Lambda Means Clustering: Automatic Parameter Search and Distributed Computing Implementation

Marcus Comiter
Harvard University
marcuscomiter@g.harvard.edu

Miriam Cha
Harvard University
miriamcha@g.harvard.edu

H.T. Kung
Harvard University
kung@harvard.edu

Surat Teerapittayanon
Harvard University
steerapi@seas.harvard.edu

***Abstract*—Recent advances in clustering have shown that ensuring a minimum separation between cluster centroids leads to higher quality clusters compared to those found by methods that explicitly set the number of clusters to be found, such as k-means. One such algorithm is DP-means, which sets a distance parameter λ for the minimum separation. However, without knowing either the true number of clusters or the underlying true distribution, setting λ itself can be difficult, and poor choices in setting λ will negatively impact cluster quality. As a general solution for finding λ, in this paper we present λ-means, a clustering algorithm capable of deriving an optimal value for λ automatically. We contribute both a theoretically-motivated cluster-based version of λ-means, as well as a faster conflict-based version of λ-means. We demonstrate that λ-means discovers the true underlying value of λ asymptotically when run on datasets generated by a Dirichlet Process, and achieves competitive performance on a real world test dataset. Further, we demonstrate that when run on both parallel multicore computers and distributed cluster computers in the cloud, cluster-based λ-means achieves near perfect speedup, and while being a more efficient algorithm, conflict-based λ-means achieves speedups only a factor of two away from the maximum-possible.**

I. Introduction

Data clustering is a key step in unsupervised learning tasks. However, many conventional clustering methods, such as k-means, make the sometimes impractical assumption that k, the number of clusters present in the data, is known *a priori*. A recent algorithm receiving large amounts of attention from the machine learning community, DP-means [6], forms clusters of superior quality using a distance parameter λ to ensure minimum separation between cluster centroids rather than specifying k in advance, forming a new cluster when a data point is found to be more than λ distance away from all existing cluster centroids. Under an assumption that a sequence of data is drawn from a Dirichlet Process, the authors of [6] prove that there exists a λ such that when used by DP-means, the algorithm will discover the ground truth number of clusters k.

However, without knowing the underlying parameters of the Dirichlet Process generating the sequence of data, as well as for data of unknown origin, it is unclear how to find the appropriate value of λ for use with DP-means. As a solution, the authors of [6] suggest the use of a farthest-first heuristic. However, this process requires a user-provided approximation of k. As we will show in Section IV, incorrectly setting this approximate k has a marked impact on the resulting value of λ. In practical situations, setting the approximate k can be very difficult, potentially leading to a suboptimal choice of λ and, by extension, suboptimal clustering.

As a solution for finding λ without the need of heuristics such as the farthest-first heuristic based on a user-provided approximation of k, we present λ-means, a clustering algorithm that uses an efficient search procedure to find an appropriate λ, and then runs the celebrated DP-means algorithm in its inner-loop.[1] λ-means has three novel properties: (1) For data generated under a Dirichlet Process as in [6], in the asymptotic limit, λ-means converges on the λ value used in the underlying Dirichlet Process; (2) λ-means uses an efficient search method to quickly find a λ value for use with its inner DP-means loop; (3) By extending another recent work (OCC DP-means [10]) for distributed computing with Optimistic Concurrency Control (OCC), λ-means easily extends to the distributed framework and can use the number of conflicts in each epoch as a low-overhead signal to determine and accelerate λ-means convergence.

We validate λ-means on both synthetic and real world data, in which the underlying data generation process is unknown. On these datasets, we empirically demonstrate that λ-means achieves performance comparable with or exceeding DP-means without the need to know the approximate k a priori. Finally, to study the speedup achieved by λ-means in parallel computing settings, we run it on both multicore computers and distributed cluster computers in the cloud, in both scenarios achieving speedups that are either perfect or only a factor of two away from the maximum possible speedup.

II. Related Work

The problem of clustering N data points has a rich history and abundant literature. A well known clustering algorithm is k-means, a partitional method which, given a distance metric and number of clusters k as parameters, partitions the data into k clusters. However, in practice, knowing k *a priori* can be difficult. To address this problem, a number of methods for finding k automatically have been proposed. A simple heuristic for setting k involves comparing some metric, such as error, against a number of different choices of k, and selecting the value of k at the "elbow" of the resulting curve. To formalize this heuristic, Tibshirani *et al.* [12] propose the "gap statistic," which takes the difference of the logarithm of the

[1] We acknowledge that the clustering performed by λ-means in its inner-loop is DP-means. However, as we have the additional goal of estimating λ, we call our method λ-means for clarity.

pooled intra-cluster sum of squares for the data points being clustered and a reference distribution, and selects the value of k that maximizes the statistic. Hamerly and Elkan [5] introduce the G-means algorithm, a variant of k-means that, under a Gaussian assumption, proposes new clusters and uses the Andersen-Darling test to check for normality before accepting.

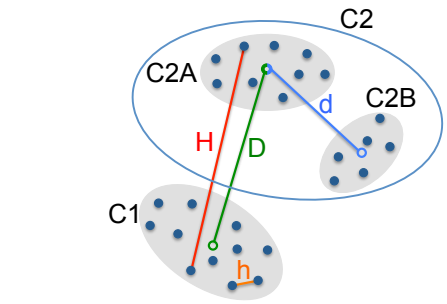
Beyond k-means, there are a number of clustering algorithms that seek to improve cluster quality while avoiding explicitly setting k. The DBSCAN algorithm [4] sets an ϵ-large neighborhood, and clusters points based on a sufficient number of points being found within the neighborhood. Unlike k-means, DBSCAN allows for non-spherical clusters such as long, thin, "snaking" clusters, but has no notions of centroids. The mean shift clustering algorithm [2] is a hierarchical agglomerative method that calculates the gradient of a density estimator in a window around each data point and iteratively moves the window towards an area of higher density until the gradient approaches zero. While mean shift clustering does not require k to be set a priori, its computational cost $\mathcal{O}(kN^2)$ is greater than that of k-means, making it an undesirable choice for large datasets. Building upon mean-shift, [1] proposes the γ-SUP algorithm, which also updates the location of the data points themselves at each iteration. Another family is based on distance metrics measuring the separation among cluster centroids, such as the previously mentioned DP-means [6]. The authors of [14] use a Bayesian framework to automatically learn a set of centroids without explicitly setting the number of clusters. Instead, the data is used to automatically find a set of centroids such that the data can be well represented as a linear combination of the centroids, and the resulting number of centroids used is selected as k.

Regardless of the chosen clustering algorithm, for large datasets, there is a need for parallel and distributed clustering algorithms. One such algorithm is MapReduce k-means, which uses the MapReduce paradigm to parallelize the k-means algorithm for a preconfigured k [3]. OCC DP-means [10] uses the principle of Optimistic Concurrency Control [7] in correcting any non-serializable cluster creation. OCC is a three phased parallelization scheme that guarantees serializability. That is, parallel execution of transactions will yield the same result as some serial execution of the same transactions where individual transactions have been reordered. However, as is the case for DP-means, OCC DP-means likewise requires a preconfigured λ parameter for cluster creation. Our proposed λ-means algorithm builds upon DP-means and OCC DP-means, and is a top-down hierarchical method capable of finding λ automatically.

III. λ-means Overview

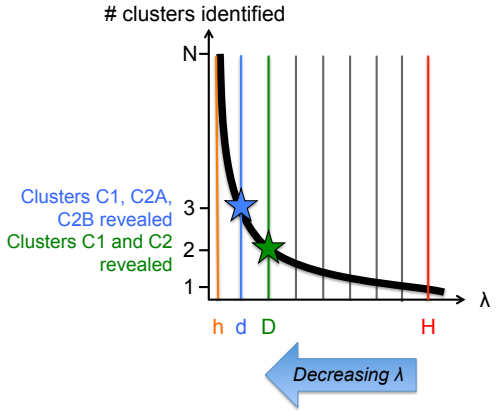
A. The Effect of Decreasing λ

Before introducing the λ-means algorithm, we first describe the effect of decreasing λ, a main mechanism of the λ-means algorithm, using the simple example depicted in Figure 1(a). In this illustrative example, we seek to cluster the N data points into three clusters (C1, C2A, and C2B) by forming a new cluster when a data point is found to be more than λ distance away from all existing cluster centroids. λ is first initialized to be the maximum distance between data points, denoted by H. At this value of λ, a single cluster is formed, as all points are within $\lambda = H$ distance of the single centroid. The single cluster persists until λ decreases below D, the maximum distance between cluster centroids, at which point the single cluster is broken into two clusters, C1 and C2. Next, when λ is decreased below d, which is the minimum distance between cluster centroids, cluster C2 is split into two clusters (C2A and C2B), inducing a total of three clusters. If λ is decreased further, these true clusters will begin to be broken into smaller sub-clusters, continuing until each point is its own cluster when $\lambda < h$, the minimum distance between data points. This example demonstrates that if λ is not estimated correctly, clustering quality would suffer, as too few or too many clusters would be found.



D, d: maximum and minimum distance between cluster centroids
H, h: maximum and minimum distance between data points
N: total # of data points

(a) Illustration of clusters emerging with a decrease in λ.



(b) λ-means finds the λ value at the elbow by dynamically decreasing λ from its initial value H.

Fig. 1: Overview of λ-means

The results of decreasing λ in the previous example are further illustrated in Figure 1(b), which plots the increasing number of clusters identified as λ decreases. Notice that there is an "elbow" in the curve, defined in this paper as the point at which the rate of increase in the number of clusters increases markedly for a decrease in λ, following a similar notion in [12]. This elbow shape corresponds to the emergence of clusters: as λ decreases from its initial large value, clusters are added slowly until all true clusters have

been found. Past this point, these true clusters are effectively being split into meaningless sub-clusters. As the distribution of points *within* the clusters are denser than the distribution of the clusters themselves, at this point a small change in λ causes a proportionally larger increase in the number of clusters. Where these two different patterns in the slope of the curve meet effectively forms the elbow of the curve. As such, because the elbow represents the point at which the true clusters have been identified but have not been broken into meaningless sub-clusters, the λ value at this point should induce the optimal number of clusters to be found, a notion we validate in an information-theoretic sense in Section V.

B. The λ-means Algorithm

We now formally introduce the λ-means algorithm. We first note that an exhaustive linear search for λ at the elbow of the curve would first require producing the entire curve, as seen in Figure 1(b), and then locating the elbow. However, λ-means, presented in Algorithm 1, employs a more efficient search method. In this section, we first describe the cluster-based version of λ-means, which uses the cumulative number of clusters formed as a signaling method and allows for theoretical guarantees, as described in Section III-D. Following this, we describe the conflict-based version of λ-means, which allows for faster algorithm execution when data is well separated. We compare the two methods in Section III-C.[2]

i) Cluster-based λ-means: In the cluster-based version of λ-means, the algorithm first generates a portion of the "elbow" curve efficiently, and then uses the *L-method* set forth in [11]. We now describe the curve generating and elbow search process. λ is first initialized to be the largest distance between any two points in the dataset. The inter-cluster variance (ρ) of the dataset is estimated from the data points. With this estimate of ρ, λ is decreased such that roughly an equal number of clusters are admitted each round. More formally, under a Gaussian-assumption for the distribution of clusters and points, we decrease λ by an amount guided by the PDF of the Gaussian distribution parameterized by the mean of the distribution from which the clusters are drawn and ρ, such that the decrease in λ corresponds to an equal amount of area under the PDF. (Note that for data characterized by a different distribution, the same method can be used with that distribution's PDF). This method has the effect of decreasing λ rapidly at the beginning (while λ is far away from the elbow) and progressively more slowly when λ begins to approach the elbow. This continues until the stopping criterion (described below) is reached. Such procedure has the effect of generating the curve as seen in Figure 1(b) incrementally *from the right to the left*, which is important from an efficiency standpoint, as we can identify the elbow without having to generate the entire curve.

At every round of λ-means, two lines are fit to the partial curve generated up to that point using a slightly modified version of the L-method algorithm introduced in [11]. The algorithm is modified such that we instead use a user-defined

[2]Note that in Algorithm 1 we purposely do not split cluster-based λ-means and conflict-based λ-means into two separate algorithms, as the two share a number of inputs and subroutines.

Algorithm 1: λ-means

Input: data set $\mathbf{X} = \{\mathbf{x}_i | i = 1, \ldots, N\}$
Input: random partitioning of $\mathbf{X}$ into epoch set $\{\mathcal{B}(t) | t = 1, \ldots, T\}$ of equal size, i.e., $|\mathcal{B}(t)| = \frac{N}{T}$
if *cluster-based λ-means* **then**
  Input: γ area under Normal pdf corresponding to how many clusters to introduce with each round of algorithm
  Input: τ threshold for change in slope between the two lines used in L-method
  Input: n number of points to use in fitting line
if *conflict-based λ-means* **then**
  Input: multiplicative decrease factor α and additive decrease factor β
  Input: τ threshold for the number of conflicts c
  Input: T epochs and P processors
Output: cluster centroids $\mathbf{D}$ and assignments $\mathbf{Z} = \{\mathbf{z}_i | i = 1, \ldots, N\}$
$\lambda \leftarrow$ the distance of the furthest point from $\texttt{mean}(\mathbf{X})$
$\mathbf{D} \leftarrow \{\texttt{mean}(\mathbf{X})\}$
if *cluster-based λ-means* **then**
  $c \leftarrow 0$
  $\hat{\rho} \leftarrow \frac{\sum_{i=1}^{N}(X_i - \bar{X})}{N-1}$
  $b \leftarrow \infty$
  while $m \leq \tau$ **do**
    $\lambda \leftarrow \lambda$ s.t. $\int_{\lambda}^{b} \frac{\Phi(x-\mu)}{\rho}\,\mathrm{d}x = \gamma$
    $\hat{\mathbf{D}}, \mathbf{Z} \leftarrow \lambda$**Assign**$(\lambda, \mathbf{D}, \mathbf{X})$
    $\tilde{\mathbf{D}}, \mathbf{Z}, c \leftarrow \lambda$**Validate**$(\lambda, \hat{\mathbf{D}}, \mathbf{Z})$
    $\mathbf{D} \leftarrow \mathbf{D} \cup \tilde{\mathbf{D}}$
    $\mathbf{D} \leftarrow$ **Update**$(\mathbf{D}, \mathbf{Z}, \mathbf{X})$
    $b \leftarrow \lambda$
if *conflict-based λ-means* **then**
  `// Phase 1 iteration`
  while $c \leq \tau$ **do**
    $\hat{\mathbf{D}}, \mathbf{Z} \leftarrow \lambda$**Assign**$(\lambda, \mathbf{D}, \mathbf{X})$
    $\tilde{\mathbf{D}}, \mathbf{Z}, c \leftarrow \lambda$**Validate**$(\lambda, \hat{\mathbf{D}}, \mathbf{Z})$
    $\mathbf{D} \leftarrow \mathbf{D} \cup \tilde{\mathbf{D}}$
    $\mathbf{D} \leftarrow$ **Update**$(\mathbf{D}, \mathbf{Z}, \mathbf{X})$
    `// Multiplicative decrease`
    $\lambda \leftarrow \lambda \times \alpha$
  `// Phase 2 iteration`
  while $c > \tau$ **do**
    $t \leftarrow$ a random sample of epoch index set $\{1, \ldots, T\}$
    $\hat{\mathbf{D}}, \mathbf{Z} \leftarrow \lambda$**Assign**$(\lambda, \mathbf{D}, \mathcal{B}(t))$
    $\tilde{\mathbf{D}}, \mathbf{Z}, \tilde{c} \leftarrow \lambda$**Validate**$(\lambda, \hat{\mathbf{D}}, \mathbf{Z})$
    $\mathbf{D} \leftarrow \mathbf{D} \cup \tilde{\mathbf{D}}$
    $c \leftarrow \tilde{c} \times T$
    $\mathbf{D} \leftarrow$ **Update**$(\mathbf{D}, \mathbf{Z}, \mathbf{X})$
    `// Additive decrease`
    $\lambda \leftarrow \lambda - \beta$
`// Termination process`
while *not converged* **do**
  $\hat{\mathbf{D}}, \mathbf{Z} \leftarrow \lambda$**Assign**$(\lambda, \mathbf{D}, \mathbf{X})$
  $\tilde{\mathbf{D}}, c \leftarrow \lambda$**Validate**$(\lambda, \hat{\mathbf{D}}, \mathbf{Z})$
  $\mathbf{D} \leftarrow \mathbf{D} \cup \tilde{\mathbf{D}}$
  $\mathbf{D} \leftarrow$ **Update**$(\mathbf{D}, \mathbf{Z}, \mathbf{X})$

Algorithm 2: λAssign

Input: a threshold λ, a set of cluster centroids $\mathbf{D}$, data set $\mathbf{X}$
Output: a set of proposed cluster centroids $\hat{\mathbf{D}}$ and assignments $\mathbf{Z}$

Partition data set $\mathbf{X}$ into P sets $\{\mathcal{X}(p)|p = 1, \ldots, P\}$ of equal size where P is the number of processors
$\hat{\mathbf{D}} \leftarrow \emptyset$
for $p = 1, 2, \ldots, P$ **do in parallel**
  for $\mathbf{x}_i \in \mathcal{X}(p)$ **do**
    $\mathbf{d}^* \leftarrow \arg\min_{\mathbf{d} \in \mathbf{D}} \|\mathbf{x}_i - \mathbf{d}\|_2$
    `// Optimistic transaction`
    if $\|\mathbf{x}_i - \mathbf{d}^*\|_2 \geq \lambda$ **then**
      $\mathbf{z}_i \leftarrow \mathbf{x}_i$
      $\hat{\mathbf{D}} \leftarrow \hat{\mathbf{D}} \cup \mathbf{x}_i$
    else
      $\mathbf{z}_i \leftarrow \mathbf{d}^*$

Algorithm 3: λValidate

Input: a threshold λ, a set of proposed cluster centroids $\hat{\mathbf{D}}$ and assignments $\mathbf{Z}$
Output: a set of cluster centroids $\mathbf{D}$ that pass the validation, updated assignments $\mathbf{Z}$ an the number of conflicts c

$c \leftarrow 0$
for $\mathbf{x}_i \in \hat{\mathbf{D}}$ **do**
  $\mathbf{d}^* \leftarrow \arg\min_{\mathbf{d} \in \mathbf{D}} \|\mathbf{x}_i - \mathbf{d}\|_2$
  if $\|\mathbf{x}_i - \mathbf{d}^*\|_2 < \lambda$ **then**
    $\mathbf{z}_i \leftarrow \mathbf{d}^*$ `// Rollback`
    $c \leftarrow c + 1$
  else
    $\mathbf{D} \leftarrow \mathbf{D} \cup \mathbf{x}_i$

Algorithm 4: Update

Input: a set of cluster centroids $\mathbf{D}$, assignments $\mathbf{Z}$ and data set $\mathbf{X}$
Output: a set of updated cluster centroids $\mathbf{D}$
for $\mathbf{d}_j \in \mathbf{D}$ **do**
  $\mathbf{d}_j \leftarrow \texttt{mean}(\{\mathbf{x}_i \in \mathbf{X} | \mathbf{z}_i = \mathbf{d}_j\})$

n points of the curve in forming each line, which allows for early termination. We find that for well-behaved situations (well-separated clusters each with a reasonable number of points, corresponding to a large $\frac{\rho}{\sigma}$ value under the formulation presented in Section V-A), generally $n = 10$ works well, as the local "elbow" in the curve identified is in fact the global "elbow" under this assumption. Once the slope of the two lines differs by at least threshold τ, the λ value corresponding to the number of clusters formed at the point of intersection is chosen as the optimal λ, and the DP-means algorithm is then run as λ-means' inner loop for additional rounds with that value of λ.

ii) Conflict-based λ-means: We now describe the conflict-based version of λ-means, which uses OCC. Under this framework, the dataset is divided into one or more epochs, where each epoch is a random non-overlapping subset of the input data. As described in [10], for a given iteration, each epoch is executed sequentially, and at the end of each epoch, the OCC validation step checks for conflicts, rolling back when proposed cluster centroids are in *conflict* with other centroids (*i.e.*, when new centroids are less than λ distance away from the other centroids). Specifically, the algorithm uses the notion of *conflicts per epoch*, an attribute of the OCC DP-means framework, as a low-overhead signal for two additional purposes: (1) determining whether an optimal λ value has been reached, and (2) controlling λ's rate of decrease. The use of conflicts as a signal to control the search of λ is novel and beyond the original use of conflicts for validation in the OCC DP-means framework.

More specifically, λ-means decreases λ in a two-phase process, using the number of conflicts as an indicator for when to transition between phases, as well as an indicator of when the optimal λ has been reached, causing the algorithm to enter the termination process. Initially, during Phase 1 (fast multiplicative-decreasing phase), when the number of conflicts is low, λ is lowered *multiplicatively*. Once the number of conflicts begins to rise, indicating true clusters are being discovered, the algorithm enters Phase 2 (slow additive-decreasing phase), in which λ is lowered *additively*. When there are zero or sufficiently few conflicts detected, indicating the optimal number of clusters has been found, the algorithm starts the termination process. We find that in practice a τ threshold in Algorithm 1 set around κPN, where κ is a constant (roughly 0.1), works well.

C. Comparing Cluster-based and Conflict-based λ-means

Both cluster-based and conflict-based λ-means seek to find the correct λ at the "elbow" of the generated curve. However, the two variants each have their own inherent strengths and weaknesses. While cluster-based λ-means is theoretically motivated (see Section III-D), at every iteration, the algorithm must cluster the data to completion, even though these intermediate clusters will not be used in the final clustering, increasing runtime and computation cost. Due to the use of conflicts as a signaling mechanism, conflict-based λ-means can immediately stop once a sufficient number of conflicts occur and proceed to the next round with a smaller λ value, decreasing runtime. In the last rounds of the algorithm, which are the most expensive, this early termination can save significant run time. However, when the data is not well separated, the conflict-based signaling mechanism may not be as robust as the cluster-based signaling mechanism. The user may choose the appropriate variant based on run-time concerns and characteristics of the data, such as inherent separability (see Section V-A for a discussion of data separability).

D. Analysis

We prove that the λ found automatically in Algorithm 1 is the same value as λ_{dp}, the value used in [6]. In this formulation, ρ denotes inter-cluster variance and σ denotes intra-cluster variance. We prove the following:

Theorem 1. *Suppose that σ approaches zero with a fixed ρ for data generated from a Dirichlet Process. Then in the limit,*

when λ decreases in Algorithm 1, the ratio between the number of new clusters when $\lambda < \lambda_{\text{dp}}$ and the number of new clusters in the case of of $\lambda > \lambda_{\text{dp}}$ becomes greater than $1 + \omega$ for a fixed positive value of ω.

Proof: Note that when λ decreases past the true underlying value λ_{dp}, λ-means precipitates a surge of clusters, as true clusters are being broken up into sub-clusters. We discuss the behavior of λ-means for all values of λ relative to λ_{dp}. First, consider the case when $\lambda = \lambda_1$, where $\lambda_1 > \lambda_{\text{dp}}$. In this case, the number of new clusters introduced each epoch will be small relative to the other case, when $\lambda < \lambda_{\text{dp}}$. The number of new clusters C_{new_1} that will be added is proportional to the probability shown below:

$$\begin{aligned} C_{\text{new}_1} &\propto P(\text{new}|d_{\min} > \lambda_1)P(d_{\min} > \lambda_1) \\ &\leq P(\text{new}|d_{\min} > \lambda_{\text{dp}})P(d_{\min} > \lambda_{\text{dp}}) \end{aligned} \quad (1)$$

where d_{min} is the minimum distance between two existing clusters. The inequality for (1) follows from the fact that the first quantity holds as the inter-cluster variance σ approaches zero, and the second quantity holds by the assumption that $\lambda > \lambda_{\text{dp}}$. Next, we consider the other case, when $\lambda = \lambda_2$, where $\lambda_2 < \lambda_{\text{dp}}$. The number of new clusters C_{new_2} that will be added is proportional to the probability shown below:

$$\begin{aligned} C_{\text{new}_2} &\propto P(\text{new}|d_{\min} > \lambda_2)P(d_{\min} > \lambda_2) \\ &\geq (P(\text{new}|d_{\min} > \lambda_{\text{dp}}) + \\ &\quad P(\text{new}|\lambda_2 < d_{\min} < \lambda_{\text{dp}}))P(d_{\min} > \lambda_2) \end{aligned} \quad (2)$$

The inequality for (2) follows from the fact that as σ approaches zero, $P(\text{new}|\lambda_2 < d_{\min} < \lambda_{\text{dp}})$ similarly approaches zero, and $P(d_{\min} > \lambda_2) \geq P(d_{\min} > \lambda_{\text{dp}})$ by the assumption that $\lambda < \lambda_{\text{dp}}$. Therefore, from these two cases, we have the following inequality:

$$\frac{C_{\text{new}_2}}{C_{\text{new}_1}} \geq \frac{P(d_{\min} > \lambda_2)}{P(d_{\min} > \lambda_{\text{dp}})}. \quad (3)$$

Since for a fixed positive value of ω

$$\frac{P(d_{\min} > \lambda_2)}{P(d_{\min} > \lambda_{\text{dp}})} > 1 + \omega, \quad (4)$$

we have

$$\frac{C_{\text{new}_2}}{C_{\text{new}_1}} > 1 + \omega \quad (5)$$

Therefore, we have shown that as λ decreases past λ_{dp}, the ratio between the number of new clusters when $\lambda < \lambda_{\text{dp}}$ and the number of new clusters in the case of of $\lambda > \lambda_{\text{dp}}$ becomes greater than $1 + \omega$ for a fixed positive value of ω. ■

IV. Comparison with DP-means

In this section, we compare our algorithm with the DP-means algorithm by directly examining the use of the farthest-first heuristic for finding λ, as suggested by the authors of the original DP-means paper [6]. We show that λ-means is a more robust method for finding the true λ. The farthest-first heuristic requires an approximation k to the true number of clusters. However, we assert that setting this initial k is challenging, especially when the user does not know the characteristics of the data. More importantly, we show that if the initial approximation to k is wrong, it negatively affects finding the correct λ. To demonstrate this, we generate a dataset from a Dirichlet Process and then use the farthest-first heuristic with a number of different values of k to derive λ. Figure 2 shows the results of this experiment. As the figure demonstrates, the derived λ value is quite sensitive to changes in the initial k. As a result, in order to derive the true λ value, the initial approximation must be very exact.

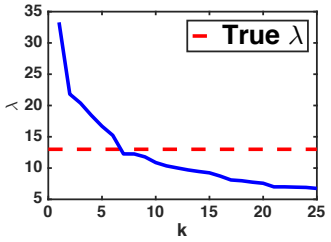


Fig. 2: Derived λ via farthest-first heuristic using different approximate k. The dashed red line denotes the true λ.

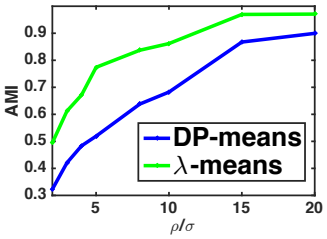


Fig. 3: Comparison of performance of DP-means (blue) and λ-means (green) for different ratio values $\frac{\rho}{\sigma}$

Importantly, this incorrect estimation of λ could potentially cause the DP-means algorithm to identify an incorrect number of clusters, negatively impacting the outcome of the clustering process. As such, the drawbacks of the farthest-first heuristic are clear: the method is brittle to small changes in the approximation of k, having a large impact on the derived value of λ as well as potentially on the resulting cluster quality. In contrast, λ-means automatically finds the λ value that maximizes AMI without an initial approximation for k. Therefore, λ-means achieves its goal of being a more robust alternative method to the farthest-first heuristic for use with DP-means.

Finally, we also note that as compared to that of DP-means, the running time of λ-means is directly proportional to the number of iterations used in its inner-loop. However, this running time can be decreased with the conflict-based version of λ-means, as described in Section III-B.

V. Experiments

We provide experimental evaluation of λ-means on both synthetic and real world data using normalized mutual information (NMI) [9] and adjusted mutual information (AMI) [13]. While AMI was proposed more recently and is normalized against chance, NMI is often used in the literature, and therefore we present results using both metrics.

A. Synthetic Data

We generate synthetic data from a Dirichlet Process as in [6]. Under this formulation, we control both the inter-cluster variance ρ and the intra-cluster variance σ. We use the ratio $\frac{\rho}{\sigma}$ as a measure of the difficulty of clustering the dataset. When $\frac{\rho}{\sigma}$ is large, the clusters are relatively separated and compact,

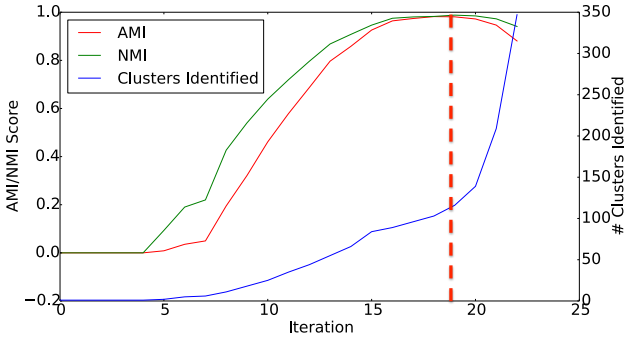


Fig. 4: The optimal λ found by λ-means during iteration 19. At this value, the peaks of AMI and NMI are reached and the correct number of clusters (100) is approximately identified.

TABLE I: Comparison on λ-means and DP-means using synthetic (Syn.) and MNIST datasets

	Dataset					
	Syn. $\frac{\rho}{\sigma} = 15$		Syn. $\frac{\rho}{\sigma} = 5$		MNIST	
Algorithm	AMI	NMI	AMI	NMI	AMI	NMI
λ-means	0.97	0.98	0.77	0.82	0.43	0.53
DP-means	0.87	0.92	0.52	0.78	0.32	0.38

while when $\frac{\rho}{\sigma}$ is small, the clusters are less separated (*e.g.*, larger overlap), making clustering more difficult.

Figure 4 shows AMI and NMI scores using the synthetic data with a high value of $\frac{\rho}{\sigma}$. We note that λ-means is able to automatically find the λ value that maximizes AMI and NMI scores; the vertical red dotted line indicates the iteration (iteration 19) at which the λ-means algorithm terminates. As shown, the AMI and NMI are both maximized at this point. Further, both metrics immediately drop in the additional iterations past termination. (Note that these further iterations are performed artificially after λ-means terminates for the purposes of generating the graph and showing that a maxima is reached.) This therefore demonstrates that the λ value found by λ-means at the elbow of the curve is optimal by this metric. We can further judge λ-means by evaluating its ability to identify the correct number of clusters. The blue curve in Figure 4 plots the cumulative number of clusters identified at each iteration. The vertical red dotted line indicates the iteration at which the λ-means algorithm terminates. We note that at this point, λ-means approximately recovers the correct number of clusters (100).

Additionally, we compare the AMI and NMI scores for λ-means and DP-means in Table I for additional values of $\frac{\rho}{\sigma}$. For DP-means, the λ value is selected with the farthest-first heuristic with k chosen to be the ground truth, and 5 iterations to ensure convergence. For a fair comparison, λ-means also uses 5 iterations in the termination process. We note that for both $\frac{\rho}{\sigma} = 15$ and the more difficult case when $\frac{\rho}{\sigma} = 5$, λ-means outperforms DP-means.

We now discuss the impact of data separability on performance. In Figure 3, we show the performance (in terms of AMI) of λ-means and DP-means on data generated with a Dirichlet Process with varying $\frac{\rho}{\sigma}$. The λ value for DP-means is estimated with the farthest-first heuristic, where the initial approximation for k is chosen using the G-means algorithm [5]. There are two important conclusions to draw from this experiment. First, we see that both algorithms perform better when data is more separable. Second, we find that to achieve a given AMI value, λ-means can cluster a "harder" dataset, while DP-means must use an "easier" dataset (where difficulty is measured in terms of relative values of $\frac{\rho}{\sigma}$). Especially at high levels of AMI, $\frac{\rho}{\sigma}$ must be increased relatively more for DP-means than for λ-means as compared with lower AMI levels.

B. *Real World Data*

We compare the performance of λ-means and DP-means on the MNIST dataset [8], a real world dataset that, in contrast to the synthetic data used in the results presented previously in Section V-A, does not necessarily follow a Dirichlet Process. The results are shown in Table I.[3] For DP-means, in order to make the most direct comparison possible, in choosing the initial k for use with the farthest-first heuristic, we choose the initial approximation for k based on the number of clusters found by λ-means. As the results show, λ-means outperforms DP-means in terms of both AMI and NMI due to its ability to find an optimal k. Further, note that while we do not directly compare with other clustering methods here, the DP-means algorithm (which λ-means outperforms) was shown in [6] to have comparable or better performance than k-means on a number of datasets.

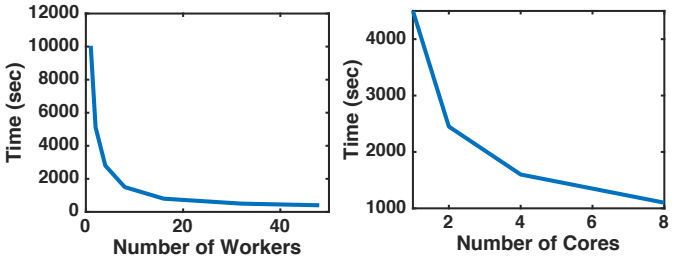
VI. Speedup Results

We measure the running time and speedup of both cluster-based and conflict-based λ-means with OCC on distributed clusters in the cloud and parallel multicore computers.

For the distributed case, we run both versions of λ-means on $N = 10$M points on a distributed cluster in the cloud. We use Amazon Web Services, and use m3.2xlarge EC2 instances located within the same geographical region. We run conflict-based λ-means using 1-, 2-, 4-, 8-, 16-, 32-, and 48-machine clusters. The wall clock running times are shown in Figure 5a. We note that we achieve approximately a 26x speedup on a 48-machine cluster, which is only a factor of 2 away from perfect speedup. We additionally run cluster-based λ-means in the distributed setting, and find that we achieve near perfect speedup on up to a 16-machine cluster. However, as discussed in Section III-C, the wall clock running time of cluster-based λ-means is greater than that of conflict-based λ-means, taking 8,048 seconds and 5,116 seconds, respectively.

For the multicore case, we run conflict-based λ-means using 1, 2, 4, and 8 cores on $N = 1$M points on a 2.6 GHz Intel Xeon E5-2670 v2 (Ivy Bridge) machine with 8 available cores and 30 GB of RAM. The wall clock running times are shown in Figure 5b. Comparing the run time between 1 and 8

[3]MNIST provides empirical evidence that λ-means works on real world data. We could test λ-means on other real world datasets, but due to the space limitations, we focus on the theoretical aspects of the algorithm while showing just one real world data example in this paper.



(a) Distributed cluster in the cloud (b) Multicore

Fig. 5: Wall clock running times of conflict-based λ-means under two parallel computing settings.

cores, we achieve a 4x speedup, only a factor of 2 away from perfect speedup.

VII. Conclusion

λ-means is a clustering algorithm with automatic parameter search as a solution for finding the λ value for use with its DP-means inner-loop without relying on heuristics such as farthest-first. In the asymptotic limit, λ-means automatically converges on the λ value used in the underlying Dirichlet Process, and performs this search efficiently. We introduce two variants of the algorithm: cluster-based λ-means, which provides sound theoretical guarantees, and conflict-based λ-means, which has a faster running time. We validate λ-means on both synthetic and real world datasets. Finally, we demonstrate that λ-means can be adapted for a distributed system. To study the speedup achieved by λ-means in parallel computing settings, we run it on both multicore and distributed settings in the cloud, in both scenarios achieving speedups only a factor of two away from the maximum possible speedup with conflict-based λ-means, and nearly maximum speedup with cluster-based λ-means.

VIII. Acknowledgements

This work is supported in part by gifts from the Intel Corporation and in part by the Naval Supply Systems Command award under the Naval Postgraduate School Agreements No. N00244-15-0050 and No. N00244-16-1-0018. Marcus Comiter is supported by the Smith Family Graduate Science and Engineering Fellowship.

References

[1] T. L. Chen, D. N. Hsieh, H. Hung, I. P. Tu, P. S. Wu, Y. M. Wu, W. H. Chang, S. Y. Huang, et al. γ-sup: A clustering algorithm for cryo-electron microscopy images of asymmetric particles. *The Annals of Applied Statistics*, 8(1):259–285, 2014.

[2] Y. Cheng. Mean shift, mode seeking, and clustering. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 17(8):790–799, 1995.

[3] C. Chu, S. K. Kim, Y. A. Lin, Y. Yu, G. Bradski, A. Y. Ng, and K. Olukotun. Map-reduce for machine learning on multicore. *Advances in Neural Information Processing Systems*, 19:281, 2007.

[4] M. Ester, H. P. Kriegel, J. Sander, and X. Xu. A density-based algorithm for discovering clusters in large spatial databases with noise. In *KDD*, volume 96, pages 226–231, 1996.

[5] G. Hamerly and C. Elkan. Learning the k in k-means. *Advances in Neural Information Processing Systems*, 16:281, 2004.

[6] B. Kulis and M. I. Jordan. Revisiting k-means: New algorithms via bayesian nonparametrics. *Proceedings of the 23rd International Conference on Machine Learning*, 2012.

[7] H. T. Kung and J. T. Robinson. On optimistic methods for concurrency control. *ACM Transactions on Database Systems*, 6(2):213–226, 1981.

[8] Y. LeCun, C. Cortes, and C. J. Burges. The mnist database of handwritten digits, 1998.

[9] C. D. Manning, P. Raghavan, and H. Schutze. *Introduction to information retrieval*. Cambridge university press Cambridge, 2008.

[10] X. Pan, J. E. Gonzalez, S. Jegelka, T. Broderick, and M. I. Jordan. Optimistic concurrency control for distributed unsupervised learning. In *Advances in Neural Information Processing Systems*, 2013.

[11] S. Salvador and P. Chan. Determining the number of clusters/segments in hierarchical clustering/segmentation algorithms. In *Proceedings of the Sixteenth IEEE International Conference on Tools with Artificial Intelligence*, pages 576–584, 2004.

[12] R. Tibshirani, G. Walther, and T. Hastie. Estimating the number of clusters in a data set via the gap statistic. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 63(2):411–423, 2001.

[13] N. X. Vinh, J. Epps, and J. Bailey. Information theoretic measures for clusterings comparison: Variants, properties, normalization and correction for chance. *The Journal of Machine Learning Research*, 11:2837–2854, 2010.

[14] M. Zhou, H. Chen, L. Ren, G. Sapiro, L. Carin, and J. W. Paisley. Non-parametric bayesian dictionary learning for sparse image representations. In *Advances in neural information processing systems*, 2009.