PyPandas: A data cleaning toolkits for Spark

Pei-Lun Liao
New York University
pll273@nyu.edu

Chia-Hsien Lin
New York University
chl566@nyu.edu

Shang-Hung Tsai
New York University
st3127@nyu.edu

ABSTRACT

Data cleaning is a huge challenge in data mining and machine learning tasks. In this paper, we propose PyPandas, a data cleaning toolkits running on Spark, to provide further solutions to data cleaning tasks. PyPandas provides advance cleaning features such as outlier detection, data scaling and text pattern searching and replacement. We have released PyPandas on GitHub and Python Package Index (PyPI).

1 INTRODUCTION

In recent years, as storage devices become cheaper, storing big data in thousands of computers is easier than before. People start thinking about how to extract useful information from the huge datasets. The term, Big Data, was created to describe this phenomenon[17]. As the result, data mining[12, 22] and machine learning[27] get popular nowadays. The quality of data is one of the key factors to successful data mining[3]. Hence, data cleaning becomes a challenge in the first step of data management and analysis[5, 13, 23].

To process huge datasets efficiently, distributed systems and parallel computing frameworks[7, 8, 28] were introduced. Spark[29] is one of the most popular open source projects for industry and academia. It is built on Hadoop[28] and provides a way to manage distributed memory. PySpark[20] is an extended library for Python programmers to work with Spark.

In the Python machine learning and data mining ecosystem, Pandas[18] is the most popular data management library. Pandas dataframe provides convenient ways to collect data and several data management methods for missing value filling, data indexing and data profiling. In PySpark SQL module, PySpark provides Pandas-liked DataFrame to handle data indexing, data integration, and simple data cleaning tasks such as missing value handling and duplication removal.

However, PySpark DataFrame does not support advance data processing features like outlier detection and data scaling. Although there exists several clustering and scaling algorithms in the PySpark ML module, the integration with PySpark DataFrame is not helpful and can be difficult to use for data cleaning. User has to go back and forth to process their dataset.

In this paper, we propose PyPandas, a data cleaning toolkits built on top of PySpark, that integrates PySpark ML module to support user friendly data processing features such as outlier detection and numerical data scaling. These features are important for the machine learning tasks and are supported in Scikit-Learn[21], a mainstream machine learning library, as well.

Furthermore, text data is another common data type in our storage. Text data is hard to process because of different languages, newly invented words, typos, abbreviations, urls and emoji etc. In this paper, we also implement easy to use text cleaning methods and provide flexibility for users to customize their cleaning processing.

The features of our library are summarized below.

- User friendly data cleaning toolkits built on PySpark
- Outlier detection and numerical data scaling
- Text processing such as url detection, punctuation removing, pattern searching and replacement.

2 RELATED WORK

There exists a number of open-source library for data cleaning and parallel data computing.

- Optimus[14] is a framework that can perform distributed data cleaning and preprocessing. This framework works well with Spark and its DataFrame can scale in big cluster. Optimus comes with helpful tools such as removing special characters and replacing null values.
- Dask[24] is another open-source library that supports parallel analytic computing. It provides scalable parallel data structures that extend interfaces like Pandas. At the same time, it offers low latency and high responsiveness.
- SparklingPandas[16] attempts to combine the power of Spark and Pandas to scale data analysis. It provides a Pandas-like API that is built using Spark's DataFrame class. Unfortunately, it only support spark v1.4 and Python 2.7, and its development has ended.

3 PROBLEM FORMULATION

In this paper, we focus on three major tasks, outlier detection, data scaling, and text pattern searching and replacement.

3.1 Outlier Detection

Outlier or anomalies detection is a challenging task in the field of data cleaning[4]. In this paper, we define outlier as the data that is far away from the its cluster centroid. Hence, given a dataset X, a clustering algorithm A[22], a distance measurement d, and the user-defined threshold t, the outlier is defined as following.

$$outlier = \{x | x \in X, d(x, c_{A_X(x)}) > t\}$$

where $A_X(x)$ assigns x into a cluster given the dataset X and $c_{A_X(x)}$ is the centroid of cluster to which x belongs. Our library implements the following clustering algorithms.

- K-Means and Bisecting K-Means Algorithm[15, 26]
  These two algorithms partition data points into k clusters, where each point belongs to the clusters that is closest to that point. In these two models, the distance measurement d is the Euclidean Distance.

$$d(x, c_{KMeans_X(x)}) = \sqrt{\sum_{i=1}^{n} (x_i - c_i)^2}$$

https://github.com/shtsai7/PyPandas
PyPandas can be installed by pip.

- Gaussian Mixture Model[19]
  The algorithm assumes a probabilistic model where data points are generated from some Gaussian distributions. The Gaussian distributions are parameterized by a mean $\vec{\mu}$ and a covariance matrix S. The distance measurement d is defined to be Mahalanobis Distance, which measures the distance between a point and a distribution.

$$d(x, c_{GMM_X(x)}) = \sqrt{(\vec{x} - \vec{\mu})^T S^{-1} (\vec{x} - \vec{\mu})}$$

3.2 Scaling and Normalization

Scaling and normalization are common preprocess in data cleaning. It is crucial to several machine learning models and optimization methods. Scaling increases the convergence speed and chance to find good optimum[2]. The scaling problem could be formulated as below. Given an one dimensional numerical dataset X, we provide a scaling method S such that scaled elements are bounded in same reasonable range $[a, b]$.

$$S_X(x) \in [a, b]$$

where $x \in X$ and $|a - b|$ should not be large.

To define the problem formally, we consider and implement several specific scaling methods.

- Standard scale

$$S(X) = \frac{X - \mu}{\sigma} \in \text{proper deviation range}$$

  where μ is the mean of X and σ is the standard deviation of X
- Min-Max scale

$$S(X) = \frac{X - min(X)}{max(X) - min(X)} \in [0.0, 1.0]$$

- Max-Abs scale

$$S(X) = \frac{X}{max(X)} \in [-1.0, 1.0]$$

- P-norm Normalization

$$S(X) = \frac{X}{\|X\|_p} \in [-1.0, 1.0]$$

3.3 Pattern Searching and Replacement

Pattern searching and replacement is a common technique in cleaning text data. Given a string t and a regular expression[25] re, we replace the matched pattern with a user-defined string s. The replacement function R could be defined as following. Consider t as a concatenation of non-overlapping substrings $t = t_1 ... t_n$.

$$R(t) = r(t_1) ... r(t_i) ... r(t_n)$$

$$r(t) = \begin{cases} s & \text{if } M \text{ accept } t \\ t & \text{otherwise} \end{cases}$$

such that M does not accept any substring of $R(t)$

where M is the generalized nondeterministic finite automaton[25] corresponding to the regular expression re. We do not define replacement order and simply ignore it.

4 METHODS, ARCHITECTURE AND DESIGN

In this project, we build our library on top of PySpark. We handle PySpark DataFrame and implement useful data cleaning functions in our library. We integrate PySpark ML module and customize user define functions to perform advanced cleaning features such as pattern searching and replacement with regular expression. We survey the PySpark[20], Pandas[18] and Scikit-Learn[21] libraries to find out key features and create Pandas-like and Scikit-Learn-like API to provide easy usage and seamless adaptation.

4.1 Outlier Detection

4.1.1 Design. Our goal in outlier detection is to integrate the existing clustering algorithms in PySpark ML module to provide user-friendly solution for outlier detection. Users are able to select existing clustering algorithms. Furthermore, users are given the power to decide their own threshold value based on their knowledge about the dataset to filter outliers. To help users identify outliers and pick a proper threshold, we provide statistics summary of the clustering.

4.1.2 Architecture. The implementation of the outlier detection functionality integrates the PySpark ml package. Our architecture hides the complexity of training and tuning clustering models, while it exposes easily accessible APIs for users to remove outliers. In addition, our data cleaning library implements User Defined Functions to compute distance between data points and cluster centers, as well as other summary statistics. At this time, our outlier remover can handle outliers with KMeans clustering, Bisecting KMeans, and Gaussian Mixture Model. We use Factory Design Pattern to construct different types of outlier removers with the generic APIs.

4.1.3 Methods.

- KMeans[15]: Users can specify the number of clusters k to use based their knowledge about the dataset. By default, The system will choose random initial values, and perform 20 iterations of updates. Finally, a summary of the clusterings will be generated, which includes cluster centers, cluster sizes, distances, etc. User can then filter out outliers in each cluster by specifying a minimum distance from the cluster center, and all data points that are beyond that distance will be removed from the dataset.
- Bisecting KMeans[26]: Bisecting Kmeans clustering algorithm is similar to KMeans. However, the algorithm does not start with k clusters. Instead, it gradually divides clusters and generates k clusters after a number of iterations.
- Gaussian Mixture[19]: Similarly, users can specify the number of clusters (i.e. Gaussian distributions). The algorithm performs 20 iterations of updates by default. A summary that contains mean, covariance matrix and average mahalanobis distance of each cluster can be returned. User can specify a minimum distance to filter out outliers.

4.2 Scaling and Normalization

4.2.1 Design. Standardization is a common requirement for many machine learning algorithms, since some optimization methods will not work properly without scaling or normalization[2]. Our

goal in scaling and normalization is similar to the goal of outlier detection. We will intergrate existing scaling algorithms in PySpark ML module into our library. User can choose existing scaling algorithms to scale their datasets.

4.2.2 Architecture. The architecture of scaling is simple. We wrap scaling logic into single function which allows user to transform their PySpark DataFrame easily.

4.2.3 Methods.

- Standard Scale: Performs basic scaling on a particular column or a list of columns in a dataframe so that the values have unit standard deviation and zero mean.
- Min-max Scale: Rescales columns to a user defined range(e.g. [0, 1]) using min max scaling.
- Max-abs Scale: Transforms columns to range between -1 and 1 by dividing through the maximum absolute value in the columns. This operation does not shift/center the data, and thus does not destroy any sparsity.
- Normalization: Given a list of columns, the normalize function can generate a normalized feature vector having unit norm. The default p-norm value for normalization is 2.0, yet users can optionally specify the p-norm value.

4.3 Pattern Searching and Replacement

4.3.1 Design. To provide a useful text cleaning toolkits, we need to have the flexibility for user to search any pattern and replace the matched ones with any user defined string. At the same time, we have to provide common patterns so that users won't need to reinvent the wheel. For example, url pattern, leading space, trailing space or numbers are common search patterns that we should provide. Moreover, users should be able to organize their cleaning logic and apply them to the text dataset easily.

4.3.2 Architecture. In order to achieve those design issues, we extract the common logic and column feature transformation into our core functions. We design the architecture such that user defined functions can be applied on a single or multiple columns, and a new DataFrame will be provided. The user defined function could be a regular expression substitution function, a column scaling function or a natural language processing function in NLTK[1]. Our architecture ensures the flexibility to various customized processing methods. Futhermore, we apply Decorator Design Pattern to enable users to register processing functions and customize their cleaning process.

4.3.3 Methods. We define several common regular expression patterns in table 1 and provide simple cleaning functions to handle those patterns in text data.

5 EXPERIMENT

5.1 Dataset

Three datasets are chosen for the experiments. Each of them has different properties and can be used to test different parts of the library. The summary of datasets could be found in table 2.

- 311 Service Requests[9] This dataset contains all 311 Service Requests since 2010. The majority of data has string type, including both categorical data (e.g. City, Status) and variable length data (e.g. Address, Description). This can be useful in testing the string-related features of the library.
- DOB Permit Issuance[10] The dataset consists of a list of permits for buildings in NYC issued by Department of Building. It has diverse data types, including categorical, string, numerical, and geographical data. Hence, it provides a good test case for the versatility of the library.
- DOB Job Application Filings[11] The dataset stores job applications filed through the Borough Offices in NYC. There are many columns containing numerical data, such as fee and number of stories. It can be used to test the data scaling and outlier detection features.

Table 1: Common Regular Expression Pattern

Common Pattern	Regular Expression
URL	URL regular expression[6]
Leading space	`'^ +'`
Trailing space	`' +$'`
Consecutive space	`' +'`
Number	`'\d+'`
Not a word	`'[^\w\d\s]+'`
Blank	`'_+'`

Table 2: Selected dataset

	(Row, Col)	Size	Property
311 Service Requests	(9M, 53)	6.01 GB	Text
Permit Issuance	(3M, 60)	1.43 GB	Mixed Type
Job Application Filings	(5M, 89)	2.88 GB	Numerical Value

5.2 Evaluation

We evaluate our project by comparing it against other existing open-source libraries, including Optimus[14] and Dask[24]. In addition, we examine the scalability of the library.

5.2.1 Features. We compare our data cleaning features with previous works, particularly those frequently used functionality in data science such as null value handling, special character removal, duplicate detection, etc.

5.2.2 Performance. We evaluate the performance of our library by measuring the time consumed and drawing comparison with existing libraries. This performance evaluation shows that this library is efficient and can be useful in production.

5.2.3 Scalability. We examine the scalability of our library by running the same task with different numbers of worker nodes. We measure the time consumed under different configuration and analyze the scalability. This evaluation shows how well the library can scale.

5.3 Results

After running our experiments on the Dumbo clusters of NYU initially, we migrate all our settings to Amazon Web Service because we could not obtain the privileges needed on Dumbo. While on

Table 3: Evaluation: Features Comparison

	PySpark DataFrame[20]	PyPandas	Optimus[14]	SparkingPandas[16]	Dask[24]
Data manipulation					
Data integration	✓	✓	✓		✓
Data aggregation	✓	✓	✓	✓	✓
Data filtering	✓	✓	✓		✓
Data profiling	✓	✓	✓	✓	✓
Data sorting	✓	✓	✓		
Basic data cleaning					
Missing value filling	✓	✓	✓		✓
Missing value removing	✓	✓	✓		✓
Duplication removing	✓	✓	✓		✓
Outlier Detection					
Deviation with median			✓		
Clustering algorithms		✓			
Scaling and Normalization					
Standard scaling		✓			✓
Scale in range		✓	✓		✓
Normalization		✓	✓		
Text Cleaning					
Special character removing	✓	✓	✓		
Pattern searching	✓	✓	✓		
Pattern searching and replacement		✓	✓		
Common patterns replacement		✓			

Amazon Web Service, we can obtain the privileges needed to install other packages and we are able to perform comparison with other related works. In addition, on Amazon Web Service, we are able to customize the clusters we need, such as setting the number of worker nodes. This allows us to perform the scalability experiments.

5.3.1 Features. In this section, we compare our features with previous works. The result is shown in table 3. We find that PySpark DataFrame provide easy-to-use data manipulation functions. Since PyPandas and Optimus extend the DataFrame, both of them support the same data manipulation features. In SparkingPandas, the project only aims to provide grouping methods on Pandas-liked dataframe and miss other important features in data manipulation. Dask also support many features in data manipulation.

PySpark DataFrame, Optimus, Dask, and our library PyPandas all provide the basic data cleaning functions like missing value handling. In the outlier detection, Optimus detects outliers by calculating the deviation with the median of data. On the other hand, PyPandas provides advance detection methods with clustering algorithms. Dask do support several clustering algorithms. Nonetheless, they are not integrated in their Dask DataFrame. Users have to manually train and tune the models to detect the outliers.

In scaling and normalization, Optimus, Dask and PyPandas provide basic scaling methods like min-max scaling. However, Dask does not provide normalization on their Dask ML preprocessing module. And Optimus does not support standard scaling.

For the text cleaning, Optimus and PyPandas provides more powerful pattern searching and replacement feature to let user find out any pattern and replace the matched one with their desired value. Moreover, PyPandas defines several common patterns for users to clean their text quickly without repeating themselves and users are able to register their cleaning logic to customize their process in text cleaning. In summary, PyPandas extends PySpark DataFrame and provides more easy-to-use features for users to clean their data.

5.3.2 Performance. We compare the performance of functions including min max scaling, normalization and special character cleaning in our library against those in Optimus on AWS EMR with the identical environment configuration. The total of execution times of each task, including the time for Spark initialization, data loading, and data preprocessing, are all recorded and listed below. Even though the execution times seems to be slightly slower than Optimus, we do provide more features in PyPandas.

- Scaling and Normalization: In this experiment, the min max scaling and the normalization are performed on the 311 Service Requests Dataset. We select all the numeric columns in the dataset to do the experiment. The experiment results can be found in table 4.

Table 4: Scaling and Normalization Experiment Result

	Min-Max Scaling	Normalization
PyPandas	391.11s	105.90s
Optimus	142.46s	61.92s

Table 5: Special Character Cleaning Experiment Result

	311 Service Requests	Permit Issuance	Job Application Filings
PyPandas	104.18s	62.35s	60.07s
Optimus	86.19s	43.67s	43.04s

Table 6: Outlier Detection - Scalability Experiment Result

Number of Worker Nodes	KMeans	Gaussian Mixture
2	650.1718 s	1408.2266 s
4	394.6293 s	862.6323 s
6	299.1153 s	622.7715 s
8	270.4852 s	592.1259 s

- Special Character Cleaning: In this experiment, the special character cleaning is performed on the 311 Service Requests Dataset. We execute the cleaning function to the entire dataset. The results are shown in table 5.

5.3.3 Scalability. We evaluate the scalability of different functionalities of PyPandas. In our measurement, we uses the total execution times, which includes the time for Spark initialization, data loading, and data preprocessing.

For each task, we use 4 configurations of the clusters. The 4 configurations are exactly the same, except they have different numbers of worker nodes. On each configuration, the same task is performed 3 times, and the average time consumed is record.

From the experiment, we find that PyPandas outlier remover has great scalability. Each time we double the number of worker nodes, the performance improves significantly. Note that the improvement is not perfectly linear, because we include the spark initialization and data loading time in our measurement. As the number of workers increases, the overhead of the cluster also increases.

- Outlier Detection: This experiment is performed on the 311 Service Requests Dataset. We perform outlier detection on one particular column that contains numerical values. We use both K-Means and Gaussian Mixture Outlier Remover with k value of 5. The results are shown in table 6.
- Scaling and Normalization: This experiment is performed on the 311 Service Requests Dataset. We choose all the columns containing numeric values in the dataset to do scaling and normalization. The results are shown in table 7.
- Text Cleaner: This experiment is performed on the 311 Service Requests Dataset. We perform the sub_with_pattern function to replace all the special characters appear in the dataset with empty value. The results are shown in table 8.

Table 7: Scaling and Normalization - Scalability Experiment Result

# of Nodes	standard_scale	min_max_scale	max_abs_scale	normalize
2	471.40s	461.14s	470.56s	148.06s
4	306.26s	283.11s	291.03s	103.33s
6	229.64s	233.15s	226.09s	97.17s
8	210.12s	200.69s	200.07s	94.43s

Table 8: Text Cleaner - Scalability Experiment Result

# of Nodes	311 Service Requests	Permit Issuance	Job Application Filings
2	152.39s	82.42s	87.85s
4	74.56s	64.77s	65.69s
6	81.51s	62.73s	63.12s
8	67.59s	60.67s	62.87s

6 CONCLUSIONS

Due to the lack of advanced data cleaning library on Spark ecosystem, we propose PyPandas, a data cleaning toolkit built on PySpark. We provide useful advanced data cleaning features such as outlier detection, data scaling, and text pattern searching and replacement. The library is easy to install and has user friendly APIs. PyPandas also has good performance and scalability.

REFERENCES

[1] Steven Bird, Ewan Klein, and Edward Loper. 2009. *Natural Language Processing with Python* (1st ed.). O'Reilly Media, Inc.

[2] Stephen Boyd and Lieven Vandenberghe. 2004. *Convex Optimization.* Cambridge University Press, New York, NY, USA.

[3] Li Cai and Yangyong Zhu. 2015. The Challenges of Data Quality and Data Quality Assessment in the Big Data Era. *Data Science Journal* 14, 2 (2015), 1–10. https://doi.org/10.5334/dsj-2015-002

[4] Varun Chandola, Arindam Banerjee, and Vipin Kumar. 2009. Anomaly Detection: A Survey. *ACM Comput. Surv.* 41, 3, Article 15 (July 2009), 58 pages. https://doi.org/10.1145/1541880.1541882

[5] Xu Chu, Ihab F. Ilyas, Sanjay Krishnan, and Jiannan Wang. 2016. Data Cleaning: Overview and Emerging Challenges. In *Proceedings of the 2016 International Conference on Management of Data (SIGMOD '16).* ACM, New York, NY, USA, 2201–2206. https://doi.org/10.1145/2882903.2912574

[6] Dans. [n. d.]. Regular Exp Url. ([n. d.]). https://www.regextester.com/53716

[7] Jeffrey Dean and Sanjay Ghemawat. 2008. MapReduce: Simplified Data Processing on Large Clusters. *Commun. ACM* 51, 1 (Jan. 2008), 107–113. https://doi.org/10.1145/1327452.1327492

[8] Sanjay Ghemawat, Howard Gobioff, and Shun-Tak Leung. 2003. The Google File System. In *Proceedings of the Nineteenth ACM Symposium on Operating Systems Principles (SOSP '03).* ACM, New York, NY, USA, 29–43. https://doi.org/10.1145/945445.945450

[9] NYC government. 2018. NYC OpenData. (2018). https://data.cityofnewyork.us/Social-Services/311-Service-Requests-from-2010-to-Present/erm2-nwe9

[10] NYC government. 2018. NYC OpenData. (2018). https://data.cityofnewyork.us/Housing-Development/DOB-Permit-Issuance/ipu4-2q9a/data

[11] NYC government. 2018. NYC OpenData. (2018). https://data.cityofnewyork.us/Housing-Development/DOB-Job-Application-Filings/ic3t-wcy2

[12] Jiawei Han, Micheline Kamber, and Jian Pei. 2011. *Data Mining: Concepts and Techniques* (3rd ed.). Morgan Kaufmann Publishers Inc., San Francisco, CA, USA.

[13] Ihab F. Ilyas and Xu Chu. 2015. Trends in Cleaning Relational Data: Consistency and Deduplication. *Foundations and TrendsÂő in Databases* 5, 4 (2015), 281–393. https://doi.org/10.1561/1900000045

[14] Iron. 2018. Optimus. (2018). https://github.com/ironmussa/Optimus

[15] Tapas Kanungo, David M. Mount, Nathan S. Netanyahu, Christine D. Piatko, Ruth Silverman, and Angela Y. Wu. 2002. An Efficient k-Means Clustering Algorithm: Analysis and Implementation. *IEEE Trans. Pattern Anal. Mach. Intell.* 24, 7 (July 2002), 881–892. https://doi.org/10.1109/TPAMI.2002.1017616

[16] Holden Karau and Juliet Hougland. 2015. SparklingPandas. (2015). https://github.com/sparklingpandas/sparklingpandas
[17] A. Katal, M. Wazid, and R. H. Goudar. 2013. Big data: Issues, challenges, tools and Good practices. In *2013 Sixth International Conference on Contemporary Computing (IC3)*. 404–409. https://doi.org/10.1109/IC3.2013.6612229
[18] Wes McKinney. [n. d.]. pandas: a Foundational Python Library for Data Analysis and Statistics. ([n. d.]).
[19] G. McLachlan and D. Peel. 2004. *Finite Mixture Models*. Wiley. https://books.google.com/books?id=7M5vK8OpXZ4C
[20] Amit Nandi. 2015. *Spark for Python Developers*. Packt Publishing.
[21] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. 2011. Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research* 12 (2011), 2825–2830.
[22] Anand Rajaraman and Jeffrey David Ullman. 2011. *Mining of Massive Datasets*. Cambridge University Press, New York, NY, USA.
[23] Vijayshankar Raman and Joseph M. Hellerstein. 2001. Potter's Wheel: An Interactive Data Cleaning System. In *Proceedings of the 27th International Conference on Very Large Data Bases (VLDB '01)*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 381–390. http://dl.acm.org/citation.cfm?id=645927.672045
[24] Matthew Rocklin. 2015. Dask: Parallel Computation with Blocked algorithms and Task Scheduling. In *Proceedings of the 14th Python in Science Conference*, Kathryn Huff and James Bergstra (Eds.). 130 – 136.
[25] Michael Sipser. 1996. *Introduction to the Theory of Computation* (1st ed.). International Thomson Publishing.
[26] Michael Steinbach, George Karypis, and Vipin Kumar. 2000. A comparison of document clustering techniques. In *In KDD Workshop on Text Mining*.
[27] Vladimir N. Vapnik. 1995. *The Nature of Statistical Learning Theory*. Springer-Verlag New York, Inc., New York, NY, USA.
[28] Tom White. 2009. *Hadoop: The Definitive Guide* (1st ed.). O'Reilly Media, Inc.
[29] Matei Zaharia, Mosharaf Chowdhury, Michael J. Franklin, Scott Shenker, and Ion Stoica. 2010. Spark: Cluster Computing with Working Sets. In *Proceedings of the 2Nd USENIX Conference on Hot Topics in Cloud Computing (HotCloud'10)*. USENIX Association, Berkeley, CA, USA, 10–10. http://dl.acm.org/citation.cfm?id=1863103.1863113