Cloud Based Web Scraping for Big Data Applications

Conference Paper · November 2017

DOI: 10.1109/SmartCloud.2017.28

Ram Sharan Chaulagain
Tribhuvan University

Santosh Pandey
Stevens Institute of Technology

Subarna Shakya
Tribhuvan University

Cloud Based Web Scraping for Big Data Applications

Ram Sharan Chaulagain*, Santosh Pandey†, Sadhu Ram Basnet‡, Subarna Shakya§
Department of Electronics and Computer Engineering
Institute of Engineering(IOE), Tribhuvan University
Lalitpur, Nepal
Email: (*tangent.rams,†santosh.pandey2222, ‡sadhupaper)@gmail.com, §drss@ioe.edu.np

***Abstract*—With the penetration of new technologies, there is a rapid growth of internet users and data (mostly unstructured) generated by those users on the internet. As scraping is one of the major sources for extraction of unstructured data from the Internet, we have analyzed the scraping process when introduced to a bulk of data extraction. We faced several challenges while scraping large amount of data, such as encountering captcha, storage issue for a large volume of data, need for intensive computation capacity and reliability of data extraction. In this paper, we investigate cloud-based web scraping architecture able to handle storage and computing resources with elasticity on demand using Amazon Web Services(Elastic Compute Cloud and DynamoDB). Our solution tries to address both scraping and feasibility for big data applications in a single cloud-based architecture for data-based industries. We discuss selenium as one of our tool for web scraping because of web drivers it supports which simulates a real user working with a browser. We also analyze the scalability and performance of the proposed cloud-based scrapper and describe the advantages of the proposed cloud-based scraping over other cloud-based scrapers.**

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

I. INTRODUCTION

The presence of an abundance of data in World Wide Web can be both blessing and curse. A lot of relevant information as an enormous source is presented; however, it is challenging to acquire that information. Most information on the web is presented as HTML document, a sketch to accord information to humans, not for computers and such data are Unstructured data [1]. The data growth rate on the internet is soaring every year and the data is predominantly unstructured [3]. Since unstructured data do not follow any data model, information churning is not easy. Extraction of a large volume of data from the internet and saving for future work is our concern. Such extraction can be realized using software solution called Web Scraping Software. We introduce a cloud-based scraping architecture for unstructured data acquisition from the web.

Data acquisition is a preliminary job for any research and development related to data science, the step where we get the data either from the private sources like company's sales reports, financial reports or public sources like journals, websites like opendata or by purchasing data. Likewise, Web Scrapper technology or simply copy-pasting from the web browser are the sources of data. The concept of Scraping can be decomposed and studied as Web Renderer Automation, Parser, Filter and Formatting of results and Storage. Likewise, handling massive requests of scraping can be studied under the domain of Distributed Computing. Scrappers can be designed using various libraries like BeautifulSoup, Scrapy, and Requests[7]; however, they work only on static pages. We select Selenium and web driver API as a tool for automating web pages especially used in web application testing. Similarly, HTMLParser library of Python, which serves as the basis for parsing text files formatted in HTML and XPath, language for finding any element on the web page using XML path expression, is used to extract data from the HTML content downloaded using Requests library [2].

We will be using EC2(Elastic compute cloud) of Amazon web services for implementing our architecture in the cloud. The reason behind choosing EC2 is due to its flexibility for scalable deployment of applications by using instances of the virtual machine per requirement. Those instances can be created, launched and terminated as per requirement(hence the term elastic). Our architecture will also integrate functionality required for big data applications like analyzing data and visualizing information from data. We will be deploying our application over amazon web server for handling both scraping and big data tools. Unlike other web scrapers, our cloud-based scraper based on selenium provides us better efficiency on data extraction due to its human mimicing web automation methodology. It is flexible and more scalable than other present scrapers as it is designed with the capability of launching unlimited numbers of parallel instances in the cloud, modular design and customization ability for the steps of scraping. The rest of the paper is organized as follows: in Section II, we discuss related works of existing cloud-based web scraper systems. In Section III, we talk about our methodology including a background of traditional scrapers. We also introduce our proposed cloud-based web scraper, present its architecture and its implementation in Amazon Web Services. In In Section IV, we provide results of our experiments . In Section V, we discuss results and analysis from the experiments. We also discuss advantages of the cloud-based scraper, comparison with other existing scrapers. In Section VI, finally, we conclude our proposed research.

II. BACKGROUND AND RELATED WORK

Cloud Computing, due to its capability of highly intensive computing and support for elastic resources with pay-per-use (on demand) model, makes it the best choice for individual business to run their own scraping starting with a small resource. Web Scraping, a form of web content mining, is also termed as web harvesting or web data scraping. Various prominent cloud-based tools and software can be found for web-scraping both free(for the trail) and paid. Some of the cloud-based scraping tools are briefly introduced below.

A. Import io

Import io [8] is a web-based platform for scraping data from different websites without coding. It provides scraping as a SaaS model in the cloud. All extracted data is stored on its cloud server. It allows us to download the data as CSV, Excel, Google Sheets, JSON or accessed via API. Scraping can be integrated into third-party analytics and visualization software using API. Its distinctive feature includes automatic data and image extraction, auto page linking, monitoring, scheduling and no need of coding.

B. Webscraper.io

Webscraper.io[4] is another Visual 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 (sitemap) implying how and what to scrape.

C. Scrapy

Scrapy [17] is a free, open-source and collaborative framework for web scraping which can also be used to extract data using APIs. It can be used for both web scraping and web crawling. Scrapy is written in Python. It can be deployed in Scrapy Cloud which allows us to scale on demand. Its feature includes visual or code option, wise resource management, full API access etc.

Present existing solutions mainly focus on scraping or data acquisition only which is the first step of big data. The data collected from scraping needs to pass through multiple processes which are not accounted on the above-mentioned system. We propose to build a cloud-based distributed scraping model taking account of further processes of big data i.e. a single system capable of scraping and working on scraped data on the cloud using parallel machines.

III. METHODOLOGY

A. Scraper Background

Figure 1: shows the overview of a normal scraper where the client provides URLs to scrape with the configuration to scrape-hub. The scrape-hub program initiates, handles and monitors all tasks in the system. It is written in python language. Then scrape-hub starts the scraping engine, the task of which is to initiate a web page, automate it to the desired state using selenium library and parse it using request library. Different scrapers may use a different library for parsing and automate web pages. Here, we have mentioned only selenium and requests. All data are stored in the database. This model represents the traditional sequential method having the capability of scraping one site at a time only. This model is able to extract limited data on a given period of time.

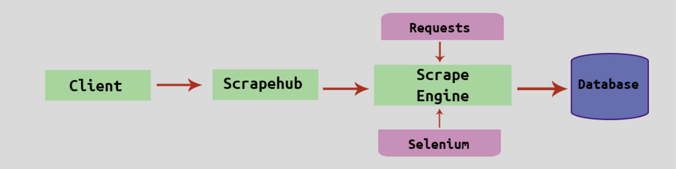

Fig. 1: Normal Scrapper

The limitation can be removed by adding more resources for computing and storage but it will be costly and difficult to maintain with increasing demand. This solution would not be flexible.

B. Cloud-based Scraper

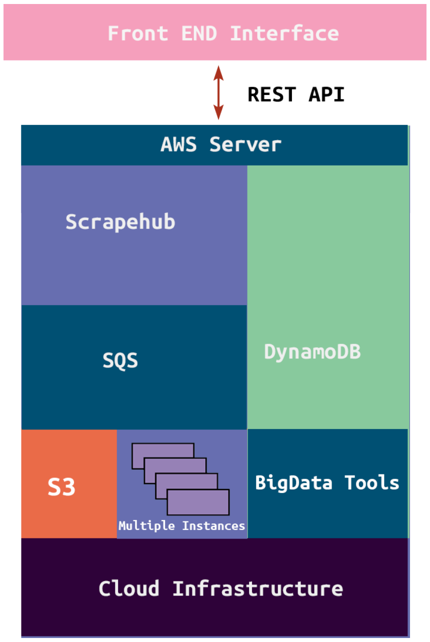

Fig. 2: Scrapper Ecosystem

Our solution addresses this problem using *Amazon Elastic Compute Cloud (EC2)* [18] services with *Simple Queue Service (SQS)* [14] and *Simple Storage Service(S3)* [14] which will enable us to use only the required amount of resource and scale on demand. Amazon S3 is used for storing Amazon Machine Image which creates an instance of the EC2 virtual machine.Using EC2 we will be able to create as many instances of virtual computing machines as needed. Cloud-based model is flexible with elastic resources and cost-effective. Amazon charges using pay per use model which is cheaper than creating own private resources. The figure illustrates the architecture of proposed web scraper. *Front-end platform* for the architecture will be *web-browser*. Front-end is connected to amazon web server in the cloud using *REST API*. *scrape-hub*, residing at cloud will handle all client requests of scraping as well as handling and monitoring events. It is also responsible for assigning the required number of instances for computing depending upon URLs chunk. scrape-hub uses amazons SQS for maintaining a list of URLs to be scraped. It handles two queues; one is *request queue* which maintains requests from scraper hub and another is *response queue*. It acts as an interface when *scrape engine* is not able to process every request. URLs are divided into numbers of chunk by scrape-hub and stored in S3 bucket, waiting to be fetched by an instance of *scrape engine*. *Scrape engine* is a program which takes URL as input and scrapes the content of page residing in EC2 instance. Each instance of EC2 is a virtual machine running the scraping engine. Instances of scrape engine to be created is decided by scrape-hub, allocating an instance of the machine per certain numbers of URL's decided by the user.

DynamoDB[12], a free and open-source cross-platform document-oriented NoSQL database program provides storage for scraped data. When any user initializes scraping request to the cloud-based scraper, the request goes to scrape-hub. The request consists of two information: URLs to scrape and configuration of the scraper. At first, scrape-hub creates an instance of machines depending on URLs, then divides the URLs to multiple chunks and forwards them to S3 bucket. SQS request queue and response queue are created and are given a name or id. Concurrently, it generates a service request to EC2 machines with the help of SQS request queue. The order of processing request is done in First-In-First-Out(FIFO) scheduling fashion. Each task on the request queue created by scrape-hub has a specific id. Scrape-hub stores metadata about the tasks generated and their status. It continuously monitors the progress of overall scraping through response queue where the machines reply the result of the scraping. CloudWatch[5] an also be used for monitoring Amazon services. The result is then stored in DynamoDB and users can download results instantly in CSV, TSV or JSON format.

Figure 3 shows the core of *Scrape engine*. Users request scraping service to scrape-hub using UI, providing the URLs and their configurations. The configurations contain the respective URL selenium configurations like mouse events and keyboard events and Parser configurations like XPath, CSS selector and Regex. *urls and Configurations* are stored in the database in JSON format and a request is generated in SQS by scrape-hub. Scrape engine reads URLs and Configuration from JSON file. URLs are then introduced to *URL Check* where URLs are checked for their validity. The invalid URLs are reported to users and valid URLs are forwarded to Selenium Renderer. Selenium Renderer uses web driver to render the given URL. Selenium Configuration is provided to web driver for various essential steps like page automation and reaching the required state of the page. When the required page is reached, python requests library is used to extract the DOM(Document Object Model) and stored in the database. Various element selectors for the parser are used with HTML parser library to extract the required content from the DOM. The contents are then formatted according to the configuration, filtered and stored in Database.

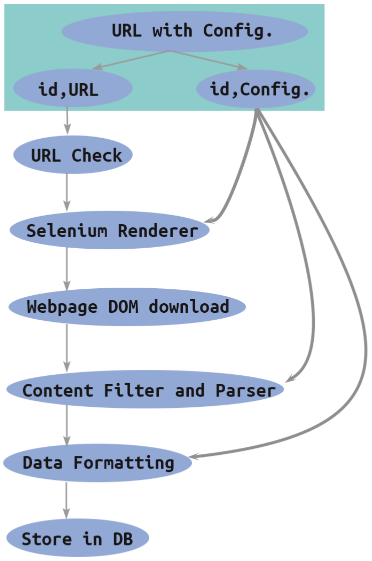

Fig. 3: Scrape Engine

Multiple instances of EC2 machines are created by scrape-hub depending upon a load of URLs to scrape. Basically, URL are grouped in pre-defined fashion i.e. 40,50,60 per instance at a time. The machines are geographically distributed so that scraping could be done more effectively. Geographical distribution helps us in two ways, one is avoiding captcha and another is overcoming a geographical restriction issue of some websites. Figure 4 shows how multiple machines are created and how they communicate. scrape-hub puts the tasks in the request queue. The task is then assigned to one of the instances of the machine. Our architecture can easily support for big data applications with some integration on the same cloud architecture or as a tool for scraping only. Applications like data cleaning or data transformation can be performed by integrating tools in scrape-hub. For computation, same EC2 machines and for storage DynamoDB can be used. Map Reduce[11] distributed computation technique can be easily implemented using EC2 machines. MapReduce and data cleaning application are integrated into the architecture which can be accessed using scrape-hub. The extracted data are used by big data applications for visualization and analysis. EC2 machines are created according to the requirement or the amount of data to be analyzed. More exploration of other useful applications for an organization could be a fruitful work.

After all scraping is completed, result could be downloaded in CSV,TSV,text,XML or HTML format. The result will be stored in dynamoDB.

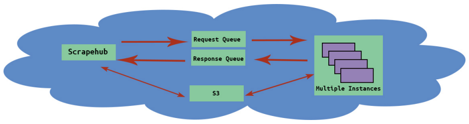

Fig. 4: Instances of Scrape Engine in Cloud

IV. Experimental Setup

In this experiment, we assigned multiple instances as parallel workers. Each instance is AWS(EC2) machine. All nodes share AWS DynamoDB, S3, and SQS as shared resources. These machines accept inputs from the request queue where URLs split are provided. The nodes parallelly scrape the URLs and results are stored in DynamoDB. The web interface is designed for the client to input URL list and create the virtual machine instances in AWS. Two constraints especially affect the performance of the architecture. They are total instances of the machine that can be created and allocated URLs per machine. We analyzed the performance of the system with increasing number of URLs to scrape. A number of URLs of Amazon, Google shopping, and other e-commerce sites were included in the test. Chunks of the file containing 50 URLs were used so that each machine scrapes 50 websites. Selenium was used as web renderer and request library was used to download the content of the website. The certain delay was used in selenium to mimic human behavior for avoiding captcha which made it little slower. The architecture was tested over Amazon's cloud services. The experiment was performed using single local node,20 machines and 30 machines of Amazon's EC2 general purpose T2 instance. Image, text, and URLs were extracted with the scraper from above-mentioned websites using XPath of the items required. *t2.nano*[6] an instance of EC2 were used for this experiment. It is the smallest instance provided by EC2 which can also provide burstable performance if needed. The lowest-cost general purpose instance type has 1 High-Frequency Intel Xeon Processor vCPU and 0.5 GiB memory.

V. Result and Analysis

The graph in figure 5 shows the time required by the scraper to complete the scraping. The x-axis represents the number of URLs and the Y-axis represents the time required by the scraper to complete scraping in minutes. It demonstrates the required scraping time of the proposed cloud-based scraper with the increase in a number of URLs up to 60,000. The lines shown in the graph represents the scraping time using 20 machines, using 30 machines, in the non-cloud implementation of the same architecture using single computing machine and non-cloud implementation using multiple threads on a single node. Using a single-node local computer, data couldn't be scraped efficiently over 4000 web pages which could be due to captcha as servers restrict a single IP to access limited data in limited time. Same goes with the implementation of concurrent threads in a single node. Though computation is much faster than the sequential node, the efficiency of scraper decreases with increase in URLs. Our cloud-based scraper approaches this problem with the use of multiple geographically distributed scraping machines. We can see that more machine yields less computing time, but comes with a cost, which is an important and limited resource of any person or organization.

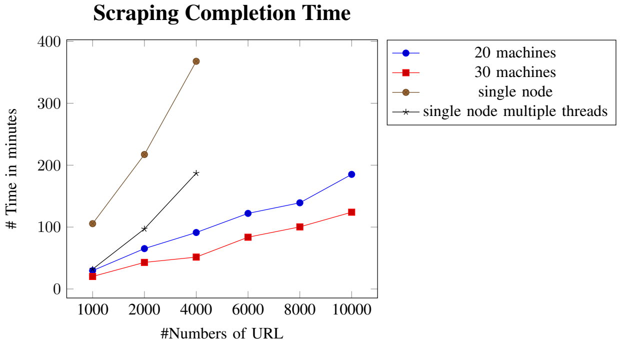

Fig. 5: Average Scraping Time

The scraping time in both lines is approximately linear. This indicates that the architecture is performing normally with the gradual increase in the numbers of URLs to scrape. The graph is used to analyze the performance and scalability of the proposed scraper. It can be inferred, from above experiment, that the cloud-based architecture is scalable with an increase in URLs to scrape. Scraping time indicated in the graph includes request time, queuing time, processing time and response time from the cloud. The scraping task could be made faster with the use of other libraries[13] famed for faster performance for scraper engine than we used here.

A. Comparison with other systems

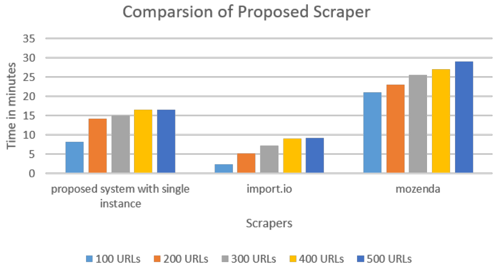

Fig. 6: Scraper's in cloud

Many cloud-based web scraping solutions are available but most of them are commercial whose working model is not public. So, a comparison is done with some existing open-source models such as Scrapy, webextractor360[10], webscraper.io, and import.io. The comparison of the performance of our proposed system is compared import.io and mozenda in the graph showed above. Only one machine instance was used for the experiment. Some of these models follow centralized computing model which can be a bottleneck sometimes. Scrapy Cloud provides only 10 parallel agents for scraping[16]. Our model, compared to other models, is distributed and there

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

TABLE I: Feature Comparision

is no limitation on instances of computing resource that can be used. All resources can be used flexibly with the help of dynamoDB in the cloud. Although our scraping time is more than that of import.io, our scraper provides more efficiency and reliability for scraping with the use of selenium. As selenium is better at human behavior mimicing, it provides flexibility in designing scraping pattern and avoiding captcha. Its scraping performance is comparable with other scrapers, currently present in the market, in terms of data extraction. Our model can be coupled with other big data applications as data acquisition is only the first step of big data. Eventually scraped data will enter the big data processes. So we have made our architecture compatible to use with other open source big data applications. It allows the access of data stored in Dynamodb from scraping and creation of machines for data analysis(i.e MapReduce). The table I compares the features provided by various scrapers.

B. Scalability of proposed Cloud-based Scrapper

Scalability is defined as the ability to handle increased workload by extending a system's capacity repeatedly applying a cost-effective strategy[9]. With the cloud, we get unlimited hardware resources on demand for scaling out or scaling up. The software components, deployed on cloud, also support for system's scalability. The experiment performed on the above section, showing scraping time with respect to a various number of virtual machines and URLs, infers our system capable of handling increasing scraping workload(numbers of URLs) normally without any changes in the software or code. Our proposed architecture utterly utilizes the underlying resources for handling increased scraping workload.

C. Advantages of Cloud-based Scrapper

There are lots of advantages while switching to cloud from traditional methods. Some of them are described below.

1) Cost-efficient: With cloud pay-per-use model, it will be cheaper to use cloud resources rather than maintaining your own. Expenses of buying own resources, maintaining it and scaling it up with demand are too much higher. So, it is reasonable to use cloud resources for cheap prices with the facility to scale up or down on demand. Cloud Vendors provides almost everything that we may need from storage to processing capability. Scaling won't be a tedious task with the cloud.

2) Elastic Resources: Cloud provides flexibility with the resources that we need including computing power and storage. Its not required to buy all the resources that may be required in coming future. It allows us to start small and scale as we will need later on. Our architecture is capable of adapting to the changing requirements of a user.

3) High Performance: Cloud Vendor provides reliable computing machines with the configuration as we need. We can create and use multiple instances of this machines per requirement. Vendors provide a large list of options for choosing our platform. We can choose the right processing power according to our performance needs for computing.

D. Advantages of using Selenium tool

Although the use of selenium may introduce a certain delay in scraping time, its shortcomings are surpassed by the advantages. It is open source, free, cross-platform and is used by major online enterprises[15]. Selenium supports multiple browsers and programming languages. As it is mostly used as web automation tool, it is capable of mimicking human behavior on the web which is the main reason behind choosing selenium as our tool. Selenium also supports to use web browsers running on remote machines using selenium grid and can cope with javascript rendered dynamic pages.

E. Big Data Support

Any big data application requires four infrastructure: storage, processing, analytics software and networking[19]. Our cloud-based scraper architecture, a scalable system, fulfills all the requirements for implementation of big data. With elastic resources on demand, we can have abundant resources for analytics software. DynamoDB, a NoSQL database, provides higher availability resulting in a better performance. Analytics software can be programmed on this architecture according to the needs of a user. With Amazon's EC2, we can scale our machines both on quantity and computing power. Although this architecture does not complete all big data requirement, further program integration and amendment can make it more robust for big data tasks.

VI. CONCLUSION

It can be concluded that a sophisticated technology like cloud computing provides a better platform for the web scrapper. Proposed web scrapper provides reliable data extraction from the web. Cloud provides elastic resources for computing and storage in a distributed environment on demand. We tested the scalability and performance of the architecture with the experiment above on top of Amazon's cloud services. We analyzed the proposed cloud-based web scraping architecture implemented using Amazon's EC2, S3 and SQS service for scraping websites using multiple instances of virtual computing machines. This architecture can also be implemented using other cloud service provider. We suggest using the alternative of selenium web renderer if better performance is the only requirement without web automation. As it is based on completely scalable resources, big data applications can be implemented over this architecture. The proposed architecture includes almost all resources required for big data solutions. We discussed advantages of our architecture and briefly compared it with traditional/other web scraping architecture.

ACKNOWLEDGEMENT

We would like to thank Prof. Dr. Subarna Shakya for his encouragement throughout this research, as well as Saghan Mudhbhari 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-interesting-facts. Accessed: 2017-08-10.

[2] Python requests library. http://docs.python-requests.org/en/master/. Accessed: 2017-08-10.

[3] Unstructured data. https://en.wikipedia.org/wiki/Unstructured_data. Accessed: 2017-08-12.

[4] Visual scraping using browser extensions. https://data-lessons.github.io/library-webscraping/03-visual-scraping. Accessed: 2017-08-5.

[5] K Alhamazani. An overview of the commercial cloud monitoring tools: Research dimensions, design issues, and state-of-the-art. 2013.

[6] Inc. Amazon.com. Amazon ec2 instance types. https://aws.amazon.com/ec2/instance-types. Accessed: 2017-08-7.

[7] Vinay Babu. Web scraping and data analysis using selenium webdriver and python. http://www.datasciencecentral.com/profiles/blogs/web-scraping-and-data-analysis-using-selenium-webdriver-and, July 2,2016. Accessed: 2017-08-1.

[8] Osmar Castrillo-Fernndez. Web scraping: Applications and tools. *European Public Sector Information Platform*, pages 13–15, 2007.

[9] John B. Goodenough Charles B. Weinstock. On system scalability, cmu/sei-2006-tn-012. March, 2006.

[10] SCM de S Sirisuriya. A comparative study on web scraping. *8th International Research Conference, KDU*, pages 135–140, November 2015.

[11] Jeffrey Dean and Sanjay Ghemawat. Mapreduce: Simplified data processing on large clusters. *OSDI'04: Sixth Symposium on Operating System Design and Implementation*, December, 2004.

[12] Giuseppe DeCandia. Dynamo: Amazons highly available key-value store. *Amazon.com*, 2007.

[13] EliteDataScience. 5 tasty python web scraping libraries. https://elitedatascience.com/python-web-scraping-libraries, Oct 28, 2016. Accessed: 2017-08-7.

[14] Simson Garfinkel. An evaluation of amazon's grid computing services: Ec2, s3, and sqs. *Harvard Computer Science Group Technical Report, TR-08-07*, 2007.

[15] Optimus Information. Selenium testing: Advantages and disadvantages. http://www.optimusinfo.com/seleniumtestingadvantagesanddisadvantages, December 22,2015. Accessed: 2017-08-10.

[16] 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.

[17] 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.

[18] 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.

[19] FedTech Staff. 4 infrastructure requirements for any big data initiative. https://fedtechmagazine.com/article/2016/12/4-infrastructure-requirements-any-big-data-initiative, December,2016. Accessed: 2017-08-03.