KeyBin2: Distributed Clustering for Scalable and In-Situ Analysis

Xinyu Chen
University of New Mexico
xychen@cs.unm.edu

Jeremy Benson
University of New Mexico
jeremybenson@cs.unm.edu

Matt Peterson
University of New Mexico
mpeterson@unm.edu

Michela Taufer
University of Tennessee
taufer@utk.edu

Trilce Estrada
University of New Mexico
estrada@cs.unm.edu

ABSTRACT

We present KeyBin2, a key-based clustering method that is able to learn from distributed data in parallel. KeyBin2 uses random projections and discrete optimizations to efficiently clustering very high dimensional data. Because it is based on keys computed independently per dimension and per data point, KeyBin2 scales linearly. We perform accuracy and scalability tests to evaluate our algorithm's performance using synthetic and real datasets. The experiments show that KeyBin2 outperforms other parallel clustering methods for problems with increased complexity. Finally, we present an application of KeyBin2 for in-situ clustering of protein folding trajectories.

CCS CONCEPTS

• **Theory of computation → Random projections and metric embeddings**; **Unsupervised learning and clustering**; *MapReduce algorithms*; Numeric approximation algorithms;

KEYWORDS

Clustering, Random Projection, Map-Reduce, Privacy Preserving

ACM Reference Format:
Xinyu Chen, Jeremy Benson, Matt Peterson, Michela Taufer, and Trilce Estrada. 2018. KeyBin2: Distributed Clustering for Scalable and In-Situ Analysis. In *ICPP 2018: 47th International Conference on Parallel Processing, August 13–16, 2018, Eugene, OR, USA.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3225058.3225149

1 INTRODUCTION

State of the art, high-throughput scientific simulations enable better understanding of natural phenomena. And while many methods are able to divide a large problem into small tasks and take full advantage of distributed resources to perform parallel processing [28, 31, 37, 46], they lack when it comes to performing global analyses, as they usually rely on expensive data movement and centralized processing. In domains where data volume is continuously growing (e.g., climate simulations - 32 PB [44], astronomy - 200 GB/day [55] to 30 TB/day [42], and high-energy physics - 500 EB/day [12]), data movement represents a performance bottleneck that increases resource pressure to storage, bandwidth, memory, and CPU [57]. Coupling simulations with data analysis yields promising performance gains [6, 19, 57, 61]. Some algorithmic approaches have been successfully used to decentralize data analysis in specific domains and provide in-situ analytics (i.e, executing analysis tasks alongside simulations or data acquisition tasks) [5, 21, 23, 30, 35, 40, 52]. However, deficiencies, in terms of scalability, accuracy, or generality, are still present. In order to scale analysis at the same rate as simulations, there is a need for improved techniques for efficient dimensionality reduction, clustering, pattern recognition, and anomaly detection, all considering and constraining data movement.

In this work, we deal specifically with clustering, or the process of identifying groups of data points that are related in a particular space. Clustering techniques often employ a distance metric to evaluate whether two points are *close* in space or not. When dealing with very large data volumes or data that is distributed, distance calculations are a major bottleneck for scalability in clustering algorithms. Our goal is to design an algorithm that extracts knowledge in a way that is embarrassingly parallel (i.e., there are no data dependencies and parallelism is effortless), data agnostic (i.e., data specifics are irrelevant), and can deal with batch processing and streams.

In previous work, we presented KeyBin [13] which is a clustering method based on distributed key calculations and binning to produce clusters from distributed data. The underlying idea in KeyBin is to assign each point a hierarchical key in space. This key can be computed independently based only on the point's features and does not need additional knowledge of other data points. Keys are assigned to bins per every dimension (i.e., feature) in the dataset. These computations are parallel and can be performed efficiently in distributed environments. To perform a final clustering assignment, it is enough to communicate just bin densities per every dimension (i.e., an histogram). From this summarized information, uninformative or noisy dimensions can be collapsed and a final clustering assignment is done by partitioning and merging the binning histograms. An important aspect of KeyBin is that all the computations on the original data can be done on site, in parallel, and without computing pairwise distances. The only information that is communicated across nodes, processes, or distributed sites are the binning

ICPP 2018, August 13–16, 2018, Eugene, OR, USA

ACM ISBN 978-1-4503-6510-9/18/08…$15.00
https://doi.org/10.1145/3225058.3225149

histograms. Bins are orders of magnitude smaller than the original data and cannot be used to trace back or reconstruct the original information. As such, this approach is ideal for distributed and privacy sensitive scenarios. In this work, we present an updated version of our KeyBin algorithm; in particular, we address three major limitations of our initial design:

Orthogonality assumption. Although we do not perform pairwise distance computations, distance between points affects how they are assigned into bins. Imagine the distance between points p and q is $d_x(p,q) = |x_2 - x_1|$ on $dimension_x$. If we include an orthogonal $dimension_y$, the actual distance in the 2D space is $d(p,q) = \sqrt[2]{(x_2-x_1)^2 + (y_2-y_1)^2} \geq |x_2 - x_1|$. Then we can use the bin number of x_1, x_2 and y_1, y_2 to infer the actual positions of p and q in 2D space. However, if instead there exists a nonorthogonal $dimension'_y$, the actual distance $d(p,q)$ in the 2D space might be shortened for some cases. That is to say, we cannot rely on the bin number of x_1, x_2 and $y_1^{'}, y_2^{'}$ to infer the actual position of p and q in the 2D space.

Projection overlapping. The binning histograms on each dimension can be seen as a partial view of the data projected into a particular dimension. If the projections of two clusters overlap, KeyBin cannot distinguish them. If p_i is the probability of two clusters overlapping on the ith dimension, then the probability that our algorithm cannot separate the two clusters is the product of such overlapping probabilities on all dimensions $\prod_{i=1}^{N} p_i$.

Partitioning heuristics. The only centralized operation in KeyBin consists on partitioning the binning histograms in order to perform a final clustering assignment. This operation used a simple heuristic based on bin densities to determine partitioning locations. As the density landscape is not known in advance and every dataset has its own inherent behavior, partitioning through heuristics is not deemed to be robust and presents a serious limitation to accurately identify clusterings.

In this paper we address these limitations by introducing an efficient process for dimensionality collapsing based on random projections, and a robust partitioning mechanism based on discrete optimization. Our specific contributions are:

(1) We introduce *KeyBin2*, a significant extension of the original KeyBin. The enhanced algorithm, takes advantage of random projections and discrete optimization to provide robustness and improve its ability to separate difficult datasets.
(2) We perform accuracy and scalability tests to evaluate our algorithm's performance using synthetic and real datasets.
(3) We present a case study for in-situ clustering of protein folding trajectories using KeyBin2.
(4) We provide an implementation of *KeyBin2* that runs on multicore GPU clusters, using both, a cuda-python compiler and mpi4py.

The remainder of this paper is organized as follows: Section 2 briefly summarizes related work. Section 3 presents the details of *KeyBin2*. Section 4 presents the evaluation of KeyBin2 in terms of scalability and accuracy. Section 5 presents a case study for in-situ analysis of protein folding trajectories. Finally Section 6 concludes the paper.

2 RELATED WORK

A variety of clustering methods have been proposed to address dimensionality and scalability concerns [32, 59] in three main categories: partitional clustering approaches represented by the K-Means[41] family, density based clustering approaches like DBSCAN [20] and DENCLUE[27], and hierarchical clustering like CLIQUE [3] and MAFIA [24]. In the following paragraphs, we elaborate on these categories and discuss the relationship to our body of work.

Partitional clustering like K-means[41], while a tried and tested clustering technique, has a few key drawbacks: the number of clusters, K, must be provided. X-means [50] proposes handling this drawback with Bayesian Information Criterion (BIC) in order to automatically select the optimal K values. A second limitation of K-means is that it is unable to find non-convex clusters [59]. Modern K-means with multi-threaded[54] or multi-core parallelization [38] are able to deal with large amounts of data, but they still inherit the above limitations. From our experiments, K-means performs well in finding sphere-shape clusters but has a tendency to mislabel points on the corners of box-shaped clusters. This is because points on the diagonal of a box may be closer to other cluster's center than to their real cluster center. In contrast, KeyBin2 determines automatically the number of clusters, is able to deal well with convex clusters, and can handle points in box corners, as well.

Density based clustering. Density based clustering algorithms have the advantage of finding non-convex clusters and that outliers and noise have less impact on the overall clustering accuracy. Their basic idea is to consider regions with high density to be clusters. DBSCAN[20] uses counting techniques to find dense regions through the number of points within the radius of a neighborhood that exceeds a given threshold. DBSCAN maintains high accuracy in clustering data when given the appropriate radius and threshold parameters. PDSDBSCAN[47] and its successor, BD-CATS[48], are the state of art density based clustering algorithms, capable of clustering up to a trillion points with up to 3-dimensions. Their limitation is that they scale with the number of data points but they are not able to handle very large dimensional data (i.e., inputs with a very large number of features). DENCLUE[27] uses influence functions to estimate probability densities of regions. The influence functions yield results comparable to Gaussian kernel density estimations (KDE)[56]. The clusters are centered at influence attractors, which are the local maxima of influence values. DBSCAN-MR[15] apply Map-Reduce methods to build a distributed index and optimize load balance. On top of this Map-Reduced algorithm, GA-DBSCANMR[29] uses genetic algorithms to improve accuracy by iteratively adjusting parameters such as minimum number of points per cluster and maximum distance. HPDBSCAN[25] combines OpenMP and MPI to parallelize DBSCAN and improve the computation time. DOC[49, 52] finds and merges areas with density higher than the expected uniform distribution density to detect clusters and outliers. When applied to system's log analysis, the DOC algorithm also implicitly uses ensemble results from multiple bins and multiple features to improve accuracy.

Hierarchical clustering. Subspace Clustering algorithms belong to grid-based, hierarchical clustering algorithms and use bottom-up search strategies, assuming that higher dimensional clusters

are composed of lower dimensional subspace clusters. They also assume some dimensions are not useful or are noise. Finding clusters in subspaces can help exclude noisy dimensions. CLIQUE [3] and MAFIA [24] find dense regions in lower-dimensional spaces, and merge these lower-dimensional regions into bigger higher-dimensional hyper-cubes if no common-face exists between two regions. A comprehensive parallel framework [26] uses MPI to manage big amounts of data points. Due to the expensive combinations of all possible lower dimensional regions, the time complexity of grid-based subspace clustering algorithms is $O(c^{k'})$ [24]. The limitation of these algorithms is in their inability to scale when the dimensionality is high. In our work, we are able to scale linearly with the number of dimensions because we build higher dimensional clusters upon 1-dimensional groupings. With this approach, we do not need to check for combinations of clusters in intermediate subspaces.

Fern et al. [22] use random projections combined with an ensemble approach for clustering. Under each individual projection, the clustering algorithm learns part of the underlying patterns and the final clustering labels are aggregated from previous results based on the change of similarities. Our approach is different because we do not perform pairwise comparison of data points and we use the internal clustering dispersion to evaluate a projection rather than an ensemble.

3 KEYBIN2

KeyBin2, our extension to KeyBin [13], is a linear-time algorithm to perform distributed clustering. Assume a dataset D^0 of size $M \times N$, where M is the number of data points and N is the number of dimensions (i.e., features). This algorithm extrapolates for data streams with $M = 1$ and for distributed datasets with multiple D's. KeyBin2's clustering process is as follows:

(1) *Projecting into a lower space.* To address some of KeyBin's limitations, we perform a random linear transformation to each data point $x \in \mathbb{R}^N \rightarrow x' \in \mathbb{R}^{N_{rp}}$. This transformation has two advantages: it rotates the data, reducing the likelihood of projection overlapping in all of the dimensions, and it produces a much smaller dimensional space, making processing more efficient. For a projected point x' is ith feature is $x'_i = |x| \cos\theta_i$, where θ_i is the angle between x and the random vector used for projection. As long as $\cos\theta_i \neq 0$, the relative ordering of points in dimension i does not change and the binning process is not negatively impacted.

(2) *Assigning a key to a point.* Given the ith point x'_i, we denote it's jth feature, for $i \leq M, j \leq N_{rp}$ as $x'_{i,j}$. For a predefined maximum depth d_{max}, and for a predetermined space range $[r_{min}, r_{max}]$; $x_{i,j}$ is assigned a key. This key is calculated by concatenating the bin label b_d corresponding to $r_{min} + b\left(\frac{r_{max}-r_{min}}{2^{d'}}\right)$, if $x'_{i,j} \in b_d$ for $d = 1$ to d_{max}. Each point is identified by a key and assigned to a hierarchy of bins. As more points are seen by the system, either as a batch or as a stream, bins update their density in the form of a local histogram.

(3) *Communicating binning histograms.* For every dimension in the projected sub-space, KeyBin2 computes at most d_{max} binning histograms. The histograms are communicated back to a central location where the clustering assignment is made. In practice we have observed that for convex problems, 2 to 4 histograms per dimension suffice to accurately clustering the data. As noted in [13], this communication step does not necessarily have to be made to a central authority. The algorithm works as well for a ring topology.

(4) *Partitioning binning histograms.* In order to consolidate an integrated view of the data, binning histograms are partitioned by their modes. This process uses discrete optimization identify inflexion points in the underlying data distribution and to maximize the inter-clustering distance while minimizing intra-clustering distance. However, still no pairwise distance is computed and all of the operations are performed only through the histograms. The partitions are broadcast back to the data collection sites, processes, or nodes.

(5) *Performing clustering assignments.* Once binning histograms are partitioned, primary clusters are locally built. Primary clusters are sets of bins organized in a partial clustering assignment for a single dimension. Primary clusters are analogous to a space map where keys can be directly assigned to form global clusters. Every point can be mapped to a specific set of primary clusters in all of its dimensions, leading to a final global clustering assignment.

(6) *Assessing projected sub-spaces.* Before finally accepting a clustering assignment and because the randomly projected data may or may not exhibit a separable behavior, we perform bootstrapping. During the analysis phase we calculate the within- and between-cluster dispersion. Our goal is to select the projection that produces the most compact and separable clusters. This evaluation is performed only on the binning histograms rather than the input points and therefore it scales independently of the input size.

Steps 2, 3, and 5 were first introduced in KeyBin version 1. Thus, for additional details and discussions, please refer to its publication in [13]. Steps 1, 4, and 6 are specific improvements for KeyBin2 and will be explained in further detail in the remainder of this section.

3.1 Projecting into a Lower Space

To address KeyBin's limitations in orthogonality assumption and the projection overlapping, we use linear *Random Projections* [1, 9, 17, 18] to transform the original data into multiple, lower dimensional spaces. A random projection is a mapping $f : D^N \Rightarrow D^{N_{rp}}$ from an original N-dimensional space into N_{rp} dimensions. In high dimensional spaces, there are a large number of orthogonal vectors[9]. If column vectors in A are close to orthogonal, the transformation reduces the dimensionality and rotates the data points, all while preserving their lengths and angles. To illustrate the rotation effect, we show two clusters in a 2D space in figure1. The original data points (a) are correlated and exhibit a projection overlapping in both of their dimensions. In these conditions, our previous method would not have been able to separate the two clusters. The other 5 sub-figures (b, c, d, e, f) depict 5 different random projections of the original data. Projections b and c are able to decorrelate the data, but d and f exacerbate the problem.

The most widely used way to project data into a lower dimensional orthogonal subspace, capturing as much variation of the

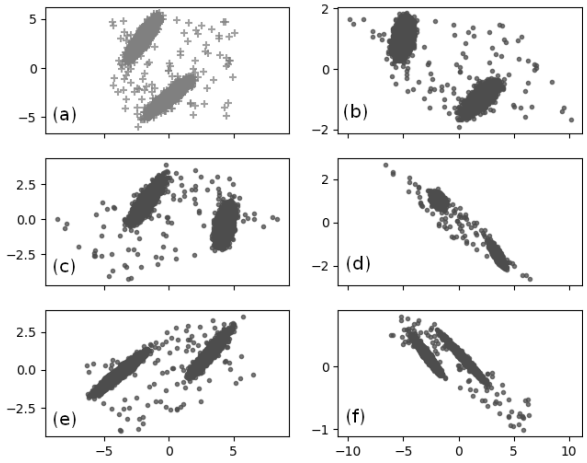

Figure 1: (a) original 2D data points. (b)-(f) projected points in space.

data as possible for the chosen N_{rp} is principal component analysis (PCA) [34]. However, PCA is expensive, even when using a singular value decomposition. Its time complexity is $O(NcM)$, where N and M are the number of dimensions and number of points respectively, and c is the average number of non-zero values per column. For the projection method to work at scale, it needs to be independent of the number of points in the dataset. By randomly constructing projection matrices, the time complexity only depends on N, N_{rp} and a constant C. To construct our projection matrix first we need to determine the target dimension size N_{rp}. Unlike the bound of lower dimensionality proposed by Daspupa.el [18], which states that the lower bound of reduced dimensionality is $4(\epsilon^2/3 - \epsilon^3)^{-1} ln(N)$ to preserve internal distances within an arbitrary error ϵ; we argue that our algorithm is able to handle even lower dimensions because it does not rely on pairwise distances preserved to some bounded range.

The process is as follows: to change the basis of the original feature space into a lower dimension, we use a transformation matrix that columns vectors are unit vectors. Consider $x = (x_1, x_2, .., x_N)$, a data point in the original feature space $\mathbb{R}^N$, and the transformation matrix:

$$A_{(N \times N_{rp})} = \begin{bmatrix} a_1 & a_2 & a_3 & \dots & a_{N_{rp}} \end{bmatrix}$$

where $a_1, ..a_{N_{rp}}$ are unit column vectors. Then the data point x is projected to $x' = (x'_1, x'_2, .., x'_{N_{rp}})$ with $N_{rp} << N$. Let θ be the angle between x and a_1, then the new coordinate $x'_1 = |x| \cos\theta$. The binning histogram we build on this new dimension$_1$ reflects how data points are ordered along the direction of vector a_1.

The lower bound of dimensionality proposed by Dasgupta [18] aims to preserve the pairwise distance of data points within an arbitrary error ϵ. This distance-preserving condition is very important to algorithms such as K nearest neighbors. As KeyBin does not depend on actual pairwise distance computations, the only necessary condition of the projection is that the ordering along some column vector a_i provides a decent spread of the data points. The optimal reduction is challenging to estimate because we do not know the *intrinsic* dimension of the original feature space. Recall that one of the constraints that we established for our algorithm is that it should scale to very large datasets and also work for streams. Thus, computing for example the covariance matrix is strictly off limits.

Then, assuming that the dataset D has an *intrinsic* dimension R, with $R < N$, where $N - R$ are redundant, noisy or uninformative dimensions. Without replacement, we want to select $z = N_{rp}$ useful projected dimensions. The probability $P(Z = z)$ follows the hypergeometric distribution. The lower bound of the target dimensionality reduction should satisfy that at least one separable dimension is selected. The expectation of this distribution is $E(Z) = N_{rp} \cdot \frac{R}{N}$. This means we want $N_{rp} \geq \frac{N}{R}$.

$$P(z) = \begin{cases} \frac{\binom{R}{N_{rp}}}{\binom{N}{N_{rp}}} & \text{when } N_{rp} < R \\ \\ \frac{\binom{N-R}{N_{rp}-R}}{\binom{N}{N_{rp}}} & \text{when } N_{rp} \geq R \end{cases} \quad (1)$$

Although we do not know the exact values for R and N_{rp}, in the first case we want the probability of $P(z = N_{rp})$ to be high, and in the second case, we want the probability of $P(z = N_{rp} - R)$ to be low. In both cases, we want N_{rp} to be small. Thus, we choose a logarithmic growth for the determination of $N_{rp} = 1.5log(N)$ as a reasonable small number, which empirically achieves good results.

After calculating a projection matrix for the specific dataset, every data point is projected into the lower dimensional space. In this new space, a key is computed (as explained in Step 2) and used to update the binning histogram for each new dimension. These steps are fully parallel, per data point and per dimension, as long as each point has access to the projection matrix. If data is being processed in batches, then histograms are communicated after a batch is over. If data is processed as a stream, histograms are communicated periodically (i.e., after a number of updates, or after a specific period of time). Once histograms are collected, statistically anomalous dimensions are identified with the Kolmogorov-Smirnov test [39] and collapsed. The following step consists of identifying a partition in the histogram space and broadcasting out to the nodes, processes, or distributed sites.

3.2 Partitioning Binning Histograms

A key aspect of our binning approach is that the size of the bins directly affects the accuracy of our method. Bins that are too large can easily confound a multimodal distribution. Bins that are too small can result in an artificially inflated number of clusters. Because of this, we produce multiple histograms with different bin sizes. Still, a key question for our algorithm is: *based on the binning density, how do we partition a specific dimension?*. For the original KeyBin, the previous heuristic to find partitioning locations was based on a density threshold, which is not robust under streaming scenarios, or when the size of the data and expected densities are difficult to estimate. In turn, a necessary extension was needed in the partitioning mechanism.

KeyBin's histograms represent discrete approximations to the underlying probability distributions observed across each dimension of the distributed datasets. To optimize the dimension partitioning,

it is important to find a smoothed representation of said probability distributions. The first step is to merge all of the histograms collected from the distributed sites. Approximating the probability distribution is not trivial; the number of modes in the distribution and their shape is not known in advance, and it is expected that noise and confounding factors will be present. We perform a non-parametric discrete optimization to determine points in the 1-dimensional space of histograms where the density of the different distributions is minimized. To this end, we smooth possible noise in the histogram by applying a moving average model that uses a window size equal to the square root of the number of bins in the histogram ($w = \sqrt[2]{log2^2(M)}$). For each window and its neighbors, we apply local regression. The first derivative determines the slope of the tangent line of the linear function at the specific bin. The second derivative identifies the inflection points in the function, indicating regions of sudden change. Once a partition is decided per every dimension, this information is broadcast to the processing nodes, which use it along with the points' keys to perform a global clustering assignment, as was done in the previous version [13].

The kernel density estimation (KDE) [56] is an alternative method that can produce an approximation of the true probability density function. The estimated distribution curves are continuous and differentiable. DENCLUE [27] provides a method to do this, but needs to compute pairwise distances between all data points to find the influence of each data point. Our simpler method reaches similar accuracy compared to KDE curves, but our smoothing technique is much faster than the kernel density estimation.

3.3 Assessing Projected Subspaces

As illustrated in Section 3.1, some projections are better than others and there is always the possibility of encountering a pathological transformation. To rate clustering models, we use a modified version of the Calinksi-Harabaz [10] index, which is the ratio of *between-cluster dispersion* and *within-cluster dispersion*. The traditional computation of this index involves computing pairwise distances between cluster centroids and the points assigned to them. For scalability reasons, we avoid pairwise distance computation involving the data points and instead rely solely on the histogram and keys' space. Recall that a point is identified by a key. This key corresponds to the set of bins to which the point belongs to in all of its dimensions (e.g., if point x belongs to bin 35 in dimension 1, 64 in dimension 2 and 06 in dimension 3 its key is simply "356406"). Global clusters are formed by a range of bins in each dimension. This set of bins is found by the partitioning mechanism discussed in Section 3.2. For a set Q of global clusters and a specific cluster $C_q \in Q$, the value of Calinski-Harabaz index can be calculated using only the the keys and density of each bin $b \in Bins$ with the following equations [10]:

$$cal = \left[\frac{B_Q}{W_Q}\right] \times \left[\frac{|Bins| - |Q|}{|Q| - 1}\right] \times log_2(|Q| - 1) \tag{2a}$$

$$W_Q = \sum_{q \in Q} \sum_{j} \sum_{b \in C_q} (b[j] - c_q[j])^2 \times Density_b[j] \tag{2b}$$

$$B_Q = \sum_{q \in Q} \sum_{j} (c_q[j] - c[j])^2 \times \sum_{b \in C_q} Density_b[j] \tag{2c}$$

Where j denotes a specific dimension, $b[j]$ and $Density_b[j]$ are the key of bin b and its density in dimension j respectively. c_q refers to the center of a local cluster C_q and c refers to the global center in the dataset, calculated by finding bins corresponding to the *50th percentile* in each dimension.

Figure 2 is an intuitive view of what this process entails. The figure depicts a 2-dimensional space consisting of 6 clusters. Histograms for each dimension are shown top and right of the figure. The grid inside of the space represents the specific partitions found by KeyBin2. For a specific cluster C_q, we identify the range of bins in each dimension belonging to Cq. Within this range, we identify the cluster's centroid given by the histograms modes. The within-cluster dispersion W_q is computed by summing the square difference between each bin and the cluster's centroid, multiplied against the density of that bin. This process is repeated for every dimension. To calculate the between-cluster dispersion the operations are similar; in this case, the difference is calculated between each cluster's centroid cq and the dataset center c. The center c is found by calculating the $50th$ percentile of the histogram in each dimension. Again, it is important to note that this process is performed completely on the histogram's space and is therefore scalable; calculations do not involve comparisons between data points.

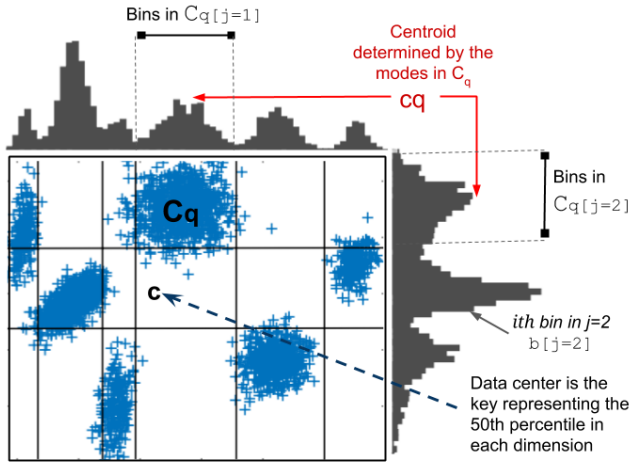

Figure 2: Assessing projection subspaces in a 2-dimensional example

3.4 Complexity Analysis

The entire *KeyBin2* algorithm breaks down into three major steps: (1) projecting the data, (2) computing the keys and updating bins, and (3) collapsing dimensions and computing partitions. The overall time complexity is the summed complexity of each step multiplied by the number of bootstrapping. In [14], we determined that the time complexity to assigning keys to M points with N dimensions, when we use B bins is $O(MNlogB)$. As we reduce the dimensionality from N to $N_{rp} = 1.5logN$, the number of bins turns $B = logM$ and the complexity of building keys is $O(MlogNloglogM)$ and building histograms is $O(Mlog^2N)$. Before reducing dimensionality, we need

to project the data using a random matrix. The time complexity for this step is $O(MNN_{rp}) = O(MNlogN)$. M in those operations can be combined, as every point needs to be read once, then multiplied by the random matrix to reduce its dimensionality, and assigned a key. After that, the point can be either discarded or sent to secondary storage awaiting its final clustering assignment.

Partitioning histograms, which entails smoothing, differentiating, and finding cuttings, is $O(N_{rp}Bw) = O(logNlog^2M)$. Bootstrapping iterates over the previous two steps, in addition to evaluating the cluster assignment up to a number of t trials. The evaluation step takes two passes over the histogram - one to find the centroids, and one to calculate the dispersions, with a time complexity of $O(B)$ $O(log^2N)$. Finally, assigning clusters for the whole dataset is $O(MN_{rp})$; one pass to concatenate keys generated in previous steps and another pass to aggregate points with the same key to form clusters. The time complexity for labeling is $O(MN_{rp}) = O(MlogN)$. In summary, the time complexity for KeyBin2 is $t \times [O(MlogNloglogN) + O(logNlog^2M) + O(log^2N)] + O(MlogN)$. Further optimization would perform t simultaneous random projections to M points, taking out M from the t bootstrapping steps. The communication cost between K locations is $O(2KN_{rp}B)$. With the reduced dimensionality N_{rp} and the number of bins B, the required communication is as small as several Kbytes.

3.5 Implementation Details

For the implementation of KeyBin2, we use a master-worker topology and mpi4py[16] to establish communication between processes. Each process generates a fraction of the data and leverages the GPU to assign keys to projected data points and build histograms on each dimension. This step is done in parallel per data point and per dimension. We accelerate this part of the computation with Numba[36], an LLVM-based compiler that enables Python to use CUDA[45]. When histograms are completed, they are communicated to the master process, which reduces the information, computes global densities, and broadcasts the aggregated information back. With the aggregated histograms, workers utilize the GPU again to find a final clustering labels for the reduced data points. The code, datasets, and documentation for everything are available online at: *https://lobogit.unm.edu/DataSci/keybin2.git*.

4 EVALUATION

To evaluate *KeyBin2*, we compared its performance (i.e., scalability and accuracy) to other well-known and widely-used clustering algorithms: (1) K-means++, an optimized version of the popular K-means algorithm from scikit-learn 0.17.1. (2) parallel-kmeans by [38], which has been shown to achieve speedup by distributing the entire dataset into many MPI ranks. (3) PDSDBSCAN [47], an HPC implementation of DBSCAN. We also attempted a comparison with the GPU implementation of MAFIA (GPUMAFIA [11]), however GPUMAFIA was unable to converge under our particular setup.

We present two sets of experiments: first, we measure scalability as a function of the number of dimensions. For our second experiment, we measure scalability as the number of points and processes grow. Most parallel clustering approaches report scalability, but as clustering is an unsupervised learning algorithm, they almost never report accuracy. To avoid this pitfall, we were sure to use classification problems for our tests and then quantify the clustering accuracy. We report precision, recall, and f1-score. In the clustering context, precision is the ratio $tp/(tp + fp)$ where tp is the number of true positives (i.e., point pairs assigned to same clusters that actually belong to same clusters) and fp the number of false positives (i.e., point pairs assigned to same clusters that do not belong to same clusters). The precision is the ability of the clustering not to assign a point to a cluster C that does not belong to it. Recall is the ratio $tp/(tp + fn)$ where fn the number of false negatives (i.e., the number of point pairs that belong to same clusters were not identified as same clusters). Recall is the ability to find all the data points that belong to a cluster. The f-score is the harmonic mean of precision and recall.

Our experiments ran on the Xena cluster at the Center for Advanced Research Computing of the University of New Mexico. Xena is a PowerEdge R730 / Intel Xeon CPU E5-2640 at 2.6 GHZ with 32 nodes, 16 cores per node, Infiniband interconnect, and 4GB of RAM per core. Each also has a NVIDIA Tesla K40m graphic card.

In the following experiments, we test scalability. Synthetic data is generated from 4 mixed Gaussian distributions with a diagonal covariance matrix. The data is produced and stored on K MPI processes. Our first experiment test how KeyBin2 scales with regards to the dataset size. We fix the number of processes to 16 and increase the dimensionality of the data from 20 to 1280 at intervals of 4× (see table 1). In the second experiment, we fix the dimensionality of the data to 1280 dimensions and we perform a doubling experiment by increasing the number of MPI processes from 1 to 16. The amount of data doubles too, as each process handles 80,000 points (see table 2. In both experiments, we provide the true number of clusters $k = 4$ to kmeans++ and parallel-kmeans, so they always find the correct number of clusters. Similarly, we provide the optimal ϵ and *minPoint* parameters to PDSDBSCAN[47]. We report confidence intervals for 20 independent runs per each experimental design point.

For some of the runs, we were unable to get results from kmeans++ and pdsdbscan because they would crash from memory issues. Table 1 does not include results for pdsdbscan because it could not handle more than 100,000 points. Table 2 does not include results for kmeans++ because it stopped converging with as little as 100 dimensions. For both runs we can see that KeyBin2's scalability is linear and grows slower than parallel-kmeans (except when the number of dimensions is very small, e.g., 20).

In terms of accuracy, KeyBin2 finds a larger number of clusters than the ground truth number. Recall that we provided the correct number of clusters to kmeans++ and parallel-kmeans and the optimal parameters to PDSDBSCAN. This information was not made available to our KeyBin2 algorithm, as it is non-parametric. KeyBin2 found a larger number of clusters, but some of them were small outliers from noise in the data. In more detail, we observe that, as problems become larger (i.e., increasing dimensions or increasing data points), KeyBin2's recall and precision both drop. This indicates the ordering of data points becomes more ambiguous in high dimensional spaces (i.e, points are lying in a thin shell [2, 8, 17]) and the effect of the Calinski-Harabaz index to accurately select the best model is decreasing. Still, KeyBin2 outperforms the other methods as shown by the F1 scores.

Table 1: 1.28 million data points on 16 MPI processes (80,000 per proc.)

Method	Clusters	Recall	Precision	F1 score	Time (sec)
20 dimensions					
KeyBin2	7.37 ± 1.49	0.836± 0.04	0.929± 0.05	**0.877± 0.03**	**42.11± 2.21**
kmeans++	4.0± 0	0.714± 0.01	1.0± 0	0.829± 0.01	509.12 ± 5.94
parallel-kmeans	4.0± 0	0.724± 0	1.0± 0	0.840± 0	**20.01 ± 0.81**
80 dimensions					
KeyBin2	9.73 ± 2.48	0.828± 0.07	0.990± 0.01	**0.898± 0.04**	**45.31 ± 2.07**
kmeans++	3.0± 0	0.692± 0.01	1.0± 0	0.815± 0.08	638.26 ± 8.0
parallel-kmeans	4.0± 0	0.871± 0	0.613± 0	0.719± 0	58.90 ± 5.08
320 dimensions					
KeyBin2	10.07 ± 1.75	0.752± 0.05	0.955± 0.02	**0.838± 0.04**	**58.26 ± 4.50**
kmeans++	–	–	–	–	–
parallel-kmeans	4.0± 0	0.695± 0	0.627± 0	0.659± 0	200.90 ± 20.60
1280 dimensions					
KeyBin2	11.00 ± 1.41	0.774± 0.04	0.967± 0.02	**0.857± 0.03**	**285.20 ± 22.25**
kmeans++	–	–	–	–	–
parallel-kmeans	4.0± 0	0.557± 0.0	0.574± 0.0	0.565± 0.0	1086.85 ± 35.27

Table 2: 1280-dimensional data points on multiple MPI processes (80,000 per proc.)

Method	Clusters	Recall	Precision	F1 score	Time (sec)
1 process (80,000 data points)					
KeyBin2	10.40 ± 2.57	0.850± 0.09	0.991± 0.01	**0.912± 0.06**	**20.75 ± 1.66**
parallel-kmeans	4.0± 0	0.831± 0.15	0.962± 0.06	0.879± 0.07	62.34 ± 9.38
pdsdbscan	1.0± 0.0	1.0± 0.0	0.286± 0.0	0.445± 0.0	1816.20 ± 148.20
2 processes (160,000 data points)					
KeyBin2	12.00 ± 3.16	0.797± 0.05	0.978± 0.01	**0.877± 0.03**	**28.58 ± 2.79**
parallel-kmeans	4.0± 0	0.698± 0.09	1.0± 0.0	0.819± 0.05	151.77 ± 25.34
pdsdbscan	–	–	–	–	–
4 processes (320,000 data points)					
KeyBin2	12.40 ± 2.05	0.766± 0.03	0.982± 0.01	**0.860± 0.02**	**38.83 ± 5.19**
parallel-kmeans	4.0± 0	0.491± 0.09	1.0± 0.0	0.654± 0.07	199.06 ± 5.19
pdsdbscan	–	–	–	–	–
8 processes (640,000 data points)					
KeyBin2	13.50 ± 4.33	0.775± 0.09	0.981± 0.02	**0.861± 0.06**	**57.41 ± 16.01**
parallel-kmeans	4.0± 0.0	0.694± 0.05	1.0± 0.0	0.818 ± 0.03	398.08 ± 5.82
pdsdbscan	–	–	–	–	–
16 processes (1,280,000 data points)					
KeyBin2	9.50 ± 0.50	0.827± 0.01	0.977± 0.01	**0.891± 0.06**	**285.20 ± 22.25**
parallel-kmeans	4.0± 0	0.557± 0.0	0.574± 0.0	0.565± 0.0	1086.85 ± 35.27
pdsdbscan	–	–	–	–	–

5 IN-SITU ANALYSIS OF PROTEIN FOLDING

To test the applicability of KeyBin2 to a challenging in-situ analysis, we apply it to protein folding trajectories. Protein folding simulations search for trajectories leading to conformations close to the native (folded) protein structure originating from an unfolded conformation. During the folding process, the protein changes its conformations into what are called meta-stable and transition stages [4]. In a metastable stage, consecutive protein conformations keep a similar structure and display only small variations. In a transition stage, consecutive protein conformations change from one meta-stable stage to another and exhibit large structural variations. In order to identify these stages, it is important to identify when one or multiple trajectories eventually converge to the same conformation. Work has been done to understand intra-trajectory and inter-trajectory convergence. These studies [7, 51, 58, 60] explore multiple folding trajectory spaces in parallel and determine what conformations are more likely to be stable.

Computational trajectory analysis usually performs a large scale comparison of trajectory frames, constructing a centralized dissimilarity matrix using all the trajectory data, reducing the dimensionality of the matrix, and then clustering the low dimensional matrix. The centralized nature of the algorithms in Best el al. [7]

and Phillips et al. [51] makes their analysis inefficient when dealing with large proteins and long trajectories. Other work in [58] analyzes simple statistical data of long trajectories at a very large scale. Our previous work [33, 60] deals with this issue in a local to global fashion, rendering the parallel analysis very efficient for large datasets and is suitable for in-situ analysis.

We used 31 simulated protein folding trajectories from MoDEL, the Molecular Dynamics Extended Library [43]. MoDEL is a large library of molecular dynamics trajectories of representative protein structures. Trajectories of all monomeric soluble structures have been studied by means of state-of-the-art atomistic molecular dynamics simulations in near-physiological conditions. Trajectories used for our analysis range from 2,000 to 20,000 time steps (i.e, number of data points M) and from 58 to 747 residues (i.e., number of dimensions N). Table 3 shows their characteristics.

Table 3: Characteristics of 31 MoDEL Trajectories

Characteristic	Mean	Stdev	Min	Max
Number of residues	193.06	145.29	58	747
Simulation time (ps)	9,779.03	3,425.85	2,000	20,000

5.1 Trajectory analysis

We perform the protein folding trajectory analysis as if it was a clustering problem. Simulations can be performed in parallel, with different nodes taking care of different segments of a trajectory, or, more accurately, different trajectories given particular starting conditions. As simulations progress, in-situ analysis is necessary to determine what conformational spaces have been analyzed and whether the current conformation is stable or transitional. To perform this analysis in parallel, we characterize each conformation (i.e., a specific conformation associated with a trajectory frame) by its collection of secondary structures given the Ramachandran plot [53]. That is, every residue was characterized by the torsion angle phi, ϕ, (angle between the C-N-CA-C atoms) versus the torsion angle psi, ψ, (angle between the N-CA-C-N atoms), and omega ω (usually restricted to be 180 deg for the typical trans case or 0 deg for the rare cis case). Based on the constraints of the torsion angles (ϕ, ψ, and ω) as described by the Ramachandran, we can associate each amino acid residue in the protein with one of six types of secondary structures: α-helix, β-strand, Polyproline PII-helix, γ'-turn, γ-turn, and cis-peptide bonds. As a protein folds and unfolds over time, its residues may participate in very different types of secondary structures, but if conformations are revisited over time, they should cluster together. We hypothesize that by clustering secondary structures, multiple fine grained clusters associated with specific secondary structure transformations will arise over time. Sequences of fine grained clusters will form a cluster fingerprint. This fingerprint can be used to identify stable phases and to differentiate conformational search spaces.

5.2 Results

Quantitative evaluation. The first evaluation of KeyBin2 for folding trajectories measures clustering time. As shown by figure 3 KeyBin2's overhead is very small. Given the size of the data (number of dimensions times number of data points), KeyBin2's time is around 4 seconds in total, or 0.0004 seconds per frame. It's execution time is much smaller than the other algorithms. Given its small overhead and its parallel nature, KeyBin2 can be seamlessly used for in-situ analysis.

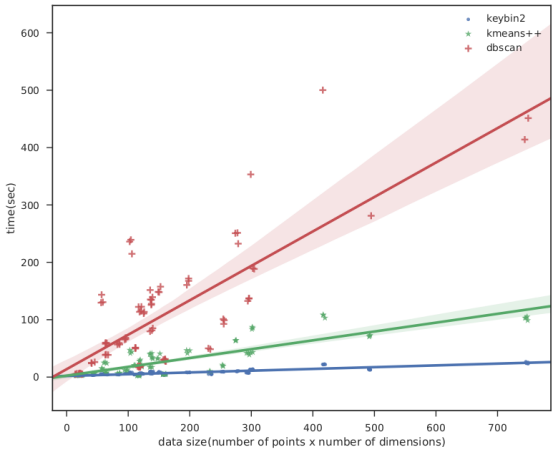

Figure 3: Execution time for clustering protein trajectories.

Qualitative evaluation. To show how KeyBin2's clustering output can be used to differentiate conformational search spaces, we validated our method using an offline probabilistic approach. After a trajectory is completed, we selected N distinct conformations sampled by using a power law distribution with respect to the distance to the mean conformation. This setup is designed to find a set of diverse representative conformations along the trajectory. For each of them, we compute the root mean squared deviation with respect to each frame in the trajectory. Given a set of root mean squared deviation time series, we preprocessed the data by converting the distance measures into probabilities that a particular time step (i.e., frame) of the trajectory is a given conformation.

$$Pr(l \text{ is stable at } i | l \in L, d_{k,i} \in D_i) = \frac{1/d_{l,i}}{\sum_{k=1}^{N} 1/d_{k,i}}, \quad (3)$$

where L is the set of N distinct conformations and D_i is the set of distance measures for each conformation at frame i. We then create a probability distribution of stability for each representative conformation, which for simplicity we denote as a label, at time step i using the previous 100 time steps. Using the probability distributions, we calculate the center of the 70% *High Density Region* (HDR) for each label. This generates a score of stability ranging from 0 to 1 for each label at time step i, where 1 indicates high label stability and 0 indicates low stability. To determine if frame i is not stable we compare the two highest label stability scores as follows,

$$Stability(s_{p,i}, s_{q,i}, w) = \begin{cases} s_{p,i} - s_{q,i} < w, & not\ stable \\ otherwise, & p\ is\ stable \end{cases}, \quad (4)$$

where w is a predefined threshold, $p, q \in L$, $s_{l,i}$ is the stability score of label l at frame i, and label p has the higher stability score at frame i.

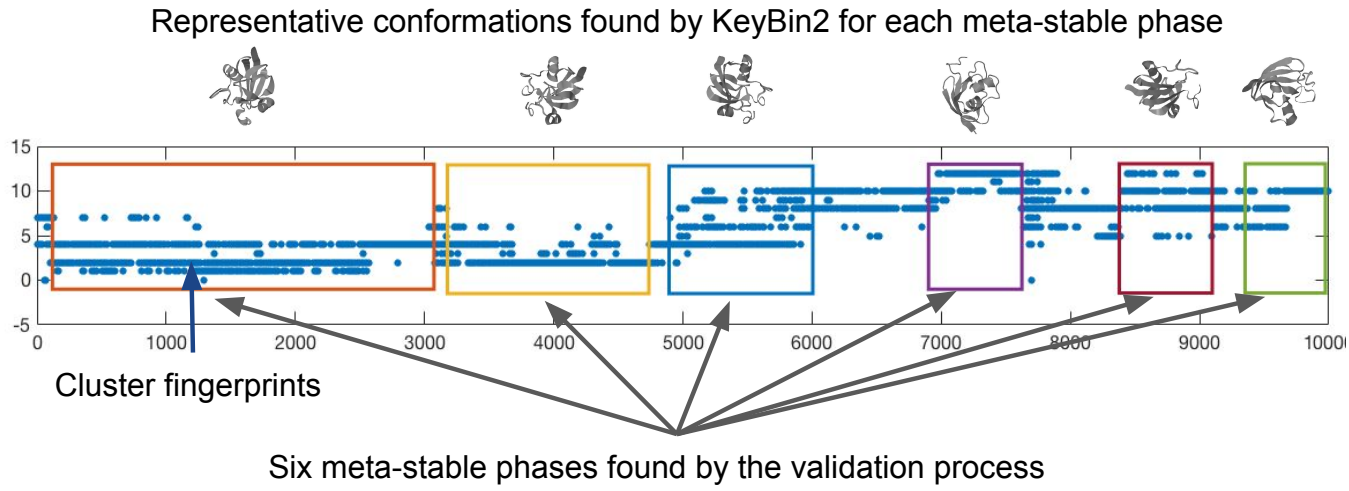

Figure 4: Qualitative clustering validation for 10,000 frames of trajectory 1a70. Rectangles represent stable segments, sequences of vertical dots represent cluster fingerprints and denote a specific conformational search

Figure 4 shows an example of how our clustering results can be used to differentiate conformational search spaces, while at the same time using this information to identify stable versus transitional conformations. The figure shows six meta-stable phases identified by the probabilistic approach and denoted by rectangles. It also shows the clustering fingerprints (i.e., each sequence of vertical dots) produced by KeyBin2. In the figure it is possible to visually inspect when clustering fingerprints change over time and how they correspond to the different meta-stable phases. The clustering fingerprints provide a richer set of information that can be used to extend the trajectory analysis with a more fine grained understanding of the different structural changes of a conformation.

6 CONCLUSIONS

In this paper we present the improved binning-based clustering algorithm *KeyBin2*. This parallel clustering algorithm uses bootstrapping and random projection methods to overcome limitations our previous method (KeyBin). The rotation effect of random projections helps to separate overlapping clusters as well as to deal with non orthogonal dimensions. In this version, we use discrete optimization to determining partitions in an non-parametric way, thus producing more robust clustering results. With these improvements, KeyBin2 improves scalability and can deal with more complex data than its predecessor. Experiments show that our algorithm scales linearly when the number of data points or as the dimensionality increases. In its current form, KeyBin2 can be used for unsupervised learning of streaming data on distributed locations without moving large data quantities to a centralized place. Finally, we show the applicability of KeyBin2 for in-situ analysis of folding trajectories.

ACKNOWLEDGMENTS

This work was supported by NSF grants *CAREER: Enabling Distributed and In-Situ Analysis for Multidimensional Structured Data* (NSF ACI-1453430) and *BIGDATA: IA: Collaborative Research: In Situ Data Analytics for Next Generation Molecular Dynamics Workflows* (NSF 1741057). We also thank the UNM Center for Advanced Research Computing for computational resources used in this work.

REFERENCES

[1] C. C. Aggarwal, J. L. Wolf, P. S. Yu, C. Procopiuc, and J. S. Park. 1999. Fast Algorithms for Projected Clustering. *SIGMOD Rec.* 28, 2 (1999), 61–72.
[2] Charu C Aggarwal and Philip S Yu. 2001. Outlier detection for high dimensional data. In *ACM Sigmod Record*, Vol. 30. ACM, 37–46.
[3] Rakesh Agrawal, Johannes Gehrke, Dimitrios Gunopulos, and Prabhakar Raghavan. 1998. Automatic subspace clustering of high dimensional data for data mining applications. *ACM SIGMOD Record* 27, 2 (1998), 94–105. DOI:http://dx.doi.org/10.1145/276305.276314
[4] David Baker. 1998. Metastable states and folding free energy barriers. *Nature Structural and Molecular Biology* 5 (1998).
[5] M. Bendechache, N. A. Le-Khac, and M. T. Kechadi. 2016. Hierarchical Aggregation Approach for Distributed Clustering of Spatial Datasets. In *2016 IEEE 16th International Conference on Data Mining Workshops (ICDMW)*. 1098–1103.
[6] J. C. Bennett, H. Abbasi, P. Bremer, R. Grout, A. Gyulassy, T. Jin, S. Klasky, H. Kolla, M. Parashar, V. Pascucci, P. Pebay, D. Thompson, H. Yu, F. Zhang, and J. Chen. 2012. Combining In-situ and In-transit Processing to Enable Extreme-scale Scientific Analysis. In *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*. IEEE Computer Society Press, 49:1–49:9.
[7] C. Best and H. C. Hege. 2002. Visualizing and identifying conformational ensembles in molecular dynamics trajectories. *Computing in Science Engineering* 4, 5 (2002).
[8] Kevin Beyer, Jonathan Goldstein, Raghu Ramakrishnan, and Uri Shaft. 1999. When is Nearest Neighbor meaningful?. In *International conference on database theory*. Springer, 217–235.
[9] Ella Bingham, Ella Bingham, Heikki Mannila, and Heikki Mannila. 2001. Random projection in dimensionality reduction: applications to image and text data. *International Conference on Knowledge Discovery and Data Mining (KDD)* (2001), 245–250. DOI:http://dx.doi.org/10.1145/502512.502546
[10] Tadeusz Caliński and Jerzy Harabasz. 1974. A dendrite method for cluster analysis. *Communications in Statistics-theory and Methods* 3, 1 (1974), 1–27.
[11] Canonizer. 2012. Implementation of MAFIA Subspace Clustering on NVidia GPUs. (2012). https://github.com/canonizer/gpumafia open source code.
[12] CERN 2018. CERN - Large Hadron Collider. (2018). http://home.web.cern.ch/
[13] X Chen, J Benson, and T Estrada. 2017. keybin: Key-Based Binning for Distributed Clustering. In *2017 IEEE International Conference on Cluster Computing (CLUSTER)*. 572–581. DOI:http://dx.doi.org/10.1109/CLUSTER.2017.96
[14] Xinyu Chen and Trilce Estrada. 2017. Index Clustering: A Map-reduce Clustering Approach using Numba. In *Proceedings of the 6th International Conference on Data Science, Technology and Applications - Volume 1: DATA,*. INSTICC, SciTePress, 233–240. DOI:http://dx.doi.org/10.5220/0006437402330240

[15] Bi-Ru Dai and I-Chang Lin. 2012. Efficient map/reduce-based dbscan algorithm with optimized data partition. In *Cloud Computing (CLOUD), 2012 IEEE 5th International Conference on*. IEEE, 59–66.

[16] Lisandro Dalcín, Rodrigo Paz, and Mario Storti. 2005. MPI for Python. *J. Parallel and Distrib. Comput.* 65, 9 (2005), 1108–1115.

[17] Sanjoy Dasgupta. 1999. Learning mixtures of Gaussians. In *Foundations of computer science, 1999. 40th annual symposium on*. IEEE, 634–644.

[18] Sanjoy Dasgupta and Anupam Gupta. 1999. An elementary proof of the Johnson-Lindenstrauss lemma. *International Computer Science Institute, Technical Report* (1999), 6–99.

[19] M. Dreher and B. Raffin. 2014. A Flexible Framework for Asynchronous in Situ and in Transit Analytics for Scientific Simulations. In *IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing*. 277–286.

[20] Martin Ester, Hans-Peter Kriegel, Jörg Sander, Xiaowei Xu, and Others. 1996. A density-based algorithm for discovering clusters in large spatial databases with noise.. In *Kdd*, Vol. 96. 226–231.

[21] Trilce Estrada and Michela Taufer. 2012. On the Effectiveness of Application-aware Self-management for Scientific Discovery in Volunteer Computing Systems. In *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*. IEEE Computer Society Press, 80:1–80:11.

[22] Xiaoli Z Fern and Carla E Brodley. 2003. Random projection for high dimensional data clustering: A cluster ensemble approach. In *Proceedings of the 20th International Conference on Machine Learning (ICML-03)*. 186–193.

[23] A. Gionis, P. Indyk, and R. Motwani. 1999. Similarity Search in High Dimensions via Hashing. In *Proceedings of the 25th International Conference on Very Large Data Bases*. Morgan Kaufmann Publishers Inc., 518–529.

[24] S Goil, H Nagesh, and A Choudhary. 1999. MAFIA: Efficient and scalable subspace clustering for very large data sets. ... *Discovery and Data Mining* 5 (1999), 443–452. DOI: http://dx.doi.org/CPDC-TR-9906-010

[25] Markus Götz, Christian Bodenstein, and Morris Riedel. 2015. HPDBSCAN: highly parallel DBSCAN. In *Proceedings of the Workshop on Machine Learning in High-Performance Computing Environments*. ACM, 2.

[26] Poonam Goyal, Sonal Kumari, Shubham Singh, Vivek Kishore, Sundar S Balasubramaniam, and Navneet Goyal. 2016. A Parallel Framework for Grid-Based Bottom-Up Subspace Clustering. In *Data Science and Advanced Analytics (DSAA), 2016 IEEE International Conference on*. IEEE, 331–340.

[27] Alexander Hinneburg and Da Keim. 1998. An Efficient Approach to Clustering in Large Multimedia Databases with Noise. *In Proceedings of 4th International Conference in Knowledge Discovery and Data Mining (KDD 98)* (1998), 58–65. DOI: http://dx.doi.org/10.1.1.44.3961

[28] L. Hong, Z. Gao, X. Pan, and K. Yang. 2011. Segmentation of high resolution remote sensing image based on hierarchically multiscale object-oriented Markov random fields model. In *IEEE International Conference on Spatial Data Mining and Geographical Knowledge Services (ICSDM)*. 343–347.

[29] Xiaojuan Hu, Lei Liu, Ningjia Qiu, Di Yang, and Meng Li. 2017. A MapReduce-based improvement algorithm for DBSCAN. *Journal of Algorithms & Computational Technology* (2017), 1748301817735665.

[30] P. Indyk and R. Motwani. 1998. Approximate Nearest Neighbors: Towards Removing the Curse of Dimensionality. In *Proceedings of the Thirtieth Annual ACM Symposium on Theory of Computing*. ACM, 604–613.

[31] J.A. Insley, L. Grinberg, and M.E. Papka. 2011. Visualizing multiscale, multiphysics simulation data: Brain blood flow. In *IEEE Symposium on Large Data Analysis and Visualization (LDAV)*. 3–7.

[32] Anil K Jain and East Lansing. 2009. Data Clustering : 50 Years Beyond K-Means 1 Anil K . Jain Michigan State University. (2009).

[33] T. Johnston, B. Zhang, A. Liwo, S. Crivelli, and M. Taufer. 2017. In situ data analytics and indexing of protein trajectories. *J Comput Chem.* 38, 16 (2017).

[34] H. Kargupta, W. Huang, K. Sivakumar, and E. Johnson. 2001. Distributed Clustering Using Collective Principal Component Analysis. *Knowledge and Information Systems* 3, 4 (2001), 422–448.

[35] H. Kawashima, R. R. Sato, and H. Kitagawa. 2008. Models and Issues on Probabilistic Data Streams with Bayesian Networks. In *Proc. of the International Symposium on Applications and the Internet (SAINT)*.

[36] Siu Kwan Lam, Antoine Pitrou, and Stanley Seibert. 2015. Numba: A llvm-based python jit compiler. In *Proceedings of the Second Workshop on the LLVM Compiler Infrastructure in HPC*. ACM, 7.

[37] Z. Li and M. Parashar. 2007. Grid-based asynchronous Replica Exchange. In *8th IEEE/ACM International Conference on Grid Computing*. 201 –208.

[38] Liao. 2018. Parallel K-Means Data Clustering. (2018). http://www.ece.northwestern.edu/~wkliao/Kmeans/index.html open source code.

[39] Hubert W Lilliefors. 1967. On the Kolmogorov-Smirnov test for normality with mean and variance unknown. *Journal of the American statistical Association* 62, 318 (1967), 399–402.

[40] Y. Liu, L. C. Jiao, F. Shang, F. Yin, and F. Liu. 2013. An Efficient Matrix Bi-factorization Alternative Optimization Method for Low-rank Matrix Recovery and Completion. *Neural Netw.* 48 (2013).

[41] Stuart Lloyd. 1982. Least squares quantization in PCM. *IEEE transactions on information theory* 28, 2 (1982), 129–137.

[42] LSST 2018. Large Synoptic Survey Telescope. (2018). http://www.lsst.org/lsst/

[43] INB. Molecular modeling and Bioinformatics Group. 2018. Molecular Dynamics Extended Library. (2018). http://mmb.pcb.ub.es/MoDEL/index.jsf

[44] NASA 2015. NASA-Center for Climate Simulation. (2015). http://www.nasa.gov/topics/earth/features/climate-sim-center.html

[45] John Nickolls, Ian Buck, Michael Garland, and Kevin Skadron. 2008. Scalable parallel programming with CUDA. In *ACM SIGGRAPH 2008 classes*. ACM, 16.

[46] J.M. Paluska, H. Pham, U. Saif, G. Chau, C. Terman, and S. Ward. 2008. Structured Decomposition of Adaptive Applications. In *Proceedings of the 6th Annual IEEE International Conference on Pervasive Computing and Communications*. 1–10.

[47] Mostofa Ali Patwary, Diana Palsetia, Ankit Agrawal, Wei-keng Liao, Fredrik Manne, and Alok Choudhary. 2012. 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*. IEEE Computer Society Press, 62.

[48] Md Mostofa Ali Patwary, Suren Byna, Nadathur Rajagopalan Satish, Narayanan Sundaram, Zarija Lukić, Vadim Roytershteyn, Michael J Anderson, Yushu Yao, Pradeep Dubey, and others. 2015. BD-CATS: big data clustering at trillion particle scale. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*. ACM, 6.

[49] Alejandro Pelaez, Andres Quiroz, James C Browne, Edward Chuah, and Manish Parashar. 2014. Online failure prediction for hpc resources using decentralized clustering. In *High Performance Computing (HiPC), 2014 21st International Conference on*. IEEE, 1–9.

[50] Dan Pelleg, Andrew W Moore, and Others. 2000. X-means: Extending k-means with efficient estimation of the number of clusters.. In *Icml*, Vol. 1. 727–734.

[51] J. Phillips. 2011. Validating clustering of molecular dynamics simulations using polymer models. *BMC Bioinformatics* 12, 1 (2011).

[52] A. Quiroz, M. Parashar, N. Gnanasambandam, and N. Sharma. 2012. Design and Evaluation of Decentralized Online Clustering. *ACM Trans. Auton. Adapt. Syst.* 7, 3 (2012), 34:1–34:31.

[53] G.N. Ramachandran, C. Ramakrishnan, and V. Sasisekharan. 1963. Multipolar representation of protein structure. *Journal of Molecular Biology* 7, 95 (1963).

[54] Conrad Sanderson and Ryan Curtin. 2017. An open source C++ implementation of multi-threaded Gaussian mixture models, k-means and expectation maximisation. *arXiv preprint arXiv:1707.09094* (2017).

[55] SDSS 2014. SDSS - Sloan Digital Sky Survey. (2014). https://www.sdss3.org/

[56] Bernard W Silverman. 1981. Using kernel density estimates to investigate multimodality. *Journal of the Royal Statistical Society. Series B (Methodological)* (1981), 97–99.

[57] D. Tiwari, S. S. Vazhkudai, Y. Kim, X. Ma, S. Boboila, and P. J. Desnoyers. 2012. Reducing Data Movement Costs Using Energy-Efficient, Active Computation on SSD. In *2012 Workshop on Power-Aware Computing and Systems*. USENIX.

[58] T. Tu. 2008. A scalable parallel framework for analyzing terascale molecular dynamics simulation trajectories. In *CM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis (SCâĂŹ08)*.

[59] Dongkuan Xu and Yingjie Tian. 2015. A Comprehensive Survey of Clustering Algorithms. *Annals of Data Science* 2, 2 (2015), 165–193. DOI: http://dx.doi.org/10.1007/s40745-015-0040-1

[60] B. Zhang, T. Estrada, P. Cicotti, and M. Taufer. 2014. Enabling in-situ data analysis for large protein folding trajectory datasets. In *IEEE International Parallel and Distributed Processing Symposium*.

[61] F. Zheng, H. Yu, C. Hantas, M. Wolf, G. Eisenhauer, K. Schwan, H. Abbasi, and S. Klasky. 2013. GoldRush: Resource Efficient in Situ Scientific Data Analytics Using Fine-grained Interference Aware Execution. In *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*. ACM, 78:1–78:12.