Integrity Verification of K-means Clustering Outsourced to Infrastructure as a Service (IaaS) Providers

Ruilin Liu[*] Hui (Wendy) Wang[†] Philippos Mordohai[‡] Hui Xiong[§]

Abstract

The Cloud-based infrastructure-as-a-service ($IaaS$) paradigm (e.g., Amazon EC2) enables a client who lacks computational resources to outsource her dataset and data mining tasks to the Cloud. However, as the Cloud may not be fully trusted, it raises serious concerns about the *integrity* of the mining results returned by the Cloud. To this end, in this paper, we provide a focused study about how to perform integrity verification of the k-means clustering task outsourced to an $IaaS$ provider. We consider the untrusted *sloppy* $IaaS$ service provider that intends to return wrong clustering results by terminating the iterations early to save computational cost. We develop both probabilistic and deterministic verification methods to catch the incorrect clustering result by the service provider. The deterministic method returns 100% integrity guarantee with cost that is much cheaper than executing k-means clustering locally, while the probabilistic method returns a probabilistic integrity guarantee with computational cost even cheaper than the deterministic approach. Our experimental results show that our verification methods can effectively and efficiently capture the sloppy service provider.

Keywords: Data-mining-as-a-service; Cloud computing; integrity; k-means clustering; Infrastructure as a Service (IaaS).

1 Introduction

Due to recent advances in network techniques, a new paradigm so called as "mining and management of data as service" has become a practical and cost effective solution for providers and consumers, ranging from business analytics to scientific computing. Cloud computing, an emerging trend of provisioning scalable computing services, provides a natural solution for the data-mining-as-a-service ($DMAS$) paradigm. There are a few active industry projects (e.g., Google's Predication APIs [2]) provide cloud-based data mining as a service. The paradigm of "mining and management of data as service" will presumably grow [6].

Although outsourcing of data mining is advantageous for data owners with limited abilities to achieve sophisticated analysis on their large volumes of data, it triggers serious security concerns. One possible security issue is the *integrity* of the mining results returned by a third party service provider that is potentially untrusted. There are many possible reasons for the service provider to cheat on its answers [9]. For instance, the service provider would like to improve its revenue by computing with minimal resources while charging for more. A basic problem is inherent in the outsourcing computing paradigm: How can a client of weak computational power verify the correctness of the received result of intensive mining computations, without any trusted verification service provider?

In the last decade, intensive efforts have been put on the security issues of the database-as-a-service (DAS) paradigm [12, 14, 19]. Only until recently some attention was paid to the security issues of the $DMAS$ paradigm. However, most of these work only focus on data confidentiality and pattern privacy [18, 21, 23]. There is surprisingly very little research [24] on result integrity verification of outsourced data mining computations in the $DMAS$ paradigm.

In this paper, we focus on k-means clustering, a popular clustering algorithm used in many applications. We consider the popular infrastructure-as-a-service ($IaaS$) model that most Cloud computing providers (e.g., Amazon EC2 [1]) offer. In this model, the client sends both data and the code of the k-means clustering algorithm to the Cloud, while the Cloud provides storage and hardware for computation. We consider the *sloppy* Cloud service providers that may terminate the iterations early to save computational cost, and thus return incorrect clustering result.

We propose both *probabilistic* and *deterministic* approaches to verify whether the $IaaS$ provider has executed the outsourced k-means clustering software faith-

[*]Stevens Institute of Technology, NJ, USA. rliu3@stevens.edu.
[†]Contact author. Stevens Institute of Technology, NJ, USA. Hui.Wang@stevens.edu.
[‡]Stevens Institute of Technology, NJ, USA. Philippos.Mordohai@stevens.edu.
[§]Rutgers University, Newark, NJ, USA. hxiong@rutgers.edu.

fully and returned correct clustering result. The deterministic approach checks whether each tuple t is assigned to its nearest cluster centroid. To speed up the verification procedure, instead of comparing the distance between t and all cluster centroids, we use the Voronoi diagram to find a small portion of centroids whose distance to t needs to be measured and compared for verification. The probabilistic approach is designed by using *synthetic clusters* (SCs). The *SC*-based approach provides quantifiable correctness guarantee to catch a sloppy server. It only needs a small number of synthetic tuples to catch the clustering result of small errors with high probability. The deterministic method returns 100% integrity guarantee with cost much cheaper than executing the k-means clustering mining locally, while the probabilistic method returns a probabilistic integrity guarantee with computational cost even cheaper than that of the deterministic approach. We provide an extensive set of experiments evaluating the performance of both deterministic and probabilistic verification approaches, with the simulation of sloppy servers. Our experimental results show that the time of our verification approaches is only 1.2% of the mining time at most. To our best knowledge, we are the first to investigate the problem of verifying the correctness of outsourced k-means clustering.

2 Preliminaries

2.1 K-means Clustering In this paper, we focus on k-means clustering, a well-known clustering mining problem. Briefly speaking, given a set of points $D = \{t_1, \dots, t_n\}$, where each point is a d-dimensional vector, k-means clustering partitions D into k groups $\mathcal{C} = \{C_1, \dots, C_k\}$, by minimizing the within-cluster sum of square distances $SD = \sum_{j=1}^{k} \sum_{t_i \in C_j} ||t_i - c_j||_2^2$, where c_j is the center of group C_j, $c_j = \frac{1}{|C_j|} \sum_{t_i \in C_j} t_i$. In the remainder of the paper, we use *cluster* and *group* interchangeably. Given a tuple t_i, let C_j be its cluster and c_j be the centroid of C_j, we say the cluster label of t_i achieves a *locally optimal solution* if $\nexists c'_j \neq c_j$ s.t. $\sum_{t_i \in C'_j} ||t_i - c'_j||_2^2 < \sum_{t_i \in C_j} ||t_i - c_j||_2^2$. We say the clustering label of t_i is *incorrect* if the cluster label of t_i does not achieve the locally optimal solution.

Minimizing the sum of square distances is NP-hard [17, 3]. Thus we consider Lloyd's algorithm [16], a popular k-means clustering algorithm. It starts from randomly picking k tuples as k initial cluster centroids $C^{(0)} = \{c_1^{(0)}, \dots, c_k^{(0)}\}$, then proceeds by repeating the following two steps:

1. Assignment: Given the current set of k cluster centroids $C^{(x)} = \{c_1^{(x)}, \dots, c_k^{(x)}\}$, assign each point t_i to the cluster whose center is the closest to t_i.

2. Update: Update the points in each cluster, and compute the new centroids of each cluster, $c_j^{(x+1)} = \frac{1}{|C_j^{(x+1)}|} \sum_{t_i \in C_j^{(x+1)}} t_i$.

The above two steps are repeated until no tuple changes its cluster assignment anymore, i.e., the clustering procedure reaches convergence.

2.2 Outsourcing Setup We consider the *infrastructure-as-a-service (IaaS)* paradigm. In this paradigm, the client outsources both data and the code of Lloyd's method to the *IaaS* service provider (server). The code is executed at the server side. The server provides storage and hardware for the computation. A typical *IaaS* example is Amazon EC2 Web Service [1]. The client configures the value of k and picks k initial centroids. After executing the mining, the server returns the clustering labels in the same order as the tuples were received. The server may return the centroids to the client too. But the centroids may not be correct, as the server may terminate the mining before executing the Update step.

Another possible computing paradigm is the *Software-as-a-service (SaaS)* paradigm, in which the client only outsources data to the server, while the server runs its own clustering software on the outsourced data. We will discuss the challenges and difficulties of verifying the k-means clustering result by *SaaS* providers in Section 5.

2.3 Types of Dishonest Servers In this paper, we consider the *sloppy* server that intends to terminate the iterations before reaching convergence to save computational cost. We aim at verifying whether there is any clustering label that does not reach local optimality.

There may exist other type of servers that have more cheating power and be able to launch more sophisticated attack. For example, it is possible the server knows the details of verification mechanism and may try to escape verification by making sure that incorrect clustering results meet what the verification procedure aims to validate. We refer to this type of server as the *malicious* server. We will discuss possible solutions to catch such malicious servers in Section 5.

2.4 Overview of Solutions We propose two solutions to verify the correctness.

(1) The *deterministic* approach verifies whether each tuple t achieves the local optimal solution with 100% certainty. The *brute-force* approach computes the distance between t and each centroid. We propose the Voronoi diagram based approach that uses the Voronoi diagram to pick the neighboring centroids of t for verification. When $k > (d+1)(d+2)/2$ and $n > k^{(\lceil d/2 \rceil - 1)}$ (i.e., large datasets with numerous clusters), the Voronoi diagram based approach only picks a small

Verification Guarantee	Verification Approach	Complexity of Verification at Client Side	
Deterministic	Voronoi-diagram based approach (Sec. 3)	If $k > (d+1)(d+2)/2$ and $n > k^{(\lceil d/2 \rceil - 1)}$	Otherwise
		$O(nk_d + klogk + k^{\lceil d/2 \rceil})$	$O(nk)$
Probabilistic	Synthetic-cluster approach (Sec. 4)	$O(m)$	

Table 1: Summary of verification approaches (n: $|D|$; m: # of artificial tuples; k: number of clusters in D; d: # of attributes of D; k_d: the average number of centroid neighbors in Voronoi diagram).

portion of centroids for verification. Therefore, the Voronoi diagram based approach is especially suitable to $DMAS$ paradigm where clustering on big datasets with large cluster numbers are common.

(2) The *probabilistic* approach returns a probabilistic correctness guarantee of the returned cluster labels. We design the *synthetic-cluster* (SC) based approach which aims to catch the sloppy server. Under this approach, the client inserts a set of artificial tuples into the dataset before outsourcing. The artificial tuples will be clustered independently from the original dataset and their labels will be validated for verification. The *SC*-based approach only needs a small number of synthetic tuples to catch a clustering result with few errors with high integrity probability guarantee.

Table 1 summarizes the two solutions with regard to their probabilistic guarantee and complexity. The complexity of both solutions is much cheaper than that of k-means clustering ($O(iknd)$, where i is the number of iterations). Furthermore, with probabilistic integrity guarantees, the SC-based approach has much lower verification complexity than that of the deterministic approach. We discuss the details of these two approaches in Sec. 3 and Sec. 4 respectively.

3 Deterministic Approach

The deterministic approach for verifying that the output of a server is a local minimum of the k-means criterion, i.e. for verifying that clustering has converged and all tuples are assigned to the nearest centroid, comprises two steps: i) verifying that the returned centroids, if available, are consistent with the label assignments or computing the centroids from the labels, and ii) verifying that there is no centroid that is closer to a tuple than its currently assigned one. These steps correspond to the steps of the k-means algorithm, since we cannot know which was the last step performed if the server terminated early. Performing only one of these tests is insufficient to conclude that the results are correct. On the other hand, any inconsistency immediately raises a red flag and tests can be terminated.

First, we verify whether the returned centroids, if there is any, are indeed the centroids of the returned clusters taking the returned labels for granted. The client computes the centroids ($O(n)$) and compares them to those returned by the server. If there is any

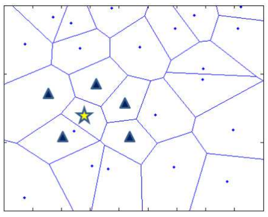


Figure 1: An Example of Deterministic Approach

discrepancy, the server was not honest. This step is also required in case the server only returns labels, but not the coordinates of the centroids.

Second, we verify that each tuple has been assigned to the nearest centroid. The brute force version of this test is $O(nk)$ since the distances from each tuple to all centroids have to be computed. We propose here a faster procedure, which is still guaranteed to detect any discrepancy and requires some pre-processing to significantly reduce the factor k of the complexity. Specifically, we propose to compute the Voronoi diagram of the centroids returned by the server, which will be the *input sites* of the diagram. The Voronoi diagram subdivides the space into regions such that any point in a region is closer to the input site of that region than any other input site. Fig. 1 shows an example of the Voronoi diagram in 2D. Based on the Voronoi diagram, we verify the clustering label of the tuple marked with the star in Fig. 1 by using centroids marked with triangles. The expected complexity for computing the Voronoi diagram is $O(klogk + k^{\lceil d/2 \rceil})$ in high-dimensional spaces [7, 8]. The dual of the Voronoi diagram is the Delaunay triangulation and its edges connect input sites that share a Voronoi boundary (are neighboring). We will use the Delaunay triangulation as a neighborhood graph which has edges only between neighboring sites (centroids). The expected number of neighbors of a site in d dimensions is $k_d = (d+1)(d+2)/2$. When $k > (d+1)(d+2)/2$ and $n > k^{(\lceil d/2 \rceil - 1)}$ (i.e., $k > k_d$ and $O(n)$ dominates $O(k^{\lceil d/2 \rceil})$), we use the Voronoi diagram to pick the neighboring centroids for verification. Otherwise, we use the brute-force approach and check all centroids for verification.

Given the neighborhood graph, we will verify the assignment of each tuple by comparing the distance to its currently assigned centroid c_i and the neighbors of c_i in the graph. Next we show that this test is sufficient and no other distances have to be computed. Assume that a tuple t has been assigned to an incorrect centroid c_f. This means that the segment from t to c_f crosses one of the boundaries of the Voronoi diagram neighboring c_f. (If the segment does not cross a boundary, then the assignment is correct.) The centroid of the boundary that was crossed, denoted by c_c, is nearer to t than c_f is to t and the algorithm detects this discrepancy. Note that c_c does not have to be the correct centroid for t.

In summary, if $k > (d+1)(d+2)/2$ and $n > k^{(\lceil d/2 \rceil - 1)}$, the complexity of validating the local optimality is $O(nk_d)$ since centroids have on average k_d neighbors that have to be tested, and the total complexity of verification at the client side is $O(nk_d + klogk + k^{\lceil d/2 \rceil})$. Otherwise, the total complexity of verification at the client side is $O(nk)$.

4 Probabilistic Approach

The key to the synthetic-cluster (SC) verification approach is to generate a set of *artificial tuples AT* that is *well-separated* from D so that AT will not influence the clusters of D. The client will send $D \cup AT$ to the server as a single dataset, and require $(k+w)$-means clustering, where k and w are the number of clusters of D and AT. The clustering answer by the server will be verified by the client as verifying the correctness of clustering on AT. Compared with the deterministic approach, an advantage of the SC-based approach is that the verification requires neither centroid computation nor searching for closest centroids. Therefore, the SC-based approach is much faster than the deterministic approach. Although inserting AT into D will increase the mining overhead at the server side, we will show later in this section and Section 6 that the overhead is small, since it does not need large numbers of artificial tuples to achieve high probabilistic guarantee. We note that AT can be normalized in the same way as D.

We formalize our verification goal first. Given a set of data points D, let L be the clustering labels of D after it reaches local optimal solution, and L_s be the labels returned by the server, the *precision* of L_s is defined as $R = \frac{|L \cap L_s|}{|L_s|}$ (i.e., the percentage of returned cluster labels that are correct). We aim at designing verification techniques that can provide (α, β)*-correctness.* Formally, given the clustering labels L_s returned by the server, we say a verification method M can verify (α, β)-correctness of L_s if the probability P to catch the server that returns L_s of precision $R = \beta$ satisfies that $P \geq \alpha$, where $\alpha, \beta \in [0, 1]$ are user-specified.

We assume that the sloppy server cannot distinguish original tuples from artificial ones and therefore will return wrong labels of both types of tuples with equal probability. Given a dataset D with β percentage of correct cluster labels, where β is the correctness ratio threshold given in the (α, β)-correctness requirement, if there is any artificial tuple whose clustering label is wrong, the sloppy server will be caught with 100% certainty. Otherwise, the probability P of catching the wrong clustering label of at least one real tuple is $P = 1 - p^m$, where p is the probability that the cluster label of a real tuple t is correct. Given the precision threshold β, the probability p that the cluster label of any real tuple is correct satisfies is $p = \beta$. Therefore, the probability P of catching a sloppy server with m artificial tuples is $P = 1 - \beta^m$. To satisfy (α, β)-correctness requirement, it is easy to infer that $m \geq \lceil log_\beta(1-\alpha) \rceil$. Our analysis shows that to catch a server that changes a small fraction of cluster labels with high correctness probability does not need large number of artificial tuples. For instance, when $\beta = 0.95$ (i.e., 5% of cluster labels are wrong) and $\alpha = 0.95$, $m = 58$. Note that the number of artificial tuples is independent from the size of the original dataset. This is advantageous as the SC-based approach can be used for efficient verification of clustering of large datasets.

To catch a server that may terminate the execution early, a straightforward solution is to construct artificial tuples whose number of iterations for clustering is guaranteed to be no fewer than that of D. In general, estimating the number of iterations by k-means clustering is challenging; it is largely dependent on the data and how initial centroids are chosen for clustering. [5, 13, 22] provide the theoretical upperbound of the number of iterations of the k-means algorithm. However, the upperbound may be too loose in practice; using the theoretical upperbound of the number of iterations of D may introduce tremendous amounts of artificial tuples for verification. Therefore, we aim at constructing artificial tuples whose number of iterations to reach convergence is guaranteed to be no less than a given threshold θ, where θ can be specified according to the client's outsourcing budget. One possible way is to compute $\theta = \frac{by}{x}$, where b is the client's verification budget (in monetary format), x is the hourly cost of using server's resources (as Amazon's pricing model allows), and y is the number of iterations that can be finished within an hour.

Thus AT should meet two requirements: (1) AT is *well separated* from the original dataset D, so that artificial and true tuples are clustered independently (i.e., never appear in the same cluster), and (2) the number of iterations on AT to achieve convergence is no less than a given threshold θ. Our algorithm consists of two steps: (1) construct a set of synthetic clusters $SC_1, \dots, SC_w$ that require θ iterations to reach convergence, and (2) move $SC_1, \dots, SC_w$ away from D

so that they are well-separated. The pseudo code of our algorithm can be found in our full paper [15]. Next, we explain the details of the two steps.

Step 1: Construct θ-iteration Clusters. To construct artificial tuples whose number of iterations to reach convergence is guaranteed to be no less than a given threshold θ, we adapt the gadget-based approach in [22] to our setting. Intuitively, we construct a sequence of gadgets $G_0, \ldots, G_{t-1}$. The *leaf* gadget G_0 consists of only one point F (of constant weight w_F), and one center P_0. Each gadget G_i $(i > 0)$ consists of six points $\{P_i, A_i, B_i, C_i, D_i, E_i\}$, with each point assigned a constant weight. The weights of each point specify the frequency of the point in the dataset. The rule is that t gadgets will build an instance with $2t + 1$ clusters, on which the k-means clustering will run $4t+3$ iterations (the proof is similar as in [22]). Given the required threshold θ of number of iterations, it requires $m = 325[\frac{\theta-3}{4}] + 50$ artificial tuples that are grouped into $w = 2[\frac{\theta-3}{4}] + 1$ clusters. More details of the gadget construction can be found in the full version of this paper [15]. We note that the real and artificial tuples may converge at different speeds. However, our AT approach always requires θ iterations on artificial tuples to reach convergence, and thus θ iterations on real tuples, no matter whether they have reached convergence before that or not.

Step 2: Make Clusters Well-Separated. Intuitively, clustering of two sets of tuples that are far away will not influence each other. We observe that, given two datasets D and D', if $\forall t \in D$ and $t' \in D'$, $dist(t, t') \geq max(MaxDis(D), MaxDis(D'))$, where $MaxDis(D)$ is the largest distance between any pair of tuples of D, then t and t' will never belong to the same cluster. The proof can be found in the full version of this paper [15]. A challenge is that computing the pairwise distances of D can be prohibitive for large datasets ($O(n^2)$). Therefore, we relax the lower bound of the distance between artificial tuples and the true dataset as the *possible* maximum pairwise distance. Before we explain the details, we define some notations. Given a d-dimension dataset D, we consider it as a d-hypercube H, in which the edge on the i-th dimension of H corresponds to the i-th attribute of D. We define a *corner* c of H as a tuple $t(v_1, \ldots, v_d)$ such that $\forall i (1 \leq i \leq d)$, v_i is either the minimum or the maximum value of the attribute A_i of D. We define the *hyper diagonal* of D as the longest edge between any two corners of H. It is straightforward that the length of the hyper diagonal of H is always no less than the maximum pairwise distance of D. Therefore, we use the length of hyper diagonal of D to construct synthetic clusters that are well-separated from D. In particular, given a d-dimension dataset D

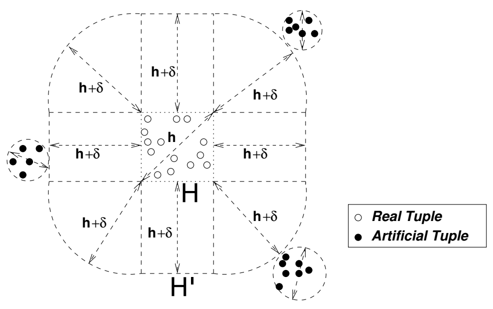


Figure 2: Construction of Synthetic Clusters

and its d-hypercube H, first, we compute the length h of the hyper diagonal of H. Second, for each attribute A_i, we expand its corresponding dimension in H by changing its maximum value max_i and minimum value min_i to be $max_i = max_i + h + \delta$ and $min_i = min_i - h - \delta$, where δ can be any positive number. Let the expanded hypercube be H'. Third, we move the synthetic clusters $SC_1, \ldots, SC_w$ constructed by Step 1 to be outside of H'. Figure 2 illustrates the construction procedure of well-separated SCs.

Maintenance of Cluster Labels of ATs. There should exist a mechanism that helps the client find the cluster labels of the ATs efficiently. One solution is that the client maintains a copy of ATs and their clustering labels locally. For those clients who cannot afford such storage overhead (e.g., those who use mobile devices to access Cloud and retrieve the clustering result), we allow them to attach the cluster membership to ATs when outsourcing, and recover the cluster membership after they receive the clustering result. In particular, before the client outsources her dataset, she assigns each tuple t (including original and artificial ones) a unique *signature* $Sig_t = Enc(s)$, where Enc is an encryption function, s is a bit vector of length ($\lfloor log_2(n+m) \rfloor + \lfloor log_2(w) \rfloor$) ($n$ and m: the number of original and artificial tuples; w: the number of synthetic clusters). The first $\lfloor log_2(n+m) \rfloor$ bits store the tuple ID, and the rest $\lfloor log_2(w) \rfloor$ bits store the cluster IDs. All original tuples are assigned the cluster ID 0, while each AC is assigned a unique cluster ID in $[1, w]$. The client will send the (encrypted) signatures together with data to the server. We assume that the server cannot decrypt the signatures. The server has to return the signatures with the clustering labels back to the client. The signatures will be decrypted by the client to retrieve the expected clustering memberships of artificial tuples. By using the signatures, the space overhead at the client side is negligible, as the auxiliary information that is stored at the client side is only the decryption keys that are used to decrypt the signature and thus recover cluster membership.

Complexity analysis. At the client side, the complexity of constructing artificial tuples is $O(n + m)$, where

n and m are the numbers of original and artificial tuples. The complexity of verification is $O(m)$, as it only involves checking the clustering label of each artificial tuple.

At the server side, the complexity of running Lloyd's method on the outsourced dataset (including artificial tuples) is $O(i(k+w)(n+m)d)$, where i is the number of iterations, k and w are the number of true and synthetic clusters, and d is the number of dimensions. As m is always much smaller than n, the main reason for possible additional overhead, if there is any, is that θ, the number of iterations of clustering on SC, is larger than the number of iterations of D. However, since θ is decided by the client's outsourcing budget, such overhead should be affordable by the client.

5 More Outsourcing Settings

5.1 Server with More Cheating Power Unfortunately, the SC-based approach cannot catch a malicious server that may obtain more cheating power (e.g., has the knowledge of the verification mechanism) and employ computationally inexpensive attacks to defeat the SC-based verification approach. For example, if the server is aware that the artificial clusters and D should be well-separated, it can seek any data segmentation methods, e.g., the max-margin clustering algorithm [11], to separate D and AT.

A possible solution to catch the malicious service provider is *sampling-based* verification. The sampling-based approach does not insert any artificial tuple into the dataset. Instead, the client will pick a set of sample tuples from D, and verify whether the clusters of these samples achieve locally optimal solution. Similar to the SC-based approach, the sampling-based approach will only return a probabilistic correctness guarantee. The challenge is how to pick the appropriate sample tuples. Picking sample tuples of specific properties, e.g., that are more likely to meet convergence in later iterations, can increase the probability of catching the incorrect labels that are produced by early termination of iterations. On the other hand, picking sample tuples of specific properties may enable the malicious server to escape from verification, if it knows the details of the verification mechanism. For instance, if it knows that the verification mechanism will pick tuples that are more likely to meet convergence in later iterations, it will ensure the clustering labels of these tuples are correct, while changing the clustering labels of other tuples on purpose. We claim that random sampling can catch such malicious server effectively. In particular, the client randomly picks sampling tuples and verifies whether these tuples are clustered correctly. If any sample fails the verification, the server is caught with 100% certainty. Otherwise, the probability P of catching a cheating server by choosing l sample tuples S is $P = 1 \ - \beta^l$. To satisfy (α, β)-correctness, the number of sample tuples l must satisfy $l \geq log_\beta(1-\alpha)$.

5.2 Handling Software-as-a-service ($SaaS$) Model In the *software-as-a-service (SaaS)* outsourcing paradigm, the client only outsources data to the server, while the server provides its own clustering software, storage, and hardware for the mining. The challenge of verifying the correctness of k-means clustering in the $SaaS$ paradigm is that the clustering results rely on a number of factors, e.g., how the initial centroids are picked, how the distance function is defined, and whether data is normalized. It is quite challenging to verify whether the server has executed the clustering faithfully without knowing the details of how the server implemented the k-means clustering algorithm. Besides, as k-means is an NP-complete problem, there may exist multiple locally optimal solutions. In this case, simply checking whether the labels computed by the client match those returned by the server (as the SC-based approach does) cannot determine whether the server's result is correct. Therefore, for the $SaaS$ paradigm, the client can verify the local optimality by running one more iteration on the cluster labels, if the server is willing to reveal the implementation details of the mining software, e.g., how the distance function is defined.

5.3 Other Centroid-based Clustering Algorithms The major difference between other centroid-based algorithms such as k-medoids and k-medians algorithms from the k-means algorithm is how the centers of clusters are computed in each iteration. We claim that our deterministic approach is still valid to catch incorrect k-medoids and k-mean clustering result, by running an additional iteration and constructing a Voronoi diagram of returned centroids. On the other hand, our probabilistic approach needs to be adapted to other centroid-based clustering algorithms. But the design philosophy should be the same: we insert a number of artificial tuples that satisfy two conditions: (1) the artificial tuples are well-separated from the real tuples so that they will not be partitioned into the same clusters of real tuples, and (2) it requires θ iterations to reach convergence on the artificial tuples (and thus θ iterations on the real tuples).

6 Experiments

In this section, we simulate the sloppy server, and report the overhead of space and performance of the three verification approaches. We implemented the three approaches in Java, and conducted an extensive set of experiments to measure both the space and time overhead of the verification. We also measured the mining overhead at the server side.

Experiment Environment. All of our experiments are evaluated on a desktop PC with a 2.4GHz Intel Core 2 Quad Q6600 CPU and 3GB RAM running Windows XP. For each experiment that measured the time performance, we ran 5 times and took the average. We pick θ heuristically based on the estimation of number of iterations. We use qhull software package[1] for the computation of Voronoi diagram.

Datasets. We used three real datasets, including the $USPS$ Handwritten Digits datasetthe $Letter$ recognition dataset from UCI repository, and a part of a large 3D point cloud acquired using range sensors from Ottawa, Canada. The $Ottawa$ dataset consists of 973547 tuples. We created five datasets from $Ottawa$ dataset by picking $100K$, $200K$, $300K$, $4000K$, and $500K$ tuples randomly. We also use a synthetic dataset called Time series (TS) dataset. The details of the datasets can be found in our full paper [15].

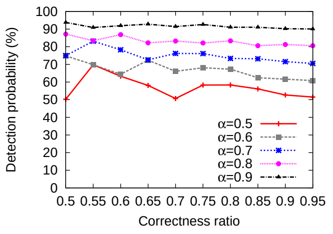


Figure 3: Detection prob. of SC-based approach

6.1 Robustness of Probabilistic Approach

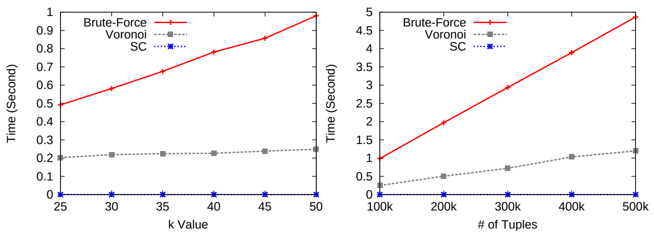
We measure the robustness of our SC-based approach by studying the probability that a sloppy server can be caught by using artificial tuples. We use the $USPS$ dataset and vary β value, i.e., the correctness ratio threshold, to control the amount of mistakes that the server can make on the cluster labels. For each β value, we randomly modify $1-\beta$ percent of cluster labels (including both true and artificial ones) so that they do not achieve local optimal solution. Then with various α values, we construct artificial tuples to satisfy (α,β)-correctness. We verify the cluster labels and record whenever there is a label of an artificial tuple that is not correct (i.e., the server is caught). We repeat this experiment 20,000 times and record the percentage of trials (as detection probability) that the server is caught. Figure 3 plots the probability for $\alpha \in [0.5, 0.9]$ and $\beta \in [0.5, 0.95]$. It shows that the detection probability is always higher than α, the required correctness guarantee threshold. This proves the robustness of our SC-based approach.

[1]http://www.qhull.org/

6.2 Overhead of Verification Preparation at Client Side

In this section, we measure the time and space overhead of verification preparation of the SC-based approach. We do not measure such overhead for the deterministic approach as it does not need any verification preparation at the client side.

First, we measure the space overhead introduced by adding artificial tuples and signatures. The overhead is quantified as $o = \frac{(|D'|-|D|)}{|D|}$, where D and D' are the datasets before and after inserting artificial tuples. We vary the α value from 0.5 to 0.9, and observe that the space overhead is always very small. For the TS dataset that is small, the overhead is around 1.6%, while for the $USPS$ dataset that is large, it never exceeds 1% even with larger α values. Second, we measure the time of constructing ATs for the SC-based approach. Our observation is that the time is always small (no more than 12 seconds), even for large α value. We omit the results due to space limit.



(a) Various # of clusters (dataset size: $100K$)

(b) Various dataset size ($k = 50$)

Figure 4: Verification time at client side ($Ottawa$ dataset, $\alpha, \beta = 0.9$)

6.3 Deterministic Approach Versus Probabilistic Approach

We measure and compare the verification time at the client side of both deterministic and probabilistic verification approaches. For the deterministic verification, we measure the verification time of brute-force and Voronoi diagram approaches, while for the probabilistic verification, we measure the verification time of the SC-based approach. Figure 4 (a) and (b) show the result when we vary the value of k for a fixed dataset, and vary the size of the dataset for a fixed value of k, respectively. We observe that, first, the SC-based approach has much less verification overhead than that of both deterministic verification approaches. This is not surprising as the probabilistic approach only can deliver probabilistic guarantees. Second, the verification time of the SC-based approach stays the same when either k or the size of dataset grows. This is because the number of artificial tuples that are verified by the SC-based approach only relies on α and β, not k or the database size. On the other hand, the verification time of both deterministic approaches grows with increasing dataset size, as they need to traverse the dataset at least once. In particular, the verification time of the brute-force approach increases much faster than the Voronoi diagram approach. Third our Voronoi approach is much faster than the brute-force approach (60% - 80% savings); the speedup is more significant with larger k. Therefore the Voronoi diagram approach is preferred when k is large. Similar results are observed when increasing the dataset size (Figure 4 (b)). Our Voronoi diagram approach is superior to the brute-force approach with at least 50% savings; these savings increase when the dataset size grows. We also use *letter* dataset ($d = 10$) and compare the time performance of both brute-force and Voronoi diagram approach. Our experiments show that the Voronoi diagram approach loses to the brute-force approach as its size ($n = 20K$) does not satisfy that $n > k^{(\lceil d/2 \rceil)-1)}$ ($k = 50$). This follows our complexity analysis in Sec. 3.

6.4 Mining Overhead at the Server Side We also measure the additional overhead that is incurred at the server side due to verification. We only measure the overhead by the SC-based approach as the deterministic approach does not change the database size and thus mining time. We define the overhead as $\frac{T'-T}{T}$, where T and T' are the mining time before and after inserting artificial tuples. Figure 5 reports the overhead at the server side for the SC-based approach on $USPS$ dataset, with various k values. We notice that for the same α, the mining overhead increases with k, since larger k values will slow the mining process. The mining overhead also increases with larger α values, as there are more artificial tuples to be inserted. However, the overhead is always very small; it is always less than 9%.

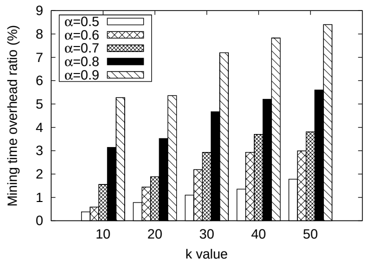
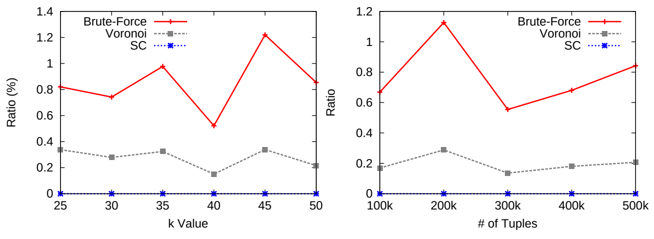


Figure 5: Mining overhead at server side ($USPS$ dataset)

6.5 Outsourcing Versus Mining Locally We compare the time performance of outsourcing mining while executing verification locally with that of mining locally. We measure the ratio $r = T'/T$, where T' is the time of outsourcing mining while executing verification locally, including the total time of verification preparation (if there is any) and verification, and T is the time of executing k-means locally. Figure 6 (a) and (b) show the result when we vary k value and dataset size respectively. It shows that our verification approaches only



(a) Various # of clusters (dataset size: $100K$)

(b) Various dataset sizes ($k = 50$)

Figure 6: Verification time vs mining time ($Ottawa$ dataset, $\alpha, \beta = 0.9$)

takes at most 1.2% time of mining locally. This proves that it is possible that the client of weak computational power can outsource expensive data mining computations to third-party service providers, while verifying the correctness of the mining result with high integrity guarantee and cost that is much cheaper than mining by herself.

7 Related Work

The issue of providing verification assurance for outsourced database management was initially raised in the database-as-a-service (DAS) paradigm [12]. The goal is to assure the correctness and completeness of SQL query evaluation over the outsourced databases. The proposed solutions include Merkle hash trees [14], signatures on a chain of paired tuples [19], and counterfeit records [25] for SQL point and range queries. These techniques cannot be used directly in the data-mining-as-a-service ($DMAS$) system for verification of data mining results.

The problem of protecting the data confidentiality and pattern privacy has caught some attention recently [18, 21, 23]. The main solution is to use encryption techniques (possible with artificial records added) on the outsourced dataset so that even with some adversary background knowledge, the attacker's probability of decrypting the encrypted dataset (and thus the patterns) is always no less than a given threshold.

On the other hand, enforcing result integrity, another security issue of the $DMAS$ paradigm, has been rarely studied. Wong et al. [24] propose the verification techniques for outsourcing of frequent itemset mining. They generate an artificial database such that all itemsets in the database are guaranteed to be frequent and their exact support counts are known. By hosting the artificial database with the original one and checking whether the server has returned all artificial itemsets, the data owner can verify whether the server has returned correct and complete frequent itemsets. To our best knowledge, Wong et al. [24] are the first (and the only) authors that address the verification issue of the outsourced data mining computations. However, their techniques on frequent itemset mining cannot be directly applied to k-means clustering.

There is a line of work on verifiable computations by using interactive proofs [10] and probabilistically checkable proofs (PCPs) [4, 9]. Unfortunately, this body of theory is impractical [20], due to the complexity of the algorithms and difficulty to use general-purpose cryptographic techniques for data mining problems.

8 Conclusion

In this paper, we investigated how to provide integrity guarantee for k-means clustering that is outsourced to infrastructure-as-a-service ($IaaS$) providers. We considered the service provider that may return cheap and incorrect clustering result by terminating the mining early. Along this line, we developed both deterministic and probabilistic integrity verification techniques that can provide high correctness guarantees with complexity much cheaper than that of k-means clustering. As demonstrated in the experiments, our verification methods can effectively and efficiently capture the sloppy service provider.

Regarding the future work, an interesting research direction is to adapt our verification techniques to other clustering methods, e.g., k-medoids clustering method. It is also interesting to explore how to define a budget-driven model to allow the client to specify her verification needs in terms of budget (possibly in monetary format) besides the (α, β)-correctness requirement.

Acknowledgement

This research was partially supported by grants from National Science Foundation (NSF) via grant numbers CCF-1018151 and DUE-1241315.

References

[1] Amazon Elastic Compute Cloud (Amazon EC2). http://aws.amazon.com/ec2/.

[2] Google Prediction APIs. http://code.google.com/apis/predict/.

[3] D. Aloise, A. Deshpande, P. Hansen, and P. Popat. Np-hardness of euclidean sum-of-squares clustering. *Machine Learning*, 75:245–248, May 2009.

[4] S. Arora, C. Lund, R. Motwani, M. Sudan, and M. Szegedy. Proof verification and the hardness of approximation problems. *Journal of ACM*, 45:501–555, May 1998.

[5] D. Arthur and S. Vassilvitskii. How slow is the k-means method? In *Proceedings of the 22nd annual symposium on Computational geometry*, 2006.

[6] R. Buyya, C. S. Yeo, and S. Venugopal. Market-oriented cloud computing: Vision, hype, and reality for delivering it services as computing utilities. In *HPCC*, 2008.

[7] B. Chazelle. An optimal convex hull algorithm in any fixed dimension. *Discrete & Computational Geometry*, 10:377–409, 1993.

[8] M. De Berg, M. Van Kreveld, M. Overmars, and O. Schwarzkopf. *Computational Geometry: Algorithms and Applications*. Springer, 1997.

[9] R. Gennaro, C. Gentry, and B. Parno. Non-interactive verifiable computing: outsourcing computation to untrusted workers. In *CRYPTO*, 2010.

[10] S. Goldwasser, S. Micali, and C. Rackoff. The knowledge complexity of interactive proof systems. *SIAM Journal of Computing*, 18:186–208, 1989.

[11] R. Gopalan and J. Sankaranarayanan. Max-margin clustering: Detecting margins from projections of points on lines. In *IEEE Conf. on Computer Vision and Pattern Recognition*, 2011.

[12] H. Hacigümüş, B. Iyer, C. Li, and S. Mehrotra. Executing SQL over encrypted data in the database-service-provider model. In *SIGMOD*, 2002.

[13] S. Har-Peled and B. Sadri. How fast is the k-means method? In *SODA*, 2005.

[14] F. Li, M. Hadjieleftheriou, G. Kollios, and L. Reyzin. Dynamic authenticated index structures for outsourced databases. In *SIGMOD*, 2006.

[15] R. Li, H. Wang, P. Mordohai, and H. Xiong. Integrity verification of k-means clustering outsourced to infrastructure as a service (IaaS) providers (Full Paper). *http://www.cs.stevens.edu/ hwang4/SDM13-clusterIntegrity-full.pdf*

[16] S. P. Lloyd. Least squares quantization in pcm. *IEEE Transactions on Information Theory*, 28, 1982.

[17] M. Mahajan, P. Nimbhorkar, and K. Varadarajan. The planar k-means problem is NP-hard. In *Proceedings of the 3rd International Workshop on Algorithms and Computation*, 2009.

[18] I. Molloy, N. Li, and T. Li. On the (in)security and (im)practicality of outsourcing precise association rule mining. In *ICDM*, 2009.

[19] H. Pang, A. Jain, K. Ramamritham, and K.-L. Tan. Verifying completeness of relational query results in data publishing. In *SIGMOD*, 2005.

[20] S. Setty, A. J. Blumberg, and M. Walfish. Toward practical and unconditional verification of remote computations. In *Proceedings of the 13th Workshop on Hot Topics in Operating Systems (HotOS)*, 2011.

[21] C.-H. Tai, P. S. Yu, and M.-S. Chen. k-support anonymity based on pseudo taxonomy for outsourcing of frequent itemset mining. In *SIGKDD*, 2010.

[22] A. Vattani. k-means requires exponentially many iterations even in the plane. In *Proceedings of the 25th annual symposium on computational geometry*, 2009.

[23] W. K. Wong, D. W. Cheung, E. Hung, B. Kao, and N. Mamoulis. Security in outsourcing of association rule mining. In *VLDB*, 2007.

[24] W. K. Wong, D. W. Cheung, B. Kao, E. Hung, and N. Mamoulis. An audit environment for outsourcing of frequent itemset mining. In *PVLDB*, volume 2, 2009.

[25] M. Xie, H. Wang, J. Yin, and X. Meng. Integrity auditing of outsourced data. In *VLDB*, 2007.