PatchWork, a Scalable Density-Grid Clustering Algorithm

Frank Gouineau
Development and Internet Technologies
Computer Research Institute of Montreal, Canada
frank.gouineau@gmail.com

Tom Landry
Vision and Imaging
Computer Research Institute of Montreal, Canada
tom.landry@crim.ca

Thomas Triplet*
Development and Internet Technologies
Computer Research Institute of Montreal, Canada
thomas.triplet@crim.ca

ABSTRACT

Clustering is a fundamental task in Knowledge Discovery and Data mining. It aims to discover the unknown nature of data by grouping together data objects that are more similar. While hundreds of clustering algorithms have been proposed, many are complex and do not scale well as more data become available, making then inadequate to analyze very large datasets. In addition, many clustering algorithms are sequential, thus inherently difficult to parallelize. We propose PatchWork, a novel clustering algorithm to address those issues. PatchWork is a distributed density clustering algorithm with linear computational complexity and linear horizontal scalability. It presents several desirable characteristics in knowledge discovery, in particular, it does not require *a priori* the number of clusters to identify, and offers a natural protection against outliers and noise. In addition, PatchWork makes it possible to discover spatially large clusters instead of dense clusters only. PatchWork relies on the map/reduce paradigm to parallelize computations and was implemented using Apache Spark, the distributed computation framework. As a result, PatchWork can cluster a billion points in a few minutes only, a 40x improvement over the distributed implementation of *k-means* in Spark MLLib.

CCS Concepts

•Theory of computation → MapReduce algorithms; *Distributed computing models;* **•Computing methodologies → Cluster analysis;**

Keywords

Clustering, Big-data, Hadoop, Spark, Real-time

*Corresponding author.

SAC 2016,April 04-08, 2016, Pisa, Italy

 ISBN 978-1-4503-3739-7/16/04...$15.00
DOI: http://dx.doi.org/10.1145/2851613.2851643

1. INTRODUCTION

Clustering, also called *unsupervised learning*, is a fundamental tool in Knowledge Discovery and Data mining to extract knowledge from data with little or no ground truth. Given a set of n points in a d-dimensional feature-space D and a similarity function, the purpose of clustering is to group data points in k homogeneous sets that are maximally separable.

1.1 Applications

Many applications now rely on massive amounts of data that may require processing in real-time. Notable use-cases of clustering algorithms in the context of big-data analysis include the Internet-of-Things (IoT), smart cities and remote sensing.

The Internet of Things (IoT) is one of the key applications of big-data and machine-learning. It is a recent domain that emerged from the proliferation of devices that are connected to the Internet or to other devices. Examples of such connected devices include smartphones, drones, wearable electronics (e.g. clothes and watches), light bulbs, home appliances, etc... The IoT also has a tremendous potential in numerous industries such as domotics, transportation, retail, health or resource consumption. Those devices are usually equipped with a variety of sensors that can collect data in real-time or several times per minute. They also often include a geolocation tracker or can be paired with a device that has a geolocation tracker. Those connected devices thus generate very large amounts of data with a strong spatio-temporal component. Those devices rely on the integration of many spatio-temporal data sources, including the geolocation of the user (from their connected smartphone) and meteorological data. One of the key benefits of the IoT is to enable machine-to-machine communication thereby facilitating the automation of various tasks. Automation relies on the spatio-temporal data collected by the devices, as well as rules to define triggers and actions. Web services to facilitate the implementation of those rules have emerged and are gaining popularity as new devices become connected. Some devices are now relying on machine learning algorithms to learn how they are being used. Connected home thermostats for instance are now capable of learning from the habits of the home owners to automatically define rules and triggers to adjust the temperature.

Another key application of large scale information systems is the modeling of public infrastructures for the development of

smarter cities. Smart cities such as Barcelona, Stockholm, or Montreal heavily rely on digital technologies to reduce resource consumption and to engage more effectively with their citizens. An example of smart city projects, that also relies on the IoT, is the public bicycle sharing systems such as Bixi[1] that are implanted in many large cities. In such a system, bikes are equipped with GPS trackers, allowing the operator to monitor the usage and adjust the service accordingly [27]. Studies of the usage of public bicycle sharing systems using clustering algorithms [33] were also conducted to reveal structures of social communities in major cities.

Medical area and disaster monitoring.

As transportation means have allowed to travel faster around the globe, more effective infection monitoring tools are needed to help in the control of disease outbreaks as illustrated by the recent H1N1 flu and Ebola pandemics. The ability to quickly analyze the evolution of the disease, and to discover patterns in the data, is critical to understand the root cause of the pandemics and take appropriate measures to control an emerging disease situation and prevent their further spread.

Epidemiological data consist of spatio-temporal data describing the evolution of the disease in both space and time. Key challenges of epidemiological data are the ability of analyze new trends and patterns in pseudo-real-time as the disease spreads, as well as the recursive nature of those patterns, that is, patterns from previous pandemics are likely to give important clues for the prediction of the evolution of the current pandemics. Note that those challenges are not unique to pandemics, and other disasters that have strong geospatial and temporal dimensions, such as tornadoes, water flooding or oil spills, share the same characteristics.

1.2 Distributed systems and scalability

It is possible to scale a system *vertically* to some extent, that is add extra storage, more memory or a faster CPU to a single machine. This approach is however usually costly, remains sensible to failures, and does not scale well with the number of users or as more data become available.

As an alternative, it is also possible to scale a system *horizontally*, that is, combine several commodity servers to solve a problem too complex for a single machine. Besides the sheer amount of data, distributed architectures have many benefits compared to monolithic systems, including greater reliability, higher availability and better performance depending on the use-cases. However, distributed systems also present a number of challenges, network latency and hardware failure to name a few.

To be effective, this horizontal approach also requires specially designed algorithms that can be efficiently distributed between the different machines. Yet, while data clustering has been extensively studied and proved to be tremendously useful in data mining and knowledge discovery, clustering of big-data presents several challenges.

In this paper,we introduce PatchWork, a novel clustering algorithm to address those issues. PatchWork is a distributed density clustering algorithm with linear computational complexity and near-linear horizontal scalability.

The rest of the paper is organized as follows. Section 2 presents a review of basic concepts and related work. Section 3 describes the algorithm in details. Section 4 describes some experiments on synthetic and real datasets, and discusses their significance. Finally, Section 6 gives some concluding remarks and open problems.

[1]http://www.publicbikesystem.com/

2. RELATED WORK

While hundreds of algorithms are now available, clustering techniques can be categorized into three broad categories: distance-based, density-based and grid-based algorithms. An extensive survey of the principal clustering algorithms in the context of Big-Data applications was conducted by Fahad *et al.* [5]. In this section, we briefly describe properties of the algorithms that are important for their parallelization in a distributed environment, then we introduce two widely popular distributed computation engines Hadoop and Spark.

2.1 Distance-based algorithms

Distance-based algorithms rely on the distance between data points in the feature space to establish the clusters. Distance-based algorithms assume that clusters to find are of similar shapes and enforce a certain notion of spatial homogeneity between neighborhood points. They will perform well if this hypothesis is verified by the actual data. It is also possible to further categorize clustering techniques depending on the output of the algorithm [14]: hierarchical or partitioning.

2.1.1 Partitioning algorithms

They define mutually exclusive hard clusters, or soft clusters that allow a certain degree of overlap measured by a membership function. Fuzzy clustering techniques are typical of this kind of soft partitioning approach [15]. The most popular partitioning algorithm is k-means clustering [20, 19]: given k clusters to find, the technique determines the centers of the clusters, and updates the membership of each cluster iteratively using the distance to the center of the cluster. The approach can be easily parallelized and distributed given that computing the distance to the cluster centers has no dependencies. Several distributed implementations are available, in particular for Hadoop and Spark, as part of the Mahout and MLLib libraries respectively (see section 2.4).

2.1.2 Hierarchical algorithms

They represent the output clusters as a nested set of partitions, that is, as a tree also called dendrogram. This family of algorithms is particularly useful to discover evolutionary relationships between data points and is very popular in bioinformatics and genomic and phylogenetic studies. Hierarchical techniques implementing a divisive top-down strategy where a larger cluster is split into several sub-clusters, have been proposed. However, hierarchical agglomerative clustering (HAC) is the most popular strategy. It is a bottom-up strategy that iteratively groups together the two most similar clusters to form a new cluster. The com-

putation of the similarity measurement between two clusters depends on the linkage method and is one the main factor that differentiates HAC methods. In single-linkage clustering, the link between two clusters is made by a single element pair of the two elements (one in each cluster) that are closest to each other. In complete-linkage clustering, the link between two clusters considers all element pairs, and the distance between clusters equals the distance between those two elements (one in each cluster) that are farthest away from each other. Other linkage methods such as UPGMA[25] have been proposed and often used in bioinformatics. However, HAC algorithms rely on a global distance matrix, and are notoriously difficult to parallelize. Furthermore, most of those algorithms have a quadratic computational complexity[9] with the number of data points and do not scale well.

2.2 Density-based algorithms

Density-based algorithms on the other hand make use of the density of the data points within a region to discover the clusters. Unlike distance-based techniques, density-based algorithms can uncover clusters of arbitrary shapes, but assume that they are of similar density. They present several desirable characteristics in data mining, in particular, they do not require *a priori* the number of clusters to identify, and offers a natural protection against outliers and noise.

Density-based algorithms are easier to parallelize and more scalable as they usually rely on local search techniques to identify dense regions in the feature space. One of the most popular density-based algorithm is DBSCAN [21], and many distributed variants [10, 22, 17, 23, 29] have been devised. Another popular example of density-based algorithm is DenClue [12] and its recent improvements [11], which is also suitable for segmenting and clustering raster data.

2.3 Grid-based algorithms

Multiple regions can be explored simultaneously by discretizing the input feature space into a finite number of grid cells, then apply the clustering method within each cell. Existing algorithms include STING [26] and WaveCluster [24, 16]. Parallel grid-based clustering further divide cells into sub-cells, process each sub-cell and combine the individual results to build the final clusters [28, 34]. This approach is particularly useful to mine spatial data as the grid can be defined using Hilbert or Z-order space filling curves [3, 13].

2.4 Distributed computation engines

While they have been studied for many years, horizontally distributed computing systems became widely popular in 2004 when Google described the architecture of its distributed file system GFS [6], and the MapReduce [4] paradigm they modernized to process and index billions of web pages.

2.4.1 Apache Hadoop

The popular open-source framework Apache Hadoop[2] is implements and reproduces the distributed architecture developed by Google. It implements in particular HDFS, a distributed file system, and MapReduce which relies on a form of divide-and-conquer technique to simplify some of the challenges of distributed computing: a complex task is first decomposed into simpler tasks that are executed on the machines of the cluster (*map*), the individual results are then aggregated (*reduce*) to produce the desired output. This approach is very effective for batch processing or when subsets of the data can be processed individually.

One of the most popular machine learning library for Hadoop is Apache Mahout[3]. It implements a wide variety of algorithms from supervised learning to collaborative filtering and clustering (including several variants of *k-means*, see section 2.1).

2.4.2 Apache Spark

More recently, the MapReduce paradigm was generalized and improved to handle streaming data, which are routinely needed for numerous applications like smart cities, and several alternative open-source frameworks supporting this use-case are now available, including Apache Storm[4] and Apache Spark [31]. The key feature of Spark is the introduction of Resilient Distributed Datasets [30], a distributed memory abstraction that lets programmers perform in-memory computations on large clusters in a fault-tolerant manner. As a result, Spark can improve the performance of Hadoop by two orders of magnitude, and its popularity is now surpassing Hadoop [8].

One of the core components of Spark is MLLib, a machine learning library that implements several key supervised and unsupervised learning algorithms. The Spark MLLib implementation of *k-means* relies on *k-means*‖ [1], a significantly faster variant that features many refinements to the base algorithm. Additional Spark Packages[5] are also available to extends the features of Spark, including a distributed implementation of the density-based DBSCAN algorithm [18].

In our experiments, we evaluated and compared our approach against the Spark implementations of *k-means*‖ and DBSCAN.

3. PATCHWORK ALGORITHM

PatchWork is a distributed density clustering algorithm with linear computational complexity and near-linear horizontal scalability. It is a mixture of density and grid-based methods. As all density algorithms, PatchWork presents several desirable characteristics in data mining, in particular, it can discover clusters of arbitrary shapes, does not require *a priori* the number of clusters to identify, and offers a natural protection against outliers and noise.

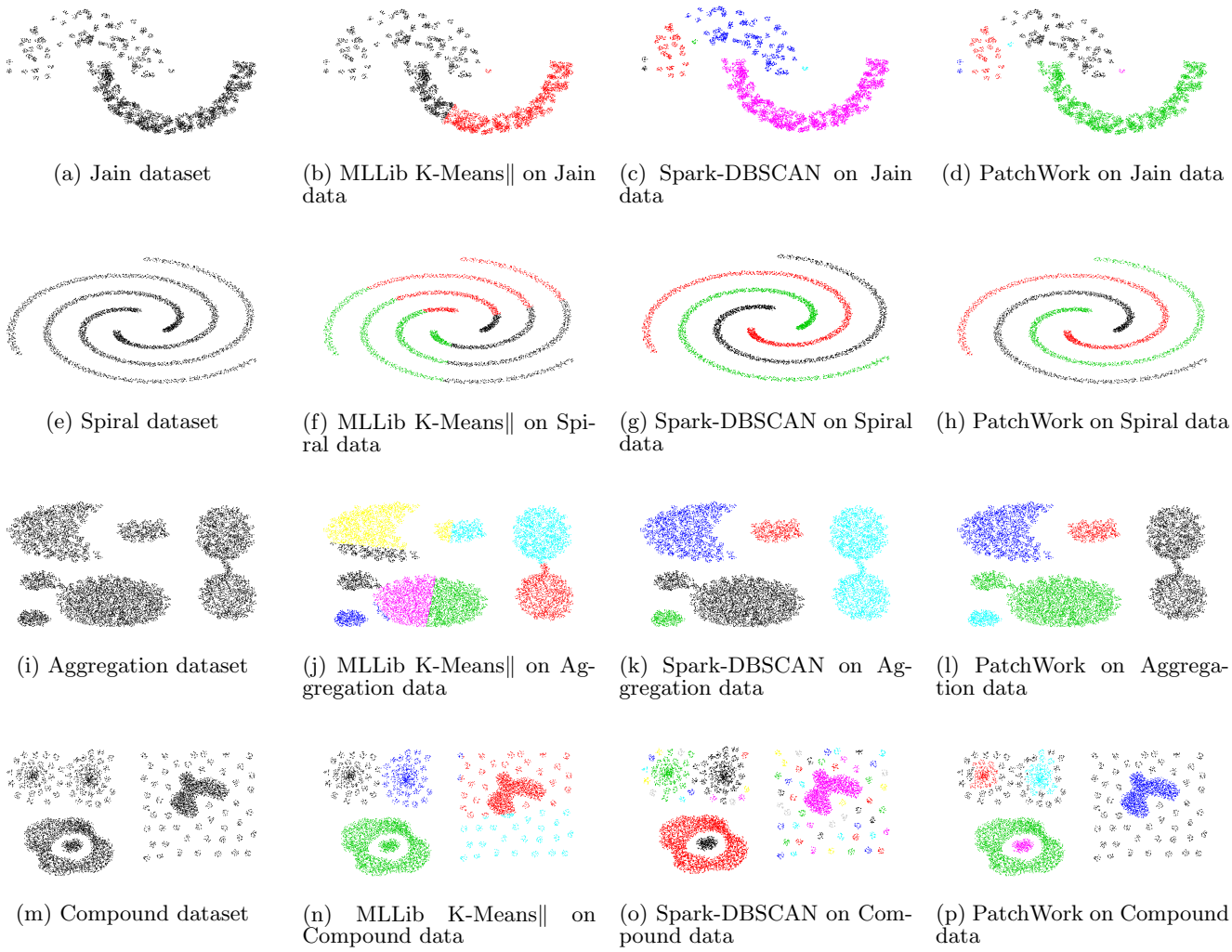
The remaining of this section is organized as follows: we first discuss the theoretical foundations of the algorithm, then we describe the main steps of our approach, the parallelization using $map()$ et $reduce()$ functions and details about the implementation using Spark.

[2]http://hadoop.apache.org/

[3]http://mahout.apache.org/

[4]http://storm.apache.org/

[5]http://spark-packages.org



(a) Jain dataset (b) MLLib K-Means‖ on Jain data (c) Spark-DBSCAN on Jain data (d) PatchWork on Jain data

(e) Spiral dataset (f) MLLib K-Means‖ on Spiral data (g) Spark-DBSCAN on Spiral data (h) PatchWork on Spiral data

(i) Aggregation dataset (j) MLLib K-Means‖ on Aggregation data (k) Spark-DBSCAN on Aggregation data (l) PatchWork on Aggregation data

(m) Compound dataset (n) MLLib K-Means‖ on Compound data (o) Spark-DBSCAN on Compound data (p) PatchWork on Compound data

Figure 1: Synthetic datasets. Clustering results of K-Means, DBSCAN and PatchWork on the *Jain*, *Spiral*, *Aggregation* and *Compound* datasets. For clarity, each dataset shown here contains $n = 5000$ data points.

3.1 Definitions and parameters

This paragraph defines several terms used in the description and proves a theorem used below to explain the algorithm.

Point: a point is an instance of the data located in the D-dimensional feature space.
Cell: a cell is a hypercube in the D-dimensional feature space. A cell has a list of points that it contains and a reference to the cluster it belongs to.
CellSize (ϵ): ϵ is a user-defined parameter that determine the cell size. Rather that a single value, ϵ is a D-dimensional array, allowing the user to adjust the size for all dimensions independently. This feature is particularly useful when the range of possible values varies greatly between dimensions or when data are not normalized.
Density: number of points inside the cell.
MinPts: optional user-defined threshold to specify the minimum number of data points needed in a cell to consider it as part of a cluster. If not specified, all cells are processed.
CellID: unique identifier of a cell. Consider a point $P = (p_1, ..., p_i, ..., p_D) \in \Re^D$. P belongs to the cell K such that $K = (k_1, ..., k_i, ..., k_D)$ with $\forall i\ [1, D], k_i = \lfloor p_i/\epsilon_i \rfloor$
Ratio: user-defined parameter to control the density threshold used to expand existing clusters.
MinCell: optional user-defined threshold to filter out clusters with too few cells. This parameters allows users to keep only spatially large clusters rather that dense ones only.

3.2 Procedure

PatchWork rely on the general map/reduce paradigm to parallelize computations and was implemented using Apache Spark, a fast and general engine for large-scale data processing.

First, the algorithm divides the multi-dimensional feature space into a grid. This grid allows us to efficiently parallelize local search computations to identify dense regions. The size of the grid ϵ is an input parameter of the algorithm

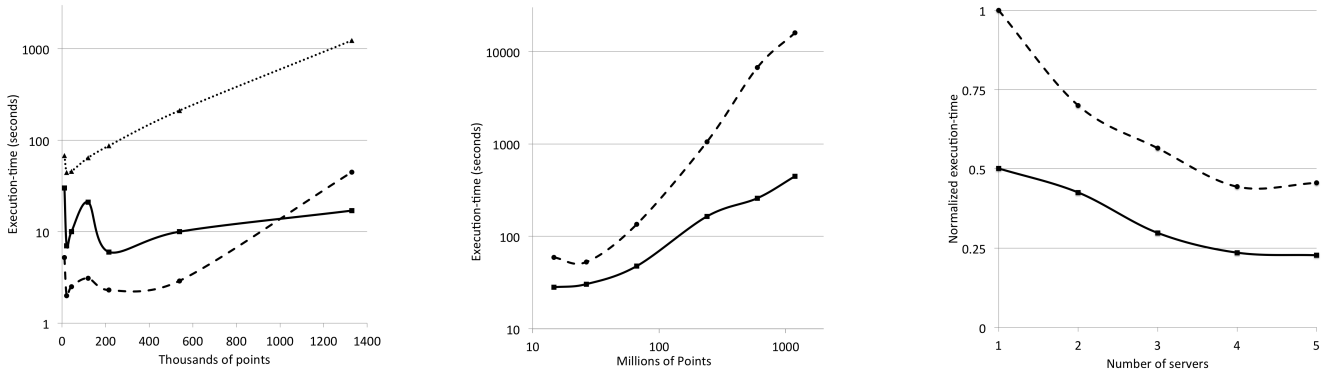


Figure 2: Performance on synthetic datasets. The dashed line represents Spark MLLib *k-means‖*, the dotted line Spark-DBSCAN, and the solid line PatchWork. (a) Running-time in seconds for smaller datasets. Note that the Y-axis is logarithmic. (b) Running-time in seconds for larger datasets. Note that both X and Y-axis are logarithmic. For large datasets, the running-time of Spark-DBSCAN became unpractical. (c) Horizontal scalability of the algorithms using 100 millions data points. Running-time are normalized.

and can impact its performance. A key feature of our algorithm is that it is possible to compute the id of the cells each point belongs to, without any preprocessing or first pass on the data. As a result, PatchWork can process data point independently of the $n-1$ other points, which is an important property to efficiently distribute the computations between servers. As a bonus, this characteristics makes PatchWork suitable to analyze data streams, such as those generated by connected devices.

Computing the CellID for each data point can be efficiently performed using a mapping function executed on the RDD collection of data points. For each data point P, the mapping function returns the tuple $(cellID(P), 1)$ and can be executed in constant time $O(1)$:

$$map(P(p_1, ..., p_D)) => ((\lfloor p_1/\epsilon_i \rfloor, ..., \lfloor p_D/\epsilon_i \rfloor), 1)$$

The collection (RDD) of n tuples $(cellID(P), 1)$ is then shuffled using $cellID$ as key, and reduced using a sum operator. The output the of the reduction step is a collection (RDD) of m_ϵ tuples $(cellID, density)$, where m_ϵ is the number of cells in the grid and depends on ϵ. When processing a data streams, new data points can be processed in constant time to update the collections of cells and their density.

Second, the algorithms creates the clusters using the collection of cells and their cardinality as input. In practical applications, we have $m_\epsilon << n$, which dramatically reduce the computational complexity of the algorithm.

If $minPoints$ is defined by the user, it is possible to filter out cells with only a few data points. This optional filter can help reduce the noise if needed. The collection of filtered cells is sorted by decreasing density. It create a first cluster C with the first cell, that is, the cell with the highest density of data points. Nearby cells C_i are explored, and expand the current cluster if their density is sufficient, that is, if $Density(C_i) > Density(C) * Ratio$. This step is repeated until all cells are processed.

The final set of clusters may be filtered to keep only those with enough cells. This optional filter ensure that clusters are *spatially* large enough and consistent with the intended application, and also helps reduce the noise caused by tiny clusters. This parameter adds a new dimension to clustering, and makes it possible to discover large spatial clusters instead of dense clusters only.

3.3 Limitations

The main limitation of the current implementation is the handling of high-dimensional datasets: the computational complexity to determine the neighborhood of a cell with D dimensions is exponential with D. Dimensionality reduction algorithms may be used to preprocess the data and keep running-times acceptable for the intended application.

In addition, like all density and grid-based clustering algorithms, the choice of the parameter ϵ can impact the performance of the algorithm.

4. EXPERIMENTAL RESULTS

We evaluated and compared PatchWork against the Spark MLLib native implementation of the *k-means‖* algorithm, as well as a Spark implementation of DBSCAN. Spark implementations were selected over MapReduce/Mahout because of the performance improvements of Spark over MapReduce.

4.1 Experimental setup

We benchmarked the performance of PatchWork and other algorithms using a cluster of six commodity servers, each with an Intel Xeon CPU E5-2650 processor with 8 cores at 2.60GHz, 192GB of RAM and 30TB of storage. The servers are interconnected using 10 Gigabit Ethernet. Experiments were conducted using HDFS 2.6.0 and Apache Spark 1.3.0 packaged in the Cloudera CDH 5.4.0 distribution[6].

[6]http://www.cloudera.com

4.2 Synthetic datasets

We first evaluated PatchWork using four synthetic datasets commonly used to evaluate clustering algorithms. Those datasets simulate different distributions of the data: *Jain*, *Spiral* [2], *Aggregation* [7] and *Compound* [32] datasets. The size of those datasets is very limited and not suitable for big-data applications. Hence, for each of those four datasets, we generated datasets of increasing size, up to 1.2 billion entries. Those datasets have the same data distribution as the original datasets while allowing us to test the scalability of the algorithms.

The parameters of each algorithms were empirically tuned for each dataset. The results of these experiments are shown in Fig.1.

First, we notice that MLLib k-means is not capable of discovering clusters of arbitrary shapes, as anticipated. Overall, PatchWork and Spark-DBSCAN have very similar results and can perfectly handle arbitrary shapes and paths (Fig.1h and Fig.1g).

On the other hand, k-means performs well when identifying clusters of similar shapes (Fig.1j on the right and Fig.1n, top-left corner). The two density algorithms wrongly merged the two clusters joined by noisy data points. The *Compound* dataset shows that unlike DBSCAN, PatchWork is aware of the spatial size of the cluster and can recognize small noisy clusters.

Fig.2a shows the running-time of the three algorithms on small datasets (up to 1.4 millions points). The Y-axis is logarithmic and represents the running-time in seconds. The quadratic complexity of DBSCAN significantly impact the running-time of the algorithm. For large datasets with over 10 millions entries, Spark-DBSCAN was unable to terminate in a timely fashion, hence is not shown on Figs.2b&2c. In fact, in half the time, PatchWork could cluster 1 billion points while Spark-DBSCAN could only cluster 1.2 million points, a 1000-fold improvement.

For very small datasets (up to one million entries), MLLib k-means is the fastest algorithm, followed by PatchWork. For datasets above one million entries, the trend reverses and PatchWork becomes faster than k-means.

Both PatchWork and k-means have a linear computation complexity. However, for very large datasets (Fig.2b), PatchWork is very significantly faster than the native Spark implementation of k-means: when clustering 1 billion points, PatchWork is 40 times faster.

Fig.2c shows that, both MLLib k-means and PatchWork have a near-linear horizontal scalability, a critical property of distributed algorithms for big-data applications.

4.3 Smart-city datasets

A key application of large scale analysis of geospatial big-data is the modeling of public infrastructures for the development of smarter cities. Smart cities such as Barcelona, San Francisco, or Montreal heavily rely on digital technologies to reduce resource consumption and to engage more effectively with their citizens.

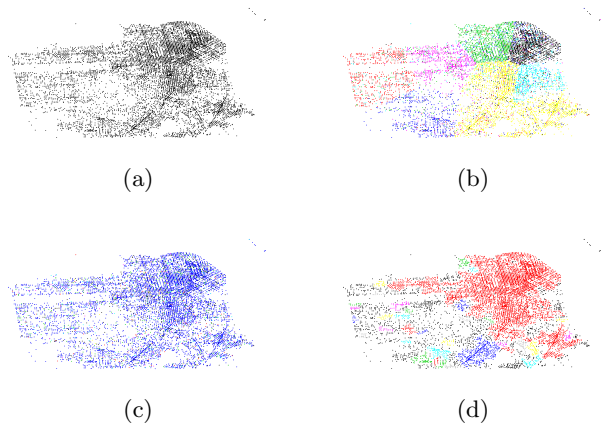


Figure 3: SFPD Incidents dataset. (a) Raw data. Clustering results of (b) MLLib k-means‖, (c) Spark-DBSCAN and (d) PatchWork.

In an effort to foster innovative services for smart cities, an increasing number of cities are making their data freely available. For example, the Open Data initiative of New-York City now comprises 1300 datasets, while San Francisco released 836 datasets.

We tested PatchWork using the *SFPD Incidents* dataset made available as part of the San Francisco Open Data initiative. It collects incidents derived from San Francisco Police Department (SFPD) Crime Incident Reporting system. The dataset is updated daily and comprises over 1.7 millions incidents recorded from 1/1/2003 and 12 dimensions. To test PatchWork and the reference algorithms, we considered 3 dimensions: *latitude*, *longitude*, and the *category* of the crime. The category of the crime is a non-numeric value. We thus preprocessed the data to convert those values into a numeric equivalent representation.

Fig.3 shows the results of the three clustering algorithms. MLLib k-means roughly divides the city into $k = 7$ clusters of equal size, while Spark-DBSCAN creates a very large cluster that encompasses nearly the entire city with numerous noisy data points. PatchWork creates a large cluster, plus several mid-sized clusters to consider for further analysis. The benefits of our approach against DBSCAN are two folds: first the ability to define ϵ for each dimension is key to better model the input dataset. The spatial-size awareness of PatchWork is also beneficial to remove small noisy clusters.

5. AVAILABILITY

PatchWork is implemented in the Scala functional programming language. It is released as a Spark Package[7] and is freely available at http://github.com/crim-ca/patchwork under the MIT license.

[7]http://spark-packages.org

6. CONCLUSION AND FUTURE WORK

An increasing number of applications are now generating very large volumes of data. For example, the number of sensors that collect geocoded data is increasing exponentially, from 500 million devices in 2003 to an anticipated 50 billion sensors in the next 5 years, resulting in volumes of data far exceeding the computing power of a single machine. However, many clustering algorithms which were developed in the past two decades rely on an iterative approach, which is inherently difficult to parallelize and distribute efficiently. While, a few parallelized variants of popular algorithms, such as k-means, have been proposed and implemented using the MapReduce paradigm, the variety of clustering algorithms that can be efficiently distributed is limited. PatchWork was designed to address those challenges. PatchWork has strong theoretical foundations and presents several desirable characteristics in knowledge discovery, in particular, it can discover clusters of arbitrary shapes, does not require *a priori* the number of clusters to identify, and offers a natural protection against outliers and noise. PatchWork was implemented using Apache Spark, and has linear computational complexity and near-linear horizontal scalability. As a result, PatchWork is suitable to process and cluster a billion data points in a distributed manner in a few minutes only. Last, our approach adds a new dimension to density-based clustering algorithms, and makes it possible to discover *spatially* large clusters instead of dense clusters only.

Further improvements to the algorithm include a better handling of data stream clustering in real-time, as well as performance optimizations for high-dimensional data.

7. REFERENCES

[1] B. Bahmani, B. Moseley, A. Vattani, R. Kumar, and S. Vassilvitskii. Scalable K-Means ++. *Proceedings of the VLDB Endowment (PVLDB)*, 5:622–633, 2012.

[2] H. Chang and D. Y. Yeung. Robust path-based spectral clustering. *Pattern Recognition*, 41(1):191–203, 2008.

[3] H. Dai and H. Su. Approximation and analytical studies of inter-clustering performances of space-filling curves. In C. Banderier and C. Krattenthaler, editors, *Discrete Random Walks, DRW'03, Paris, France, September 1-5, 2003*, volume AC of *Discrete Mathematics and Theoretical Computer Science Proceedings*, pages 53–68. DMTCS, 2003.

[4] J. Dean and S. Ghemawat. MapReduce: Simplified data processing on large clusters. In *Proceedings of the 6th Conference on Symposium on Opearting Systems Design & Implementation - Volume 6*, OSDI'04, pages 10–10, Berkeley, CA, USA, 2004. USENIX Association.

[5] A. Fahad, N. Alshatri, Z. Tari, A. Alamri, I. Khalil, A. Zomaya, S. Foufou, and A. Bouras. A survey of clustering algorithms for big data: Taxonomy and empirical analysis. *Emerging Topics in Computing, IEEE Transactions on*, 2(3):267–279, Sept 2014.

[6] S. Ghemawat, H. Gobioff, and S.-T. Leung. The google file system. In *Proceedings of the 19th ACM Symposium on Operating Systems Principles*, SOSP '03, pages 29–43, New York, NY, USA, 2003. ACM.

[7] A. Gionis, H. Mannila, and P. Tsaparas. Clustering aggregation. In *Proceedings - International Conference on Data Engineering*, pages 341–352, 2005.

[8] Google-Trends. Popularity of Spark and Hadoop, 2014.

[9] I. Gronau and S. Moran. Optimal implementations of UPGMA and other common clustering algorithms. *Information Processing Letters*, 104(6):205–210, 2007.

[10] Y. He, H. Tan, W. Luo, S. Feng, and J. Fan. MR-DBSCAN: a scalable MapReduce-based DBSCAN algorithm for heavily skewed data. *Frontiers of Computer Science*, 8(1):83–99, 2014.

[11] A. Hinneburg and H.-H. Gabriel. DENCLUE 2.0: Fast clustering based on kernel density estimation. In *Proceedings of the 7th International Conference on Intelligent Data Analysis*, IDA'07, pages 70–80, Berlin, Heidelberg, 2007. Springer-Verlag.

[12] A. Hinneburg and D. A. Keim. An efficient approach to clustering in large multimedia databases with noise. In R. Agrawal, P. E. Stolorz, and G. Piatetsky-Shapiro, editors, *Proceedings of the Fourth International Conference on Knowledge Discovery and Data Mining (KDD-98), New York City, New York, USA, August 27-31, 1998*, pages 58–65. AAAI Press, 1998.

[13] X. Hong-bo, H. Zhong-xiao, and H. Qi-Long. A clustering algorithm based on grid partition of space-filling curve. In *Internet Computing for Science and Engineering (ICICSE), 2009 Fourth International Conference on*, pages 260–265, Dec 2009.

[14] E. Hruschka, R. Campello, A. Freitas, and A. de Carvalho. A survey of evolutionary algorithms for clustering. *Systems, Man, and Cybernetics, Part C: Applications and Reviews, IEEE Transactions on*, 39(2):133–155, March 2009.

[15] R. E. J. C. Bezdek and W. Fullh. Fcm: The fuzzy c-means clustering algorithm. *Comput. Geosci.*, 10(2-3):191–203, 1984.

[16] J. Jestes, K. Yi, and F. Li. Building wavelet histograms on large data in MapReduce. *Proc. VLDB Endow.*, 5(2):109–120, Oct. 2011.

[17] S. Kisilevich, F. Mansmann, and D. Keim. P-DBSCAN: A density based clustering algorithm for exploration and analysis of attractive areas using collections of geo-tagged photos. In *Proceedings of the 1st International Conference and Exhibition on Computing for Geospatial Research & Application*, COM.Geo '10, pages 1–4. ACM, 2010.

[18] A. Litouka. DBSCAN clustering algorithm on top of Apache Spark, 2015.

[19] S. Lloyd. Least squares quantization in PCM. *Information Theory, IEEE Transactions on*, 28(2):129–137, Mar 1982.

[20] J. MacQueen. Some methods for classification and analysis of multivariate observations. In *Proc. 5th Berkeley Symp. Math. Stat. Probab.*, 1967.

[21] X. X. Martin Ester, Hans-peter Kriegel, Jörg S. A density-based algorithm for discovering clusters in large spatial databases with noise. In *Data Mining and Knowledge Discovery*, pages 226–231, 1996.

[22] M. Noticewala and D. Vaghela. MR-IDBSCAN:

Efficient parallel incremental DBSCAN algorithm using mapreduce. *International Journal of Computer Applications*, 93(4):13–18, May 2014.

[23] M. A. Patwary, D. Palsetia, A. Agrawal, W.-k. Liao, F. Manne, and A. Choudhary. A new scalable parallel DBSCAN algorithm using the disjoint-set data structure. In *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*, SC '12, pages 62:1–62:11, Los Alamitos, CA, USA, 2012. IEEE Computer Society Press.

[24] G. Sheikholeslami, S. Chatterjee, and A. Zhang. Wavecluster: A multi-resolution clustering approach for very large spatial databases. pages 428–439, 1998.

[25] R. R. Sokal and C. D. Michener. A statistical method for evaluating systematic relationships. *University of Kansas Scientific Bulletin*, 28:1409–1438, 1958.

[26] W. Wang, J. Yang, and R. R. Muntz. STING: A statistical information grid approach to spatial data mining. In *Proceedings of the 23rd International Conference on Very Large Data Bases*, VLDB '97, pages 186–195, San Francisco, CA, USA, 1997. Morgan Kaufmann Publishers Inc.

[27] J. Wood, O. O'Brien, A. Slingsby, and J. Dykes. Visualizing the dynamics of London's bicycle-hire scheme. *Cartographica: The International Journal for Geographic Information and Geovisualization*, 46(4):239–251, 2011.

[28] C. Xiaoyun, C. Yi, Q. Xiaoli, Y. Min, and H. Yanshan. PGMCLU: A novel parallel grid-based clustering algorithm for multi-density datasets. In *Web Society, 2009. SWS '09. 1st IEEE Symposium on*, pages 166–171, Aug 2009.

[29] Y. Yu, J. Zhao, X. Wang, Q. Wang, and Y. Zhang. Cludoop: An efficient distributed density-based clustering for big data using hadoop. *International Journal of Distributed Sensor Networks*, Feb. 2015.

[30] M. Zaharia, M. Chowdhury, T. Das, A. Dave, J. Ma, M. McCauley, M. J. Franklin, S. Shenker, and I. Stoica. Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing. In *Proceedings of the 9th USENIX Conference on Networked Systems Design and Implementation*, NSDI'12, pages 2–2, Berkeley, CA, USA, 2012. USENIX Association.

[31] M. Zaharia, M. Chowdhury, M. J. Franklin, S. Shenker, and I. Stoica. Spark: Cluster computing with working sets. In *Proceedings of the 2Nd USENIX Conference on Hot Topics in Cloud Computing*, HotCloud'10, pages 10–10, Berkeley, CA, USA, 2010. USENIX Association.

[32] C. Zahn. Graph-Theoretical Methods for Detecting and Describing Gestalt Clusters. *IEEE Transactions on Computers*, C-20(1), 1971.

[33] M. Zaltz Austwick, O. O'Brien, E. Strano, and M. Viana. The structure of spatial networks and communities in bicycle sharing systems. *PLoS ONE*, 8(9):e74685, 09 2013.

[34] H. Zhang, Y. Zhou, J. Li, X. Wang, and B. Yan. Analyze the wild birds' migration tracks by mpi-based parallel clustering algorithm. In *Proceedings of the 6th International Conference on Advanced Data Mining and Applications: Part I*, ADMA'10, pages 383–393, Berlin, Heidelberg, 2010. Springer-Verlag.