YADING: Fast Clustering of Large-Scale Time Series Data

Rui Ding, Qiang Wang, Yingnong Dang, Qiang Fu, Haidong Zhang, Dongmei Zhang
Microsoft Research
Beijing, China
{juding, qiawang, yidang, qifu, haizhang, dongmeiz}@microsoft.com

ABSTRACT

Fast and scalable analysis techniques are becoming increasingly important in the era of big data, because they are the enabling techniques to create real-time and interactive experiences in data analysis. Time series are widely available in diverse application areas. Due to the large number of time series instances (e.g., millions) and the high dimensionality of each time series instance (e.g., thousands), it is challenging to conduct clustering on large-scale time series, and it is even more challenging to do so in real-time to support interactive exploration.

In this paper, we propose a novel end-to-end time series clustering algorithm, YADING, which automatically clusters large-scale time series with fast performance and quality results. Specifically, YADING consists of three steps: sampling the input dataset, conducting clustering on the sampled dataset, and assigning the rest of the input data to the clusters generated on the sampled dataset. In particular, we provide theoretical proof on the lower and upper bounds of the sample size, which not only guarantees YADING's high performance, but also ensures the distribution consistency between the input dataset and the sampled dataset. We also select L_1 norm as similarity measure and the multi-density approach as the clustering method. With theoretical bound, this selection ensures YADING's robustness to time series variations due to phase perturbation and random noise.

Evaluation results have demonstrated that on typical-scale (100,000 time series each with 1,000 dimensions) datasets, YADING is about 40 times faster than the state-of-the-art, sampling-based clustering algorithm DENCLUE 2.0, and about 1,000 times faster than DBSCAN and CLARANS. YADING has also been used by product teams at Microsoft to analyze service performance. Two of such use cases are shared in this paper.

1. INTRODUCTION

Fast and scalable techniques are becoming increasingly important in interactive data exploration in the era of big data. When conducting interactive data exploration, users often leverage different analysis techniques such as clustering, matching, filtering, and visualization, etc. to perform various tasks, e.g., understanding data characteristics, spotting patterns, and validating hypotheses, etc. These tasks help users obtain insights and make informed decisions [2]. The process of completing these tasks is usually iterative, which requires analysis techniques to be fast and scalable in order to create real-time and interactive exploration experiences.

Proceedings of the VLDB Endowment, Vol. 8, No. 5
Copyright 2015 VLDB Endowment 2150-8097/15/01.

Time series are a common data type and they are widely used in diverse application areas, such as finance, economics, communication, automatic control, and online services, etc. Clustering time series is to identify the homogeneous groups of time series data based on their similarity. It is an important and useful technique for exploratory study on the characteristics of various groups in a given time series dataset.

For instance, large datacenters may have tens of thousands or even more servers running and hosting different services [3] . In order to ensure service quality, various types of performance counters (e.g., CPU usage, disk I/O, network throughput, etc.) are continuously collected on each server. For analysis purpose, they are often aggregated at pre-defined time intervals (e.g., 5 minutes) on each server, resulting in time series representing certain performance characteristic of the service(s) under monitoring.

In practice, such time series are a rich and important data source for software engineers to perform quality assurance tasks, including detecting abnormal states, diagnosing performance issues, and understanding the overall service health status, etc. Fast and scalable clustering of time series is essential to completing these tasks since grouping provides common performance profiles across servers or services. Moreover, clustering transforms time series data into categorical attributes, thus making it possible to analyze time series data together with other categorical attributes. For example, if a group of servers have similar network usage profiles that consist of unusual spikes, and they also share the same router, then it is reasonable for the engineers to suspect that the router may have issues related to the high network usage.

Time series are well known for their high dimensionality. When combined with high dimensionality, the large scale of time series datasets poses great challenges to clustering with high performance, which is often required in interactive analysis in real practice. In the above example of service monitoring, the scale of datasets of performance counters ranges from ~10,000 to ~100,000 instances each with ~100 to ~1,000 dimensions. The scale can easily increase to millions on e-Commerce related datasets that involve millions of users or transactions.

Although the topic of time series clustering has been studied for about two decades [1], and hundreds of papers have researched on various aspects of time series data [4][5], few of the research works in literature are able to address the high-performance and scalability challenges.

In this paper, we propose a novel end-to-end time series clustering algorithm, YADING, which automatically groups large-scale time series data with fast performance and quality results. YADING consists of three steps: sampling the input dataset, conducting clustering on the sampled dataset, and assigning the rest of the input data to the clusters generated on the sampled dataset. In particular, we provide theoretical proofs at the first two steps of YADING to guarantee both high efficiency and high accuracy in clustering.

At the sampling step, we provide theoretical proof on the lower and upper bounds of the size of the sampled dataset, which ensures the high accuracy of clustering results while significantly reducing the computational cost. This sample size is primarily determined by how well the distribution of the input dataset is preserved on the sampled dataset. It should be noted that the bounds of the sample size are independent of the size of the input dataset, which provides theoretical guarantee on YADING's high performance.

Our main clustering scheme employs L_1 norm as similarity measure combined with multi-density based clustering. We provide theoretical bound on the robustness of YADING to the time series variations caused by phase perturbation and random noise. Specifically, our theoretical analysis reveals the relationships among cluster quality, data size and degree of variation. Such relationships provide quantitative estimation on the optimal parameters, e.g., the tolerance for preserving a certain clustering quality under a given data size. In addition, the simplicity of L_1 computation is compliant with our target of high efficiency.

It should be noted that no single similarity measure can work well on every time series dataset due to various domain characteristics. In this paper, we focus on the domains where L_1 (which is typical in L_p family) is a reasonable similarity measure. In fact, a wide range of real time series applications (e.g., performance counters in datacenters) belong to these domains [12].

In addition, we also propose two algorithms to automatically estimate the parameters used at different stages of YADING based on the given time series dataset. The first algorithm determines the frame length in PAA (Piecewise Aggregate Approximation) [13], which is used to reduce the dimensionality of a time series instance. In multi-density based clustering, the second algorithm calculates the number of densities using a model-based approach. These two algorithms remove the need of manually setting these parameters, thus making YADING fully automatic.

Several product teams at Microsoft have used YADING to conduct quality analysis for online services, and provided positive feedback. The high performance of YADING is key to enabling the engagement with those teams.

The contributions of our work are summarized as follows:

- We propose a novel end-to-end time series clustering algorithm, YADING, which clusters large-scale time series data with fast performance and quality results.
- We provide theoretical proof on the bounds to determine the size of the sampled dataset which is independent of the size of the input dataset, thus guaranteeing YADING's high performance, as well as the distribution consistency between the sampled dataset and the input dataset.
- We provide theoretical bound on the robustness of YADING (i.e., L_1 distance combined with multi-density based clustering) to the time series variations due to phase perturbation and random noise. Specifically, our theoretical analysis reveals relationships among cluster quality, data size and degree of variation.
- We have conducted thorough testing on both simulation datasets and real datasets to demonstrate YADING's high performance and quality. Several teams at Microsoft have used YADING to help analyze time series data and provided positive feedback.

This paper is organized as follows. Section 2 discusses the related work. Section 3 describes in detail the YADING algorithm. Section 4 reports the evaluation results. Section 5 illustrates the application examples in practice. Section 6 discusses YADING's future work, and Section 7 concludes the paper.

2. RELATED WORK

The topic of time series clustering has received a lot of attention in the research community. Two survey papers [4][5] provide extensive studies on the large amount of work published on this topic. In this section, we first summarize the work specifically focusing on time series clustering, which is highly relevant to our work. Then we discuss three most commonly used techniques in time series clustering, which are also general to clustering problems: similarity measurement, clustering method, and data reduction. As an end-to-end solution, YADING leverages all of these techniques.

2.1 Overview of Time Series Clustering

Most of the existing time series clustering algorithms fall into two categories, depending on whether the similarity measures are defined directly on input data, or on the features extracted from input data.

In the first category, Golay et al. [27] studied three different distance measures of time series, the *Euclidean distance* and two cross-correlation based distances. Liao et al. [28] adopted DTW (Dynamic Time Warping)and employed genetic clustering to group the time series data generated from battle simulation.

In the second category, time series instances are considered to be generated by underlying models or probability distributions. Two time series instances are considered similar when their corresponding models are similar. ARIMA (Auto-Regressive Integrated Moving Average) is used in [29]. Other models such as Gaussian mixture model [30] are also used to measure the similarity of time series. The time complexity of the model-based techniques is usually high due to the complexity of model learning.

Most of the above algorithms focus on improving the clustering accuracy by proposing or leveraging sophisticated similarity measures. While the aforementioned similarity measures can help improve the clustering accuracy, their time complexity is usually high. In the context of fast clustering, similarity measures with low computational cost are preferred.

2.2 Similarity Measurement

We list as follows the similarity measures that are commonly used in clustering problems.

L_p norm is the most popular class of dissimilarity measures [12]. Among the measures with different p values, L_1 is known for its robustness to impulsive noise [12]. The measures with $p \geq 3$ often suffer from the *curse of dimensionality* [18].

Pearson's correlation [6] is another widely used similarity measure. Its primary advantage is its invariance to linear transformation. Since its calculation involves standard deviation and covariance, it is about 4~10 times slower than calculating L_p norm, according to our simulation tests.

DTW (Dynamic Time Warping) [7] calculates the optimal match of two time series instances with certain constraints. It allows the two instances to have different lengths, and it is robust to phase perturbation. Generally, computing DTW requires $O(D^2)$ (D is the length of each time series instance), which is one order of magnitude slower than calculating L_p norm.

2.3 Clustering Methods

Two categories of clustering methods [8] are most related to our context: partitioning methods and density-based methods.

Partitioning methods identify k partitions of the input data with each partition representing a cluster. Partitioning methods need manual specification of k as the number of clusters. *k-means* and *k-medoid* are typical partitioning algorithms. CLARANS [19] is an improved *k-medoid* method, and it is more effective and efficient.

Density-based algorithms [22] treat clusters as dense regions of objects in the spatial space separated by regions of low density. A well-known algorithm in this category is DBSCAN [17]. Density-based methods are able to identify clusters with arbitrary shapes, and they can detect outliers as well. The time complexity of DBSCAN is fairly low. DBSCAN has two major issues. One issue is that it requires manual specification of two parameters: Eps, which is the maximum radius of the neighborhood; and MinPts, which is the minimum number of points in an Eps-neighborhood of that point. The other issue is that it can only handle datasets with single density. Various techniques [23] [24] are further proposed to address these issues.

In addition, grid-based methods slice the object space into cells [21], and perform clustering on the grid structure of the cells. Other clustering methods such as Hierarchical clustering and model-based clustering [20] usually have high computational cost.

2.4 Data Reduction

Data reduction can significantly impact the clustering performance on large-scale datasets. Data reduction includes reducing the dimensionality of time series instances and reducing the size of time series.

Discrete Fourier Transform (DFT) is used to represent time series in frequency domain [9], and Discrete Wavelets Transform (DWT) is used to provide additional temporal information [10]. Singular Value Decomposition (SVD) and Piecewise Aggregate Approximation (PAA) are used in [11] and [13], respectively. Compared to other techniques, PAA is faster and simpler for mining time series [25].

Sampling techniques are adopted to reduce the size of time series dataset [16]. DENCLUE 2.0 [36] employs random sampling with different sample sizes to reduce the number of iterations for density attractor estimation. The experiment results show that 20% sampling rate is sufficient to achieve good clustering results on a specific synthetic dataset. However, the sample size is chosen in an ad hoc manner in DENCLUE 2.0, and how to set proper sample size for arbitrary large-scale time series datasets is still unknown and challenging. Kollios et al. [14] utilize biased sampling to capture the local density of datasets. Zhou et al. [15] leverage random sampling due to its simplicity and low complexity.

3. ALGORITHMS

In this section, we introduce the YADING algorithm in detail. Overall, YADING consists of three steps: data reduction, clustering, and assignment. Data reduction is conducted via sampling the input dataset, and reducing the dimensionality of the input time series instances. Clustering is then conducted on the sampled dataset. Finally, all the input time series are assigned to the clusters resulted from the sampled dataset. In the rest of this section, we will introduce in detail our choice of algorithm at each step, discuss how high performance and good accuracy are achieved, and how parameters are estimated automatically.

3.1 Data Reduction

Suppose that there are N input time series instances denoted as $\mathcal{T}_{N\times D} \coloneqq \{T_1, T_2, \dots, T_N\}$. Each time series instance, also referred to as object for brevity, is denoted as $T_i \coloneqq (t_{i1}, t_{i2}, \dots, t_{iD}), T_i \in \mathcal{T}_{N\times D}$, where D indicates the length of T_i. T_i is a D-dimensional vector, and $\mathcal{T}_{N\times D}$ is a set of N such vectors. The goal of data reduction is to determine values s and d, such that a sampled dataset $\mathcal{T}_{s\times d}$ of $\mathcal{T}_{N\times D}$ satisfies the following constraints: $s \le N$, $d \le D$, and $\mathcal{T}_{s\times d}$ preserves the underlying distribution of $\mathcal{T}_{N\times D}$.

3.1.1 Random Sampling

Sampling is the most effective mechanism to handle the scale of the input dataset. Since we want to achieve high performance and we do not assume any distribution of the input dataset, we choose random sampling as our sampling algorithm.

In practice, a predefined sampling rate is often used to determine the size s of the sampled dataset. As *N*, the size of the input dataset, keeps increasing, s also increases accordingly, which will result in slower clustering performance on the sampled dataset. Furthermore, it is unclear what impact the increased number of samples may have on the clustering accuracy.

We come up with the following theoretical bounds to guide the selection of s.

Assume that the ground truth of clustering is known for $\mathcal{T}_{N\times D}$, i.e. all the $T_i \in \mathcal{T}_{N\times D}$ belong to k known groups, and n_i represents the number of time series in the *i*th group. Let $p_i = \frac{n_i}{N}$ denote the population ratio of group *i*. Similarly, $p_i' = \frac{n_i'}{s}$ denotes the population ratio of the *i*th group on the sampled dataset. $|p_i - p_i'|$ reflects the deviation of population ratio between the input dataset $\mathcal{T}_{N\times D}$ and the sampled dataset $\mathcal{T}_{s\times d}$. We formalize the selection of the sample size s as finding the lower bound s_l and upper bound s_u such that, given a tolerance ϵ and a confidence level $1-\alpha$, (1) group i with p_i less than ϵ is not guaranteed to have sufficient instances in the sampled dataset for $s < s_l$, and (2) the maximum of ratio deviation $|p_i - p_i'|$, $1 \le i \le k$, is within a given tolerance for $s \ge s_u$. Intuitively, the lower bound constrains the smallest size of clusters that are possible to be found; and the upper bound indicates that when the sample size is greater than a threshold, the distribution of population ratio on the sampled dataset is representative of the input dataset. Such distribution consistency tends to produce close enough clustering results on both datasets, which coincides with intuition and is also evidenced by experimental results (Section 4).

Lemma 1 (lower bound): Given *m*, the least number of instances present in the sampled dataset for group i, and the confidence level $1-\alpha$, the sample size $s \geq \frac{m+z_\alpha\left(\frac{z_\alpha}{2}+\sqrt{m+\frac{z_\alpha^2}{4}}\right)}{p_i}$ satisfies $P(n_i' \geq m) > 1-\alpha$. Here, $z_{\alpha/2}$ is a function of α, $P(Z > z_{\alpha/2}) = \alpha/2$, where $Z \sim N(0,1)$.

With confidence level $1-\alpha$, Lemma 1 provides the lower bound on sample size s that guarantees *m* instances in the sampled dataset for any cluster with population ratio higher than p_i. For sample, if a cluster has $p_i > 1\%$, and we set $m = 5$ with confidence 95% (i.e. $1-\alpha = 0.95$), then we get $s_l \geq 1{,}030$. In this case, when $s < 1{,}030$, the clusters with $p_i < 1\%$ have perceptible probability (>5%) to be missed in the sampled dataset.

It should be noted that the selection of *m* is related to the clustering method applied to the sampled dataset. For example, DBSCAN is a density-based method, and it typically requires 4 nearest neighbors of a specific object to identify a cluster. Thus, any cluster with size less than 5 is difficult to be found. The consideration on clustering method also supports our formalization for deciding s_l.

Proof: Based on the property of random sampling, the probability of a sample belonging to the *i*th group is p_i. Since each sample is independent, the number of samples belonging to the *i*th group follows *Binomial distribution* $n_i' \sim B(s, p_i)$.

Considering $B(s,p_i) \sim N(sp_i, sp_i(1-p_i))$ when s is large, and the distribution is not too skewed, n_i' approximately follows $N(sp_i, sp_i(1-p_i))$. Denote $\sigma = \sqrt{sp_i(1-p_i)}$.

Event $\{n_i' \geq m\} \leftrightarrow \left\{\frac{n_i'-sp_i}{\sigma} \geq \frac{m-sp_i}{\sigma}\right\} \leftrightarrow \left\{Z \geq \frac{m-sp_i}{\sigma}\right\}$. $\frac{n_i'-sp_i}{\sigma} = Z$ is true because $n_i' \sim N(sp_i, sp_i(1-p_i))$. Therefore,

$P(n_i' \geq m) > 1-\alpha \leftrightarrow P\left(Z \geq \frac{m-sp_i}{\sigma}\right) > 1-\alpha \leftrightarrow \frac{m-sp_i}{\sigma} \leq -z_\alpha$, where $m \leq sp_i$.

The last inequality above can be equally transformed to

$$\left(p_i - \frac{m+z_\alpha^2/2}{s+z_\alpha^2}\right)^2 \geq \left(\frac{m+z_\alpha^2/2}{s+z_\alpha^2}\right)^2 - \frac{m^2}{s(s+z_\alpha^2)}, m \leq sp_i$$

Considering that the value of z_α usually falls in [0, 3], which implies $s \gg z_\alpha^2$, we get $s + z_\alpha^2 \approx s$. When applying this to the inequality above, we get a simplified version $s \geq \frac{m+z_\alpha\left(\frac{z_\alpha}{2}+\sqrt{m+\frac{z_\alpha^2}{4}}\right)}{p_i}$, hence the lemma is proven.

Lemma 2 (upper bound): Given tolerance $\epsilon \in [0,1]$, and the confidence level $1-\alpha$, we have sample size $s \geq \frac{z_{\alpha/2}^2}{4\epsilon^2}$ satisfying $P(|p_i - p_i'| < \epsilon, 1 \leq i \leq k) > 1-\alpha$.

Lemma 2 implies that the sample size s only depends on the tolerance ϵ and the confidence level $1-\alpha$, and it is independent of the input data size. For example, if we set $\epsilon = 0.01$ and $1-\alpha = 0.95$, which means that for any group, the difference of its population ratio between the input dataset and the sampled dataset is less than 0.01, then the lowest sample size to guarantee such setting is $s \geq \frac{z_{0.025}^2}{4\times 0.01^2} \sim 9{,}600$; and more samples than 9,600 are not necessary. This makes $s = 9{,}600$ the upper bound of the sample size. Moreover, this sample size does not change with the size of the input dataset given ϵ and $1-\alpha$.

Proof: $p_i' = \frac{n_i'}{s} \sim N(p_i, \frac{p_i(1-p_i)}{s}) \rightarrow Y := p_i' - p_i \sim N(0, \frac{p_i(1-p_i)}{s})$.

Event $\{|p_i - p_i'| < \epsilon\} \leftrightarrow \{|Y| < \epsilon\} \leftrightarrow \{|Z| < \frac{\epsilon}{\sigma}\}$ where $\sigma = \sqrt{\frac{p_i(1-p_i)}{s}}$. So when $\frac{\epsilon}{\sigma} > z_{\alpha/2}$, $P\left(|Z| < \frac{\epsilon}{\sigma}\right) > 1-\alpha$ holds. Expanding σ in $\frac{\epsilon}{\sigma} > z_{\alpha/2}$, we get the range value of $s \geq \frac{p_i(1-p_i)}{\epsilon^2} z_{\alpha/2}^2$. Note that $p_i(1-p_i) \leq \frac{1}{4}$, so $s \geq \frac{z_{\alpha/2}^2}{4\epsilon^2}$, which is independent of i, and satisfies $P(|p_i - p_i'| < \epsilon, 1 \leq i \leq k) > 1-\alpha$. Hence the lemma is proven.

Lemma 2 provides a loose upper bound since we replace $p_i(1-p_i)$ with $\frac{1}{4}$ to bound all the ratio values. Hence, a sample size smaller than 9,600 may still preserve reasonable ratio distribution while increasing clustering performance. In practice, we vary the sample size s from 1,030 (lower bound) to 10,000 and conduct clustering accordingly on real datasets. We choose $s = 2{,}000$ since we achieve close clustering accuracy when $s > 2{,}000$.

3.1.2 Dimensionality Reduction

We adopt PAA for dimensionality reduction because of its computational efficiency and its capability of preserving the shape of time series. Denote a time series instance with length *D* as $T_i := (t_{i1}, t_{i2}, \ldots, t_{iD})$. The transformation from T_i to $T_i' := (\tau_{i1}, \tau_{i2}, \ldots, \tau_{id})$ where $\tau_{ij} = \frac{d}{D}\sum_{k=\frac{D}{d}(j-1)+1}^{\frac{D}{d}j} t_{ik}$ is called PAA. PAA segments T_i into d frames, and uses one value (i.e. the mean value) to represent each frame so as to reduce the length of T_i from D to d.

One key issue in applying PAA is to automatically determine the number of frames d. As proved by the Nyquist-Shannon sampling theory, any time series without frequencies higher than *B* Hertz can be perfectly recovered by its sampled points with sampling rate *2*B*. This means that using *2*B* as sampling rate can preserve the shape of a frequency-bound time series. Although some time series under our study are often imperfectly frequency-bound signals, most of them can be approximated by frequency-bound signals because their very high-frequency components usually correspond to noise. Therefore, we transform the problem of determining *d* into estimating the upper bound of frequencies.

In this paper, we propose a novel auto-correlation-based approach to identify the approximate value of the frequency upper bound of all the input time series instances. The number of frames *d* is then easily determined as the inverse of the frequency upper bound.

In more details, we first identify the *typical frequency* of each time series instance T_i by locating the first local minimum on its auto-correlation curve g_i, which is denoted as $g_i(y) = \sum_{j=1}^{D-y} t_{ij} t_{i(j+y)}$, where y is the *lag*. If there is a local minimum of g_i on a particular lag y', then y' relates to a typical half-period if $g_i(y') < 0$. In this case, we call $1/y'$ the *typical frequency* for T_i. The smaller y' is, the higher the frequency it represents.

After detecting all the typical frequencies, we sort them in ascending order; and select a high percentile, e.g., the 80th percentile, to approximate the frequency upper bound of all the time series instances. The reason why we do not use the exact maximum typical frequency is to remove the potential instability caused by the small amount of extraordinary noise in some time series instances.

In our implementation, the auto-correlation curves $g_i(y)$ can be obtained efficiently using the *Fast Fourier transforms*: (1) $F_g(f) = FFT[T_i]$; (2) $S(f) = F_g(f)F_g^*(f)$; (3) $g(y) = IFFT[S(f)]$, where IFFT is inverse *Fast Fourier transforms*, and the asterisk denotes complex conjugate. Table 1 shows the algorithm of automatically estimating the frame length.

The time complexity of obtaining the number of frames and applying PAA is $O(sD \log D)$ and $O(ND)$, respectively. Therefore, the overall complexity of data reduction is $O(sD \log D + ND)$.

Table 1. Auto estimation of the frame length

```
FRAMELENGTH($\mathcal{T}'_{s\times D}$)
  for each $T_i \in \mathcal{T}'_{s\times D}$)
    $g_i(y)$ ← auto-correlation applied to $T_i$
    $y_i^*$ ← get first local minimum of $g_i(y)$
  $y^*$ ← 80% percentile on sorted $\{y_1^* \sim y_s^*\}$
  return $y^*$
```

3.2 Multi-Density Based Clustering

Using L_1 distance as the similarity measure, we perform multi-density based clustering on the sampled dataset $\mathcal{T}_{s\times d}$. In this section, we first discuss why we choose the combination of L_1 and multi-density based clustering. Then we propose a novel algorithm to automatically estimate the densities, which are key to multi-density based clustering algorithms. We present the complete procedure of our algorithm at the end of the section.

3.2.1 Overview

We adopt L_1 distance as similarity measure because of both its high computational efficiency and its robustness to impulsive noise. We do not have any assumptions on the distribution of the input dataset, which means that the input dataset may have arbitrary shapes and different densities. We select multi-density based clustering as our clustering algorithm because it is the most suitable approach to handle datasets with such characteristics.

Datasets with irregular shapes and different densities are common in practice. Using a time series dataset $\mathcal{T}_{N\times D}$ obtained from monitoring online services, we visualize its spatial distribution by projecting each instance onto the 2-dimensional principal subspace using PCA. In the example in Figure 1, the projected time series instances preserve 91% of the total energy of $\mathcal{T}_{N\times D}$, which means that the 2D projection is a good approximation of the spatial distribution of $\mathcal{T}_{N\times D}$. The time series groups shown in Figure 1 have irregular shapes and different densities. There are also noisy instances that do not belong to any group.

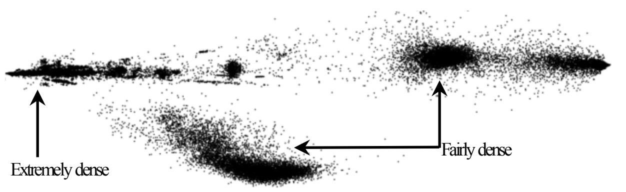

Figure 1. A real time series dataset projected onto 2D PCA map with 91% energy preserved

In addition to irregular shapes and different densities, time series themselves have variations such as phase perturbations and random noise, which also impacts clustering accuracy. The combination of L_1 distance and multi-density based clustering makes YADING robust to time series variations, i.e., phase perturbations and random noise.

Two intuitions help understand the aforementioned robustness. One is that, if two time series instances have small phase differences, then their L_1 distance is also small. The other is that, as the size of the input dataset increases, two time series instances with large phase difference can still be clustered together by density-based clustering, because they can be connected by a series of intermediate time series instances each with small phase shift.

In order to obtain quantitative analysis results, we provide theoretical bound on the tolerance to phase perturbations and noise. We formalize the problem of bound estimation as finding the relationship among three variables: the data size *n*, the upper bound Δ of phase perturbation, and the probability *P* corresponding to the event: {time series instances with phase perturbation less than Δ are successfully clustered together}. Based on such relationship, Δ can be determined given *n* and *P*. Next, we first define some notations and then present the lemma followed by proof.

Denote a time series $T(a) = \{f(a+b), \dots f(a+mb)\}$, where a is the initial phase, b is the time interval at which time series are sampled, and m is the length. $T(a)$ is generated by an underlying continuous function $f(t)$. Here, we assume $f(t)$ is an *analytic function*. Another time series with phase perturbation δ is represented by $T(a-\delta) = \{f(a+b-\delta), \dots f(a+mb-\delta)\}$. Now suppose we have n time series $T(a-\delta_i)$, which only differ from $T(a)$ by a phase perturbation δ_i. Without loss of generality, let $\delta_i \in [0, \Delta]$. We assume that these time series are generated independently, with δ_i uniformly distributed in the interval $[0, \Delta]$. Denote the distance threshold in the density-based clustering as ε. If the distance between a specific object and its kth Nearest Neighbor (kNN) is smaller than ε, then it is a core point (same definition as [17]). Denote event:

$$E_n := \{T(a-\delta_i), i = 1, 2, \dots n.\ belong\ to\ same\ cluster\ \}.$$

Lemma 3: $P(E_n) \geq 1 - n(1 - \frac{\varepsilon}{mMk\Delta})^n$

Proof: Step 1, we prove the following inequality:

$$\exists M, s.t., L_1(T(a), T(a-\delta)) := \sum_{i=1}^{m} |f(a+ib) - f(a+ib-\delta)| \leq mM\delta$$

It indicates that the L_1 distance can be arbitrarily small as long as the phase perturbation is small enough. The detailed proof is provided at our project website [35] due to page limit.

Step 2, we divide the interval $[0, \Delta]$ into buckets with length equal to $\frac{\varepsilon}{mMk}$. According to the mechanism of DBSCAN, if each bucket contains at least one time series, then $L_1(T(a), its\ kNN) \leq k \times mM \times \frac{\varepsilon}{mMk} = \varepsilon$. Therefore, all the objects become core points, and they are density-connected [17]. Hence, all the time series are grouped into one cluster.

Step 3, denote event $U_j := \{j^{th} bucket\ is\ empty\}$, then

$P(at\ least\ one\ bucket\ is\ empty) = P(\bigcup_j U_j) \le \sum_j P(U_j) = n(1-\frac{\varepsilon}{mMk\Delta})^n$. Note that event $\{no\ empty\ bucket\}$ is a subset of E_n, so $P(E_n) \ge P(no\ empty\ bucket) = 1 - P(\bigcup_j U_j) \ge 1 - n\left(1-\frac{\varepsilon}{mMk\Delta}\right)^n$. Hence the lemma is proven.

The theoretical analysis on the robustness to random noise is available at [35] due to page limit.

3.2.2 Density Estimation

Density estimation is key to density-based clustering algorithms. It is performed either manually or with slow performance in most of the existing algorithms [17][23][24]. In this section, we define a concept *density radius* and provide theoretical proof on its estimation. We use density radius in YADING to identify the core points of the input dataset and conduct multi-density clustering accordingly.

We define k_{dis} of an object as the distance between this object and its kNN. A k_{dis} curve is a list of k_{dis} values in descending order. Figure 2 shows an example of k_{dis} curve with $k = 4$. We define density radius as the most frequent k_{dis} value. Intuitively, most objects contain exactly k nearest neighbors in a hyper-sphere with radius equal to *density radius*.

We transform the estimation of density radius to the identification of the inflection point on k_{dis} curve. Here, inflection point takes general definition of having its second derivative equal to zero. Next, we provide the intuition behind this transformation followed by theoretical proof.

Intuitively, the local area of an inflection point on k_{dis} curve is the flattest (i.e. the slopes on its left-hand and right-hand sides have the smallest difference). On k_{dis} curve, the points in the neighborhood of a inflection point have close values of k_{dis}.

For example, in Figure 2, there are three inflection points with corresponding k_{dis} values equal to 800, 500, and 200. In other words, most points on this curve have k_{dis} values close to 800, 500, or 200. According to the definition of density radius, these three values can be used to approximate three density radiuses.

We now provide theoretical analysis on estimating density radius by identifying the inflection point on k_{dis} curve. We first prove that the Y-value, k_{dis}, of each inflection point on the k_{dis} curve equals to one unique density radius. Specifically, given a dataset with single density, we provide an analytical form to its k_{dis} curve, and prove that there exists a unique inflection point with Y-value equal to the density radius of the dataset. We further generalize the estimation to the dataset with multiple densities.

To make the mathematical deduction easier, we use $EDF_k(r) := \frac{|\{objects\ whose\ k_{dis} \le r\}|}{N}$ to represent k_{dis} curve equivalently. EDF is short for Empirical Distribution Function. It is the k_{dis} curve rotated 90 degrees clockwise with normalized Y-Axis. The X-value of inflection point on $EDF_k(r)$ equals to the Y-value of inflection point on k_{dis} curve.

Lemma 4: $EDF_k(r) \approx \sum_{m=k+1}^{N} P(E_{m,r})$, where $P(E_{m,r}) = C_{N-1}^{m-1} P_r^{m-1} (1-P_r)^{N-m}$, $P_r = \frac{V_r}{V} = \frac{c_d \times r^d}{V}$, C_n^k is the notation of combination, and $V_r := c_d \times r^d$, is the volume of the hyper-sphere in the d-dimensional L_p space, e.g., in Euclidean space, $c_d = \frac{\pi^{d/2}}{\Gamma(\frac{d}{2}+1)}$.

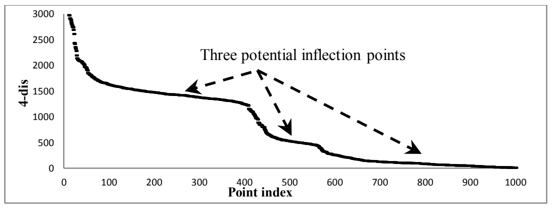

Figure 2. 4-dis curve of a time series dataset

Lemma 4 provides an analytical form of the $EDF_k(r)$ of a dataset with single density, which implies the existence of inflection point (the detailed proof is available at [35]).

Lemma 5: Y-value of inflection point on k_{dis} curve is density radius.

Proof: Denote r_i as the X-value of inflection point of $EDF_k(r)$, so $\frac{d^2 EDF_k(r)}{dr^2}|_{r=r_i} = 0$ by definition. Recall that the first derivation of $EDF_k(r)$ is the probability density function, so the second derivation equal to zero indicates that $r = r_i$ has the maximum likelihood. In other words, r_i is the most frequent value of k_{dis}, which conforms to the definition of density radius. Since the X-value of inflection point on $EDF_k(r)$ equals to the Y-value of inflection point on k_{dis} curve, the lemma is proven.

Based on lemma 4 and 5, we generalize the estimation to datasets with multiple densities. For a dataset with *l* different density radiuses, there exist *l* corresponding inflection points on the k_{dis} curve.

In our implementation, we first find the inflection point with the minimum difference between its left and right-hand slopes. We then recursively repeat this process on the two sub-curves segmented by the obtained inflection point, until no more significant (a pre-defined threshold to indicate the significance) inflection points are found (Table 2).

Table 2. Algorithm for estimating density radiuses

```
Function 1   DENSITYRADIUSES($\boldsymbol{k_{dis}}$)
                 length ← |$\boldsymbol{k_{dis}}$|
                 allocate res as list
                 INFLECTIONPOINT($\boldsymbol{k_{dis}}$, 0, length, res)
                 return res
Function 2   INFLECTIONPOINT ($\boldsymbol{k_{dis}}$, s, e, res)
                 r ← -1, diff ← -1
                 for i ← s to e
                     left ← SLOPE ($\boldsymbol{k_{dis}}$, s, i)
                     right ← SLOPE ($\boldsymbol{k_{dis}}$, i, e)
                     if left or right greater than threshold1
                         continue
                     if |left - right| smaller than diff
                         diff ← |left-right|
                         r ← i^th element of $\boldsymbol{k_{dis}}$
                 if diff smaller than threshod2 /*record the
             inflection point, and recursively search*/
                     add r to res
                     INFLECTIONPOINT ($\boldsymbol{k_{dis}}$, s, r-1)
                     INFLECTIONPOINT ($\boldsymbol{k_{dis}}$, r+1, e)
```

The time complexity of estimating density radiuses is as follows. The generation of k_{dis} curve costs $O(ds^2)$ due to the calculation of distance between each pair of objects on the sampled dataset. Multi-density estimation costs $O(s \log s)$ since it adopts the divide-and-conquer strategy.

3.2.3 Clustering

Once we obtain the density radiuses, the clustering algorithm is straightforward. With each density radius specified, from the smallest to the largest, DBSCAN is performed accordingly (Table 3). In our implementation, we set $k = 4$, which is the MinPts value in DBSCAN. The complexity of DBSCAN is $O(s \log s)$. Since it is performed at most s times, the total complexity is $O(s^2 \log s)$.

Table 3. Algorithm for multi-density based clustering

```
/* p: the sample data set.
   radiuses: the density radiuses. */
MULTIDBSCAN(p, radiuses)
   for each radius ∈ radiuses
      objs ← cluster from DBSCAN(p, radius)
      remove objs from p
   mark p as noise objects
```

3.3 Assignment

After clustering is performed on the sampled dataset, a cluster label needs to be assigned to each unlabeled time series instance in the input dataset. Intuitively, the assignment process involves computing the distance between every pair of unlabeled and labeled instances, which has complexity $O(Nsd)$. Such complexity is the highest among all the steps. Therefore, it is important to design an efficient assignment strategy to significantly reduce the time cost of this step, which further improves the overall performance of YADING. In this section, we propose an efficient pruning strategy as well as the corresponding data structure design, which aim to save unnecessary distance computation between the unlabeled time series instances and the labeled instances.

Assignment criteria. For an unlabeled instance, find its closest labeled instance. If their distance is less than the density radius of the cluster that the labeled instance belongs to, then the unlabeled instance is considered to be in the same cluster as the labeled instance. Otherwise, it is labeled as noise.

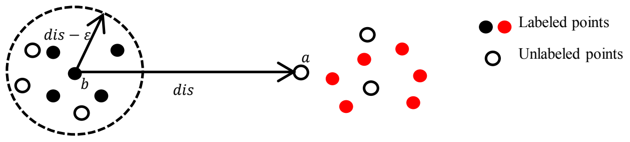

Figure 3. Illustration of the pruning strategy in assignment

Let us use an example to illustrate our pruning strategy. In Figure 3, there are a set of labeled and unlabeled points. If an unlabeled object a is far from a labeled object b, i.e. their distance dis is greater than the density radius ε of b's cluster, then the distances between a and the labeled neighbors of b (within $dis - \varepsilon$) are also greater than ε (according to triangle inequality). Therefore, the distance computation between a and each of b's neighbors can be saved.

We design a data structure named Sorted Neighbor Graph (SNG) to achieve the above pruning strategy. When performing density-based clustering on the sampled dataset, if an instance b is determined to be a core point, then b is added to SNG, and its distances to all the other instances in the sampled dataset are computed and stored in SNG in ascending order. Quick-sort is used in the construction of SNG, so the time complexity of SNG is $O(s^2 \log s)$. Such pruning can improve the performance by 2~4 times in practice depending on data distribution, which reduces time cost significantly. In the worst case, $O(Nsd)$ is still needed. Table 4 shows the detailed algorithm.

Table 4. Algorithm for assignment

```
/* uObj: the list of unlabeled objects*/
ASSIGNMENT(SNG, uObj)
   for each obj ∈ uObj
      set the label of obj as "noisy"
      for each o ∈ {keys of SNG}
         if o has been inspected
            continue;
         dis ← L1 distance between o and obj
         if dis less than density radius of o
            mark obj with same label of o
            break
         mark o as inspected
         jump ← dis - density radius of o
         i ← BINARYSEARCH(SNG[o], jump)
         for each neighbor ∈ SNG[o] with index greater than i
            if density radius of neighbor is less than jump
               mark neighbor as inspected
            else break /*this is a sorted list*/
```

3.4 Time Complexity

Based on the discussion on time complexity in the previous subsections, the overall time complexity of YADING is $O(Nsd + ND + s^2 \log s + sD \log D)$. Considering $s \leq N, d \leq D$ and the independence of s with respect to N (Section 3.1.1), in practice, the last two items are much smaller than the first two when N is large. When the last two items are ignored, our algorithm is approximately linear to the size of the input time series dataset, and the length of each time series instance.

4. EVALUATION

In this section, we evaluate YADING on both simulation datasets and one real dataset to answer the following research questions.

RQ1. How efficiently can YADING cluster time series data?

RQ2. How does sample size affect the clustering accuracy?

RQ3. How robust is YADING to time series variation?

4.1 Setup

Hardware Our experiments were conducted on a machine with 2.4 GHz Intel Xeon E5-2665 processor and 128 GB RAM. To set the comparison ground, single-thread implementation is used for YADING and the four comparison clustering algorithms.

Datasets We first utilize simulation data in order to perform the evaluation on datasets with large scale and ground-truth labeling. We use five different stochastic models (AR(1), Forced Oscillation, Drift, Peak, and Random Walk [35]) to generate the simulation time series data. These models cover a wide range of characteristics of time series. Using these stochastic models, we create data generation templates to generate simulation datasets with configurable parameters N and D, representing the size of the dataset and the length of each time series, respectively. Specifically, we use Template A for RQ1 and RQ2. For RQ3, we use Template A and Template B to evaluate against random noise and phase perturbation, respectively.

Each template consists of a set of time series groups. Each group is generated by a randomly selected model with arbitrarily chosen parameters. Represented by the population ratio, the group size varies significantly. For example, Template A consists of 15 groups, with group size ranging from 0.1% to 30%. According to the requirements of each evaluation, the datasets are generated accordingly with given *N* and *D*. Both the templates and the simulation datasets are available at [35].

In addition, we select StarLightCurves from the UCR library [34] as the real dataset. StarLightCurves is the largest-scale (data_size * dimensionality) time series dataset (Table 5) in the UCR library. This dataset is also labeled with clustering ground truth. More real datasets are available at [35]. Our results on these datasets are similar to those on StarLightCurves in terms of performance and clustering quality.

Table 5. One real large-scale dataset

Name	# clusters	data size	dimensionality
StarLightCurves	3	9,236	1,024

Measures We use execution time to evaluate performance. We use two widely used measures to evaluate clustering accuracy. One is Normalized Mutual Information (NMI) [31], which incorporates both precision and recall into one score in [0, 1]. The value of NMI equals 1 when the clustering result exactly matches the labels, and 0 when the result is completely irrelevant. The other accuracy measure is NCICluster which is the number of correctly identified clusters.

Comparison algorithms We select DENCLUE 2.0, DBSCAN, and CLARANS as end-to-end comparison algorithms. In addition, we compare YADING's multi-density estimation component against DECODE [24].

DENCLUE 2.0 is a highly efficient density-based clustering algorithm. It improves the efficiency of DENCLUE 1.0 by adopting a fast hill-climbing procedure and random sampling to accelerate the computation. DBSCAN is a widely used density-based clustering algorithm. CLARANS is an efficient partitioning-based clustering algorithm. We implement these three comparison algorithms carefully by following the corresponding instructions, pseudo-code and parameter recommendations. The source code of the three algorithms is available at [35]. For DENCLUE 2.0, we use the same sample size as used in YADING, since no guidelines are provided on how to specify sample size. For DBSCAN, we set $MinPts$ to 4 (same as YADING's setting) and ε as the best-estimated density radius (since DBSCAN only supports single density, we set it to the most significant density radius used in YADING's setting). CLARANS requires the number of clusters as input, so we set it to the ground-truth value for each simulation dataset.

It should be noted that, DENCLUE 2.0 typically deals with relatively low-dimensional datasets. The dimensionality is 16 for the evaluation dataset in [36]. When evaluated on our high-dimensional datasets, e.g., *D*=1,024, both its performance and clustering quality are far from satisfactory when using the default parameters. Therefore, we carefully tuned several key parameters to make DENCLUE 2.0 suitable to dealing with high-dimensional datasets. The corresponding details are available at [35].

DECODE is the most recent algorithm for estimating multiple densities of a spatial dataset. It models the multiple densities as a mixture of point processes, and conducts multi-density estimation by constructing a Bayesian mixture model and using the Markov Chain Monte Carlo (MCMC) algorithm. We use the source code of DECODE provided by the authors, and set the parameters according to the recommendations in [24].

4.2 Results

We present the evaluation results for the three research questions in this section.

4.2.1 Results of RQ1

Using the aforementioned Template A in Section 4.1, 10 simulation datasets are generated with dimensionality $D = 10^3$ and size N ranging from 10^4 to 10^5; and 5 datasets with $N = 10^5$ and D ranging from 64 to 1,024. The performance results of YADING, DENCLUE 2.0, DBSCAN, and CLARANS against *N* and *D* on the simulation datasets are shown in Figure 4. Note that logarithmic scale is used on the Y-Axis in both charts. The comparison indicates that across all the datasets with different scales, YADING is about 1-order-of-magnitude faster than DENCLUE 2.0, and 3-order-of-magnitude faster than DBSCAN and CLARANS. Another observation is that YADING's performance is approximately linear to the size of input dataset and dimensionality, which is consistent with our time complexity estimation in Section 3.4.

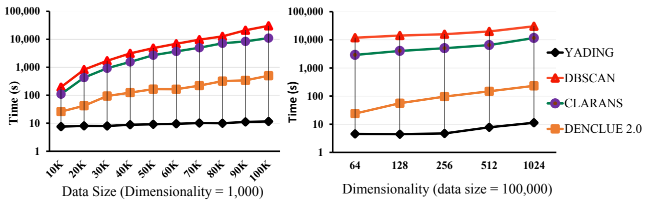

Figure 4. Performance comparison against data size *N* (left) and dimensionality *D* (right)

To further demonstrate YADING's high performance, we evaluate it on a simulation dataset with 1 million time series and dimensionality equal to 2,000. The execution time of YADING is 91.7 seconds, demonstrating consistent linearity to *N* and *D*.

The performance results of YADING, DENCLUE 2.0, DBSCAN, and CLARANS on StarLightCurves are shown in Table 6. Although the scale of StarLightCurves is much smaller than that of our simulation datasets, YADING still significantly outperforms the other three algorithms.

The NMI results in Table 6 show that YADING's accuracy on StarLightCurves is comparable to that of DBSCAN and CLARANS. The reason why the NMI of the three algorithms (except DENCLUE 2.0) is not high is because of the specific distribution of this dataset. The time series instances with cluster label 1 and 3 are relatively close. Therefore, all three algorithms group these instances together into one cluster.

We use the aforementioned 15 datasets to evaluate the multi-density estimation. The densities estimated by YADING and DECODE are based on the same sampled sub-datasets, with sample size equal to 2,000. The different densities are represented by *density radiuses* (Section 3.2.2). Here, the corresponding parameter *k* is set to 4 for both YADING and DECODE (note that this is also the default value in DECODE).

Table 6. Evaluation results on StarLightCurves

Algorithm / Measure	YADING	DENCLUE 2.0	DBSCAN	CLARANS
Performance (second)	3.1	28.6	145.1	91
NMI	0.60	0.23	0.55	0.61

Table 7. Multi-density estimation results on Template A

How Found	Number of Densities
Ground truth	58
YADING	55
DECODE	40
Found by both YADING and DECODE	37

We use the number of different *sharp changes* on the k_{dis} curve as ground truth for the number of densities. As shown in Table 7, there are 58 densities in total; YADING identifies 55, and DECODE identifies 40. In addition, YADING identifies most (37 out of 40) of the densities identified by DECODE. The detailed results on each individual dataset is available at [35]. Regarding the performance of density identification, the average time cost for running the multi-density estimation module in DECODE is more than 500 seconds, while the corresponding time cost of YADING is less than 100ms.

4.2.2 Results of RQ2

In order to evaluate YADING's clustering accuracy with varied sample size, we generate two datasets with $\{N = 10^4, D = 10^3\}$ and $\{N = 10^5, D = 10^3\}$, respectively, based on Template A. We conduct clustering on these datasets with different sample size s, and repeat this experiment 100 times. The average NMI of clustering and the average NCICluster are shown in Table 8 and 9, respectively.

Table 8. NMI with different sample sizes

Sample Size / Data Set	200	500	1K	2K	5K	Full
DS1 (10K)	0.915	0.932	0.948	0.952	0.956	0.965
DS2 (100K)	0.857	0.919	0.939	0.943	0.946	0.960

When the sample size $s = 200$ and $s = 500$, although the corresponding NMI scores are not significantly lower than those with $s \geq 10^3$ (Table 8), the corresponding numbers of correctly identified clusters are much lower (Table 9). In fact, 8-out-of-15 and 6-out-of-15 groups are missed on Dataset 1 when $s = 200$ and $s = 500$, respectively, and these missed groups include all the groups with $< 1\%$ population. When $s = 10^3$, the 4 smallest groups are missed, but the other two small groups, with 0.5% and 0.8% population ratio, respectively, are preserved in final clustering result. Further inspection reveals that, the 4 unidentified groups have fewer than 5 instances in the sampled dataset (based on their population ratio). These results are consistent with our theoretical analysis, and they demonstrate the effectiveness of the lower bound of sample size (1,030) as proved in Section 3.1.1.

The NMI score of $s = 2{,}000$ is close to that of $s > 2{,}000$ (Full means no sampling in Table 8 and 9), and NCIClusters are similar when $s \geq 2{,}000$. This indicates that when the sample size exceeds 2,000, there is little improvement on the clustering accuracy by adding more samples. This result is consistent with the concept of upper bound on the sample size discussed in Section 3.1.1. Regarding the value of the upper bound, 2,000 is a good choice as shown in the evaluation results. Although the theoretical value is 9,600 with 1% deviation on the population ratio (Section 3.1.1), in practice, this upper bound may be set lower as suggested in this evaluation.

Table 9. NCICluster with different sample sizes

Sample Size / Data Set	200	500	1K	2K	5K	Full
DS1 (10K)	7.0 ±0.7	8.9 ±0.3	10.6 ±0.1	11.3 ±0.1	11.6 ±0.1	12
DS2 (100K)	6.3 ±0.6	8.7 ±0.4	10.3 ±0.1	11.3 ±0.1	11.3 ±0.1	12

4.2.3 Results of RQ3

We evaluate the robustness of YADING to random noise and phase perturbation, respectively. We perform clustering using YADING, DENCLUE 2.0, DBSCAN and CLARANS on the 15 datasets generated in Section 4.2.1 (10 datasets with varied *N* values, and 5 datasets with varied *D* values). Since all the 15 datasets have random noise added to each of its time series data during generation, these datasets are suitable for evaluating YADING's robustness against random noise. For each clustering method, we compute its clustering accuracy as the average NMI score across the 15 datasets. The results in Table 10 show that YADING achieves high clustering accuracy (0.925) with small variance (0.027) on the datasets, thus demonstrating its robustness to random noise. In addition, YADING's average accuracy is higher than that of DENCLUE 2.0, DBSCAN and CLARANS on the 15 datasets.

Table 10. Average clustering accuracy on 15 datasets

Algorithm / Measure	YADING	DEN-CLUE 2.0	DBSCAN	CLARANS
NMI	0.925± 0.027	0.523± 0.057	0.820± 0.071	0.804± 0.018

We use Template B to evaluate YADING's robustness against phase perturbation. Template B consists of 8 time series groups. Two of the 8 groups are generated using the forced oscillation model. We change the phase parameter β in the model to create phase perturbation. Two datasets are created based on Template B with $\{N = 10^4, D = 10^3\}$ and $\{N = 10^5, D = 10^3\}$, respectively. The detailed information of Template B and the corresponding datasets can be found in [35].

On both datasets, YADING achieves high clustering accuracy, 0.982 and 0.988, respectively; and it correctly groups the two sets of time series with phase perturbation. Figure 5 plots the time series instances in one of the groups identified by YADING. Using the black curve (with no phase perturbation and random noise) as reference, the amount of phase perturbation and random noise is illustrated.

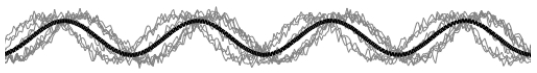
Figure 5. Time series instances with phase perturbation and random noise grouped by YADING

5. APPLICATIONS IN PRACTICE

Based on YADING, we have built an interactive multi-dimensional analysis tool for time series. With easy-to-use user interface, our tool allows users to group large-scale time series data on the fly. The grouping results transform time series into categorical attributes, which enables users to analyze time series data along with other categorical attributes.

Engineers from several online-service teams at Microsoft have used our tool to analyze time series data related to service performance. We present two usage examples below to demonstrate how YADING helps with analysis in practice via its high performance.

5.1 Clustering Performance Counters

Team A manages a large number of servers across several datacenters. Performance counters are an important data source for Team A to monitor, understand, and trouble-shoot performance issues of servers and services. Such performance counters include hardware and OS level metrics, e.g., CPU and memory utilization; as well as application level metrics, e.g., response time of service requests and request throughput, etc.

Hundreds of performance counters are continuously collected at each server. For analysis purpose, the data for each type of performance counter is often aggregated at pre-defined time interval, e.g., 5 minutes, thus resulting in a time series instance. Due to the large number of servers, the dataset of a performance counter usually contains tens of thousands of time series instances. Typically, the analysis window varies from hours to days or a week. If the aggregation time interval is 5 minutes, then the length (i.e. dimensionality) of each time series is in the range of (~100, ~1,000).

One dataset that Team A loaded into our tool consists of two types of data for a server: two time series datasets and one categorical attribute named ClusterID (servers are organized in clusters in datacenters). The time series datasets represent CPU and memory utilization, respectively, for 46,000 servers; and each time series instance has 1,008 data points. The analysis task is to obtain an overall understanding on the CPU and memory utilization across all the servers, and find out how these two aspects relate to each other.

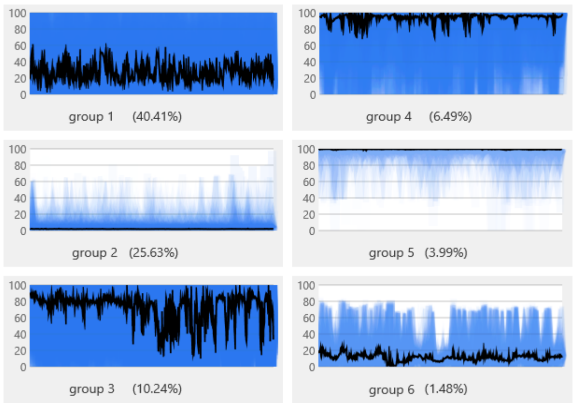

Figure 6. Top groups of CPU utilization

After the above dataset is loaded into our tool, 21 groups of CPU utilization are immediately obtained in about 4.5 seconds. The major groups with population ratio greater than 1% are shown in Figure 6. In the chart for each group, all the time series instances are plotted in blue. The single black curve is the median of the group, (i.e. median is used to aggregate all the time series instances in the group). Such design of visualization provides an intuitive understanding of the group characteristics.

The above groups reveal different patterns of CPU utilization of different servers. For example, servers in group 2 have very low CPU utilization; servers in group 1 and 6 have medium CPU workload; and servers in group 3, 4 and 5 have high CPU usage with different patterns.

YADING's high-performance clustering algorithm enables real-time and interactive drill-down analysis in our tool. In this analysis task, following the observation on the CPU utilization groups, one question worth exploring is why the CPU usage is so low on about a quarter of servers (group 2, 25.63%). The servers in group 2 may then be pivoted on other attributes, e.g., memory utilization. When doing so, the time series of memory utilization for the servers in group 2 are grouped immediately in our tool. Please note that this clustering is performed on-demand, and it is performed in real-time due to the high performance of YADING.

The above analysis task can be carried down further with more attributes such as topology information, job information, etc., to reveal where the servers in group 2 are located, and how computation tasks are scheduled to them, etc. This will help quickly gain insights and take appropriate actions to improve the resource allocation.

5.2 Clustering Error Distributions

Team B runs a cloud service that spans multiple datacenters, and the team closely monitors the health status of their servers. For this purpose, one important type of data they collect and analyze is called "error distribution". Specifically, Team B defines a set of error events, and records the occurrences of the events that are associated with the state change of server health.

An error distribution is a discrete curve $Occur(e_i)$ to represent the error pattern of a server within a time window, where e_i is an error type, and the value is the occurrences of e_i within a given time window. The error types are listed in a fixed order in order for the error distribution to be well defined. Typically, there are about 100 error types and 10,000 Machines; the time window is usually set to 24 hours.

Error distribution is an important data source for diagnosing service performance issues since it can provide clues on problematic servers or service components. Team B lacked effective methods to quickly analyze error distribution data at scale. In order to evaluate whether our tool is capable to meet their requirements, the engineers in Team B selected a historical dataset and loaded it into our tool. This dataset consisted of 1-day error distribution data from a large number of servers. Some of the servers experienced a performance problem. Our tool clustered all the error distribution curves into 16 groups in less than 1 second. Two example groups are shown in Figure 7 (the data scale and detailed error information are skipped for confidentiality reason).

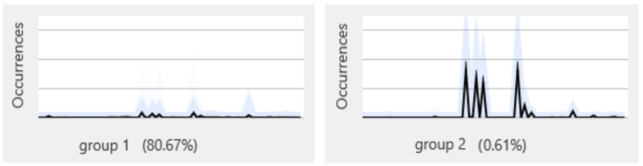

Figure 7. Two example groups generated by YADING

Combining the grouping results with their domain knowledge on error distribution, the engineers of Team B easily found out that group 1 was relatively healthy, and group 2 might have experienced problems. They further drilled down into group 2 by pivoting on other attributes of the servers. When pivoting on the topological attribute, they found out that all the servers in group 2 belong to server cluster X. When drilling down one step further, they identified that there were versioning issues for the software installed on those servers. These insights helped the engineers rapidly narrow down the investigation scope to cluster X and the particular version of service components. Using our tool, it took only several minutes for the engineers to complete the analysis process . In comparison, they spent almost a full day to obtain these findings when they diagnosed this performance issue in the past.

Team B was satisfied with our tool due to its effectiveness and its highly interactive experience enabled by the high performance of YADING. Team B started to consider applying YADING in more diagnosis scenarios, e.g., constructing a knowledgebase for historical performance issues. In this scenario, large amount of error distribution data related to past performance issues would be grouped by YADING. The resultant groups could be labeled with description and resolution to solving the corresponding problems. Such knowledgebase can be used to reduce the diagnosis effort when new suspicious error distribution occurs.

6. DISCUSSION

The current implementation of YADING is single-threaded. We use the performance of this implementation to compare with other clustering methods, and as the baseline for us to further improve YADING's performance. From the discussion of YADING's algorithm in Section 3, it is obvious that YADING's implementation can be easily parallelized, especially the assignment step which is most expensive. Thus, YADING's performance can be significantly improved through parallelization.

Locality-Sensitive Hashing (LSH) [32] can be used to speed up the assignment step of YADING. Considering that the assignment step can be easily parallelized and the potential impact on clustering accuracy brought by LSH, we will evaluate the practical benefit of employing LSH as part of our future work.

iSAX [37] is a technique used for indexing time series data. It discretizes the real values at the epochs of time series instances into symbols, and obtains a compact representation that significantly reduces the storage size, and improves the query performance. However, the cardinality b, which indicates the granularity of the discretization, needs to be manually specified. In addition, as iSAX is a lossy compression, it could potentially bias the density estimation. Therefore, we did not utilize such indexing techniques in our approach.

We utilize the straightforward pair-wise calculation to obtain the kNN when conducting density estimation during multi-density clustering. We also tried other sophisticated data structures such as R^*tree [26] to speed up this computation, and we found that the performance is worse due to the high dimensionality of the time series data. A thorough analysis on how dimensionality impacts the performance of such techniques can be found in [33].

Currently, YADING requires that the input time series have equal lengths. In the real applications discussed in Section 5, the equal-length property is guaranteed by the underlying data pipeline. In general, it is common for datasets to have equal-length time series. In order to handle time series with variable lengths, preprocessing is needed before data is input into YADING. One possible technique is to apply down-sampling to align the lengths of time series. Specifically, down-sampling by integer factor (retaining only every m^{th} sample creates a new time series) is simple and efficient with trade-off on accuracy. More sophisticated sampling techniques, such as interpolation filter, incur smaller error with higher computation cost.

We have demonstrated that by selecting L_1 distance as similarity measure combined with multi-density based clustering, YADING achieves high effectiveness in clustering. In addition, the framework of YADING can also support other types of similarity measures, as long as those measures satisfy the mathematical properties of metric: non-negativity, symmetry, and triangularity. This is because YADING adopts multi-density estimation, and the concept of *density* is meaningful only in a metric space. The general L_p norm can be adapted into YADING's framework almost seamlessly, and it only needs a modification of the volume coefficient of the d-dimensional hyper-sphere used for multi-density estimation, which can be found in mathematical handbooks. Some similarity measures used in model-based clustering can also be adapted into the framework of YADING. For example, Piccolo [29] proposed the Euclidean distance on cepstral coefficients derived from the ARIMA model parameters. To adopt YADING, preprocessing is needed to transform each time series instance into the model-parameter space. Some other similarity measures, such as DTW [7] and Pearson correlation [6], cannot be supported by YADING easily, because they are not mathematical metrics.

Most steps of YADING can be adapted to handle streaming data except PAA (Section 3.1.2). The calculation of L_1 distance between two time series can naturally be updated incrementally. A distance matrix is needed to record the L_1 distance between all pairs of time series instances, which provides sufficient information for estimating densities, performing multi-density based clustering, and conducting assignment. Apparently, the memory consumption and execution time are constant with respect to the length of time series, which complies with the efficiency requirement of data streams clustering [38]. The dimensionality reduction step is difficult to adapt to handle data streams, because the auto-correlation estimation cannot be updated incrementally. We leave it to the future work of YADING to design a dimensionality reduction algorithm suitable for streaming data.

7. CONCLUSION

We present YADING, a novel end-to-end clustering algorithm that automatically clusters large-scale time series with fast performance and quality results. The theoretical proof on the sampling bounds guarantees that YADING's time complexity is linear to the scale of input datasets. When selecting L_1 distance as similarity measure and the multi-density approach as the clustering method, we provide theoretical bound to ensure YADING's robustness to phase perturbation and random noise of time series. We also report how YADING is used in practice to help analyze service monitoring data with its high performance.

8. ACKNOWLEDGEMENTS

We thank the anonymous reviewers for their valuable comments. We thank Jian-Guang Lou, Qianchuan Zhao, Yaowei Zhu, Fei Liu, and Shulei Wang for their helpful discussions on this work. We also thank our partners in Microsoft product teams who used our tool and provided valuable feedback. We thank Tao Pei for sharing with us the source code of DECODE.

9. REFERENCES

[1] Debregeas, A., and Hebrail, G. 1998. Interactive interpretation of Kohonen maps applied to curves. *In Proc. of KDD'98*. 179-183.

[2] Derrick, K., Bill, K., and Vamsi, C. 2012. Large scale/big data federation & virtualization: a case study.

[3] D. A. Patterson. 2002. A simple way to estimate the cost of downtime. In *Proc. of LISA' 02*, pp. 185-188.

[4] Eamonn, K., and Shruti, K. 2002. On the need for time series data mining benchmarks: a survey and empirical demonstration. In *Proc. of KDD'02*, July 23-26.

[5] T. W. Liao. 2005. Clustering of time series data—A survey *Pattern Recognit*., vol. 38, no. 11, pp. 1857–1874, Nov.

[6] X. Golay, S. Kollias, G. Stoll, D. Meier, A. Valavanis, P. Boesiger. 1998. A new correlation-based fuzzy logic clustering algorithm for fMRI, *Mag. Resonance Med*.

[7] B. K. Yi, H. V. Jagadish, and C. Faloutsos. 1998. Efficient retrieval of similar time sequences under time warping. In *IEEE Proc. of ICDE*, Feb.

[8] J. Han, M. Kamber. 2001. Data Mining: Concepts and Techniques, Morgan Kaufmann, San Francisco, pp. 346–389.

[9] Chu, K. & Wong, M. 1999. Fast time-series searching with scaling and shifting. In *proc. of PODS*. pp 237-248.

[10] Popivanov, I. & Miller, R. J. 2002. Similarity search over time series data using wavelets. In *proc. of ICDE.*

[11] Keogh, E., Chakrabarti, K., Pazzani, M. & Mehrotra, S. 2001. Locally adaptive dimensionality reduction for indexing large time series databases. In *proc. of ACM SIGMOD*.

[12] Yi, B. & Faloutsos, C. 2000. Fast time sequence indexing for arbitrary lp norms. In *proc. of the VLDB*. pp 385-394.

[13] E. J. Keogh, K. Chakrabarti, M. J. Pazzani, and S. Mehrotra. 2001. Dimensionality Reduction for Fast Similarity Search in Large Time Series Databases. *Knowl. Inf. Syst.*, 3(3).

[14] G. Kollios, D. Gunopulos, N. Koudas, and S. Berchtold. 2003. Efficient biased sampling for approximate clustering and outlier detection in large datasets. *IEEE TKDE*, 15(5).

[15] Zhou, S., Zhou, A., Cao, J., Wen, J., Fan, Y., Hu. Y. 2000. Combining sampling technique with DBSCAN algorithm for clustering large spatial databases. In *Proc. of the PAKDD.*

[16] Stuart, Alan. 1962. Basic Ideas of Scientific Sampling, Hafner Publishing Company, New York.

[17] M. Ester, H. P. Kriegel, and X. Xu. A density-based algorithm for discovering clusters in large spatial databases with noise. *ACM SIGKDD 1996*, pages 226–231.

[18] M. Steinbach, L. Ertoz, and V. Kumar. 2003. Challenges of clustering high dimensional data. In L. T. Wille, editor, *New Vistas in Statistical Physics – Applications in Econophysics, Bioinformatics, and Pattern Recognition*. Springer-Verlag.

[19] Ng R.T., and Han J. 1994. Efficient and Effective Clustering Methods for Spatial Data Mining, In *proc. of VLDB*.

[20] S. Guha, R. Rastogi, K. Shim. 1998. CURE: an efficient clustering algorithm for large databases. *SIGMOD*.

[21] W. Wang, J. Yang, R. Muntz, R. 1997. STING: a statistical information grid approach to spatial data mining, *VLDB'97*.

[22] Hans-Peter Kriegel, Peer Kröger, Jörg Sander, Arthur Zimek 2011. Density-based Clustering. WIREs Data Mining and Knowledge Discovery 1 (3): 231–240. doi:10.1002/widm.30.

[23] Mihael Ankerst, Markus M. Breunig, Hans-Peter Kriegel, Jörg Sander. 1999. OPTICS: Ordering Points To Identify the Clustering Structure. ACM SIGMOD. pp. 49–60.

[24] Tao Pei, Ajay Jasra, David J. Hand, A. X. Zhu, C. Zhou. 2009. DECODE: a new method for discovering clusters of different densities in spatial data, Data Min Knowl Disc.

[25] C. Guo, H. Li, and D. Pan. 2010. An improved piecewise aggregate approximation based on statistical features for time series mining. KSEM'10, pages 234–244.

[26] Beckmann, N.; Kriegel, H. P.; Schneider, R.; Seeger, B. 1990. "The R*-tree: an efficient and robust access method for points and rectangles". In *proc. of SIGMOD*. p. 322.

[27] X. Golay, S. Kollias, G. Stoll, D. Meier, A. Valavanis, P. Boesiger, A new correlation-based fuzzy logic clustering algorithm for fMRI, Mag. Resonance Med. 40 (1998).

[28] T.W. Liao, B. Bolt, J. Forester, E. Hailman, C. Hansen, R.C. Kaste, J. O'May, Understanding and projecting the battle state, 23rd Army Science Conference, Orlando, FL, 2002.

[29] D. Piccolo,A distance measure for classifying ARMA models, J. Time Ser. Anal. 11 (2) (1990) 153–163.

[30] D. Tran, M. Wagner, Fuzzy c-means clustering-based speaker verification, in: N.R. Pal, M. Sugeno (Eds.), AFSS 2002, Lecture Notes in Artificial Intelligence, 2275, 2002.

[31] Danon L, D´ıaz-Guilera A, Duch J and Arenas A 2005 J. Stat. Mech. P09008.

[32] M. Datar, N. Immorlica, P. Indyk, V.S. Mirrokni. 2004. Locality-sensitive hashing scheme based on p-stable distributions, Computational Geometry, pp. 253–262.

[33] Berchthold S., Keim D., Kriegel H. P. 1996. The X-Tree: An Index Structure for High-Dimensional Data, VLDB.

[34] Keogh, E., Zhu, Q., Hu, B., Hao. Y., Xi, X., Wei, L. & Ratanamahatana, C. A. (2011). The UCR Time Series Classification/Clustering Homepage: www.cs.ucr.edu/~eamonn/time_series_data/

[35] http://research.microsoft.com/en-us/people/juding/yading.aspx

[36] A. Hinneburg and H. H. Gabriel. DENCLUE 2.0: Fast Clustering Based on Kernel Density Estimation. In IDA, 2007.

[37] Sheih, J., Keogh, E., 2009. iSAX: disk-aware mining and indexing of massive time series data sets. Data Mining and Knowledge Discovery 19 (1), 24-57.

[38] D. Barbara. Requirements for clustering data streams. SIGKDD Explorations, 3(2): 23-27, January 2002.

[39] P. Liu, D. Zhou, N. Wu. 2007. Varied Density Based Spatial Clustering of Application with Noise, *In proc.* of ICSSSM. 528-531