applied sciences

MDPI

Article

A Cloud-Based Framework for Large-Scale Log Mining through Apache Spark and Elasticsearch

Yun Li [1], Yongyao Jiang [1], Juan Gu [1], Mingyue Lu [1], Manzhu Yu [1], Edward M. Armstrong [2], Thomas Huang [2], David Moroni [2], Lewis J. McGibbney [2], Greguska Frank [2] and Chaowei Yang [1,*]

1 NSF Spatiotemporal Innovation Center and Department of Geography and GeoInformation Science, George Mason University, Fairfax, VA 22030, USA; yli38@gmu.edu (Y.L.); yjiang8@gmu.edu (Y.J.); jgu4@gmu.edu (J.G.); mlu@gmu.edu (M.L.); myu7@gmu.edu (M.Y.)

2 NASA Jet Propulsion Laboratory, California Institute of Technology, Pasadena, CA 91109, USA; Edward.M.Armstrong@jpl.nasa.gov (E.M.A.); Thomas.Huang@jpl.nasa.gov (T.H.); David.F.Moroni@jpl.nasa.gov (D.M.); Lewis.J.Mcgibbney@jpl.nasa.gov (L.J.M.); Francis.Greguska@jpl.nasa.gov (G.F.)

* Correspondence: cyang3@gmu.edu; Tel.: +1-703-993-4742

Received: 29 January 2019; Accepted: 11 March 2019; Published: 16 March 2019

Abstract: The volume, variety, and velocity of different data, e.g., simulation data, observation data, and social media data, are growing ever faster, posing grand challenges for data discovery. An increasing trend in data discovery is to mine hidden relationships among users and metadata from the web usage logs to support the data discovery process. Web usage log mining is the process of reconstructing sessions from raw logs and finding interesting patterns or implicit linkages. The mining results play an important role in improving quality of search-related components, e.g., ranking, query suggestion, and recommendation. While researches were done in the data discovery domain, collecting and analyzing logs efficiently remains a challenge because (1) the volume of web usage logs continues to grow as long as users access the data; (2) the dynamic volume of logs requires on-demand computing resources for mining tasks; (3) the mining process is compute-intensive and time-intensive. To speed up the mining process, we propose a cloud-based log-mining framework using Apache Spark and Elasticsearch. In addition, a data partition paradigm, logPartitioner, is designed to solve the data imbalance problem in data parallelism. As a proof of concept, oceanographic data search and access logs are chosen to validate performance of the proposed parallel log-mining framework.

Keywords: web usage mining; log mining; cloud computing; parallel computing; data balance

1. Introduction

Spatial data portals (SDPs) serve the Earth science community with massive geospatial data [1]. However, as the volume and variety of the data are increasing faster than ever, they pose a great challenge for SDPs to provide reliable and quality service [2]. An emerging trend in data discovery is mining user behaviors from logs for the latent linkages between users and data [3]. Taking the National Aeronautics and Space Administration (NASA) Physical Oceanography Distributed Active Archive Center (PO.DAAC) as an example, a solution was proposed to improve oceanography data discovery and access by mining user behavior data, called the Mining and Utilizing Dataset Relevancy from Oceanographic Dataset (MUDROD) [2]. The MUDROD engine manages logs and converts them into a series of sessions and domain-knowledge-like oceanographic vocabulary linkages based on Elasticsearch [4,5]. Elasticsearch is a component of the ELK stack and provides a solution to automatically index, search, analyze, and visualize logs with Logstash and Kibana.

When learning useful knowledge from web usage logs, as more logs are collected, the more comprehensive a knowledge base will be populated. The faster the log analysis performs, the sooner the knowledge base keeps up with the trends in the data portal. However, the log-mining process is compute-intensive and time-intensive because it associates with complicated data preprocessing and mining algorithms such as user identification, crawler detection, session identification, and knowledge extraction (Figure 1). Session identification, e.g., requires the mining framework to differentiate a user's log from another user's log efficiently [4]. PO.DAAC provides nearly 10 years of historical logs as starting seeds for the MUDROD search engine. It takes more than five days (~120 h) to process all logs. In addition, the changing log volume requires on-demand computing resources. History logs are gathered once, and online logs are collected for analysis periodically. Enough computing resources are an indispensable prerequisite for large historical log analysis, while, for those logs collected on the fly, fewer computing resources are needed. Therefore, a scalable framework is desirable to optimize the log-mining process with a dynamic number of computing nodes.

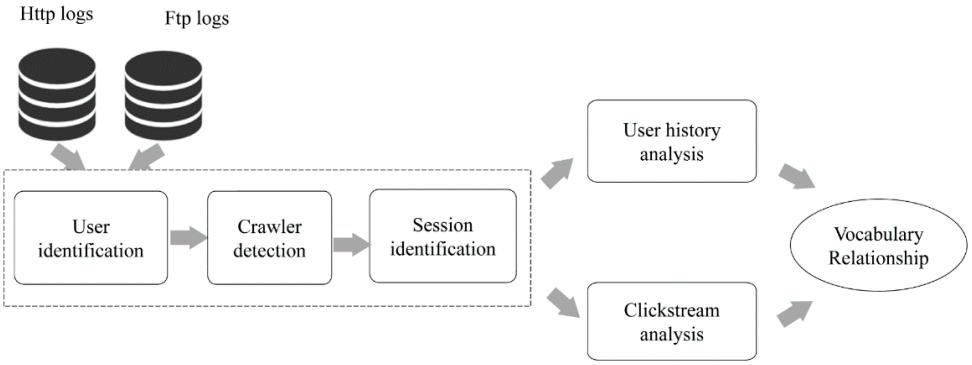

Figure 1. The components of log mining in the Mining and Utilizing Dataset Relevancy from Oceanographic Dataset (MUDROD) search engine.

Elasticsearch can search data and ingest logs with simple operations, e.g., filtering Uniform Resource Locator (URL) with the aid of Logstash. Kibana enables users to analyze and visualize logs. As an evolving open-source project, Elasticsearch provides rich resources and Application Programming Interfaces (APIs) for customizable search functionalities. We can implement log analysis functionalities using existing resources provides by the ELK stack efficiently. However, the ELK stack has an orientation toward efficient search rather than high-performance analysis. To improve performance of the log-mining framework built on Elasticsearch, a straightforward way is to adopt parallel processing frameworks. Big data frameworks, e.g., Apache Spark and Hadoop, are used to handle big data in recent years. Spark can accelerate the data analysis process with data parallelism and provide in-memory computing capability. To efficiently mine knowledge from web usage logs and make historical logs searchable and visualizable, we proposed a cloud-based framework for large-scale log mining using Spark and Elasticsearch. The framework can efficiently reconstruct sessions from offline logs and learn knowledge from user behavior data by making full use of advanced features provided by Spark and Elasticsearch, e.g., data parallelism and full-text indexing features. In addition, logs and extracted sessions are searchable on the platform. An information flow graph describes how data are flowing in the framework, and a log partitioner is designed to further speed up the log-mining process. We utilize PO.DAAC web usage log mining to demonstrate and evaluate the proposed framework and methods.

2. Related Work

As an important complementary component of the data discovery engine, log mining discovers implicit knowledge from user behavior data by learning meaningful patterns, trends, and linkages, e.g., Google explores user logs to catch what the world is searching and publishes reports of trends in

a year (https://trends.google.com/trends/), whereby it was found that the two most popular phrases searched in the Unites States were "world cup" and "Hurricane Florence" in 2018. Joachims et al. proposed the RankSVM algorithm to improve the ranking performance of a search engine by tracking user behavior on the ranking list. The assumption is that if a user clicked a link with a lower rank, the link is more relevant to the user's search than other links with a higher rank [6]. In the GIScience domain, researchers also pay close attention to log mining. Logs were collected and analyzed to assist search engines to understand queries [7], rank data intelligently [8], analyze a traveler's moving patterns for route recommendation [9], discover semantic relationships among geospatial vocabularies for domain ontology population [5], discover interesting spatiotemporal theme patterns [10], etc.

Sessionization is the process of splitting a collection of logs into multiple sessions which record a user's behavior on a website during a time period, serving as the first and fundamental step of log mining. However, it is quite a challenge to reconstruct sessions when the user identifier is missing in the raw logs. Multiple methods were proposed to group logs into sessions relying on time interval, referrer, or both [4,11,12]. In previous research, we proposed a method of reconstructing sessions from multiple logs to build a semantic knowledge base [4,5]. The knowledge extracted from logs is valuable, but the speed of the log-mining process needs improvement. For instance, for the small vertical search portal PO.DAAC, the log-mining framework built on Elasticsearch took more the one hour to finish monthly log analysis. Thus, we propose a cloud-based framework to speed up log mining with Apache Spark and Elasticsearch.

Logs are continually produced in different formats and rates, and can uncover useful information through proper and effective analysis. Cloud computing and parallel computation frameworks were introduced to support log analysis, as they can manage and analyze data of large size with a high production rate [13,14] The distributed computing paradigm can provide on-demand storage and computing resources for log mining [15,16]. The parallel computation frameworks, e.g., Apache Spark and Apache MapReduce, improve data analysis performance through data parallelism. Lin et al. proposed a unified cloud platform with batch analysis and in-memory computing capacity by combining Hadoop and Spark [16]. Therdphapiyanak et al. applied Hadoop for large-scale log analysis to efficiently detect abnormal traffic from high-volume data [17].

Given the significance of log analysis, several open-source and commercial solutions were designed for log collection, storage, search, analysis, and visualization. The ELK stack, i.e., Elasticsearch, Logstash, and Kibana, automatically collects, indexes, aggregates, and visualizes logs [18]. Prakash et al. efficiently geo-identified the website user traffic through logs using ELK stack [19]. Bagnasco et al. used the Elasticsearch ecosystem to monitor the Infrastructure as a Service (IaaS) and scientific application on the cloud, keeping track of usage and dynamic allocation of resources [20].

Both Spark and Elasticsearch have advantages and disadvantages for log mining. Spark is capable of handling big data, but it is not designed for log analysis. We have to conduct a significant amount of development if we implement a log analysis platform from scratch with Spark. Elasticsearch has the capability of indexing logs and searching specific records from billions of log records instantly; however, it only provides basic analysis functions, e.g., monthly aggregation and total count. Meaningful hidden patterns cannot be automatically detected by the ELK stack. Furthermore, Elasticsearch is a search engine, not a parallel analysis framework. A trend is to integrate Spark and Elasticsearch to implement an efficient and scalable log management and analysis system. Metha et al. proposed a streaming architecture based on ELK, Spark, and Hadoop to do anomaly detection from network connection logs in near real time [21]. Li et al. introduced a method to speed up log mining with Elasticsearch and Spark; however, Elasticsearch and Spark work independently in their framework [22]. Specifically, Spark is only used to split logs and distribute them to multiple machines prior to the log-mining process. In this research, we propose a log-mining framework integrating both Elasticsearch and Spark, and design a log partitioner approach to solve the skewed data problem. The framework aims to (1) present a platform which integrates Spark and Elasticsearch to make full use of the functionalities offered by both; (2) serve as a guide about how to speed up Elasticsearch-based data analysis with Spark.

This paper introduces our research on utilizing cloud, Spark, and Elasticsearch in an interactively and optimized fashion to mine interesting patterns from log files. In addition to the introduction (Section 1) and literature review (Section 2), Section 3 introduces the logs used in the experiment. Section 4 delineates the proposed log-mining framework and the log-partitioning algorithm. Section 5 details the experiment, and Section 6 interprets and evaluates the experimental results. Section 7 reviews the framework and discusses future work.

3. Data

Logs of both Hypertext Transfer Protocol (HTTP) and File Transfer Protocol (FTP) are used in our research. For a data archive portal, HTTP logs record a user's search and click behavior when he/she visits the website. Generally, it consists of a client Internet Protocol (IP) address, request date/time, the page requested, HTTP code, user agent, and referrer (Figure 2) in the combined log format. The combination of client IP and user agent could identify a unique user. The date illustrates when the user visited the page [23]. FTP logs track the download operations on all datasets archived. Similarly, the FTP log has IP and time fields (Figure 3). HTTP, combined with FTP logs, records a user's activities on the websites, e.g., which collection-level metadata were viewed after a user submitted a query and which granule-level data the user downloaded. Through the log-mining process, such knowledge could be mined from the raw HTTP and FTP logs (Figure 1). Other popular types of logs in geospatial data service, such as Open-source Project for a Network Data Access Protocol (OPeNDAP) [24], are not included in our research, but the framework can support these log types with minor modification.

```
185.10.104.132 - - [01/Feb/2015:00:00:02 -0800] "GET
/datasetlist?ids=Collections:Platform:AccessType:Measurement:Availability&values=CCMP:AQUA:OPEN:
Ocean%20Winds:HISTORICAL&view=list HTTP/1.1" 200 85283 "-" "Mozilla/5.0 (Windows NT 6.1;
WOW64; rv:22.0) Gecko/20100101 Firefox/22.0“

185.10.104.196 - - [01/Feb/2015:00:00:08 -0800] "GET
/datasetlist?ids=Collections:TimeSpan:Platform:GridSpatialResolution:Availability&values=CCMP:[1826%
20TO%20*]:DMSP-F14:0.25:HISTORICAL&view=list HTTP/1.1" 200 84531 "-" "Mozilla/5.0 (Windows NT
6.1; WOW64; rv:22.0) Gecko/20100101 Firefox/22.0"
```

Figure 2. Sample HTTP logs in the combined format.

```
Sun Feb  1 00:00:48 2015 1 85.142.104.170 97
/allData/ascat/preview/L2/metop_a/12km/2015/031/ascat_20150131_110900_metopa_42987_eps_o_
125_2300_ovw.l2.nc.gz.md5 b _ o a anonymous@ ftp 0 * c

Sun Feb  1 00:00:49 2015 1 85.142.104.170 97
/allData/ascat/preview/L2/metop_a/12km/2015/031/ascat_20150131_074800_metopa_42985_eps_o_
125_2300_ovw.l2.nc.gz.md5 b _ o a anonymous@ ftp 0 * c
```

Figure 3. Sample FTP log data in FTP format.

4. Methodology

4.1. Proposed Framework Architecture for Log Mining

After an extensive study of the open-source tools supporting log mining, we propose an architecture integrating Elasticsearch with Spark for log mining. Elasticsearch, a component in the ELK stack ecosystem, provides a series of log management functionalities including log ingestion, index, query, statistical analyses and visualization. Spark is an in-memory parallel computing framework for big data. The goal of this framework is to utilize existing resources and tools to establish a flexible and scalable system for the entire log-mining process. The Elasticsearch ecosystem provides the framework as a log management system, in which logs are indexed for query, visualization, and statistical analyses. Spark accelerates the mining process through parallel computing. Figure 4

shows the system architecture. A cloud infrastructure, e.g., Amazon Web Service (AWS) [25] and OpenStack [26], automatically and efficiently launches virtual resources like virtual machines and network configurations. Once a virtual machine starts, a series of open-source tools cooperate with each other to manage and analyze logs. The Hadoop Distributed File System (HDFS) stores massive logs distributedly; Elasticsearch provides the capabilities of indexing and searching logs in a near real-time manner; Spark accelerates log-mining subtasks through data parallelism. An image can be created in advance to facilitate the installation of log-mining clusters on demand, including an HDFS cluster, a Spark cluster, and an Elasticsearch cluster.

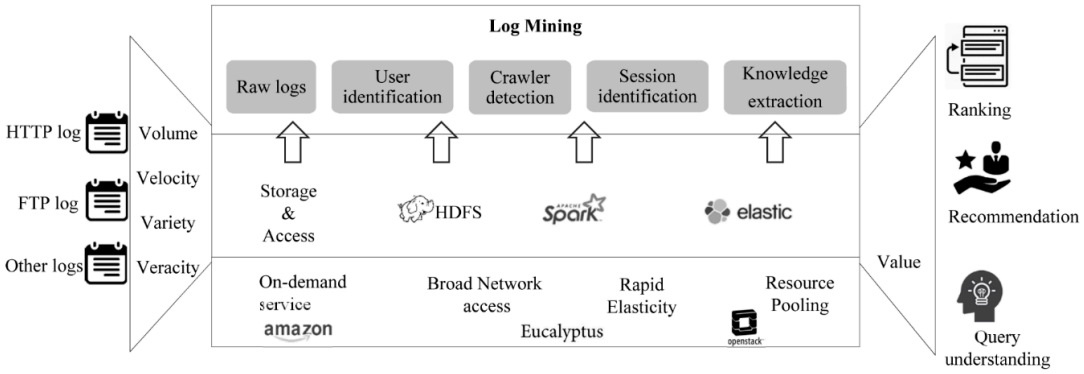

Figure 4. The framework for log mining.

The framework indexes and analyzes multi-source, heterogeneous logs through user identification, crawler detection, session reconstruction, and knowledge extraction. User identification recognizes all users from billions of logs. The IP or IP combined with other fields, e.g., agent, identifies a user. Crawler detection recognizes crawlers with well-known robots and visiting rate. Session reconstruction splits logs belonging to one user into small groups, and each group records a user's behavior in a relatively short time. Knowledge extraction summarizes a user's behavior in a session including the dataset the user clicked or viewed [4]. With the framework, big raw logs could be efficiently converted to knowledge, e.g., vocabulary linkage or metadata linkage, which can be adopted to improve search engines through better ranking, recommendation, and query understanding [2,5,8].

4.2. Information Flow

Logs flow through different stages to the final mining results. The raw log collection stage gathers multi-source logs and migrates them into the HDFS. Then, logs are loaded by Spark and indexed into Elasticsearch in parallel. During the ingestion process, Elasticsearch builds a full content index of logs for better access. Then, a list of unique users is recognized from ingested logs through the Elasticsearch term aggregation operation, which aggregates data based on a search query. For any user in the list, all logs belonging to a specific user are aggregated to visiting rates to determine if the user is a crawler. If the user is a crawler, all his/her logs will be discarded to reduce noise. Alternatively, real user logs will be kept for further analysis. The real user list is fed to Spark for identifying sessions and extracting knowledge parallelly. As shown in Figure 5 and described above, data flow between Spark and Elasticsearch during the whole process to fully leverage the advantages of both.

4.3. Log Partitioner

In the information flow, a pattern occurs repeatedly in which all users are recognized by Elasticsearch and partitioned into small groups. This repetitive process for user-level analyses can be performed in parallel, e.g., constructing sessions from a user's logs. In the process, term aggregation, one of the most beneficial features that Elasticsearch offers, returns the unique terms for a given field along with the count of matching documents on the fly [27]. Given an IP, term aggregation returns all IPs in the log and the count of logs associated with each IP. Apache Spark divides the IP list into

sub-datasets and distributes them to different computing nodes for the same analyses. HashPartitioner, the default partitioner in Spark, assigns IPs to different groups according to the hash value of the IP address.

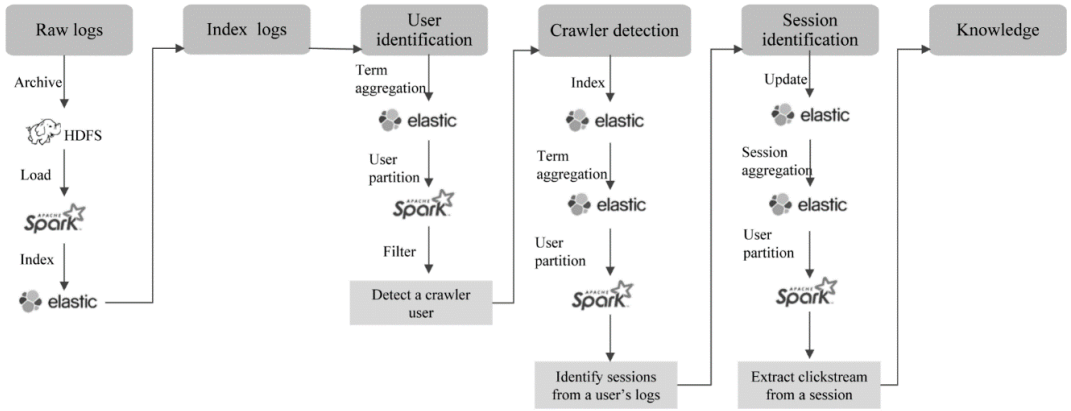

Figure 5. Information flow of the log-mining framework.

In the log-mining process, the Spark driver splits all users into *k* partitions, each of which will be passed to an executor and run as a task. Although the HashPartitioner splits the user list evenly with the hash code of IP address, the log size in each task (partition) is significantly different because the partitioner ignores the fact that different users have different behavior. Some users interacting with the website frequently produce many logs, while others who visit the website seldom populate few logs. As a result, the workloads are quite different, known as the skewed data problem (Figure 6). The skewed data problem will result in most tasks waiting for a slow task, which is the bottleneck of the whole process, since the Spark application runtime depends on the slowest task.

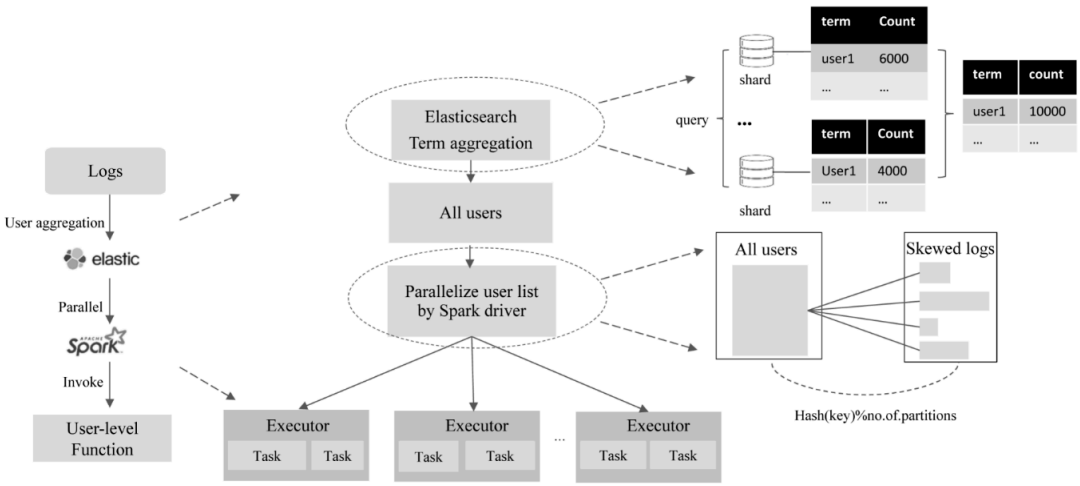

Figure 6. The skewed data problem caused by the default HashPartitioner.

To solve the problem, we propose a customized partitioner called LogPartitioner to balance logs across Spark tasks. As shown in Figures 7 and 8, the LogPartitioner algorithm consists of the following steps:

1. Calculate log records for each user. A user map stores the user and the corresponding log count as ($user_1$, log_count_1), ($user_2$, log_count_2), . . . , ($user_k$, log_count_k). The calculation module is implemented with the Elasticsearch term aggregation API and the performance depends on the cluster configuration and unique IPs in the log. Generally, the runtime increases linearly with the increment of log number; thus, the time complexity is approximately O(n).

2. Sort the user map by the log number in descending order. The higher a user is ranked in the map, the more frequently the user uses the website. The time complexity of the map sort operation is $O(n\log_2 n)$).
3. Split users into k groups using the greedy algorithm [28]. The top k users in the sorted user map are assigned to k groups in series. For the rest, each user is assigned to the group with minimal logs until all users are assigned to the k groups. A user group map stores the split strategy in the format of ($user_i$, $group_1$), ($user_j$, $group_1$), . . . , ($user_k$, $group_n$). Note that the greedy algorithm yields local optimum group strategy. The dynamic planning solutions can be adopted to split users too. In addition, to maximize the usage of all computation resources, k is suggested to be equal to or be multiples of the total number of Central Processing Unit (CPU) cores managed by Spark. The greedy algorithm is straightforward, and the time complexity is linear ($O(n)$).
4. Load all users into Spark and distribute user data across Spark tasks with a customized partitioner. The partitioner receives the user map and partition number as input and assigns each user to a partition using the corresponding group identifier instead of the hash value of the user IP. As a result, the logs are nearly evenly distributed across all partitions, which prevents significant slow tasks and improves the log-mining performance.

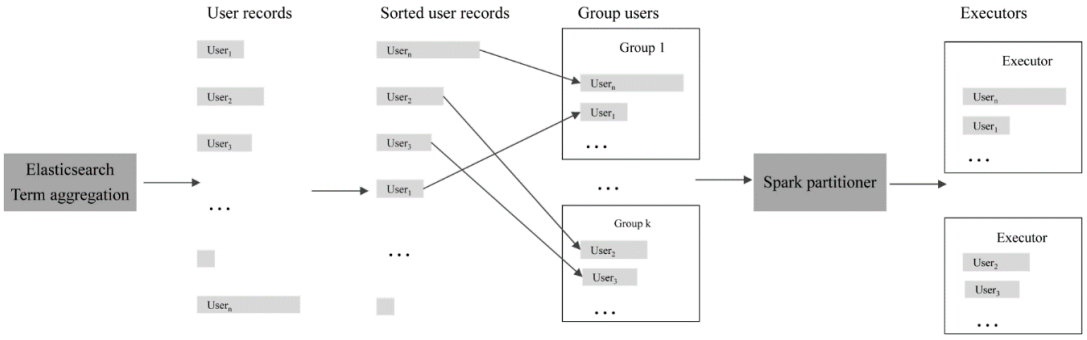

Figure 7. The workflow of the proposed LogPartitioner.

1 split raw logs into small partitions

```
function CreateLogPartitioner(logs, partitionNum)
    userlogcount ← ElasticsearchTermAggregation(logs)
    userMap ← GroupUsers(userlogcount, partitionNum)
    logPartitioner ← SparkCustomerizedPartitioner(userMap, partitionNum)
    return logPartitioner
end function

function GroupUsers(userlogcount, partitionNum) partition ← []
    sortedMap ← userlogcount.sortedByLogCount
    while i < sortedMap.size() do
        if i < partitionNum then
            partition[i].add(sortedMap[i].getUser())
        end if
        user ← sortedMap[i].getUser()
        logcount ← sortedMap[i].getLogCount()
        index ← 0
        maxLogCount ← 0
        tmpLogCount ← 0
        while j < partitionNum do
            partitionLogs ← partition[j].countLogs()
            tmpLogCount ← partitionLogs + logcount
            if tmpLogCount > maxLogCount then
                maxLogCount ← tmpLogCount
                index ← j
            end if
        end while
        partition[index].add(user)
    end while
    return groupedUsers
end function
```

Figure 8. The pseudo code of the LogPartitioner.

5. Experiments

In this research, the NASA Advanced Information System Technologies (AIST) cloud platform was used to set up a testbed for the framework effortlessly and efficiently. An image with required software, e.g., HDFS, Spark, and Elasticsearch, was deployed in advance to facilitate the creation of a log-mining cluster on demand. Conveniently, whenever the cluster needs more computing resources to process growing logs, we can start a new instance from the preconfigured image. In our experiments, six virtual machines were launched for the log-mining process. Each machine was configured with four CPU cores, 16 GB of memory, and a 2.4-GHz clock speed (Figure 9). Operations provided by the AIST cloud platform make it possible and efficient to group these virtual machines into a cluster.

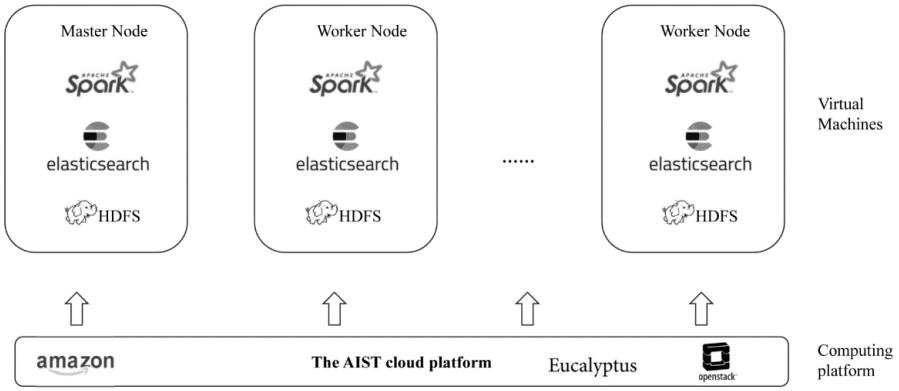

Figure 9. The testbeds for the proposed framework.

The entire 2014 PO.DAAC logs were used to test the performance of the framework. The log volume was >32.8 GB and the logs consisted of >154 million records in total. Figure 10 shows the distribution of logs in the 12 months in 2014. The user traffic changed dramatically from month to month and, in February, there were 6,727,710 (~7 million) logs in total and the log volume was >3.5 GB.

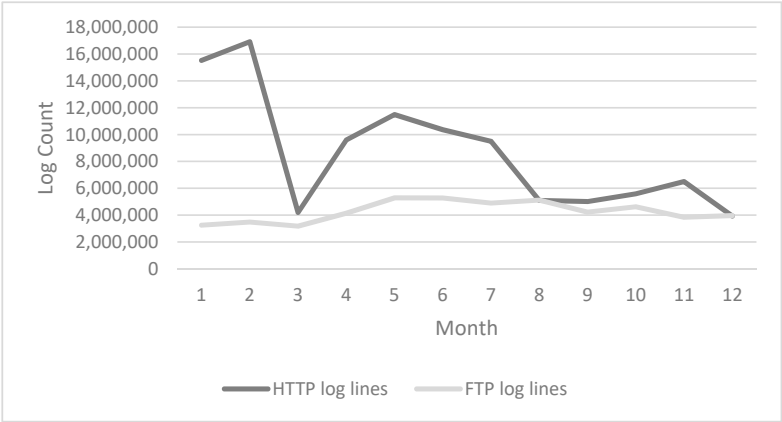

Figure 10. Monthly log distribution of the Physical Oceanography Distributed Active Archive Center (PO.DAAC).

Three experiments were conducted to evaluate the performance of the framework. Firstly, the February 2014 monthly log was randomly selected and processed with HashPartitioner and LogPartitioner separately to validate the proposed partitioner. Secondly, the entire logs were processed by the proposed framework to test performance. The log-mining algorithm implemented only on Elasticsearch with the same procedure was used for comparison. Then, the monthly log was processed with multiple clusters consisting of different numbers of workers to validate the scalability of the framework. Finally, we discussed the evaluation results.

6. Result and Evaluation

6.1. Comparison of the HashPartitioner and the LogPartitioner

To demonstrate the validity of the proposed LogPartitioner, the February 2014 monthly logs were separately parallelized through the HashPartitioner and the LogPartitioner. Table 1 shows the time spent on each log-mining step. Initially, it took more than one hour (~3873 s) to finish the entire mining process with Elasticsearch. With the proposed framework running on a cluster which consisted of four worker nodes, the processing time decreased from 3873 s to 589 s if the logs were split with the default HashPartitioner. By applying the proposed LogPartitioner, the processing time was further reduced to 310 s.

Table 1. Comparison of time cost of each step in the log-mining process.

Log-Mining Steps/Processing Time	Index Log and User Identification	Crawler Detection	Session Identification	Total Time
With Elasticsearch (s)	3140	130	603	3873
Cluster with the HashPartitioner (s)	152	108	329	589
Cluster with the LogPartitioner (s)	152	54	104	310

The cluster had one master node and four worker nodes, on each of which an executor ran tasks with four CPU cores. It means 16 tasks were executed in parallel. Figure 11 shows the sum of log processing time on each executor in the crawler detection step. The dark line displays the runtime of executors when users were grouped with the HashPartitioner. Executor 10 worked much longer than other executors, indicating that most executors need to wait for the slow executor. The light line represents the processing time of all executors after introducing the LogPartitioner. All executors finished their tasks at almost the same time and no executor needed to wait extensively for other executors. Thus, the processing time was significantly reduced by our proposed LogPartitioner.

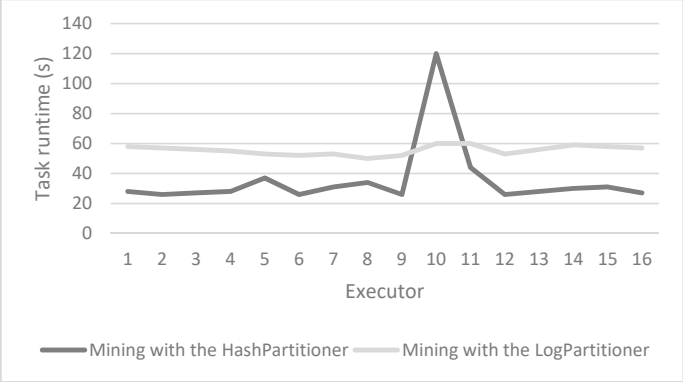

Figure 11. The sum of task runtime on each executor.

6.2. Comparison of Performance with and without the Framework

The whole 2014 PO.DAAC logs were processed with a cluster consisting of five worker nodes. For comparison, the entire logs were processed with and without the proposed framework. The log-mining process took about one hour to process the entire logs with the framework. With Elasticsearch, it took more than 10 h. Monthly processing time was not positively correlated with the log size because some records generated by robots in the raw logs were discarded during the mining process (Figure 12). Figures 13 and 14 show time spent on each stage of the log-mining process without and with the framework, respectively. The log ingestion time was significantly reduced since logs were read from the file system and written into Elasticsearch in parallel. The crawler detection and session reconstruction steps were also accelerated with the support of the proposed framework.

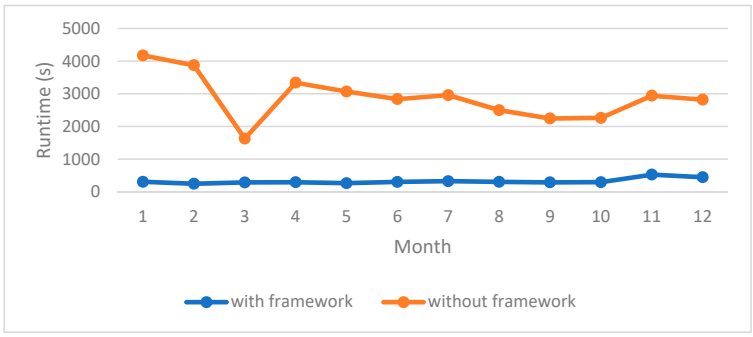

Figure 12. The processing time of logs in 2014 with and without framework.

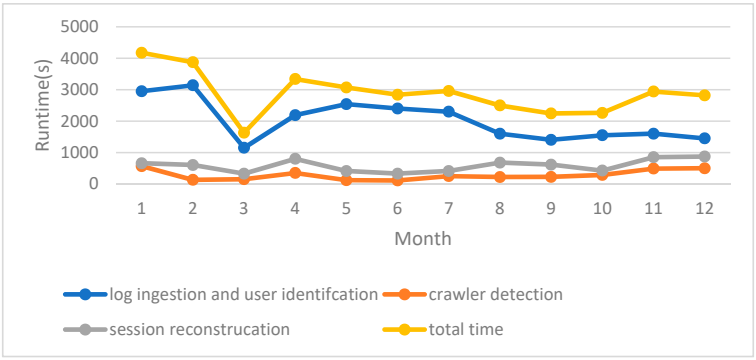

Figure 13. The stage processing time of logs in 2014 without the proposed framework.

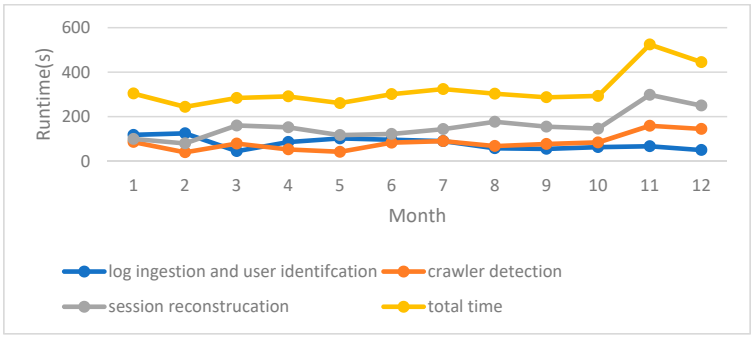

Figure 14. The stage processing time of logs in 2014 with the proposed framework.

6.3. Scalability of the Proposed Framework

To demonstrate the scalability of our proposed framework, multiple clusters were set up with one master node and different numbers of worker nodes ranging from one to five. On each worker node, an executor performed the log-mining tasks with a maximum of four CPU cores. In experiments 1 and 2, only two CPU cores were in use because the processing speed exceeded the log ingestion speed if all cores were allocated. The partition number was set to 112 in experiment 4 and 120 in all other experiments to make sure the partition number was a multiple of the number of CPU cores. Table 2 shows that the processing time decreased as the worker number increased, and the results remained the same.

Table 2. The processing time of the February 2014 monthly log on different clusters.

Experiment No.	No. of Workers	No. of Executors	No. of Cores	No. of Partitions	Time Cost	Session Number
1	1	1	2	120	2056	1073
2	2	2	4	120	1055	1073
3	3	3	12	120	386	1073
4	4	4	16	112	317	1073
5	5	5	20	120	281	1073

6.4. Sample Mining Results

Through the log-mining workflow, 11,360 sessions were reconstructed from the entire 2014 raw logs. Figure 15 shows an example of the reconstructed session. The tree structure makes it easy to track which dataset a user clicked or downloaded after he/she performed a query. In this session, the user download granule data from the "recon_sea_level_ost_l4_v1" and "alt_tide_gauge_l4_ost_sla_us_west_coast" collection after searching "sea surface topography". Such information could be used for calculating vocabulary linkage to maintain a live knowledge base for ranking, recommendation, and query suggestion [2,5]. Figure 16 lists statistical analysis results of a selected month, e.g., the most popular user input keys, the most popular datasets, session duration, and others.

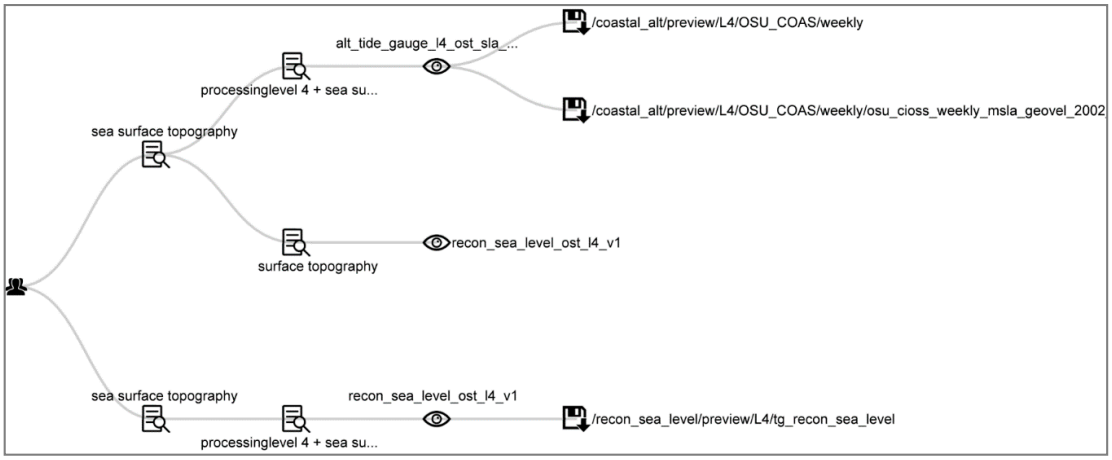

Figure 15. An example of the session tree structure [4].

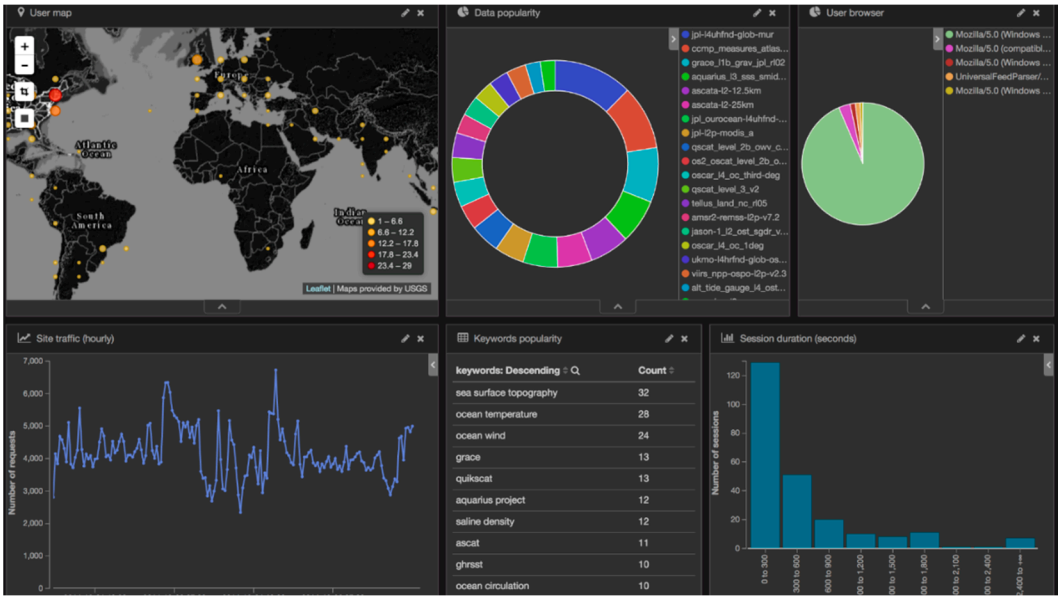

Figure 16. Statistical information for a selected month.

7. Discussion and Conclusions

Log mining is the process of reconstructing sessions from raw web logs and discovering meaningful usage patterns and implicit linkages. The mining result is valuable in improving the data discovery experience. We proposed a cloud-based framework for large-scale log mining by integrating Elasticsearch and Spark. A customized partitioner named LogPartitioner solves the skewed data issue in the log parallelism process. The cloud platform serves as the framework with virtual resources.

Experiments demonstrated that the processing time of one month of logs decreased from 60+ minutes to ~5 min with the framework. To share the framework with the public, the research results were published as a GitHub open-source project (https://github.com/apache/incubator-sdap-mudrod).

Several aspects are worth further investigation. Firstly, since the framework cannot analyze real-time logs, a real-time data streaming tool, e.g., Apache Spark Streaming, can be introduced into the framework for near real-time analysis and tracking [29]. Secondly, the network transportation cost in the cloud-based framework should be experimented for the mining process when both the Elasticsearch and Spark cluster are integrated as a whole entity. Thirdly, although the framework reported in the article focused on log data mining, one can accelerate the analysis process with a customized partitioner for other types of data, e.g., geo-tagged tweets. We will continue this research to improve the framework with more specific performance-improving techniques.

Author Contributions: C.Y. and T.H. conceptualized the study; Y.L., L.J.M., and G.F. designed and developed the study; Y.L., Y.J., E.M.A., D.M., L.J.M., M.L., and G.F. conducted the experiments; Y.L., E.M.A., L.J.M., and G.F. performed the analysis; C.Y. and E.M.A. supervised the research; Y.L. wrote a draft of the manuscript, which was iteratively improved by all other authors of the manuscript.

Funding: This project was funded by NASA AIST (NNX15AM85G) and NSF (IIP-1338925). The research was partially carried out at the Jet Propulsion Laboratory, California Institute of Technology, under a contract with the National Aeronautics and Space Administration.

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

References

1. Gui, Z.; Yang, C.; Xia, J.; Liu, K.; Xu, C.; Li, J.; Lostritto, P. A performance, semantic and service quality-enhanced distributed search engine for improving geospatial resource discovery. *Int. J. Geogr. Inf. Sci.* **2013**, *27*, 1109–1132. [CrossRef]
2. Jiang, Y.; Li, Y.; Yang, C.; Hu, F.; Armstrong, E.M.; Huang, T.; Moroni, D.; McGibbney, L.J.; Greguska, F.; Finch, C.J. A Smart Web-Based Geospatial Data Discovery System with Oceanographic Data as an Example. *ISPRS Int. J. Geo-Inf.* **2018**, *7*, 62. [CrossRef]
3. Srivastava, J.; Cooley, R.; Deshpande, M.; Tan, P.-N. Web usage mining: Discovery and applications of usage patterns from web data. *ACM Sigkdd Explor. Newslett.* **2000**, *1*, 12–23. [CrossRef]
4. Jiang, Y.; Li, Y.; Yang, C.; Armstrong, E.M.; Huang, T.; Moroni, D. Reconstructing sessions from data discovery and access logs to build a semantic knowledge base for improving data discovery. *ISPRS Int. J. Geo-Inf.* **2016**, *5*, 54. [CrossRef]
5. Jiang, Y.; Li, Y.; Yang, C.; Liu, K.; Armstrong, E.M.; Huang, T.; Moroni, D.F.; Finch, C.J. A comprehensive methodology for discovering semantic relationships among geospatial vocabularies using oceanographic data discovery as an example. *Int. J. Geogr. Inf. Sci.* **2017**, *31*, 2310–2328. [CrossRef]
6. Joachims, T. Optimizing search engines using clickthrough data. In Proceedings of the Eighth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Edmonton, AB, Canada, 23–26 July 2002; pp. 133–142.
7. Gan, Q.; Attenberg, J.; Markowetz, A.; Suel, T. Analysis of geographic queries in a search engine log. In Proceedings of the First International Workshop on Location and the Web, New York, NY, USA, 22 April 2008; pp. 49–56.
8. Jiang, Y.; Li, Y.; Yang, C.; Hu, F.; Armstrong, E.M.; Huang, T.; Moroni, D.; McGibbney, L.J.; Finch, C.J. Towards intelligent geospatial data discovery: A machine learning framework for search ranking. *Int. J. Dig. Earth* **2018**, *11*, 956–971. [CrossRef]
9. Frignani, M.; Auld, J.; Mohammadian, A.; Williams, C.; Nelson, P. Urban travel route and activity choice survey: Internet-based prompted-recall activity travel survey using global positioning system data. *Trans. Res. Record J. Trans. Res. Board* **2010**, *2183*, 19–28. [CrossRef]
10. Mei, Q.; Liu, C.; Su, H.; Zhai, C. A probabilistic approach to spatiotemporal theme pattern mining on weblogs. In Proceedings of the 15th International Conference on World Wide Web, Scotland, UK, 23–26 May 2006; pp. 533–542.

11. Kaur, N.; Aggarwal, H. A novel semantically-time-referrer based approach of web usage mining for improved sessionization in pre-processing of web log. *Int. J. Adv. comput. Sci. Appl.* **2017**, *8*. [CrossRef]
12. Dell, R.F.; Roman, P.E.; Velasquez, J.D. Web user session reconstruction using integer programming. In Proceedings of the 2008 IEEE/WIC/ACM International Conference on Web Intelligence and Intelligent Agent Technology, Washington, DC, USA, 9–12 December 2008; Volume 1, pp. 385–388.
13. Yang, C.; Yu, M.; Hu, Y.; Jiang, Y.; Li, Y. Utilizing cloud computing to address big geospatial data challenges. *Comput. Environ. Urban Syst.* **2017**, *61*, 120–128. [CrossRef]
14. Yang, C.; Huang, Q.; Li, Z.; Liu, K.; Hu, F. Big Data and cloud computing: Innovation opportunities and challenges. *Int. J. Dig. Earth* **2017**, *10*, 13–53. [CrossRef]
15. Mavridis, I.; Karatza, H. Performance evaluation of cloud-based log file analysis with Apache Hadoop and Apache Spark. *J. Syst. Softw.* **2017**, *125*, 133–151. [CrossRef]
16. Lin, X.; Wang, P.; Wu, B. Log analysis in cloud computing environment with Hadoop and Spark. In Proceedings of the 2013 5th IEEE International Conference on Broadband Network & Multimedia Technology (IC-BNMT), Guilin, China, 17–19 November; pp. 273–276.
17. Therdphapiyanak, J.; Piromsopa, K. Applying Hadoop for log analysis toward distributed IDS. In Proceedings of the 7th International Conference on Ubiquitous Information Management and Communication, Kota Kinabalu, Malaysia, 17–19 January 2013; p. 3.
18. Chhajed, S. *Learning ELK Stack*; Packt Publishing Ltd.: Birmingham, UK, 2015.
19. Prakash, T.; Kakkar, M.; Patel, K. Geo-identification of web users through logs using ELK stack. In Proceedings of the 2016 6th International Conference Cloud System and Big Data Engineering (Confluence), Noida, India, 14–15 January 2016; pp. 606–610.
20. Bagnasco, S.; Berzano, D.; Guarise, A.; Lusso, S.; Masera, M.; Vallero, S. Monitoring of IaaS and scientific applications on the Cloud using the Elasticsearch ecosystem. *Proc. J. Phys.* **2015**, *608*, 012016. [CrossRef]
21. Mehta, S.; Kothuri, P.; Garcia, D.L. Anomaly Detection for Network Connection Logs. *arXiv* **2018**, arXiv:1812.01941.
22. Li, Y.; Jiang, Y.; Hu, F.; Yang, C.; Armstrong; Huang, T.; Moroni, D.; Fench, C. Leveraging cloud computing to speedup user access log mining. In Proceedings of the OCEANS 2016 MTS/IEEE Monterey, Monterey, CA, USA, 19–23 September 2016.
23. Apache. Apache HTTP Server Version 2.4. Available online: http://httpd.apache.org/docs/current/logs.html#combined (accessed on 1 January 2016).
24. Cornillon, P.; Gallagher, J.; Sgouros, T. OPeNDAP: Accessing data in a distributed, heterogeneous environment. *Data Sci. J.* **2003**, *2*, 164–174. [CrossRef]
25. Fox, A.; Griffith, R.; Joseph, A.; Katz, R.; Konwinski, A.; Lee, G.; Patterson, D.; Rabkin, A.; Stoica, I. Above the clouds: A berkeley view of cloud computing. *UCB/EECS* **2009**, *28*, 2009.
26. Sefraoui, O.; Aissaoui, M.; Eleuldj, M. OpenStack: Toward an open-source solution for cloud computing. *Int. J. Comput. Appl.* **2012**, *55*, 38–42. [CrossRef]
27. Dixit, B. *Elasticsearch Essentials*; Packt Publishing Ltd.: Birmingham, UK, 2016.
28. Edmonds, J. Matroids and the greedy algorithm. *Math. Programm.* **1971**, *1*, 127–136. [CrossRef]
29. Meng, X.; Bradley, J.; Yavuz, B.; Sparks, E.; Venkataraman, S.; Liu, D.; Freeman, J.; Tsai, D.; Amde, M.; Owen, S. Mllib: Machine learning in apache spark. *J. Mach. Learn. Res.* **2016**, *17*, 1235–1241.