in minutes. The graph delineates the scraping time of the proposed cloudbased scraper as the number of URLs increases, reaching up to 60,000. Different lines on the graph denote the scraping times using 20 machines, 30 machines, the noncloud implementation of the same architecture using a single computing machine, and the non- cloud implementation utilizing multiple threads on a single node.
When employing a single-node local computer, the scraping efficiency was notably hindered, struggling to efficiently scrape over 4,000 web pages, possibly due to captcha restrictions where servers limit a single IP's access to a specific amount of data within a limited time. A similar challenge was encountered in the non-cloud implementation using concurrent threads on a single node. While the computation speed was faster than the sequential approach, the scraper's efficiency diminished as the number of URLs increased. The cloud-based scraper addresses this challenge by leveraging multiple geographically distributed scraping machines.
It is evident from the graph that employing more machines results in reduced computing time, but this advantage comes with a trade-off- an associated cost, which is a crucial and limited resource for individuals or organizations.

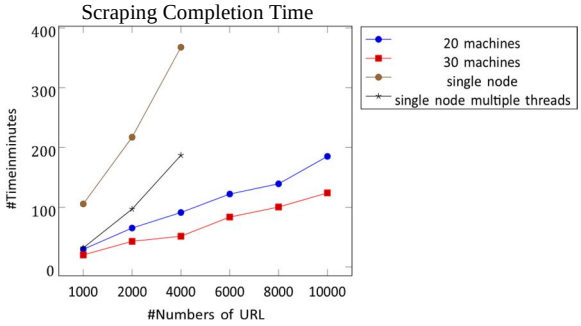

Fig. 5: Average Scraping Time

The observed linear trend in scraping time across both lines suggests that the architecture operates effectively with a gradual rise in the number of URLs to be scraped. This graph serves as a tool for assessing the performance and scalability of the proposed scraper. Based on the experiment, it can be deduced that the cloud-based architecture exhibits scalability, demonstrating its ability to handle an increased workload as the number of URLs to be scraped grows. The scraping time reflected in the graph encompasses various stages, including request time, queuing time, processing time, and response time from the cloud.

Moreover, it's worth noting that the scraping task could potentially be expedited by utilizing alternative libraries [13] known for faster performance in the scraper engine compared to the one employed in this particular experiment.

A. Comparison with other systems

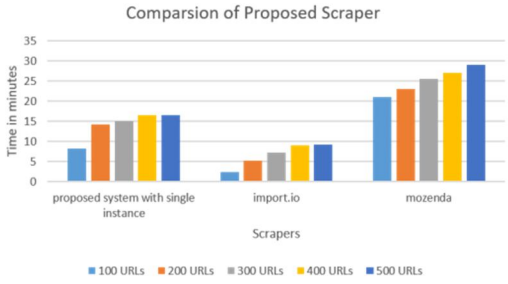

Fig. 6: Scraper's in cloud

Numerous cloud-based web scraping solutions exist, with many being commercial and lacking transparency in their working models. To offer a comparative analysis, our proposed system was evaluated against existing open-source models like Scrapy, webextractor360[10], webscraper.io, and import.io. The graph presented earlier compares the performance of our system with import.io and Mozenda, utilizing only one machine instance for the experiment. Some of the existing models follow a centralized computing model, which may become a bottleneck in certain scenarios. For instance, Scrapy Cloud only provides 10 parallel agents for scraping[16]. In contrast, our distributed model imposes no limitations on the instances of computing resources, offering flexible utilization facilitated by DynamoDB in the cloud.

TABLE I: FEATURE COMPARISION

Scraper	Coding	Output Format	Methods	Extracts
Import.io	No	CSV,Excel, Json	browser extension, webpage	text, image, URL, table
Proposed	No	TSV	webpage cloud	text, image URL, table
Mazendo	No	CSV,TSV, Json,XML	webpage software	text, image pdf, table
Scrappy	Yes	CSV,TSV, Json,XML	Web,Software, Cloud	text, image, URL, table

While our scraping time may be slightly higher than that of import.io, our scraper excels in efficiency and reliability, particularly with the integration of Selenium. Selenium's proficiency in mimicking human behavior

not only provides flexibility in designing scraping patterns but also aids in circumventing captchas. The scraping performance of our model is comparable to other market- present scrapers in terms of data extraction.

B. Scalability of proposed Cloud-based Scrapper

Scalability is characterized by the capacity to manage a growing workload efficiently through the repeated application of a cost-effective strategy[9]. Leveraging the cloud provides access to unlimited hardware resources on demand, enabling scalability through both scaling out and scaling up. Software components deployed in the cloud further contribute to the system's scalability. The experiment conducted in the previous section, where scraping time was examined concerning varying numbers of virtual machines and URLs, indicates that our system is adept at handling increasing scraping workloads (number of URLs) seamlessly, without necessitating modifications to the software or code. Our proposed architecture maximizes the utilization of underlying resources to efficiently manage escalating scraping workloads.

C. Advantages of Cloud-based Scrapper

Transitioning from traditional methods to the cloud brings forth numerous advantages, and some of them are outlined below:

Cost-efficient
The cloud's pay-per-use model offers a more economical alternative compared to maintaining and managing one's own resources. The expenses associated with acquiring, maintaining, and scaling up resources based on demand are considerably higher in traditional methods. Cloud resources, on the other hand, provide a cost-effective solution, allowing users to scale up or down as needed. Cloud vendors offer a comprehensive range of services, covering everything from storage to processing capabilities. With the cloud, scaling becomes a straightforward task, eliminating the complexities associated with traditional methods.

Elastic Resources
The cloud offers a dynamic approach to resources, encompassing computing power and storage. There's no necessity to invest in all the resources anticipated for future needs. This flexibility enables users to initiate with minimal resources and scale up as requirements evolve. Our architecture is designed to seamlessly adapt to the evolving needs of users.

High Performance
Cloud vendors furnish dependable computing machines configured to meet specific requirements. Users have the flexibility to create and utilize multiple instances of these machines based on their needs. Vendors offer an extensive array of options, allowing users to select the appropriate processing power tailored to their performance requirements for computing tasks.

D. Advantages of using Selenium tool

Despite the potential introduction of a delay in scraping time, the use of Selenium brings forth numerous advantages that outweigh its drawbacks. Selenium is an open-source, free, cross-platform tool utilized by major online enterprises[15]. Offering support for multiple browsers and programming languages, Selenium stands out as a versatile web automation tool. Its capability to mimic human behavior on the web is a primary factor in selecting Selenium for our tool. Additionally, Selenium supports the use of web browsers running on remote machines through Selenium Grid and can effectively handle dynamically rendered pages with JavaScript.

E. Big Data Support

Any comprehensive big data application necessitates four essential infrastructure components: storage, processing, analytics software, and networking[19]. Our cloud-based scraper architecture, designed as a scalable system, satisfies all the prerequisites for the implementation of big data. The elastic resources available on demand ensure an abundance of resources for analytics software.
DynamoDB, serving as a NoSQL database, contributes to higher availability, resulting in enhanced performance. The architecture allows the programming of analytics software tailored to the user's specific needs.

VI. CONCLUSION

In conclusion, cloud computing emerges as an advanced and superior platform for web scraping applications. The proposed web scraper demonstrates robust data extraction capabilities from the web, leveraging the elastic resources provided by the cloud for computing and storage in a distributed environment on-demand. Scalability

and performance were evaluated through experiments conducted on Amazon's cloud services, specifically utilizing EC2, S3, and SQS for scraping websites with multiple instances of virtual computing machines. It's worth noting that this architecture can be implemented with other cloud service providers.
For users primarily concerned with performance and not requiring web automation, we recommend exploring alternatives to the Selenium web renderer. The architecture, being entirely scalable, lends itself well to the implementation of big data applications. It includes nearly all the resources necessary for comprehensive big data solutions. The advantages of our architecture were discussed, and a brief comparison was drawn with traditional and other web scraping architectures.

ACKNOWLEDGEMENT

We would like to thank Prof. Ajay Kumar for his encouragement throughout this research, as well as for his direction and support. We would also like to think of our friends and family as they too have supported us to accomplish this goal.

REFERENCES

[1] Interesting facts of big data. https://www.waterfordtechnologies.com/ big-data-interestingfacts. Accessed: 2017-08-10.
[2] Visual scraping using browser extensions. https://data-lessons.github.io/ library-webscraping/03-visual-scraping. Accessed: 2017-08-5.
[3] K Alhamazani. An overview of the commercial cloud monitoring tools: Research dimensions, design issues, and state-of-the-art. 2013.
[4] Inc. Amazon.com. Amazon ec2 instance types. https://aws.amazon.com/ ec2/instance-types. Accessed: 2017-08-7.
[5] Osmar Castrillo-Fernndez. Web scraping: Applications and tools. European Public Sector Information Platform, pages 13–15, 2007.
[6] John B. Goodenough Charles B. Weinstock. On system scalability, cmu/sei-2006-tn-012. March, 2006.
[7] SCM de S Sirisuriya. A comparative study on web scraping. 8th International Research Conference, KDU, pages 135–140, November 2015.
[8] Jeffrey Dean and Sanjay Ghemawat. Mapreduce: Simplified data processing on large clusters. OSDI'04: Sixth Symposium on Operating System Design and Implementation, December, 2004.
[9] Giuseppe DeCandia. Dynamo: Amazons highly available key-value store. Amazon.com, 2007.
[10] EliteDataScience. 5 tasty python web scraping libraries. https:// elitedatascience.com/python-web-scraping-libraries, Oct 28, 2016. Accessed: 2017-08-7.
[11] Simson Garfinkel. An evaluation of amazon's grid computing services: Ec2, s3, and sqs. Harvard Computer Science Group Technical Report, TR08-07, 2007.
[12] Optimus Information. Selenium testing: Advantages and disadvantages. http://www.optimusinfo.com/ seleniumtestingadvantagesanddisadvantages, December 22,2015. Accessed: 2017-08-10.
[13] Zixuan Zhuang Mehdi Bahrami, Mukesh Singhal. A cloud-based web crawler architecture. 18th International Conference on Intelligence in Next Generation Networks,978-1-4799-1866-9, pages 216–223, 2015.
[14] Neha Goyal Monika Yadav. Comparison of open source crawlers(a review). International Journal of Scientific and Engineering Research,2229-5518, pages 1544–1551, September 2015.
[15] Dr C K Kumbharana Prof Vaibhav A Gandhi. Comparative study of amazon ec2 and microsoft azure cloud architecture. International Journal of Advanced Networking Applications,0975-0290, pages 117– 123.
[16] FedTech Staff. 4 infrastructure requirements for any big data initiative. https://fedtechmagazine.com/article/2016/12/ 4-infrastructurerequirements-any-big-data-initiative, December,2016. Accessed: 201708-03.