K-means and adaptive k-means algorithms for clustering DNS traffic

Qinghui Xu, Daniel Migault, Stéphane Sénécal, Stanislas Francfort
Orange Labs
38-40 rue du Général Leclerc
92794 Issy-les-Moulineaux CEDEX 9
{qinghui.xu, daniel.migault, stephane.senecal,
stanislas.francfort}@orange-ftgroup.com

ABSTRACT

Internet Service Providers' DNS traffic can be up to 120000 queries per second and increases around 8% every month. DNSSEC is expected to replace DNS and brings new challenge to naming resolution with heavy signature check. This paper provides an architecture, where incoming DNS traffic is split according to the DNS query rather than to its IP address, in order to minimize the number of signature checks. To split DNS traffic among the different nodes of the platform, k-means clustering algorithms are considered. This paper proposes an enhancement of the standard algorithm: an adaptive k-means and compares performance of both methods on simulated data from a Gaussian mixture model and on real DNS traffic data from the Orange IP network.

Keywords

Telecommunication network architecture, DNS, DNSSEC, routing, data mining, clustering, k-means.

1. INTRODUCTION

Domain Name Server (DNS, [1, 2]) is a global hierarchical database that associates information to domain names. A common use case is when DNS binds an IP address to a domain name, also called Fully Qualified Domain Name (FQDN). This makes websurfing easier as end users can type on their web browsers *www.orange.com* rather than *194.2.208.16*. When the end user types *www.orange.com*, the web browser sends a DNS query on the Internet to get the IP address of this website. Once the browser get the DNS response, it can initiate an HTTP connection and show the information expected by the end user. This is why we commonly say that DNS makes communication between names possible. DNS is involved in many end users applications such as web browsing or e-mail but the core nework also relies on DNS mechanisms.

Very popular websites like *www.google.com*, Content Distributed Network (CDN), or Video on Demand (VoD) services may use DNS to balance the traffic load between the servers and redirect the traffic, for example, to either the closest node or to the least busy one in order to enhance the Quality of Service (QoS). In fact, in the case of several IP addresses for one domain, the end user will randomly choose one or another, and the HTTP traffic will thus be split between the different servers. Very popular websites with multiple servers can then provide responses with a subset of IP addresses, usually those with the least activity, or those that are closer to the end user so to optimize the network resources. A consequence of using DNS for traffic management is that information is dynamically configured and thus data are valid only for a limited period of time. This, in conjunction with an increasing number of end users on the Internet as well as the increasing number of services available on the Internet, makes the DNS traffic to grow around 8% per month, with a rush up to 120000 DNS queries per second.

Moreover, in July 2008, Kaminsky [3] showed that DNS is sensitive to cache poisoning, and that DNSSEC [4, 5, 6] can be a long term solution. DNSSEC stands for DNS SECurity extension and provides mechanisms so that the resolver can authenticate the DNS data, i.e. can prove that the response has not been alterated during the network transmission and is really corresponding to the data hosted by the authoritative entity. This is achieved by transmitting the signature associated to the DNS data. With the signature and the public key of the server, the end user is able to authenticate the DNS data. Since DNS is hierarchical and distributed, DNSSEC must provide as well mechanisms to build a chain of trust between servers. In fact, DNSSEC should provide solutions to transfer the trust from a trusted point to its sons.

Internet Service Providers (ISP) are using DNS to make services available to their end users. In this sense, they are taking advantage of DNSSEC by protecting their services from cache poisoning attacks, and by protecting their end users from identity thefts. On the other hand, by providing Internet access, ISP provides not only IP routing services, but also naming resolution services. While DNSSEC responses carry signatures which make them larger than DNS responses, from the end user point of view, this increases network latency, and so QoS requirements. On the server side, manipulating longer responses increases the processing

VALUETOOLS 2011, May 16-20, Paris, France
Copyright © 2011 ICST 978-1-936968-09-1
DOI 10.4108/icst.valuetools.2011.245598

time, as well as memory consumption. Moreover, DNSSEC responses may require one or more signature checks. Reference [7] provides a performance point of view on DNSSEC migration with different implementations and shows that, without signature validation, the maximum load of resolving servers can be decreased by up to 14% with no signature validation and by up to 50% when signature validation is performed.

DNSSEC clearly impacts performances of resolving platforms, and actually DNSSEC completely makes its characteristics different from DNS. This makes DNSSEC to be considered more as a new protocol requiring a new design for resolving platform rather than an extension of DNS. In fact, DNS has been designed with small and costless responses. Such assumption has led to the design of current DNS resolving platforms, which are designed not to minimize the number of resolutions over the Internet, but to balance the traffic load between its nodes. With DNSSEC, signature checks and longer responses represent the cost of a resolution. DNSSEC resolving architecture should be designed so as to minimize the number of resolutions performed by the resolving platform. One solution presented in this paper is to consider that each node of the platform is responsible for resolving a given set of domain names. The traffic is thus split according to the domain names instead of IP layer information. In this paper we consider that end users queries come to a load balancing device that will redirect each query to its appropriate server node. Such a node is called a responsible node.

Data mining literature provides multiple methods to cluster data: combinatorial methods [8, 9] (k-means, k-medoids, hierarchical clustering), mixture modeling [10] (maximum likelihood or Bayesian inference via the Expectation-Maximization (EM) algorithm for instance [11, 12]), mode seeking [9]. In this paper we considered the unsupervised k-means algorithm. We adopted an unsupervised method as we seek to minimize the decision processes of the load balancer that redirect queries to the responsible node. Unsupervised methods provides the advantage to configure static table, whereas supervised methods go through multiple operations. Although the static table may change over time, we assume that they remain static during a given time and should be only updated after this period. Furthermore, the standard k-means algorithm is easily scalable with the large volume of data we need to process for our application.

In this paper we propose an adaptive extension of this algorithm in order to perform the clustering task. This extension makes it possible to grasp some latent deep-layer structure or more complexity in the data which can be critical. Since the proposed method is an extension of the standard k-means, it is equivalent to the standard algorithm when a parameter of the algorithm is appropriately tuned. We tested and compared these two clustering methods on a set of simulated data consisting of a two-dimensional Gaussian mixture with three components, and on a DNS traffic data timeframe extracted from the Orange IP network.

The paper is thus organized as follows. Section 2 describes current DNS resolving architecture and designs an architecture that fits DNSSEC requirements. In section 3 the k-means algorithm is briefly recalled and its proposed adaptive extension is presented. The numerical experiments to illustrate the performance of the algorithms are shown in section 4. Finally section 5 concludes the paper and provides perspectives of this work.

2. DNS RESOLVING PLATFORM ARCHITECTURES

2.1 DNS architecture

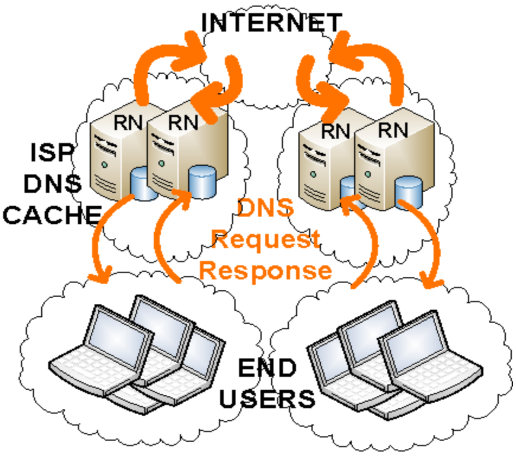

Figure 1: DNS architecture.

Figure 1 depicts current architectures for a DNS resolving platform. The traffic is split according to the IP address of the end user between the different nodes of the platform. Nodes are independent and proceed to a DNS resolution for each incoming query. More specifically, the node look first if the response is in the cache, and if the response is not proceeds to a resolution over the Internet.

In such architecture, nodes are independent, and there is no cooperation between the different nodes. Since traffic is split between the nodes according to the IP layer, popular domain names are being resolved on almost all the nodes. More specifically, when a node receives a query and does not have the response in its cache, it proceeds to a DNS resolution over the Internet. Alternatives that would avoid a DNS resolution on the Internet are:

- an unsupervised architecture: the receiving node can ask whether another node has the response in its cache, and get the response from it, or forward the query to that node.

- a supervised architecture: domain names are associated to nodes, each of them is responsible of a subset of the domain names. Furthermore, all nodes have to know for a given domain name which node is responsible for it. With such a rule, queries can be directly sent to the responsible node, or nodes can forward, or get the response from the responsible node.

Clearly the unsupervised architecture may avoid DNS resolution on the Internet, but would overload the platform. The supervised method requires additional processing that may be equivalent as proceeding to a resolution over the

Internet. Furthermore, it does not reduce the number of exchanges. On the other hand defining a splitting rule and synchronizing the nodes with such a rule add much more complexity.

In other words, cooperation between nodes in a resolving platform should be considered only when the resolution process represents a heavy cost compared to message exchanges. Otherwise, as with DNS, the platform should rather be scalable and designed to accept all incoming traffic, rather than designed to minimize the number of resolutions. This is why accepting all requests regardless of the domain name is the architecture that best fit the DNS protocol.

2.2 DNSSEC architecture

With DNSSEC, resolution involves signature checks which require much more CPU time and memory consumption. Reference [7] shows that for a DNS traffic involving a single signature check per domain name, the signature check is equivalent to 45% (resp. 70%) of the maximum load for routine BIND (resp. UNBOUND). This means that if the responses are stored in the cache, the routine BIND can treat 1.82 (resp. 3.33) times more queries.

Furthermore, reference [7] shows that the number of added queries increases exponentially with the Cache Hit Rate (CHR). For instance, changing the CHR from 0 to 70% increases the number of queries which can be processed by the platform by 250%.

Thus the benefit of CHR in term of performance motivates to investigate how we could split the DNS traffic between the nodes according to the domain name value as presented by figure 2.

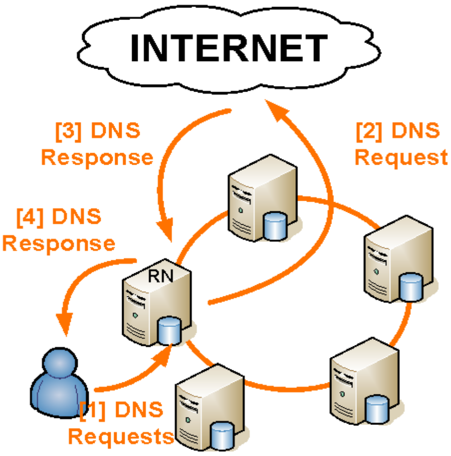

Figure 2: DNSSEC architecture.

2.3 Clustering DNS traffic data

To split the DNS traffic between the nodes of the platform, we consider the space of all domain names involved in a traffic capture. We assume that this capture is representative of the running DNS traffic. The basic idea is to associate different criteria (query rate, length of the query/response, length, number of signature check to perform among others) and then derive costs. For each domain name, we can either assign one global cost, however, since criteria are quite heterogeneous, we might have to derive multiple costs. From the costs repartition, we define classes of the different costs. We use the clustering algorithm to define the number of groups as well as for assigning a group to each domain name.

To illustrate our purpose, let us consider the single criterion case where the *query rate* is associated to the domain name. Applying a clustering algorithm to this criterion may lead to consider the groups *HeavilyRequested*, *RegularlyRequested* and *RarelyRequested*. If we want the resources to be equally distributed among the different nodes, then we should attempt to assign to each node an equal number of domain name from each class.
If we are considering different criteria, we can apply the clustering algorithm for each criterion, and then consider all classes intersections. If n_i is the number of groups for criteria κ_i, and $K = (\kappa_1, \ldots, \kappa_i, \ldots, \kappa_d)$ the list of criteria we consider, then the total number of classes to consider is given by $\prod_{\kappa_i \in K} n_i$.
Another possibility is to consider domain names as vectors in $\mathbb{R}^d$ where d is the number of criteria we consider for each domain name. When we apply a clustering algorithm to domain names represented as vectors, we are looking for domain names with similar characteristics. It becomes then quite hard to compare the cost between the different groups. However, by spreading equally the various groups among the different nodes of the resolving platform, we uniformly distribute the cost among the different nodes.

All criteria are measured for each Fully Qualified Domain Name (FQDN). Measured criteria are expected to provide metrics for cost at the network layer (from the network cards and drivers) as well as at the application layer (for CPU processing time and memory consumption). In this paper we considered the following criteria:

- End User Query Rate (EUQR) measures the number of DNS queries sent by the end user to the resolving platform.
- Resolver Query Rate (REQR) measures the number of resolutions performed on the Internet by the resolver for a given FQDN. Resolution occurs when a cache miss occurs.
- Response Time (RT) measures the response time over the Internet.
- End User Bit Rate (EUBR) measures the number of bits sent by the end user to the resolving platform.
- Resolver Bit Rate (REBR) measures the number of bits sent by the platform on the Internet.
- Transfert Bit Rate (TBR) measures the total bit rate associated to a specific FQDN.
- Client Query Response Time (CQRT) measures the mean time an end user has to wait before receiving an answer.
- REsolver OCCupancy time (REOCC) (respectively End User OCCupancy time (EUOCC) and Total OCCupancy time (TOCC)) measures the time contexts opened for the associated FQDN.

- Cache Hit Rate (CHR) measures the probability that the FQDN is already cached.
- OKR (resp. NOKR) measures the rate of correct resolutions (resp. resolutions with error) for the associated FQDN.

Since these criteria are compactly correlated, which will be illustrated in the numerical experiments, it is more interesting to consider this latter point of view, *i.e.* to apply a joint clustering methodology on our data sets.

3. ADAPTIVE K-MEANS

Cluster analysis is a very practical subject in the rapidly growing field known as exploratory data analysis and is being applied in a variety of engineering and scientific disciplines such as biology, psychology, marketing, computing science for instance. For our research, we focus on clustering in data mining. In fact, data mining adds to clustering the complications of very large datasets with many attributes of different types.

Clustering is a division of data into groups of similar objects. Intuitively, patterns within a valid cluster are more similar to each other than they are to a pattern belonging to a different cluster. Data mining deals with large databases that impose on clustering analysis additional severe computational requirements. Note that different applications make use of different data types, such as continuous variables, discrete variables, similarities, and dissimilarities. Therefore, one needs different clustering methods in order to adapt to the kind of application and the type of clusters sought. In our case, we tested some clustering techniques such as k-means, hierarchical clustering, self-organizing maps and principal component analysis to find out the most efficient method for our needs. It comes out that the k-means algorithm suits the best our application.

3.1 Standard k-means

Among the clustering methods, k-means is one of the most widely applied algorithm [8, 9]. The basic idea is that samples grouped in each cluster are more closely related one to each other than to those assigned to other clusters. In the classical k-means algorithm, this similiarity relation between two samples is defined by the Euclidean distance. Samples are thus assigned to the cluster whose centroid is the closest to them, in the sense of the Euclidean distance. Considering a set of samples $\{x_i, i = 1, 2, ...N\}$ in the space $\mathbb{R}^d$, and a set of clusters centroids $c_1, c_2, ...c_\ell$, then the criteria for assigning the sample x_i to a cluster is to choose the centroid c_j minimizing the formula

$$||x_i - c_j||^2 = \sum_{k=1}^{d} (x_{ik} - c_{jk})^2 \qquad (1)$$

The standard k-means algorithm is described in details in [8, 9]. With similarity function derived from (1), each coordinate has the same importance. However, this may be not efficient if there is not an appropriate mesurement of each coordinate. As an illustration, in our application, traffic samples have multiple dimensions: “end user query number”, “internet resolution time”, “plateform resolution time”, etc. A second or a millisecond can be used as the resolution times measurement unit, and the query number can be measured in milliseconds, seconds or in an other time interval (statistically, the query number is proportional to the measured time). With these different combinations of measurement units above, the results of the standard k-means algorithm will change for sure because of the variance of the measurements.

Let us consider for instance a set of data generated from a three components two-dimensional Gaussian mixture model [10], where the samples from each component are plotted respectively in red, violet or blue on figure 3. The samples are firstly measured in their proper units, then we take another method of measurement such that the vertical coordinate is compressed by a factor 1/60. Imagine for instance that the appropriate measurement unit is the second, but for the moment we use the minute instead, obviously the numerical value obtained is 1/60 of the original one. Now let us apply the k-means algorithm to find the three clusters (i.e. the three components of the mixture), which are represented respectively by “circle”, “square” and “diamond” symbols.

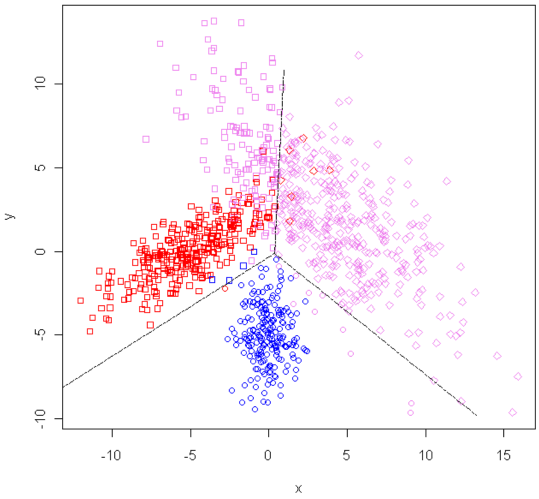

Figure 3: k-means applied to the samples of a Gaussian mixture model.

Figure 3 depicts the k-means clustering for the original Gaussian mixture data, and figure 4 depicts the k-means clustering obtained for the same data but where the vertical measurement is compressed to 1/60. For a clearer demonstration, we trace the separators between the clusters after plotting. The result of the clustering in the first case is acceptable, but in the other case when the vertical dimension is compressed to 1/60, the horizontal dimension absolutely dominates the clustering, leading to quasi-parallel vertical separators, which is not pertinent.

Also, even when the variables are measured appropriately, it is still possible that the k-means clustering does not reflect exactly the inherent structures of the data. In such contexts, the Euclidean distance to the centroids may not be relevant.

Now, let us consider another approach to tackle this prob-

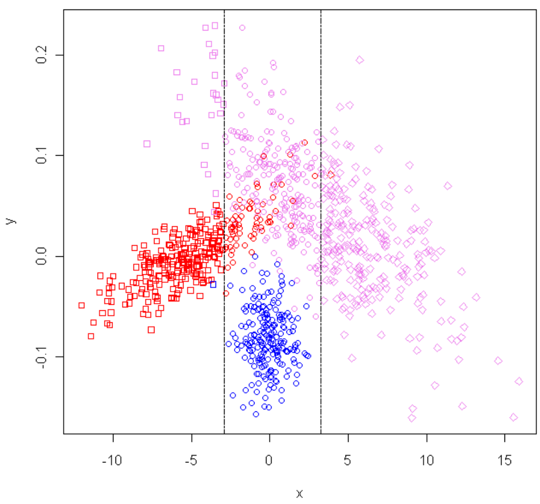

Figure 4: k-means applied to the samples of a Gaussian mixture model, vertically compressed.

lem. Considering a Gaussian mixture model, determining a cluster for each sample according to the mixture distribution demands a more sophisticated method than the classical k-means algorithm. For example, the EM-type soft k-means clustering is proposed in [9], which estimates the covariances and prior probability in the Expectation-Maximization procedure [11, 12]. In the following we propose a new EM-type method, inspired from the quadratic discriminant analysis (QDA) [13] and linear discriminant analysis (LDA) classification procedures [14, 15].

3.2 Improved k-means: adaptive k-means

In QDA, data are classified to the nearest center, after the compensation of the prior probabilities and using some spheric metric which spheres data according to the covariance matrix. Let Ω denote the within-class covariance. In QDA data are first sphered with respect to Ω and then the target sample is classified to the nearest centroid with the compensation of the prior membership probabilities.
We propose to estimate the within-class covariance Ω for each cluster, and then use it to construct a local metric which approximately behaves as the QDA metric.

The proposed algorithm consists essentially of the following steps:

1. Initialization of the clusters. First of all, we compute the covariance matrix of the sample set Σ, with

$$c = \frac{1}{N}\sum_{i=1}^{N} x_i \tag{2}$$

$$\Sigma = \frac{1}{N-1}\left(\sum_{i=1}^{N}(x_i - c)(x_i - c)^T\right) + \epsilon_1 \tag{3}$$

   Data are sphered with this metric. Initially, several points are chosen (uniformly) randomly among the samples, and all samples are divided into groups which contain the nearest chosen point. The centroids of each group are taken as the initial centroids.

2. Expectation. Estimate the covariance with each group's covariance matrix. For group G_j with the centroid c_j,

$$\Omega_j = \frac{1}{\#G_j - 1}\left(\sum_{x_i \in G_j}(x_i - c_j)(x_i - c_j)^T\right) + \epsilon_2 \tag{4}$$

3. Maximization. Compute the distance between each point and each centroid with the metric Ω_j.

$$(x_i - c_j)^T \Omega_j^{-1}(x_i - c_j) \tag{5}$$

   Classify the point to the cluster with the nearest centroid. Recompute the centroid for each group.

4. Iterate Step 2 and Step 3.

5. Stop when the centroids do not change anymore.

It should be noted that the covariances matrices mentioned above could be singular, which leads to an unstable result, thus the covariance matrices are amended with regularization factors ϵ_1 and ϵ_2. We should notice that these regularization factors should not be too important in comparison with covariances' eigenvalues. In fact, when regularization factors tend to infinity, the covariances are negligible, the method reduces exactly to the standard k-means.

Let us consider the same model and data mentioned above, applying the adaptive version of the k-means.

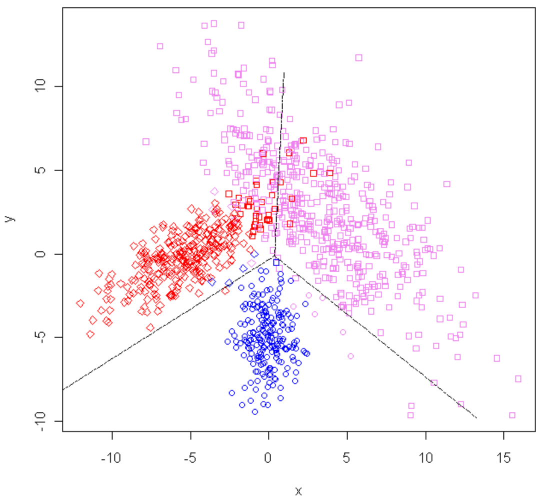

Figure 5: Samples of Gaussian mixture model, with adaptive k-means clustering.

Figure 5 is the result under the appropriate measurement, and figure 6 depicts the result under the condition that the vertical value is compressed to 1/60. On these two figures, we still trace the separators of the standard k-means for comparison. We get a satisfying result in both the appropriate measurement and the other case, furthermore, clustering is enhanced in the first case in the manner that each cluster corresponds to one component of the Gaussian mixture.

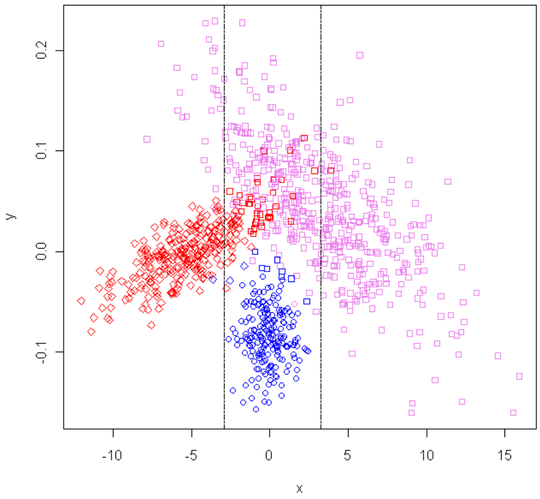

Figure 6: Samples of Gaussian mixture model, vertically compressed, with adaptive k-means clustering.

4. NUMERICAL EXPERIMENTS

In this section, numerical results of the clustering algorithms introduced in section 3 are studied. These experiments consist of two parts, one with simulated data generated by a Gaussian mixture distribution in a two-dimensional space, the other with the DNS traffic data extracted from Orange IP traffic, which is not Gaussian. We introduce an error model and measure associated error rates only for the Gaussian case.

4.1 Gaussian mixture model

To compare the standard k-means clustering and the proposed adaptive k-means clustering, we should evaluate the clusters output from both algorithms. With the simulated Gaussian mixture data, we propose a performance measure as an evaluation of our satisfaction for the clustering. Thanks to the "oracle" model that we consider (components for the samples are known), we can define two different types of "error" in the cluster, first let us examine the samples pair by pair:

- **Error of mixture**: if two samples from different components are grouped in the same cluster, this pair of samples is an error of mixture.
- **Error of separation**: if two samples from one component are grouped into two different cluster, this pair of samples is an error of separation.

Otherwise, a pair of samples not satisfying neither of the two situation mentioned above is considered as correctly grouped. We can define the sum of Error of mixture and Error of separation as the **total error rate** of "misclassification".

From this definition, we can now evaluate the performance of a clustering algorithm. In our instance, we have $N = 1000$ samples, so we should check all the $N(N-1)$ possible pairs of samples to compute the error rate, which means that the so defined error rate will probably be a large number compared to N. If we want to have a more precise idea about this error rate, and its "relative" level, we should estimate its upper bound and lower bound corresponding to the worst and the best clustering. As an illustration, we consider the following model.

Our N samples data are generated from a three component Gaussian mixture distribution, and the objective is to divide the samples into three groups (ideally the three original components):

$$\sum_{m=1}^{3} w_m \mathcal{N}(\mu_m, \Sigma_m) \tag{6}$$

The most naive clustering is when we assign uniformly randomly each sample to a group, which means that a sample has a propability $1/3$ to be assigned to each group. Finally each group would contain $\frac{1}{3}w_1 N$ samples from component 1, $\frac{1}{3}w_2 N$ samples from component 2, and $\frac{1}{3}w_3 N$ samples from component 3. We take a sample from component 1 for example, with any sample from component 2 or from component 3 which is assigned to the same group, the so formed couple will produce a mixture error. Otherwise, with any sample from component 1 which is assigned to other groups, such a couple will produce a separate error. The errors due to this sample are thus as follows:

- mixture error due to a sample from component 1:

$$\frac{1}{3}w_2 N + \frac{1}{3}w_3 N \tag{7}$$

- separate error due to a sample from component 1:

$$\frac{2}{3}w_1 N \tag{8}$$

We can also compute errors due to other samples in the same way, if we sum them up, we will get the error rates. Since each pair of samples is counted twice, the sum should be divided by 2 to obtain the error rates:

$$\text{mixture error} = \frac{1}{3}(w_1 w_2 + w_2 w_3 + w_3 w_1)N^2 \tag{9}$$

$$\text{separate error} = \frac{1}{3}(w_1^2 + w_2^2 + w_3^2)N^2 \tag{10}$$

$$\text{total error} = \frac{1}{6}(1 + w_1^2 + w_2^2 + w_3^2)N^2 \tag{11}$$

On the other hand, if the clustering corresponds exactly to the labeled components, which means no error appears, the error rate will be 0. But here the lower bound is to estimate an "unperfect" clustering, which means there is at least 1 sample assigned to a "wrong" group. So the lower bound is estimated by the case where there is only one sample badly grouped. So the lower bound is estimated by:

$$\text{min mixture error} = \min\{w_1, w_2, w_3\}N \tag{12}$$

$$\text{min separate error} = \min\{w_1, w_2, w_3\}N \tag{13}$$

$$\text{min total error} = (1 - \max\{w_1, w_2, w_3\})N \tag{14}$$

Let us test the algorithms with the following instance for model (6): prior probability $w = (0.3, 0.2, 0.5)$, the component centroids:

$$(\mu_1, \mu_2, \mu_3) = \left(\begin{pmatrix} -5 \\ 0 \end{pmatrix}, \begin{pmatrix} 3 \\ 3 \end{pmatrix}, \begin{pmatrix} 0 \\ -5 \end{pmatrix} \right) \tag{15}$$

and covariances:

$$\Sigma_1 = \begin{pmatrix} 7.07 & 3.47 \\ 3.47 & 3.09 \end{pmatrix} \tag{16}$$

$$\Sigma_2 = \begin{pmatrix} 13.49 & -10.33 \\ -10.33 & 13.53 \end{pmatrix} \tag{17}$$

$$\Sigma_3 = \begin{pmatrix} 1.09 & -0.13 \\ -0.13 & 4.17 \end{pmatrix} \tag{18}$$

Case I.

Firstly, we examine the results under the appropriate measurement. With the standard k-means procedure and the adaptive k-means algorithm, we obtain the following error rates:

Method	k-means	adaptive k-means
Error of mixture	12417	7443
Error of separation	11692	9817
Total error	24109	17260

For sure the proposed adaptive method enhances the performance of clustering, we can not only perceive the improvement from the error rate, but also we have an intuitive impression from the figure 7. Samples from three components are plotted with red, violet and blue, the standard k-means divides samples with linear separators, while the adaptive k-means clusters samples into three groups: "circle", "square", and "diamond". It is noticed that clustering performance is improved particularly on components of type "head or tail".

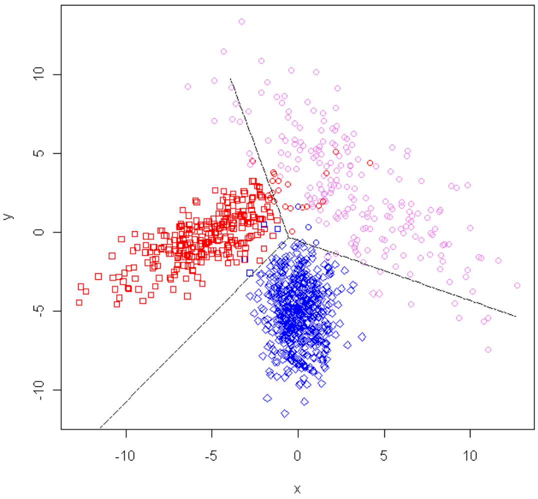

Figure 7: k-means and adaptive k-means clustering, on 1000 samples from Gaussian mixture model (6).

Case II.

Secondly, we tested these methods under an inappropriate measurement, with the numerical value of the vertical dimension multiplied by a factor 1/100.

Method	k-means	adaptive k-means
Error of mixture	91445	13370
Error of separation	26176	21764
Total error	117621	35134

We also visualize the result by plotting the "circle", "square", and "diamond" clusters and tracing the standard k-means linear separators on the same figure 8.

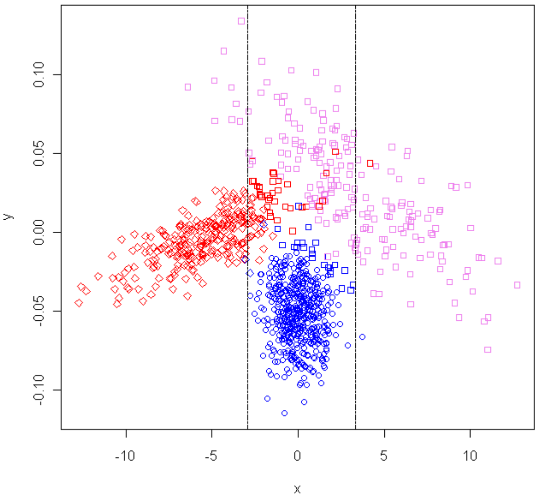

Figure 8: k-means and adaptive k-means clustering, on 1000 samples from Gaussian mixture distribution, vertically compressed by factor 1/100.

Comparing this result with the precedent, the robustness of our proposed adaptive k-means is remarkable, which leads to similar clustering results under both measurement scales.

Furthermore, after the error rates that we obtained, we would like to compare them with the lower bound and upper bound of the error rate. Calculating the total error rate's upper and lower bounds with the formula (11) and (14), we obtain this estimation:

$$500 \leq \text{total error} \leq 230000 \tag{19}$$

Knowing that one badly grouped sample will cause an error rate 500, we can estimate the number of badly grouped samples by dividing the total error rate by 500. Now we can estimate that under the appropriate measurement, k-means clustering has made wrong decisions on about 50 samples, and adaptive k-means clustering on about 35 samples, for this instance in **case I**.

When the vertical dimension is compressed by a factor 1/100, k-means clustering has made wrong decisions on about 235 samples, while adaptive method on about 70 samples, for this instance in **case II**.

In the worst situation, if we cluster the samples randomly, we will make bad decisions on about 460 samples, which is a little less than a half of the entire set.

Furthermore, it would be interesting to investigate the issue of convergence. We know that the standard k-means iterations finally converge to a local minimum of the within-class variance, and the procedure normaly stops after about 20 iterations [9]. With the proposed adaptive method, we observe the convergence is as quick as the standard k-means clustering, it also stops around 20 iterations in our tests.

In the discussion of classical k-means method, the notion of within-class variance is our principal interest. Minimizing the within-class variance is equivalent to reducing the dissimilarity within each cluster and at the same time enlarging the difference between clusters. We should note that within-class variance is indeed the square sum of Euclidean distances between samples and centroids. In the adaptive method, this notion is replaced by the normalized within-class variance, which is the square sum of distance between samples and centroids, under the metric defined in 3.2. In the numerical experiments, the normalized within-class variance is reduced with the iterations. We take **Case I** for illustration, under the appropriate measurement, the normalized within-class variance in the adaptive k-means procedure is depicted in figure 9, in which the variance decreases until the convergence.

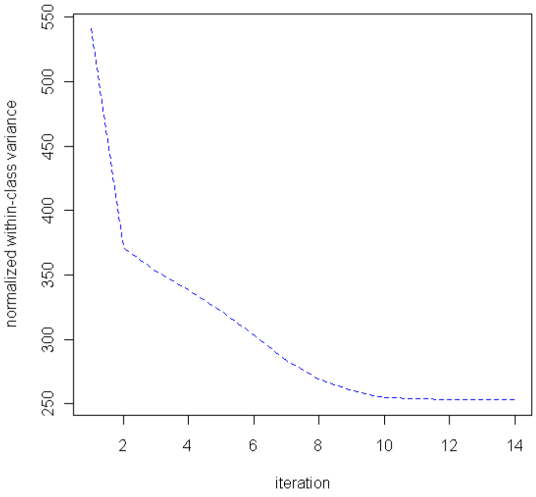

Figure 9: Normalized within-class variance with respect to iterations, for the adaptive k-means clustering of 1000 samples from Gaussian mixture model.

4.2 DNS traffic data

We have considered 16 measured parameters for FQDNs over a timeframe of 30 seconds of data traffic, leading to 167793 samples, as explained in subsection 2.3. These parameters includes the End User Request Number, the DNSSEC SIG Number, (second-order) statistics for the Internet and Platform Resolution Times, and for the TTL, the End User IP Address, the Error Code Response and the Error Code. These parameters are highly correlated one with each other as we can see on figure 10. The correlation among the 16 variables is remarkable, especially the relationships between the "EUQR" (end user query rate) variable and the other variables can always be approximated by two segments, so we prefer to process the clustering on the whole data set than respectively on each variable, as mentioned in subsection 2.3.

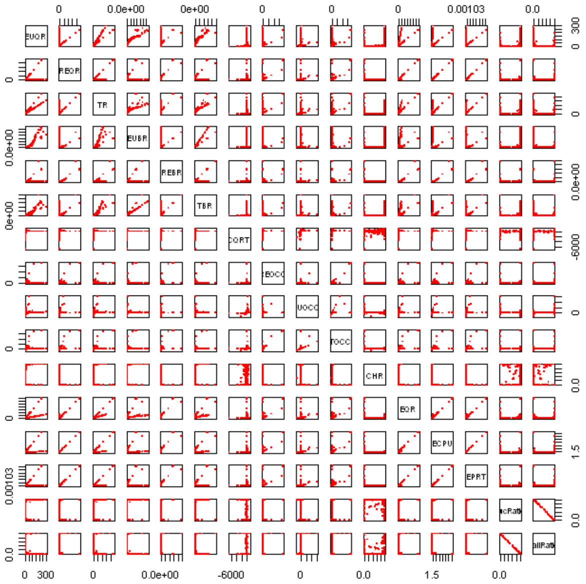

Figure 10: Plots of pairs of variables for the 16 parameters of DNS traffic data set.

When applied to these data, both standard k-means and the proposed adaptive k-means algorithm lead to the grouping of data into four clusters, from the Ward stopping criterion [9]. Standard k-means algorithm groups the samples in sets respectively of 19, 95, 167650 and 29 samples. On the other hand, the adaptive k-means algorithm groups the samples in sets respectively of 7, 57, 20 and 167709 samples.

The difference between the two methods lies in the respectively obtained within-class variance of the clusters. For the standard k-means algorithm, the within-class variance of the clusters is given by

$$(2.54 \times 10^{17}, 4.33 \times 10^{16}, 2.13 \times 10^{16}, 6.26 \times 10^{16}) \qquad (20)$$

after convergence.
The proposed method makes it possible to reduce drastically the within-class variance of the clusters. Figure 11 depicts the within-class variance of the clusters with respect to the iterations for the proposed adaptive k-means algorithm. In this instance, the adaptive algorithm converges after 16 iterations.

Also the clustering obtained from the method we proposed lead to fully connected groups for two-dimensional slices representations, on the contrary to the clustering obtained for the k-means algorithm, as shown on figures 12 and 13. We depict the two clusterings on these figures, for simplicity we only depict in 4 × 4 scatterplot matrices, with color blue, red, green and violet representing the four different groups.

It should be noted that both methods find some groups of type "Heavily Requested", "Regularly Requested" and "Rarely Requested", and that the "Heavily Requested" domain

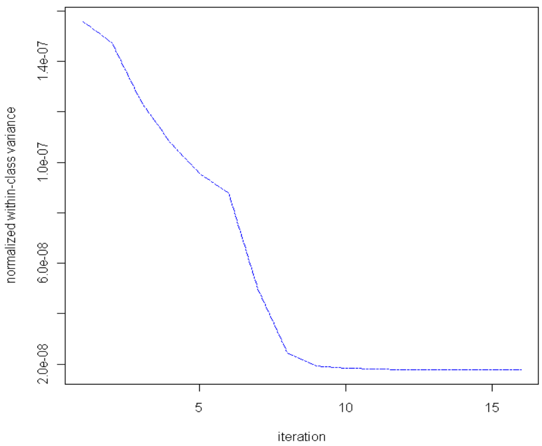

Figure 11: Within-class variance of the clusters with respect to iterations for the proposed adaptive k-means algorithm applied to the DNS traffic data set.

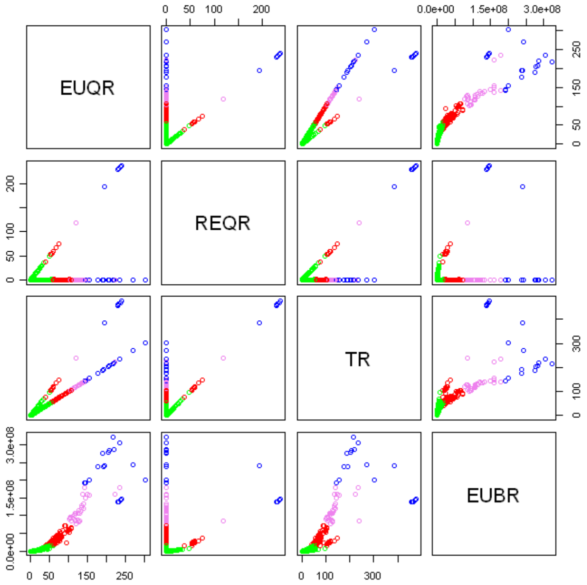

Figure 12: Scatterplot matrix of k-means clustering, on 4 criteria of DNS data.

names are not numerous (19 and 7 respectively). Now, by comparing the results, it is clear that the standard k-means method tends to group the samples according to their Euclidean norm, which leads to some donut-shaped structure, and on the contrary the adaptive method tends to distinguish the "branches" when the Euclidean norms of the samples are of the same level.

Finally, it appeared that the outcomes of the k-means and adaptive k-means algorithms on the Gaussian mixture and DNS traffic data are not dependent on the initial conditions of the procedures in all the tests we conducted.

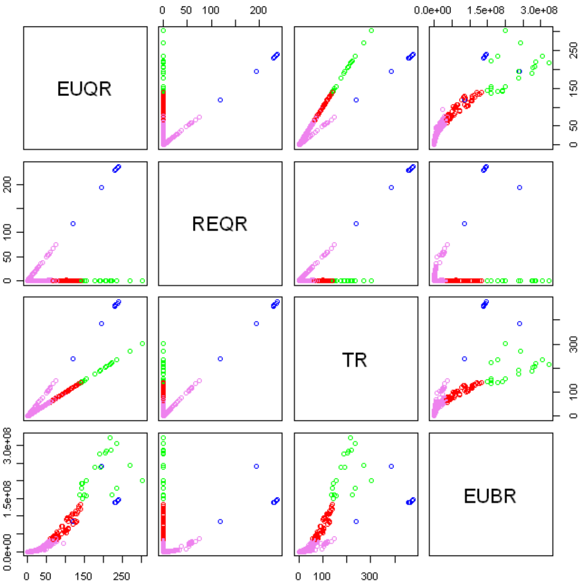

Figure 13: Scatterplot matrix of adaptive k-means clustering, on 4 criteria of DNS data.

5. CONCLUSION

In this paper we have introduced a DNSSEC architecture proposal based on the splitting of the traffic in order to cope with scalability and security issues of the current DNS architecture. Its implementation requires the clustering of DNS traffic data.
To perform this task, we have considered the standard k-means algorithm and proposed an extension of it through an adaptive version. This algorithm performs better than the standard one both on a simulated Gaussian mixture model and on the DNS traffic data we extracted from the Orange network. The adaptive extension of k-means algorithm can reflect more precisely the inherent structure of data. We also proposed three types of "error rate" to estimate the accuracy of a clustering algorithm, when some extra information about data's "label" is available. Our method indeed enhances the error rates for Gaussian mixture models, comparing to the standard k-means clustering algorithm. Concerning DNS traffic data, both algorithms find "Heavily Requested", "Regularly Requested" and "Rarely Requested" domain names, but the adaptive method manages to extract some latent structure that the standard k-means cannot find.

As a perspective, it seems interesting to investigate the classification task that could be implemented after the clustering of the DNS traffic data in order to come up with a fully adaptive and on-line system to perform the routing.
Regarding the DNS architecture, it would be necessary to carry on tests on real devices, for applying the traffic clustering.
Also, from a methodological point of view, we did not discuss the convergence of the proposed algorithm in this cur-

rent paper, we looked instead into the evolution of the normalized within-class variance with respect to iterations, and noted that it decreases drastically. As an EM type method, we can continue to discuss theoretically the convergence in future works. Here we can roughly consider that the adaptive algorithm searches a local minimum of the normalized within-class variance during the iterations, similarly to the k-means search for a local minimum of the within-class variance.

In future works, it would be also interesting to improve the performance and stability of the proposed algorithm. Noting that as an algorithm searching for a local minimum, just like k-means, initial conditions can be of premium importance, the accuracy of our algorithm depends strongly on the initial choice of the centroids. The investigation of the initialization process should be taken into account in order to enhance the accuracy. On the other hand, the regularization parameter can also influence the performance, as mentioned at the end of subsection 3.2, the more important it is, the more stable behaviors the algorithm can enjoy, but with a loss of accuracy. These issues will also be tackled as parts of future works.

6. REFERENCES

[1] P.V. Mockapetris. *Domain names - concepts and facilities*, RFC 1034 (Standard), IETF, Nov., 1987.

[2] P.V. Mockapetris. *Domain names - implementation and specification*, RFC 1035 (Standard), IETF, Nov., 1987.

[3] D. Kaminsky. *It's The End Of The Cache As We Know It. Or: "64K Should Be Good Enough For Anyone"*, IOActive Black Hat Conference, Jul., 2008.

[4] R. Arends, R. Austein, M. Larson, D. Massey and S. Rose. *DNS Security Introduction and Requirements*, RFC 4033 (Proposed Standard), IETF, Mar., 2005.

[5] R. Arends, R. Austein, M. Larson, D. Massey and S. Rose. *Resource Records for the DNS Security Extensions*, RFC 4034 (Proposed Standard), IETF, Mar., 2005.

[6] R. Arends, R. Austein, M. Larson, D. Massey and S. Rose. *Protocol Modifications for the DNS Security Extensions*, RFC 4035 (Proposed Standard), IETF, Mar., 2005.

[7] D. Migault, C. Girard and M. Laurent. *A Performance view on DNSSEC migration*, Proc. of Conference on Network and Service Management (CNSM), 2010.

[8] J.A. Hartigan and M.A. Wong. *A k-means clustering algorithm*, Applied Statistics, Vol. 28, 1979, pp. 100-108.

[9] T. Hastie, R. Tibshirani and J. Friedman. *The Elements of Statistical Learning.* Springer Series in Statistics, 2009.

[10] G.J. McLachlan and D. Peel. *Finite Mixture Models.* Wiley Series in Probability and Statistics, 2000.

[11] A.P. Dempster, N.M. Laird and D.B. Rubin. *Maximum Likelihood fom Incomplete Data via the EM Algorithm*, J. R. S. S. series B , Vol. 39, No. 1, 1977, pp. 1-38.

[12] G.J. McLachlan and T. Krishnan. *The EM Algorithm and Extensions.* Wiley Series in Probability and Statistics, 1997.

[13] S. Srivastava, M.R. Gupta, and B.A. Frigyik. *Bayesian Quadratic Discriminant Analysis*, Journal of Machine Learning Research, Vol. 8, 2007, pp. 1277-1305.

[14] P.Y. Simard, Y. LeCun, and J. Denker. *Efficient Pattern Recognition Using a New Transformation Distance.* Proc. of Advances in Neural Information Processing Systems, Morgan Kaufman, 1993, pp. 50-58.

[15] T. Hastie, A. Buja, and R. Tibshirani. *Penalized Discriminant Analysis.* Annals of Statistics, 1994.