Actor-based architecture for Cloud Services Orchestration: the case of social media data extraction

Stefano Cavalli, Stefano Cagnoni, Gianfranco Lombardo and Agostino Poggi

University of Parma

Abstract

In this paper we present a distributed system for social media scraping which aims to acquire an arbitrarily large number of information from social networks, by exploiting an actor-based solution able to orchestrate efficiently several services on cloud. Our goal is to ensure that correct operations among actors occur, thanks to a master node, based on the ActoDeS architecture, which takes care of managing communications, interface and messages exchanged by client nodes. As a use case, we consider Twitter as social media platform for the key role that is playing in the modern society, as shown by Google Trends data. However, Twitter's search API have many limitations and there is definitely no way to make it work when it comes to obtaining millions of records within a monthly or annual time range. Thus, we have designed a distributed solution that is able to overcome these constraints without breaking the current laws on this subject and the policies of Twitter.

Keywords

Actor-based systems, Cloud computing, Web-scraping, Data Analysis

1. Introduction

Nowadays, web scraping is one of the most widely used techniques to extract any type of data from a web page, thanks to multiple software that automatize this process. Over the past few years, it has been the subject of many controversies but on September 9th 2019, the Appeal from the United States District Court for the Northern District of California officially declared [1] that web scraping is not illegal, as are not methodologies that try to prevent users from working on it, provided that no access is made by the software within the platform itself. Therefore, in the United States, as well as in the rest of the world, laws are changing and more than ever the knowledge represents a truly competitive key for companies and industries.

With over 500 million tweets per day, according to their 2020 Q1 report [2], Twitter has over 166 million daily active users who generate a quantity of information that can certainly be defined as big data. In light of this, the interest in big data generated over social media is increasing and, at the same time, software solutions to perform data extraction are more and

WOA 2020: Workshop "From Objects to Agents", September 14–16, 2020, Bologna, Italy

stefano.cavalli1@unipr.it (S. Cavalli); stefano.cagnoni@unipr.it (S. Cagnoni); gianfranco.lombardo@unipr.it (G. Lombardo); agostino.poggi@unipr.it (A. Poggi)

0000-0002-3505-0556 (S. Cavalli); 0000-0003-4669-512X (S. Cagnoni); 0000-0003-1808-4487 (G. Lombardo); 0000-0003-3528-0260 (A. Poggi)

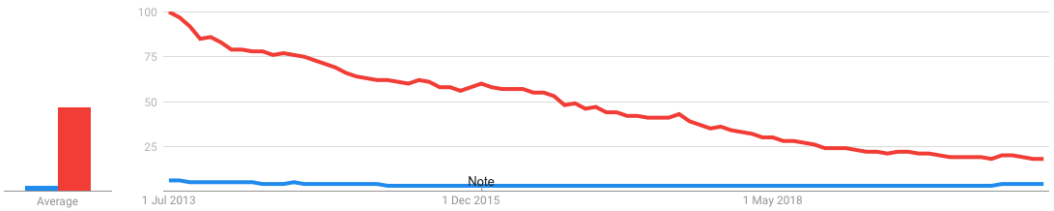

Figure 1: Google Trends data shows interest over time in Twitter (blue) and Facebook (red) expressed by people from all over the world, within a time range which goes from July 2013 to July 2020

more demanded. Several research works have also highlighted the importance of collecting contents from the main social media platforms, such as Facebook and Twitter, to perform various kinds of analysis. For example, in [3], data from Facebook are used to analyze a rare disease. In [4], social media data are used to study collaboration in firms and organizations. In [5] and [6], data from Twitter are used to train classifiers able to detect troll users.

Moreover, different sentiment analysis techniques perform really well and obtain successful results when applied to social media [7, 8, 9]. These applications are improving day by day and they are moving to cloud and to distributed systems, to better tackle countless performance issues, software and hardware conflicts, backup and replication problems. Cloud orchestration manages and coordinates all the activities coming from cloud automation, i.e., those processes that run without the need for human interaction. Nowadays, cloud orchestration brings many advantages depending on the application context: cost reduction and scaling, efficiency, improved security and enhanced visibility into resources.

When it comes to distributed computing and actor-model, one of the software frameworks for multi-agent and actor architecture modelling is ActoDeS [10]: developed entirely in Java, it takes advantage of the actor model by delegating the management of events to the execution environment. Moreover, thanks to the use of different implementations of components that manage the execution of the actors, it is suitable for the development of efficient and scalable applications in particular in the data mining domain [11]. In this environment, messages, actors and message patterns are all instances of Java classes. Every message is an object containing a set of fields, a message pattern is an object defined as a combination of constraints on the value of some message field and the actor address acts as both a system-wide unique identifier and a local and remote proxy of an actor.

In this article, we present an actor-based architecture for orchestrating cloud services. In particular, as use case, the system aims at easing the execution of data extraction from Twitter. The following tasks are available: retrieving data for a long period of time, without any limitation; distributing the computing power and computational logic on different machines and making sure that a master process will orchestrate all services.

2. Related Work

Examining the state of the art of web scraping [12], we confirm that this term is always considered in two different contexts: (*i*) data collection, in which the main focus is not a real-

time system, therefore speed performance is not an important factor and it does not affect results; (*ii*) work within a real-time environment, where it is important to provide all the information required and where a millisecond error could be very critical.

Nowadays, web scraping is widely used. Some of its applications include: job search [13], weather data [14], advertisement [15], journalism [16], and health sector [17]. Financial trading [18] is also important and requires real-time systems to be very performing. News, along with political-economic decisions, may drastically influence a price stock within a very short time range, where web scraping automation plays a fundamental role. In this case, web scraping is used to design and implement automated bots that act as trading decision support systems [19].

In 2017, Trifa *et al.* [20] described a Personalized Multi-Agent Systems (PMAS) on a distributed and parallel architecture in order to predict users' topics of interest using data coming from web scraping tools, focusing on Twitter, Facebook and LinkedIn. As previously mentioned, we have chosen to evaluate Twitter as social network because it is still increasing its user base against Facebook, which instead has set an interest decrease for seven consecutive years, based on Google trends data [21] (Figure 1) which is frequently used in data analysis and statistical topics.

Building a cloud system, in order to analyze an unlimited amount of data, avoids running into several problems and leads to multiple advantages. Integrity, which refers to data completeness, accuracy and consistency, must be kept safe; a cloud system always helps to provide data integrity [22]. Along with it, connectivity, speed, scalability and store availability are also features of cloud or cloud-hybrid systems. Working with services on a cloud-based environment can be risky and drive into several problems [23]: large computational load can be required and nodes could take too long for a decision-making process; recovery from mistakes that can arise with a centralized decision making process would not be managed by the system. By using an actor model, all these problems can be solved. This is due to the fact that each actor is able to make decisions in this cloud-based environment, where only partial information may be available. If one or more nodes fail, a readjustment process will take place in order to prevent any related problem. Dealing with actor-model also brings some advantages: the use of simple and high-level abstractions for distribution, concurrency and parallelism; an asynchronous, non-blocking and highly performing message-driven programming model can be easily designed; take advantage of a very lightweight event-driven processes.

Cloud orchestration and automation are well-known terms in tech industries. Based on their context, researchers have different opinions about them, but when web services come up, orchestration has been extensively discussed [24] along with concurrency programming. Concurrency in today's middleware is almost always thread-based and hardware still evolves towards more parallel architectures. Threads are the most used way to execute code concurrently and they should manage possible conflicts of their execution, avoiding synchronization that seems to help on correct data ordering and consistency. However, this paradigm raises several problems when it comes to deadlocks, maintaining data consistency and liveness. Most of the problems with concurrency, such as deadlocks and data corruption, result from having shared state. That is where the actors come into play: the actor model, where independent processes exchange immutable messages, is an interesting alternative to the shared state model.

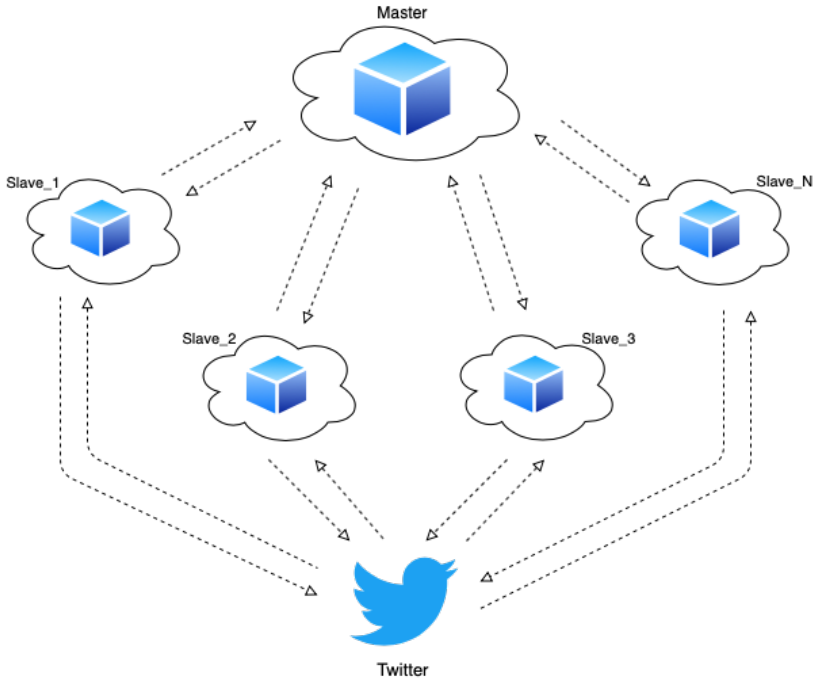

Figure 2: Actor model cloud-based architecture: each node represents an actor enclosed in its cloud environment; it can communicate exchanging messages with the master node, retrieving data coming from Twitter. *N* stands for the number of actors the system is designed for.

3. System Architecture

In this section, we describe the cloud system architecture and its components, as shown in Figure 2. The solution is composed by multiple actors that play different roles depending on their behavior and act with different logics. The overall system combines several existing technologies, which have strongly evolved during the last decades such as server, databases, web services, actor model, networks and cloud. The main purpose of our work is to obtain any number of tweets, specifying a time interval together with a language tag (i.e. "EN" for English) and a keyword which is part of the tweets themselves.

3.1. Multi-Actor model

As previously mentioned, a multi-actor model represents the optimal architectural choice, to describe the distributed cloud system we have worked on. Using a master-slave connection, we have built different nodes: the master node is designed using ActoDeS and its events management, while slave nodes are Python services running concurrently. ActoDeS allows actors to work on indipendent but communicating actor spaces. In order to make this communication available ActoDeS implements a Dispatcher which takes care of messages forwarding between different actor spaces. First of all, rather than explicit sleeping or waking up, actors react to events that come from interactions with other actors (i.e. receiving a message). If they worked

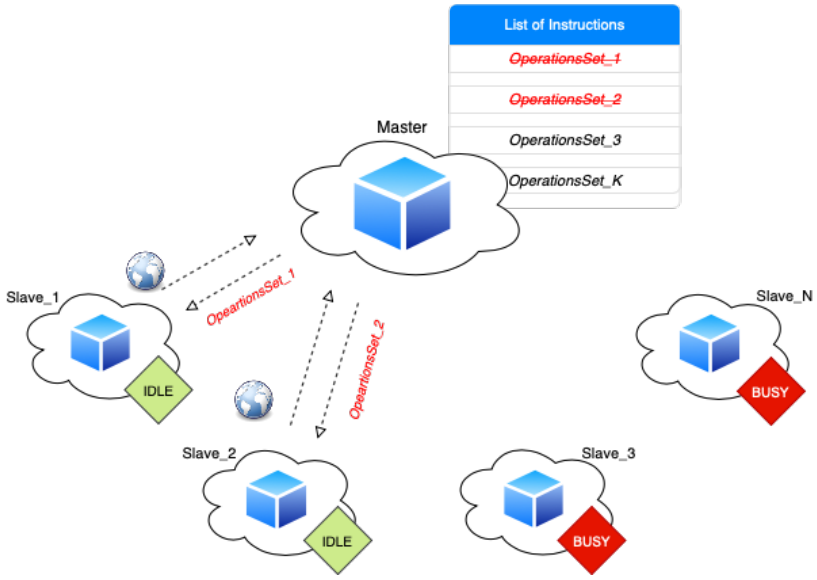

Figure 3: When clients are in IDLE state, they send a message to the master trough the WebSocket and it replies with a set of operations coming from the List of Instruction, it than update the list removing this specific item. In this example, two clients are busy for some other reasons and they are not communicating with the server.

out their previous tasks and nobody send them new messages, these actors get passive and wait for another message to receive. In this architecture, actors communicate asynchronously with each other by sending immutable messages following a specific pattern which is later discussed. A FIFO (First-In-First-Out) order is respected when an actor receive multiple messages from others. Receiving messages is always a blocking operation from the actor point of view, however, sending a new message must be non-blocking. In particular, when a slave wants to send data to the master, it keeps going on working as a web scraping service: in this case, a non-blocking operation cannot be performed at all.

The master node acts as a system orchestrator (Figure 3): firstly, it sets all the communication channels between itself and all the other nodes, obviously located on separate servers. According to the user's requests, a list of instructions is made by the master node, which keeps it up-to-date as needed during the execution time. The list of instructions contains a set of operations that must be executed by the clients. At a certain point, while it is not working, a client can get one of these instructions by establishing a WebSocket connection with the master and sending a message to it. A WebSocket is a communication protocol, located at layer seven in the OSI model and standardized by IETF as RFC 6455 in 2011, that provides full-duplex communications channels over a single TCP connection. The master, which instantly reacts to this event and wakes up, replies with a random message taken from the list of instructions and updates the list, removing the instruction itself. While multiple clients may simultaneously connect to the master, a FIFO (First-In-First-Out) strategy is applied in order to maintain consistency and optimize the workload on all clients, avoiding that some of them work more than others or stay inactive for too long, causing inefficiency. Once the client has done with its set of operations (a

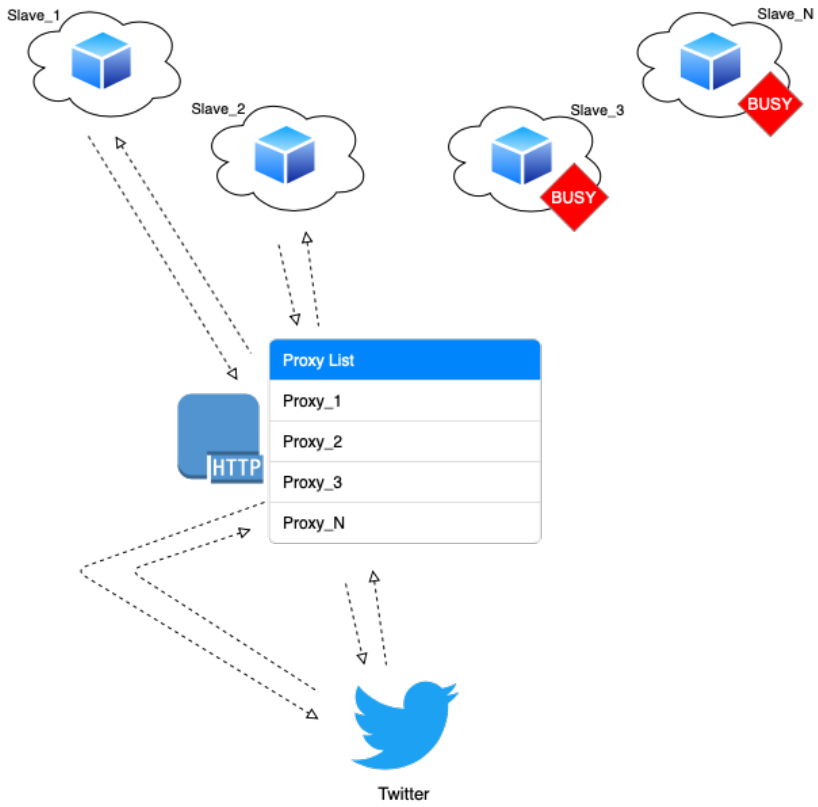

Figure 4: Once server sends a message to the clients, they send an http request to Twitter using a proxy list, in order to retrieve tweet's information. In this example, *Slave_3* and *Slave_N* are busy and they are not communicating with Twitter.

process which is extensively discussed below), it sends another message to the master using the Remote Copy Protocol (RCP) to transfer files and Secure Shell (SSH) to provide authentication and encryption. These files are all stored in the master's server and they will be removed from client's one as soon as the sending process is successfully completed. When the list of instructions is empty, it means there is no need to keep the client alive again and the server interrupts its WebSocket.

3.2. The Twitter case study

In this section we describe the structure of the instruction list, which we have introduced previously, and how the clients behave with Twitter 4. The so-called instruction list is a set of operations that clients execute, regardless of the others' behavior. In particular, each of them receives a time range together with a language and a keyword and will take care of downloading all the tweets that fall within those parameters listed in Table 1.

Each client operates asynchronously and executes instructions received from the server sequentially. By the time a client receives a server's message, containing a set of operations, it starts its job making an HTTP request, thanks to the Python3 Requests library [25], in order to

Table 1
Search parameters *Lan, Interval, Kw*

Parameter	Description
Lan	The language of a tweets..
Interval	A specific interval that includes a list of tweets.
Kw	A word, or sentence, that must be part of the tweets.

Table 2
Search parameters *Lan, Interval, Kw*

Parameter	Description
Username	Username of the tweet's creator.
FullName	Full name of the tweet's creator.
User_id	User id of the tweet's creator.
Tweet_id	Tweet's id.
Tweet_url	Tweet's url.
Timestamp	Tweet's timestamp.
Replies	Number of tweet's replies.
Retweets	Number of tweet's retweets.
Likes	Number of tweet's likes.
Text	Plain text of the tweet.
Html	Html extracted from the tweet.

Table 3
Effective parameters *Lan, Interval, Kw* used while testing the entire system.

Parameter	Value
Lan	**English**
Interval	**2009-01-01 / 2019-01-01**
Kw	**Bitcoin**

retrieve tweets from Twitter. We use a free proxy list [26] available from the web, which is a list of open HTTP/HTTPS/SOCKS proxy servers that allow clients to make indirect network connections to Twitter. This approach prevents from being banned by Twitter, once too many http requests are made.

Once an http request is made, the Python Beautifulsoup4 library is used in order to parse the content, retrieving all possible information a tweet may hold (Table 2).

4. Experimental Results

We have implemented the previously described system and a set of experiments has been performed. Using twenty-two different clients and one server, while working on a distributed and cloud environment, we have chosen the parameters described in Table 3.

The first problem we faced on is that when dealing with such systems, we will not have

the data size, nor the execution time, until the execution itself is finished. Nobody has any idea about how long it could takes to retrieve all those tweets, since we do not know how many tweets may have been posted for a specific term in a given time interval. Choosing the parameters mentioned above, we have realized that the execution time took 8 hours and 23 minutes (in a distributed system, it is the maximum execution time of the node that takes the longest time to complete its work), and 53,353,764 tweets have been collected in multiple files with comma-separated values (CSV) format, for a total physical size of 73 GB.

While working in a distributed cloud system the workload is shared among the nodes, choosing a sequential approach within a non-distributed actor-model it requires a longer execution time. The process includes one server which involves just one client. The client receives all the information coming from the server and works on each request by processing the data received from Twitter. However, this architecture shows no advantages at all, delaying the execution time up to 193 hours and 52 minutes using the same parameters we already focused on in the actor-model system.

5. Conclusion

The proposed actor-based approach shows how it is possible to obtain a large amount of data coming from Twitter, using a distributed cloud-based system. The API limits imposed by Twitter are incresingly difficult to avoid and restrict the scientific research progress in many field, such as the Natural Language Processing (NLP). The actor-model we choose optimizes every single action in order to guarantees a well distributed workload between all nodes, drastically reducing the possibility that an IP address may be banned by Twitter using a list of free proxies available on the web.

References

[1] Appeal from the United States District Court for the Northern District of California, https://parsers.me/appeal-from-the-united-states-district-court-for-the-northern-district-of-california, last accessed October 2020.

[2] Twitter Q1 2020 Financial Report, https://s22.q4cdn.com/826641620/files/doc_financials/2020/q1/Q1-2020-Earnings-Press-Release.pdf, last accessed October 2020.

[3] G. Lombardo, P. Fornacciari, M. Mordonini, L. Sani, M. Tomaiuolo, A combined approach for the analysis of support groups on facebook-the case of patients of hidradenitis suppurativa, Multimedia Tools and Applications 78 (2019) 3321–3339.

[4] E. Franchi, A. Poggi, M. Tomaiuolo, Social media for online collaboration in firms and organizations, International Journal of Information System Modeling and Design (IJISMD) 7 (2016) 18–31.

[5] P. Fornacciari, M. Mordonini, A. Poggi, L. Sani, M. Tomaiuolo, A holistic system for troll detection on twitter, Computers in Human Behavior 89 (2018) 258–268.

[6] M. Tomaiuolo, G. Lombardo, M. Mordonini, S. Cagnoni, A. Poggi, A survey on troll detection, Future Internet 12 (2020) 31.

[7] G. Lombardo, A. Ferrari, P. Fornacciari, M. Mordonini, L. Sani, M. Tomaiuolo, Dynamics of emotions and relations in a facebook group of patients with hidradenitis suppurativa, in: International Conference on Smart Objects and Technologies for Social Good, Springer, 2017, pp. 269–278.
[8] G. Angiani, S. Cagnoni, N. Chuzhikova, P. Fornacciari, M. Mordonini, M. Tomaiuolo, Flat and hierarchical classifiers for detecting emotion in tweets, in: Conference of the Italian Association for Artificial Intelligence, Springer, 2016, pp. 51–64.
[9] P. Fornacciari, M. Mordonini, Social network and sentiment analysis on twitter: Towards a combined approach., in: KDWeb, 2015.
[10] F. Bergenti, A. Poggi, M. Tomaiuolo, An actor based software framework for scalable applications, in: International Conference on Internet and Distributed Computing Systems, Springer, 2014, pp. 26–35.
[11] G. Lombardo, P. Fornacciari, M. Mordonini, M. Tomaiuolo, A. Poggi, A multi-agent architecture for data analysis, Future Internet 11 (2019) 49.
[12] R. Diouf, E. N. Sarr, O. Sall, B. Birregah, M. Bousso, S. N. Mbaye, Web scraping: State-of-the-art and areas of application, in: 2019 IEEE International Conference on Big Data (Big Data), 2019, pp. 6040–6042.
[13] A. Phaphuangwittayakul, S. Saranwong, S. Panyakaew, P. Inkeaw, J. Chaijaruwanich, Analysis of skill demand in thai labor market from online jobs recruitments websites, in: 2018 15th International Joint Conference on Computer Science and Software Engineering (JCSSE), 2018, pp. 1–5.
[14] Fatmasari, Y. N. Kunang, S. D. Purnamasari, Web scraping techniques to collect weather data in south sumatera, in: 2018 International Conference on Electrical Engineering and Computer Science (ICECOS), 2018, pp. 385–390.
[15] A. Maududie, W. E. Y. Retnani, M. A. Rohim, An approach of web scraping on news website based on regular expression, in: 2018 2nd East Indonesia Conference on Computer and Information Technology (EIConCIT), 2018, pp. 203–207.
[16] E. N. SARR, O. SALL, A. DIALLO, Factextract: Automatic collection and aggregation of articles and journalistic factual claims from online newspaper, in: 2018 Fifth International Conference on Social Networks Analysis, Management and Security (SNAMS), 2018, pp. 336–341.
[17] A. Amalia, R. M. Afifa, H. Herriyance, Resource description framework generation for tropical disease using web scraping, in: 2018 IEEE International Conference on Communication, Networks and Satellite (Comnetsat), 2018, pp. 44–48.
[18] B. B. P. Maurya, A. Ray, A. Upadhyay, B. Gour, A. U. Khan, Recursive stock price prediction with machine learning and web scrapping for specified time period, in: 2019 Sixteenth International Conference on Wireless and Optical Communication Networks (WOCN), 2019, pp. 1–3.
[19] K. Salah-ddine, K. Abouloula, E. Brahim, Money management limits to trade by robot trader for automatic trading, International Journal of Engineering 7 (2018).
[20] A. Trifa, A. H. Sbaï, W. L. Chaari, Evaluate a personalized multi agent system through social networks: Web scraping, in: 2017 IEEE 26th International Conference on Enabling Technologies: Infrastructure for Collaborative Enterprises (WETICE), 2017, pp. 18–20.
[21] R. Kruthika, P. Balasubramanian, V. Sureshkumar, Relationship between google trends

data and index returns, in: 2018 International Conference on Computation of Power, Energy, Information and Communication (ICCPEIC), 2018, pp. 042–045.
[22] A. M. Talib, R. Atan, R. Abdullah, M. Azrifah, Cloudzone: Towards an integrity layer of cloud data storage based on multi agent system architecture, in: 2011 IEEE Conference on Open Systems, 2011, pp. 127–132.
[23] F. De la Prieta, S. Rodríguez, J. Bajo, J. M. Corchado, A multiagent system for resource distribution into a cloud computing environment, in: Y. Demazeau, T. Ishida, J. M. Corchado, J. Bajo (Eds.), Advances on Practical Applications of Agents and Multi-Agent Systems, Springer Berlin Heidelberg, Berlin, Heidelberg, 2013, pp. 37–48.
[24] X. Kang, C. Zhou, X. Liu, H. Sun, Y. Huang, Improving performance for decentralized execution of composite web services, in: 2013 IEEE Ninth World Congress on Services, IEEE Computer Society, Los Alamitos, CA, USA, 2010, pp. 582–589.
[25] Python Requests Library, https://requests.readthedocs.io/, last accessed October 2020.
[26] Free Proxy List, https://free-proxy-list.net/, last accessed October 2020.