# Scalable Malware Clustering using Multi-Stage Tree Parallelization

Muqeet Ali and Josiah Hagen
*TrendMicro Research*
USA
{muqeet_ali,josiah_hagen}@trendmicro.com

Jonathan Oliver
*TrendMicro Research*
Australia
jon_oliver@trendmicro.com

***Abstract*—Similarity hashing is an important tool for searching and analyzing malware samples which are similar to known malware samples. Several similarity hashing schemes exist in the literature (like ssdeep, TLSH, sdhash). TLSH has been found to be particularly well-suited for finding related malware (and goodware) samples from known malware (and goodware) samples. In particular, TLSH has been shown to be good at finding the different variants of a given malware. Previous work has shown that TLSH hashes can be used to build fast search and clustering techniques which can scale to tens of millions of items. In this paper, we show that previous work can be made to scale to even larger data sizes by doing clustering in stages. A fast clustering algorithm (like $k$-Means) can be used in multiple stages to obtain clusters at a coarse-level, which can later be processed by other state-of-the-art clustering techniques in parallel to obtain final clusters. We show that such a multi-stage technique can be used to cluster up to 10 million items with 9-12x speedup over just using existing state-of-the-art clustering techniques. We show that the resulting cluster quality obtained by multi-stage clustering is comparable to the cluster quality obtained by existing methods. Moreover, we show how to optimize the cost (dollars spent on the cloud) or latency incurred by multi-stage clustering technique by choosing appropriate values of parameters**

***Index Terms*—K-Means, Hierarchical Clustering, Trend Locality Sensitive Hashing (TLSH)**

## I. Introduction:

Similarity hashing aims to produce hash digests which are similar when computed on similar files or byte streams. Similarity hashing allows for a class of related files or malware samples to be detected by comparing their similarity hash digests with the similarity hash digests of known files or malware samples. It is therefore a useful tool to detect different variants of a malware in an ever evolving threat landscape. TLSH has been shown to be particularly useful at identifying variants of software when minor source code changes occur, or when some code is re-used [1], [2]. This property allows TLSH to find related malware samples (and threats) when given some known malware samples (and threats). TLSH is also computationally efficient to compute. Its time complexity grows linearly in the size of the input and is easily scalable to large number of items. Moreover, TLSH works on byte streams which makes it a general-purpose technique and is also more robust to security attacks as compared to other techniques which rely on features based on metadata. Subverting a single feature which comes under the control of the attacker can compromise the accuracy of such techniques.

Searching and clustering TLSH hashes is an important tool as it allows the analyst to sift through vast amounts of goodware and malware present in the wild and make informed decisions. The sheer amount of data to be analyzed often makes it difficult for analysts to pinpoint the related threats to a given known threat. A fast clustering technique can quickly find related samples of interest, and this can greatly help analysts to infer meaningful associations between related malware samples, and families. It can also be used to develop an effective whitelisting tool where related samples to known goodware samples are whitelisted based on their proximity to the goodware samples [3]

Most previous work on clustering malware could not scale to large number of items as it either used features which were difficult to extract and compute, or used clustering methods which could not scale to large number of items (we discuss prior work in greater detail in Section II). Recently, some clustering techniques have been proposed which can scale to large number of items [4]–[6]. One such technique uses TLSH hash digests to develop a scalable searching and clustering technique which can use either Vantage Point Trees (VPT) or Forest-based index [6]. This technique is referred to as Threshold based Hierarchical Agglomerative Clustering (HAC-T). HAC-T has been shown to scale to tens of millions of items while also exhibiting good cluster quality which indicates that the items within clusters are similar to each other. In this paper, we show how to use multi-stage clustering technique to scale HAC-T clustering as described in [6] to even larger data sets. We develop a multi-stage clustering method which combines $k$-Means clustering with HAC-T clustering and works in stages. In multi-stage clustering consisting of $n$ stages, $k$-Means is used to obtain coarse-level clusters during the first $n$-1 stages. In the final stage, more expensive but accurate clustering technique like HAC-T clustering is used to obtain the final clusters. Note that all the stages after the first stage, can be made to run in paralllel which allows for significant speedup over just using HAC-T clustering. The easy parallelization allows for cloud resources to be utilized as each cluster can be assigned to its own worker machine.

We make the following contributions in this paper:

- We introduce multi-stage clustering, and show that it can be used to cluster up to 10 million items.
- We show that multi-stage clustering can be used to obtain a speedup of 9-12x over just using HAC-T clustering without compromising on cluster quality
- We show that multi-stage clustering can be optimized for reducing cloud resources (dollar cost) and/or for reducing the job completion time of the clustering task.

Next we describe the organization for the rest of the paper. In Section II we discuss prior art, and discuss recent clustering techniques. In Section III we discuss TLSH in greater detail. In Section IV we present our multi-stage clustering technique, and show experimental evaluation of its usefulness. In Section V we discuss the tradeoffs involved in choosing different parameter values of the clustering technique presented. We conclude in Section VI

## II. Prior Art:

Many studies have been performed to cluster malware samples (or files). Some recent studies present scalable clustering techniques which scale to large number of items [4]–[6]. Authors in [4] present a clustering technique which they estimate can scale to cluster millions of files. It uses sketches to represent files, and DBSCAN is used to obtain final clusters using approximate nearest neighbors. Another paper presents a variant of DBSCAN making use of approx. nearest neighbors approach based on HNSW (Hierarchical Navigable Small World) graphs [5]. A recent paper is able to scale clustering to millions of files, using TLSH hash digests, and demonstrates that it scales better than previous two approaches as outlined above [6]. This approach uses a tree-based index for fast search. There are two approaches (i) either the index can be used with Vantage Point Tree search with backtrack; or (ii) a forest may be built which is then used for fast search of TLSH hashes. The index is then used as a building block for hierarchical threshold based clustering approach (HAC-T). Since we found this approach to be the most scalable, we compare our technique with this approach throughout the paper.

A variety of hierarchical clustering techniques have been used on malware, none of which scale to millions of samples [7]–[13]. These studies either did not evaluate their techniques on a large-scale or their estimates of throughput take on the order of days to scale to a million samples. Note also that most of these technique rely on metadata which are used as features in the clustering algorithm utilized. Such an approach which relies on metadata for features is susceptible to evasion techniques which can be used by the attackers if they happen to control some of the metadata used for clustering. TLSH based clustering is less susceptible to such attacks as the TLSH hash digest is computed using the whole byte stream [14].

A comparative analysis of several clustering techniques for malware based on different distance and evaluation metrics is presented in [15], [16]. It found hierarchical and density-based approaches as winners. BIRCH clustering was also shown to be effective at clustering malware. However, the data set is restricted to a few thousand samples which does not allow us to understand the scalability of the approaches presented. BIRCH clustering is also more suited for Euclidean distance metrics, and we deal with non-Euclidean distance in this paper.

Some other approaches use more involved or complex models for clustering malware samples [17], [18]. A deep learning based architecture based on auto-encoders is used to cluster malware samples of PE files [17]. A bytes frequency based approach is used in [18] to cluster malware samples using symbolic aggregation approximation. The works [17], [18] are not directly comparable to our work as they depend on complex techniques and models which are difficult to scale. Neither of these works have evaluated their techniques on a million samples or more. Also, most techniques rely on a set of features to obtain clustering and are more susceptible to evasion.

An ensemble learning based approach to cluster malware samples is explored in [19], [20]. This approach combines different clustering algorithms with different set of features (both static and dynamic) to more effectively cluster the malware samples. However the evaluation is only limited to a few thousand samples . A coarse-level clustering technique (BIRCH) is combined with fine-grained clustering technique (hierarchical clustering) in [21]. The high-level approach used by this work is quite similar to our approach of combining coarse-level clustering with fine-grained clustering, however, it uses different clustering algorithms and aims to cluster HTTP-based malware. Moreover, it is not as scalable as our approach as it takes more processing times for clustering 100,000 samples as compared to our approach which can cluster in the millions using less time.

## III. TLSH Overview:

TLSH is a locality sensitive hash which produces a fixed-length hash digest based on the input bytes. The standard TLSH hash (which is used throughout in this paper) comes out to be 70 characters long. TLSH hash digest has the property that two similar inputs would produce a similar hash digest (the hash computation is based on statistical features of the input bytes). The hash digest is a concatenation of the digest header and digest body. The following steps are involved in computation of the standard TLSH hash:

- All 3-grams from a sliding window of 5 bytes are used to compute an array of bucket counts, which are used to form the digest body.

- Based on the calculation of bucket counts (as calculated above) the three quartiles are calculated (referred to as $q1$, $q2$, and $q3$ respectively).
- The digest body is constructed based on the values of the quartiles in the array of bucket counts, using two bits per 128 buckets to construct a 32 byte digest.
- The digest header is composed of a checksum, the logarithm of the byte string length and a compact representation of the histogram of bucket counts using the ratios between the quartile points for $q1$:$q3$ and $q2$:$q3$

Two different TLSH hash digests are compared using the TLSH distance. The TLSH distance of zero represents that the files are likely identical, and scores greater than that indicate greater degrees of dissimilarity (please see the original paper for more details on the computation of the distance) [22].

## IV. Multi-Stage Clustering:

In this section, we describe in greater detail our approach which combines a variant of $k$-Means clustering algorithm with HAC-T clustering technique. As noted in the comparative analysis of different clustering algorithms [6], $k$-Means is a fast clustering method however the resulting clustering quality is usually not very good to make it useful for similarity search and analysis of malware. On the other hand, HAC-T clustering technique presented in [6] produces clusters which are on average quite homogeneous, and are much better suited for similarity search. However, HAC-T clustering scales in $O(n \cdot logn)$ manner as compared to linear scalability of $k$-Means. We present a technique which combines these two clustering techniques, and show that it is able to achieve good clustering quality while being more scalable than using HAC-T clustering alone.

### *A. Multi-Stage Clustering Overview:*

We propose a technique which combines $k$-Means algorithm with HAC-T clustering technique to do clustering in stages. We first describe the multi-stage clustering technique for two stages, and refer to it as two-stage clustering (*2-SC*). In the two-stage clustering, we first use $k$-Means algorithm on the input data set to obtain coarse-grained clusters. In the second stage, we run HAC-T clustering on each of the clusters obtained from stage one. Note that the second stage can be done in parallel. HAC-T clustering is able to produce fine-grained clusters which are useful for similairty search and analysis. Two-stage clustering is faster as compared to HAC-T clustering as the first stage acts as a fast preprocessing step which allows HAC-T clustering to be used on the (smaller) coarse-grained clusters as compared to the original data set. Moreover, the easy parallelization allows for clustering task to be parallelized and the subtasks can be assigned to its own worker machines in the second stage.

The intuition behind such two-stage clustering approach is to use $k$-Means as a first step to divide the clusters at a coarse-level in an efficient manner. The coarse-level clusters are not suitable for finding similar malware or for hunting down a specific variant of a malware, although they are reasonably good at separating out the clusters at a large enough level so as to make way for the more expensive but better HAC-T clustering to work on the (smaller) coarse-level clusters.

Conceptually, the multi-stage clustering technique can be done in multiple stages (or levels). For a general description of the technique we consider $n$ stages (where $n$ is greater than 2), and refer to the resulting clustering technique as $n$-stage clustering (*$n$-SC*). In the first stage we perform $k_1$-Means clustering to obtain $k_1$ coarse-grained clusters. In each of the next ($n$-2) stage(s) we run $k_{n-2}$-Means algorithm respectively on each of the clusters obtained in the previous stage. Note that in each of these stages, the $k_{n-2}$-Means algorithm can be run in parallel on each of the clusters obtained from the previous stage. In the last stage, we perform HAC-T clustering on all the clusters produced by the previous stage in parallel. For example, we consider three-stage clustering (*3-SC*) in which $k_1$-Means would be used in the first stage to produce $k_1$ clusters. In the second stage, we use $k_2$-Means algorithm on each of the $k_1$ clusters to obtain $k = k_1 \cdot k_2$ clusters in total. In the third stage, HAC-T clustering is used on the $k$ clusters to obtain fine-grained clusters. A schematic diagram which describes the conceptual overview of three-stage clustering with $k_1$ = 4 and $k_2$ = 2 is presented in Figure I. In this Figure, $C_{i,j}$ is the $jth$ cluster obtained after the $ith$ stage is completed. A total of $w$ clusters are obtained after applying HAC-T clustering in the final stage.

We use a modified $k$-Means algorithm which is adapted to work on TLSH distances [6]. It essentially computes the mean by first converting the 70 character TLSH hash into a 70 coordinate vector and computing the mean of the vectors ($m$). It then computes the closest TLSH hash item to the mean ($m$) using TLSH distance and uses that as the mean in the algorithm. It is a heuristic which allows us to adapt $k$-Means algorithm for clustering TLSH hashes which use the TLSH distance as compared to Euclidean distance metric.

Below we discuss some unique features of multi-stage clustering which allow it to scale much better as compared to other clustering techniques:

- Multi-stage clustering allows for better scalability as it uses the faster method ($k$-Means) as a first step to preprocess the data and produce coarse-level clusters which can be processed using the more effective HAC-T clustering to obtain interesting clusters which are smaller and more numerous.
- It allows for effective parallelization of the original clustering problem as the clusters obtained from previous stage can be clustered in parallel
- It can also be used to optimize for total compute time (or dollars spent on the cloud) and/or completion time of the

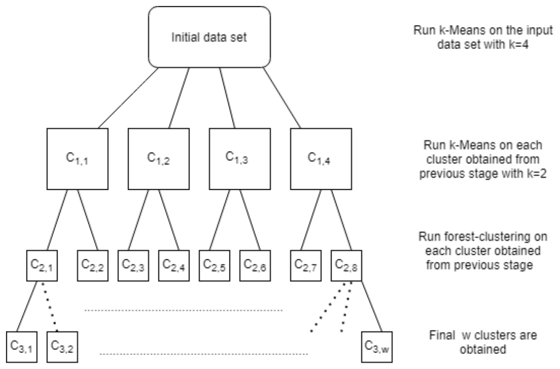
Initial data set
Run k-Means on the input data set with k=4
C1,1
C1,2
C1,3
C1,4
Run k-Means on each cluster obtained from previous stage with k=2
C2,1
C2,2
C2,3
C2,4
C2,5
C2,6
C2,7
C2,8
Run forest-clustering on each cluster obtained from previous stage
C3,1
C3,2
C3,w
Final w clusters are obtained

Fig. 1: A schematic diagram explaning the flow of three-stage clustering with $k_1$=4 and $k_2$=2. A total of $w$ clusters are obtained after the final stage.

clustering job by optimizing $k$-Means parameters (more on this is described in Section V)

Next, we discuss the experimental setup and evaluation.

## B. Experimental Setup :

Our experimental setup consists of a fleet of commodity cloud 32-core machines with 128 GB memory, and AMD EPYC 7000 series processor (with an all core turbo clock speed of 2.5 GHz). In our experiments, we used all 32 cores for the execution of $k$-Means algorithm, while a single core is used for the execution of HAC-T clustering. We used a fleet of up to 100 machines in our experiments with each stage using a different number of machines based on the number of clusters available for parallel execution.

We source data from VirusTotal data feeds. We used a random sample of VirusTotal data downloaded between September 2019 and Feb 2020. The VirusTotal data feed gives information about the scan results of all the major anti-virus vendors (it gives us 'True'/'False' labels based on whether the file was detected by the respective AV vendor or not, and includes the malware family/variant information for the files which were detected as 'True'). It also includes some auxiliary information (like SHA1, md5 etc.) for each file. We calculate the TLSH for each Win32 PE file that is submitted to VirusTotal, as provided by the data feed. We use the TLSH and 'True'/'False' labels obtained from five major anti-virus vendors (Kaspersky, Microsoft, Symantec, Sophos, and McAfee) when doing clustering analysis to ascertain the quality of the clustering.

## C. Experimental Evaluation:

We evaluated two-stage clustering (*2-SC*) and three-stage clustering (*3-SC*) with recent state-of-the-art HAC-T using forest-based implementation as described in [6]. We have done a comparative analysis of these clustering techniques using a sample size of up to 10 million items. The clustering quality is measured using labels ('True'/'False') obtained from five major AV vendors in VirusTotal data feed. We have used purity as the metric for computing cluster quality as it scales linearly with increasing sample sizes. We investigated other cluster quality metrics like silhouette coefficient but these do not scale well to millions of items so we ended up not using them. Purity score varies from 0 to 1 with 1 for the perfect cluster quality and 0 for data set with worst cluster quality.

Table I shows a comparison of the purity scores for the three clustering techniques evaluated at different sample sizes. The purity scores are reported for five major anti-virus vendors (Kaspersky, Microsoft, Symantec, Sophos, and McAfee). As the multi-stage clustering technique involves $k$-Means which tends to be dependent on the initial random seed, we have reported the results for two-stage and three-stage clustering by averaging the results over 10 executions of these techniques. We have chosen $k = 100$ as value of the parameter of $k$-Means in two-stage clustering, and we chose $k_1$=10 and $k_2$=10 as parameters for three-stage clustering (note that $k_1$ and $k_2$ are chosen such that $k_1 \cdot k_2 = 100$). We chose $max\_iterations = 10$ for every execution of the $k$-Means algorithm used in multi-stage clustering. For the purpose of this evaluation we chose the parameter $T$ as 50 which is a TLSH distance threshold used by the HAC-T clustering algorithm [6].

Table II presents an evaluation of running times and percentage of items clustered for the three clustering techniques. As can be inferred from the two tables, the two-stage clustering approach is 5-7x faster as compared to just running HAC-T clustering, and it produces similar cluster quality. The percentage of items clustered is only lagging by at most 1 percent. Also, we can infer from Table I and Table II that three-stage clustering is able to run much faster than HAC-T clustering while producing similar cluster quality. In fact, it is able to provide 9-12x speedup over using just HAC-T clustering, and it is able to cluster almost the same percentage of items as clustered by two-stage clustering. Note that the machines exchange some information (from AWS S3 bucket) regarding clusters. This is how machines communicate regarding which items belong to a given cluster, and this time is included in the time reported in Table II.

| | n=1 million | | | n= 2 million | | | n= 4 million | | | n= 10 million | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | *HAC-T* | *2-SC* | *3-SC* | *HAC-T* | *2-SC* | *3-SC* | *HAC-T* | *2-SC* | *3-SC* | *HAC-T* | *2-SC* | *3-SC* |
| **Kaspersky AV** | 0.97 | 0.97 | 0.97 | 0.97 | 0.97 | 0.97 | 0.97 | 0.97 | 0.97 | 0.97 | 0.97 | 0.97 |
| **Microsoft AV** | 0.98 | 0.98 | 0.98 | 0.98 | 0.98 | 0.98 | 0.98 | 0.98 | 0.98 | 0.98 | 0.98 | 0.98 |
| **Symantec AV** | 0.98 | 0.98 | 0.98 | 0.98 | 0.98 | 0.98 | 0.98 | 0.98 | 0.98 | 0.98 | 0.98 | 0.98 |
| **Sophos AV** | 0.98 | 0.98 | 0.98 | 0.97 | 0.98 | 0.98 | 0.98 | 0.98 | 0.98 | 0.98 | 0.98 | 0.98 |
| **McAfee AV** | 0.97 | 0.97 | 0.97 | 0.97 | 0.97 | 0.97 | 0.97 | 0.97 | 0.97 | 0.97 | 0.97 | 0.97 |

TABLE I: A comparison of purity scores for HAC-T clustering, two-stage clustering (*2-SC*) and three-stage clustering (*3-SC*) using five major AV vendors

| | n=1 million | | | n= 2 million | | | n= 4 million | | | n= 10 million | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | *HAC-T* | *2-SC* | *3-SC* | *HAC-T* | *2-SC* | *3-SC* | *HAC-T* | *2-SC* | *3-SC* | *HAC-T* | *2-SC* | *3-SC* |
| **Time Taken (s)** | 510.59 | 96 | 53.43 | 1149.45 | 181.7 | 108.84 | 2640.51 | 372.07 | 220.9 | 7782.55 | 1131.66 | 833.74 |
| **(%) Clustered** | 60 | 59 | 58 | 62 | 61 | 61 | 64 | 63 | 63 | 67 | 66 | 66 |

TABLE II: Time taken and (%) clustered for HAC-T, two-stage clustering (*2-SC*) and three-stage clustering (*3-SC*)

It could be possible to further reduce the clustering time by considering more levels of clustering (i.e., use more than three stages in multi-stage clustering as described previously) but note that the parallelization opportunity reduces with each successive level of the multi-stage clustering. Note that the parallelization opportunity would eventually go away after some (finite) number of levels.

## V. Choice of Parameters:

As noted in the previous section, we chose a particular set of values for the multi-stage clustering technique. In this section, we investigate which choice of parameter values makes more sense, and what are the different tradeoffs involved in choosing these values for the parameters. We present our results for two-stage clustering but we believe these serve as guidance for selecting parameters for multi-stage clustering in general. We first investigate how different values of $k$ affect the running time of the clustering technique as well as the dollars spent on the cloud resources utilized. We define **job completion time** as the latency measured in seconds from job start to the job end, where job end happens when all the clusters in the second stage have been processed by HAC-T clustering algorithm. We define **total time** as the time spent by all the instances when executing the two-stage clustering technique. Total time is equal to the time that we used the cloud resources to perform clustering which is proportional to the dollars spent on the cloud (as most cloud providers charge based on the time the cloud resources were used).

In Figure II we show that as $k$ is varied (from low to high), initially both total time and job completion time decrease (or remain stable) but then after a certain threshold both total time and job completion time start increasing again. This holds true for different sample sizes. We experimented with sample size range from 500k to 4 million and found a consistent behavior. Also, we averaged each reported result over 10 executions of two-stage clustering in order to account for variations owing to the random seed used by $k$-Means. We also observe that the optimal value of $k$ increases slowly (more akin to logarithmically) with increasing values of sample sizes ($n$). We chose a value of $k = 100$ for our experiments in the previous section as we believe choosing a larger value only consumes additional cloud resources and does not help in reducing job completion time either. We chose a conservative value which saves cloud resources without compromising on the job completion time. We separately confirmed that changing values of $k$ does not alter the clustering quality in any meaningful way as clustering quality is determined by HAC-T clustering algorithm.

Next, we describe how values of parameter $T$ affect the clustering quality of the clusters obtained by HAC-T clustering, and also the percentage of items clustered. Table III shows the tradeoffs involved in this regard.

| | **T=20** | **T=40** | **T=60** | **T=80** | **T=100** |
|---|---|---|---|---|---|
| **(%) Noise** | 51.10 | 43.24 | 37.77 | 32.49 | 27.72 |
| **Average purity score** | 0.982 | 0.976 | 0.974 | 0.964 | 0.956 |

TABLE III: Items clustered and average purity score achieved for $n = 1$ million

Table III shows the value of $T$ affects both the percentage of items clustered and the average purity score achieved (the purity score is calculated by computing the average purity score for five major anti-virus vendors (Kaspersky, Microsoft, Symantec, Sophos and McAfee). As we increase the value of $T$ more items can be clustered owing to a greater distance threshold allowed but the purity score decreases. We chose a value of 50 for $T$ as a best estimate which accounts for the above tradeoffs involved.

## VI. Conclusion:

TLSH has been found to be useful when comparing similar byte streams, and is particularly useful to find the many variants of a given malware family. Previous work on clustering TLSH hashes has addressed the problem of searching and clustering TLSH hashes but was confined to the limits of a single machine/core. This restriction puts a limit on the number of items that can be clustered effectively in a given amount of time. We have presented a technique by which one can easily

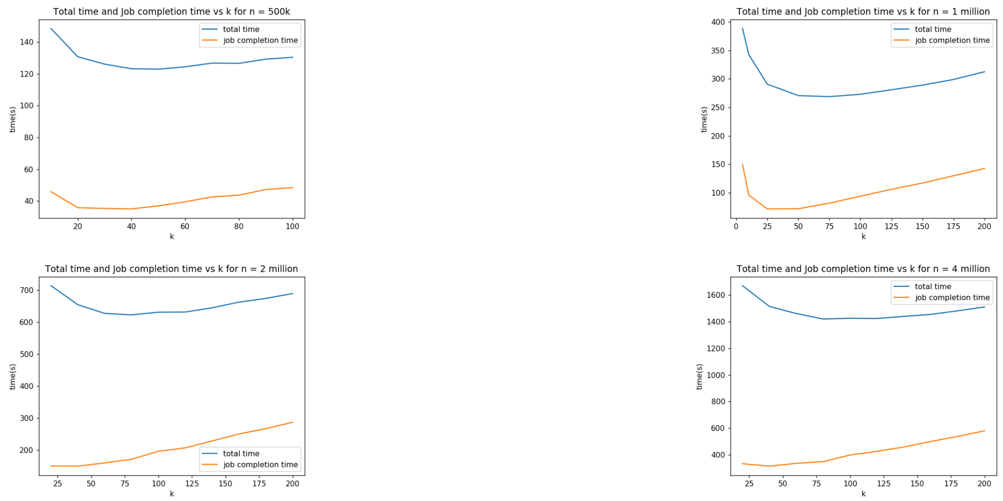

Fig. 2: The effect of $k$ on the job completion time and total time associated with the two-stage clustering technique

scale HAC-T clustering approach by making use of $k$-Means as a preprocessing time to produce clusters which can then be clustered by making use of on-demand cloud resources. The final clustering so obtained is useful for separating the files based on labels obtained from VirusTotal, however, note that TLSH hash digests are calculated based on byte streams and do not consider behavior-based features which are considered by some other clustering approaches. It might be possible to further refine the clustering by applying some behavior-based approach. However, we believe that clustering based on TLSH hash digests serves as an excellent first pass as most other approaches are computationally expensive and fail to scale to millions of items.

## References

[1] F. Pagani, M. Dell'Amico, and D. Balzarotti, "Beyond precision and recall: understanding uses (and misuses) of similarity hashes in binary analysis," in *Proceedings of the Eighth ACM Conference on Data and Application Security and Privacy*, 2018, pp. 354–365.

[2] J. Coffman, A. Chakravarty, J. A. Russo, and A. S. Gearhart, "Quantifying the effectiveness of software diversity using near-duplicate detection algorithms," in *Proceedings of the 5th ACM Workshop on Moving Target Defense*, 2018, pp. 1–10.

[3] J. Oliver and J. Pryde, "Smart Whitelisting Using Locality Sensitive Hashing," https://www.blackhat.com/asia-17/arsenal.html#smart-whitelisting-using-locality-sensitive-hashing, 2017, [Online; accessed 13-Aug-2020].

[4] K. Soska, C. Gates, K. A. Roundy, and N. Christin, "Automatic application identification from billions of files," in *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2017, pp. 2021–2030.

[5] M. Dell'Amico, "Fishdbc: Flexible, incremental, scalable, hierarchical density-based clustering for arbitrary data and distance," *arXiv preprint arXiv:1910.07283*, 2019.

[6] J. Oliver, M. Ali, and J. Hagen, "Hac-t and fast search for similarity in security," in *To Appear in IEEE COINS Conference*, 2020.

[7] Y. Li, S. C. Sundaramurthy, A. G. Bardas, X. Ou, D. Caragea, X. Hu, and J. Jang, "Experimental study of fuzzy hashing in malware clustering analysis," in *8th Workshop on Cyber Security Experimentation and Test ({CSET} 15)*, 2015.

[8] U. Bayer, P. M. Comparetti, C. Hlauschek, C. Kruegel, and E. Kirda, "Scalable, behavior-based malware clustering." in *NDSS*, vol. 9. Citeseer, 2009, pp. 8–11.

[9] X. Hu, K. G. Shin, S. Bhatkar, and K. Griffin, "Mutantx-s: Scalable malware clustering based on static features," in *Presented as part of the 2013 {USENIX} Annual Technical Conference*, 2013, pp. 187–198.

[10] J. Jang, D. Brumley, and S. Venkataraman, "Bitshred: feature hashing malware for scalable triage and semantic analysis," in *Proceedings of the 18th ACM conference on Computer and communications security*, 2011, pp. 309–320.

[11] Y. Li, J. Jang, X. Hu, and X. Ou, "Android malware clustering through malicious payload mining," in *International Symposium on Research in Attacks, Intrusions, and Defenses*. Springer, 2017, pp. 192–214.

[12] O.-B. Boţocan and G. Czibula, "Hacga: An artifacts-based clustering approach for malware classification," in *2017 13th IEEE International Conference on Intelligent Computer Communication and Processing (ICCP)*. IEEE, 2017, pp. 5–12.

[13] A. Mohaisen, O. Alrawi, and M. Mohaisen, "Amal: High-fidelity, behavior-based automated malware analysis and classification," *computers & security*, vol. 52, pp. 251–266, 2015.

[14] J. Oliver, S. Forman, and C. Cheng, "Using randomization to attack similarity digests," in *International Conference on Applications and Techniques in Information Security*. Springer, 2014, pp. 199–210.

[15] H. Faridi, S. Srinivasagopalan, and R. Verma, "Performance evaluation of features and clustering algorithms for malware," in *IEEE International Conference on Data Mining Workshops (ICDMW)*, 2018, pp. 13–22.

[16] G. Pitolli, L. Aniello, G. Laurenza, L. Querzoni, and R. Baldoni, "Malware family identification with birch clustering," in *2017 International Carnahan Conference on Security Technology (ICCST)*, 2017, pp. 1–6.

[17] C. K. Ng, F. Jiang, L. Y. Zhang, and W. Zhou, "Static malware clustering using enhanced deep embedding method," *Concurrency and Computation: Practice and Experience*, vol. 31, no. 19, p. e5234, 2019.

[18] N. Singh and S. S. Khurmi, "Bytefreq: Malware clustering using byte frequency," in *2016 5th International Conference on Reliability, Infocom Technologies and Optimization (Trends and Future Directions)(ICRITO)*. IEEE, 2016, pp. 333–337.

[19] X. Hu and K. G. Shin, "Duet: integration of dynamic and static analyses for malware clustering with cluster ensembles," in *Proceedings of the 29th annual computer security applications conference*, 2013, pp. 79–88.

[20] Y. Ye, T. Li, Y. Chen, and Q. Jiang, "Automatic malware categorization using cluster ensemble," in *Proceedings of the 16th ACM SIGKDD international conference on Knowledge discovery and data mining*, 2010, pp. 95–104.

[21] R. Perdisci, D. Ariu, and G. Giacinto, "Scalable fine-grained behavioral clustering of http-based malware," *Computer Networks*, vol. 57, no. 2, pp. 487–500, 2013.

[22] J. Oliver, C. Cheng, and Y. Chen, "Tlsh–a locality sensitive hash," in *2013 Fourth Cybercrime and Trustworthy Computing Workshop*. IEEE, 2013, pp. 7–13.