Finding a Needle in a Haystack: A Novel Log Analysis Method with Test Clustering in Distributed Systems

Jin Choi[1], Sangwoo Noh[1], Sooncheol Hong[3], Hanna Jang[4], Seonghee Yim[5], Seonil Brian Choi[6]
([1]jjin.choi; [1]sangwoo.noh; [3]star.hong; [4]hn01.jang; [5]sh.yim; [6]seonilb.choi@samsung.com)

***Abstract*- As the diversity and the complexity in digital design increases, the cost of design verification also increases. In such a trend, verification engineers are struggling with repetitive and tedious process, such as debugging common errors from different test cases. To solve this problem, verification engineers can use their own domain knowledge to prioritize test cases to find different errors, however, it takes time and efforts to do it manually. It is also not so easy tasks to understand different designs and test benches with limited time and resources. In this paper, we propose a novel approach of a test case prioritization algorithm based on clustering. Components from terabytes of simulation logs are considered as features in clustering algorithms. After clustering, the proposed algorithm reorders test cases with priority and recommends test cases that allow verification engineers to achieve a specific rate of error detection as early as possible. We also propose robust and scalable system architecture that can handle terabytes of data for practical use in design verification industry.**

I. INTRODUCTION

As demands on digital designs increase across industries such as smartphones, wearable devices, and automotive, engineers are required to design more diverse and complex digital logics. Development period is also getting shorter with shorter release cycles of finished product. In such a trend, conventional development methods in digital design and verification have their limitations to its sustainability, scalability and reusability. These limitations give digital design engineers missions to innovate development process by integrating software technologies such as machine learning and big data into hardware industry. One of the important missions is to reduce debugging efforts in design verification. The process of design verification accounts for approximately 70% of the entire process [1] [2]. There have been many efforts to reduce cost for design verification, however, verification engineers are still struggling with tedious tasks such as debugging common errors from different test cases or finding critical errors in the later stage of the development process.

In this paper, we propose a novel approach of a Test Case Prioritization (TCP) algorithm with clustering to reduce time and expenses in debugging process. The proposed algorithm prioritizes test cases based on clustering results. Clustering algorithms are unsupervised machine learning techniques to find clusters having common features, and this paper uses components as features for clustering algorithms. The components are extracted from terabytes of simulation logs and provide information about what actions are taken while running simulation. Based on components, test cases are clustered with other similar test cases, and then test cases are prioritized in each cluster.

The contributions of this paper include:

- **Clustering algorithm based on components from logs**
- **Test case prioritization algorithm in clusters**
- **System architecture for machine learning application**

First, we propose a clustering algorithm based on components from logs. Components include functions, such as Task, Class and Sequence and errors, such as UVM errors and user defined errors. Clustering algorithms consider these components as features for clustering, and each component is given weights for better clustering results depending on maturity of verification. By applying different weights on specific components, clustering is extended to prioritize more important component to find critical errors first.

Second, we propose test case prioritization algorithms. The algorithm prioritizes test cases based on how different type of errors can be found as early as possible. In the conventional debugging process, verification engineers depended on their domain knowledge of design to prioritize test cases, and this was tedious tasks; for instance,

debugging common errors in different test cases. The proposed algorithm prioritizes test cases and gives a higher priority to test cases with more uncommon errors and shorter run time. In *Algorithm* section, five different algorithms are described, and in *Result* section, they are compared to determine which algorithm is appropriate to reach a targeted rate of error detection as early as possible.

Finally, we propose robust and scalable system architecture for machine learning application in the industry. The proposed system architecture is sustainable, scalable and reusable. It enables machine learning models to be serviced in the production level and updated with new data sources. The system is also scalable in that it is composed of distributed servers and databases, and only one machine learning model is loaded regardless of the number of inference requests. Each machine learning pipeline is reusable in a process of continuous integration and development of the proposed algorithm.

II. BACKGROUNDS

Clustering Algorithms

K-means clustering algorithm is one of the most widely used clustering algorithms. This algorithm initially selects K points for centroids of clusters randomly, computes the sum of the squared distance between points and all centroids and then assigns to the closest cluster. This process is iterated until there is no change in the centroids. Density-Based Spatial Clustering of Applications with Noise (DBSCAN) algorithm is used to find clusters considering density, and it is useful when the number of clusters is not specified. DBSCAN algorithm uses two hyper parameters: epsilon ε and *minPts*, denoting distance and minimum number of clusters respectively. Ordering Points To Identify the Clustering Structure (OPTICS) algorithm is used only with *minPts*, without epsilon ε, and it is better suited for usage on large datasets than DBSCAN. This paper focuses on K-means algorithm which is suitable for the serviced system in real-time and characteristics of data.

A weighted clustering algorithm is one way to improve clustering algorithms for the status of verification progress by giving weights to specific features. In the simple K-means algorithm, each feature contributes equally to clustering, and it makes the algorithm concentrate on finding the optimal way to separate different features based on similarity. On the other hand, by giving specific features more weights, a weighted clustering algorithm can improve cohesion of clusters and it is consequently useful when finding critical errors or prioritizing important features.

In this paper, we use K-means clustering algorithm because it is suitable for our datasets and our parallel system. To find optimal K value in K-means clustering algorithm, each cluster is evaluated with Silhouette scores and Elbow method. Silhouette score indicates how clusters are well apart from each other and how similar an object is to its own cluster. Elbow method finds optimal K value until there is no remarkable improvement even with additional costs. In this paper, we compare a K-means clustering algorithm with and without weights, and set K value that maximizes Silhouette score considering slope in Elbow method graph.

Test Case Prioritization

There has been a variety of research on test case prioritization. B. Miranda proposed Function Analysis System Technique (FAST) approaches to scalable similarity-based test case prioritization by using min-hashing and locality-sensitive hashing algorithm for finding test cases within a big dataset [3]. A. Wahba, J. Hohnerlein and F. Rahman proposed a tool with machine learning techniques to detect failure and associate it with actual bugs in Register Transfer Level (RTL) in x86 processors [4]. Test case prioritization is a main issue in the software industry as well. Fang, C. et.al proposed similarity-based test case prioritization using ordering sequences of program entities [5]. They analyzed open source projects as a target. T. B. Noor and H. Hemmati proposed a similarity-based approach using historical failure data based on a class of metric that estimates test cases quality [6]. Daniel Flemstr¨om et.al studied test prioritization based on similarity-based reuse of test codes [7].

In this paper, we use a similarity-based approach - the same method as the related works - however, we focus on prioritization with a clustering algorithm, and consider not only results of test case simulation but also components of test cases.

System Architecture

System architecture is of large importance for deploying successful machine learning model. Even if a good machine learning model is developed, it is of no use when the model remains in local environment. Moreover, if development and deployment process is not automated, machine learning models would be hard to be updated with continuous stream of data in design verification process. In this paper, we introduce robust and scalable architecture for practical use of machine learning models in design verification. The system includes in-memory machine learning models, distributed systems and automated pipelines.

First, the concept of in-memory machine learning is a method to speed up model inference time by loading machine learning models in memory, not disk. In-memory machine learning is a key for real-time inference [8]. The in-memory approach is used in other data processing domains. For instance, Spark, an open source for distributed systems, reduces data processing time much more than Hadoop, well-known distributed file system by using in-memory approach [9].

Second, distributed systems also improve data analyzing and processing speed. Clustering algorithms, such as K-means clustering, are appropriate machine learning algorithms for parallel processing in distributed systems. In addition, as log data processing can be performed independently, distributed systems are efficient for handling terabytes of simulation logs in parallel. Such architecture maximizes its performance of processing large datasets.

Finally, automated pipelines enable the system to position in design verification process. In our previous work, we proposed a design verification automation system [10]. In this paper, the proposed system architecture is based on the previous system with newly added automated data analysis and machine learning pipelines. In the automated pipelines, for example, the system automatically cleanses data, finds optimal hyper parameters, such as the number of cluster K in K-means clustering algorithm, and deploys models in the automation system. Airflow, an open source to automate workflow, is used for the system.

III. ALGORITHMS

The goal of the proposed algorithm is to reduce time and cost to find errors in test cases with minimum number of regression and least resources. The algorithm is composed of three phases: data preprocessing phase, clustering phase and test case prioritization phase.

In the data preprocessing phase, duplicated components are removed, and preserved components are transformed into vectors for clustering algorithms. In addition, to maximize the accuracy of clustering algorithms, the number of features is reduced by dimensionality reduction algorithms.

In the clustering phase, test cases are clustered based on components extracted from simulation logs. Components are considered as features for a clustering algorithm. For a weighted clustering algorithm, weights can be given by specific components based on engineers' domain knowledge or the importance of components.

In the test case prioritization phase, test cases are prioritized in each cluster through one of five algorithms. The purpose of this phase is to sample representative test cases to find different errors as early as possible without running all test cases. Each TCP algorithm is chosen depending on the circumstance of verification.

The overview of the flow in the algorithm is shown in Figure 1.

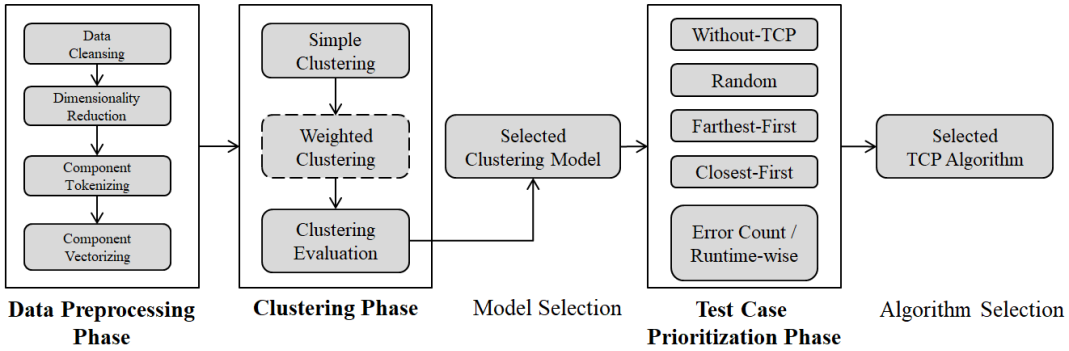


Figure 1. Flow of the Proposed Algorithm.

Data Preprocessing Phase

In the data preprocessing phase, components are preprocessed prior to be used as features for cluster algorithms. First, duplicated components are removed, and the preserved components are tokenized. Second, tokenized components are vectorized with Term Frequency and Inverse Document Frequency (TF-IDF) vectorization: a method to vectorize words considering the impact of words to documents. Finally, dimensionality is reduced by using Principal Component Analysis (PCA) or Singular Value Decomposition (SVD). PCA finds orthogonal basis to reduce dimensionality while preserving variance of the original data, and SVD decomposes complex matrix into singular vectors to reduce dimensionality. For example, the number of dimension decreases to 92% and 85%, based on PCA and SVD, respectively from our datasets. Figure 2 shows comparison results between without dimensionality reduction and with dimensionality reduction methods. Considering all components as 3,990 features, the dimensionality reduction methods help clusters separate from other clusters compared to the method without reduction.

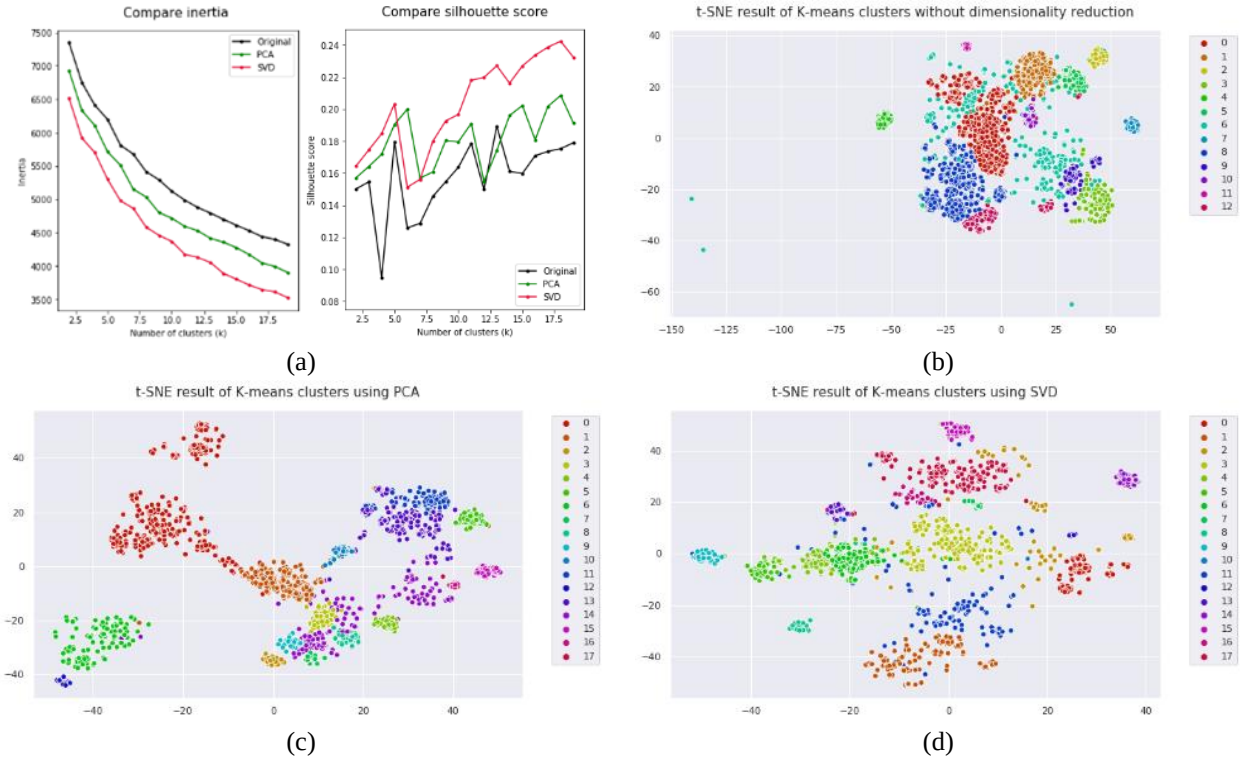
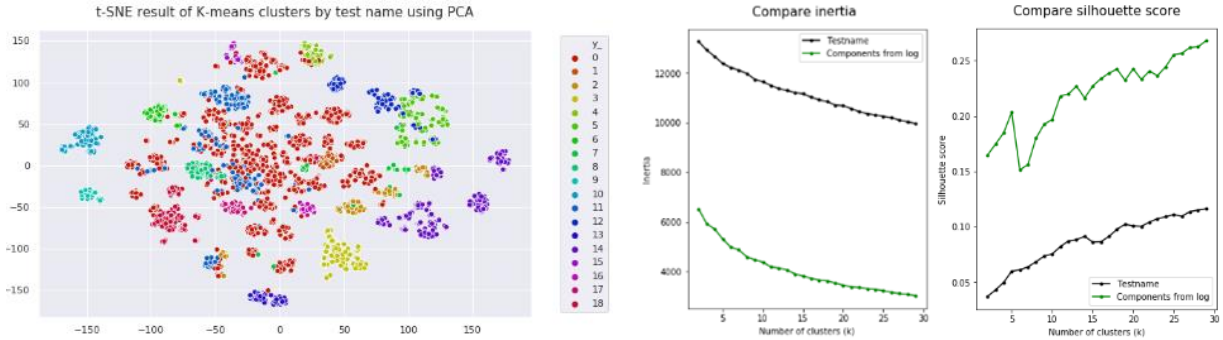


Figure 2. (a) Comparison inertia and silhouette score, (b) t-SNE results of K-Means clustering without dimensionality reduction and dimensionality reduction: (c) Principal Component Analysis (PCA) and (d) Singular Value Decomposition (SVD).

Clustering Phase

In the clustering phase, to find out the optimal K value in K-Means clustering algorithms, Silhouette score and Elbow method are used. In our automated system, an optimal K value is automatically given based on maximum Silhouette scores. Figure 3b describes how to select the largest Silhouette score between 0 and 30 of K value. K value above 30 is considered as meaningless based on Elbow method.

At first, to ensure that components are valid enough as features, a naive clustering and the proposed component-based clustering method are compared. The naive method extracts components from test names, which contain intuitive clues about test cases. The component-based clustering method, however, extracts components from log files. Both methods vectorize components with TF-IDF, the number of word occurrences in each document. Figure 3 shows results of comparing naïve method and component-based method. The points in clusters in Figure 3a are much sparser than Figure 2c, meaning components have more abundant information as features for clustering together tightly.



(a) (b)

Figure 3. (a) t-SNE result of K-means clusters by test name using PCA (b) Comparison inertia and silhouette score.

For a weighted clustering, it gives different weights to each component. Each component is not considered equally as it is in the simple clustering algorithm. For example, common and minor components can have smaller weights or some critical errors can have larger weights. As components are divided into two types: (1) Non-error type such as Task, Function or Sequence of test cases and (2) Error type included tool errors and user defined errors. The reason for such separation is to consider errors and non-errors differently on a user level. This means that users sometimes consider errors more importantly, or sometimes consider non-errors more importantly. By giving different weights on non-errors and errors, clustering algorithms can be customized depending on different situations.

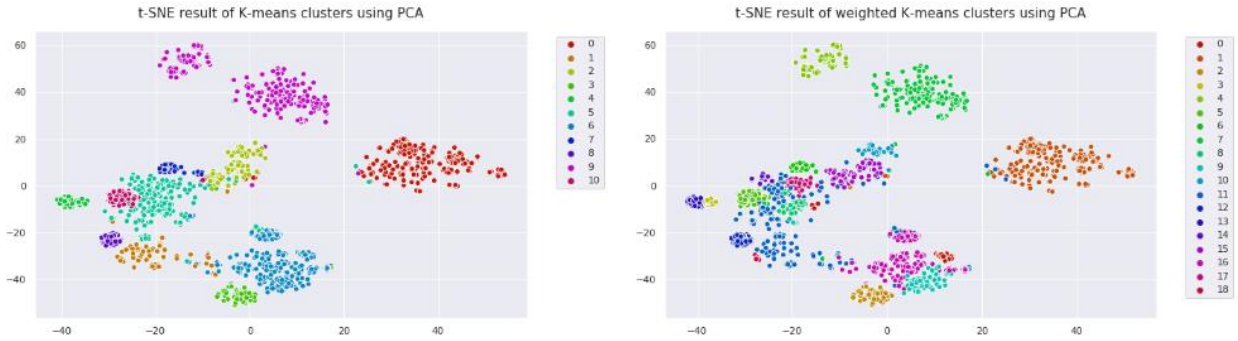
Based on how much weight each component is given, clusters are reorganized. The equation of obtaining weight values is as follows:

$$W_C = \alpha * \left(\begin{bmatrix} w_{c_1} \\ w_{c_2} \\ \dots \\ w_{c_n} \end{bmatrix} [c_1,\ c_2\ \dots\ c_n] \right),\ W_E = \beta * \left(\begin{bmatrix} w_{e_1} \\ w_{e_2} \\ \dots \\ w_{e_m} \end{bmatrix} [e_1,\ e_2\ \dots\ e_m] \right)$$

where $c \in C$, $e \in E$, C is Components, E is Errors

Specifically, α is a hyper parameter that denotes how much weight is given to all components; β is a hyper parameter that denotes how much weight is given to all errors; *w* is a hyper parameter that denotes how much weight is given in individual component and error. The matrices of weights, W_c and W_e, are multiplied to TF-IDF value of individual component ($[c_1, c_2\ \dots\ c_n]$) and errors ($[e_1, e_2\ \dots\ e_m]$).

Figure 4a shows the result of K-means clustering without weights, and Figure 4b shows the result of K-means clustering with weights on specific errors related to access registers. Hyper parameters were set to give more weight on β than α and each *w* was given equally except the specific errors. These hyper parameter settings mean test cases are clustered more considering on the specific errors. Figure 4b shows the average distance between points from the centroid is smaller than Figure 4a because the specific errors become significant components for clustering.



(a) (b)

Figure 4. (a) t-SNE result of K-means clusters, (b) t-SNE result of weighted K-means clustering.

Test case Prioritization Phase

To find an appropriate algorithm in test case prioritization, five algorithms are compared in this paper: (1) *Without-TCP*, (2) *Random*, (3) *Farthest-First*, (4) *Closest-First* and (5) *Error-Count / Runtime-wise*. *Without-TCP* has nothing to do with clustering. It is a way of running regression in the original system. *Random* algorithm samples a specific number of test cases randomly from clusters. *Farthest-First* is an algorithm selecting test cases in the order of the farthest distance from the centroid of each cluster. *Closest-First* is an algorithm selecting test cases in the order of the closest distance from the centroid of each cluster. *Error-Count / Runtime-wise* is a prioritizing algorithm that considers the number of errors in a test case and its run time. This algorithm is based on the hypothesis that when there are many error types in a test case, the test case is highly possible to be useful when extracting more possible failure cases, and when the test case has shorter run time, other test cases will run and find other errors as early as possible. Of five algorithms, depending on different environment, one algorithm is selected based on the result of how early a specific rate of error detection is achieved.

Based on the selected algorithm of five candidates, test case prioritization algorithm reorders the order of regression. For each clusters, the five highest prioritized test cases are selected and simulated. The process of selecting test cases and regression is repeated until all errors are detected. The detailed algorithm is described as follows:

Algorithm: test case prioritization from clusters

Input: C: A list of clusters, S: Sampled test cases,
I: Important components and errors, T: list of test cases, N: size of T
Output: ∅
Preprocessing;
Allocate N size of memory for T;
Sort C by occurrence of I in c;

Sort c by the selected prioritization algorithm;
foreach *c* ∈ *C* **do**
Sort c (e.g. closest-first)
end

Run regression based on test case priority;
while *C is empty* **do**
foreach c ∈ C **do**
S ← *number of test cases from c*
foreach *s* ∈ *S* **do**
Append s to T
Pop s from c
end

Run regression with T
end

IV. ARCHITECTURE

This section introduces system architecture of the proposed system. The overall architecture is shown in Figure 5. While running regression, automation pipelines run design verification automation steps such as environment setup, compile and simulation. The simulation data is stored in distributed databases and file systems for efficient access to databases. After regression is finished, machine learning pipelines are executed. In the pipelines, log files are processed by distributed computing servers, and the proposed algorithm prioritizes test cases based on clustering algorithms. Next, the machine learning model is deployed to the web server and loaded in its memory. This system architecture enables the proposed algorithm to be updated in design verification automation flow. In this section, it is discussed how each part of system contributes to practical use of machine learning in design verification process.

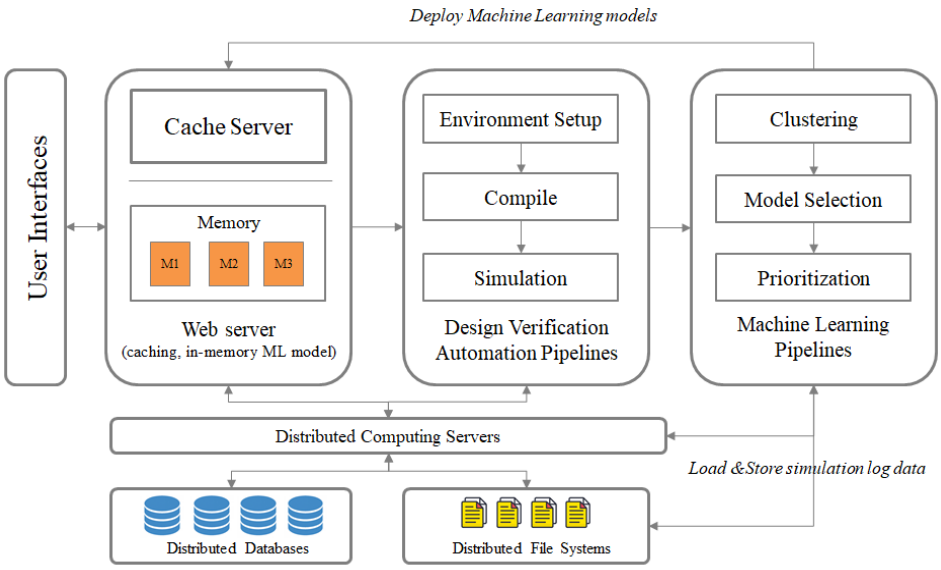


Figure 5. Overall architecture of the proposed sytstem.

In-memory machine learning model

In-memory machine learning model is literally a machine learning model that is loaded on the memory, not disk. As memory access is much faster than disk access, this method takes advantage of faster model inference time. This method is combined with software technologies such as web server and design patterns. For web server, the system consists of Nginx, an event-driven web server, Gunicorn, a Web Server Gateway Interface, and Django, one of the most widely used web frameworks in Python. Nginx is used for load balancing and caching, and it decreases response time and reduces overload to worker threads. In the experiment, model inference time took average of 20 to 40 seconds at most when a job is queued by workload balance platform such as Load Sharing Facility. However, in-memory machine learning model took only 2 seconds on average, moreover, when caching is used, it took only less than 1 second.

As workers are created with multiple threads, machine learning models are designed as a class with a thread-safe Singleton design pattern. Singleton design pattern is a software design pattern that enables an instance of class used as unique across multiple threads. The design pattern reduces space complexity of machine learning models from O(N) to O(1), where N is the number of threads. Figure 6 shows the proposed architecture of multi-threads, single model. Moreover, as the model is created in a thread-safe manner, there is no concern about multithreading problems. In-memory machine learning model and Singleton pattern enables the system to make inference as early as possible and to scale out concerning about increasing number of threads and memory usage.

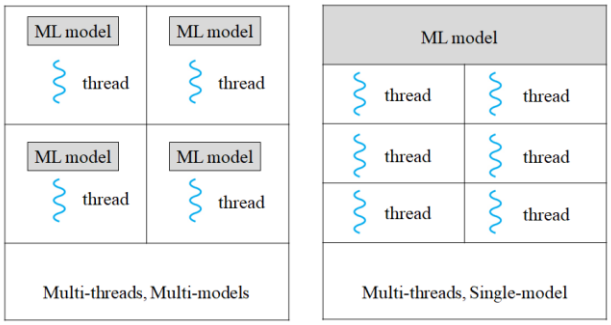


Figure 6. Multi-threads, Multi-models and Multi-threads, Single model.

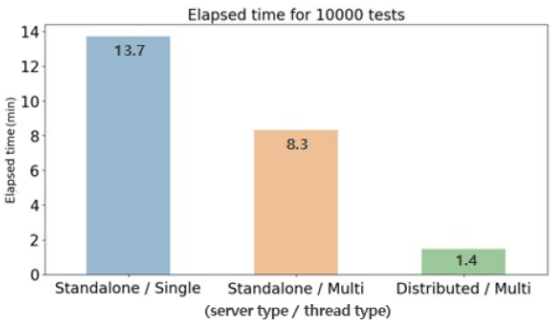


Figure 7. Performcance comparison of standalone system and distributed system.

Distributed systems

Distributed systems also contribute to scalability in the proposed system. While handling and processing terabytes of simulation log datasets, it takes long time to get results. When the number of projects increases, the processing time will increase proportionally. To prevent this, the proposed system consists of distributed servers and databases. Spark, an open source for distributed processing, and MongoDB, an open source for distributed systems, is used. Moreover, using multithreading, the system maximizes its performances. Figure 7 shows the comparison of data processing time in different types of systems: standalone and single thread system, standalone and multithread system and distributed and multithread system. The distributed and multithreading system reduces processing time remarkably. For this reason, the system can be scaled out easily even if the number of projects and datasets increases.

Automated Pipelines

Automated pipelines include pipelines from data collection to test case prioritization and automatic regression. In the proposed system, automated pipelines are implemented as Directed Acyclic Graph (DAG) in Airflow. As we have already used automatic design verification system [10], these pipelines are appended at the end of verification flow: generating logs. When test cases generate logs, the proposed pipelines in this paper pick the logs, process data and run the proposed prioritization algorithm. Finally, the algorithm recommends the optimal order of test cases. This flow works without any human intervention such as clustering manually, finding optimal hyper parameters for clustering algorithms, or reordering test cases based on domain knowledge.

V. Result

To validate the proposed test case prioritization algorithm and the system, hundreds of thousands of test cases were addressed from one of our verification projects. The dimension of components was reduced by 92% by using PCA dimensionality reduction algorithm. In clustering phase, K-means algorithm was chosen as it is suitable for our distributed system and shows better performance for our datasets compared to other clustering algorithms. In test case prioritization phase, five algorithms were compared: *Without-TCP, Random, Farthest-First, Closest-First and Error-Count / Runtime-wise* algorithm.

The evaluation was performed considering how early 90% of errors were detected in design verification process. In the original system, *Without-TCP*, total regression time was the longest with approximately 368 hours in Table 1, meaning regression was inefficiently done to detect different errors so far. On the contrary, based on the proposed algorithm, design verification time has decreased when test cases were prioritized after clustered by components.

Of all algorithms, *Farthest-First* algorithm took the shortest verification time to achieve 90% of error detection rate as shown in Figure 8. The number of test cases to find 90% of errors also significantly decreases 23.9% with *Farthest-First* algorithms in Table 1. When comparing *Closest-First* algorithm and *Farthest-First* algorithm, it was observed that test cases in the farthest position of the centroid helped to find more different errors than the closest position.

Before the experiment, *Error-Count / Runtime-wise* algorithm was expected to have the best performance, however, the result turned out to be different. The possible reason for this result is that as we compared different versions of components, the number of errors and run time were changed across different versions. However, it is expected that this problem can be resolved considering the history of errors and components in the whole cycle of design verification.

In sum, the proposed algorithm shows reduced verification time with the least number of test cases. The proposed system is flexible and data-driven in that it selects the optimal test case prioritization algorithm depending on datasets.

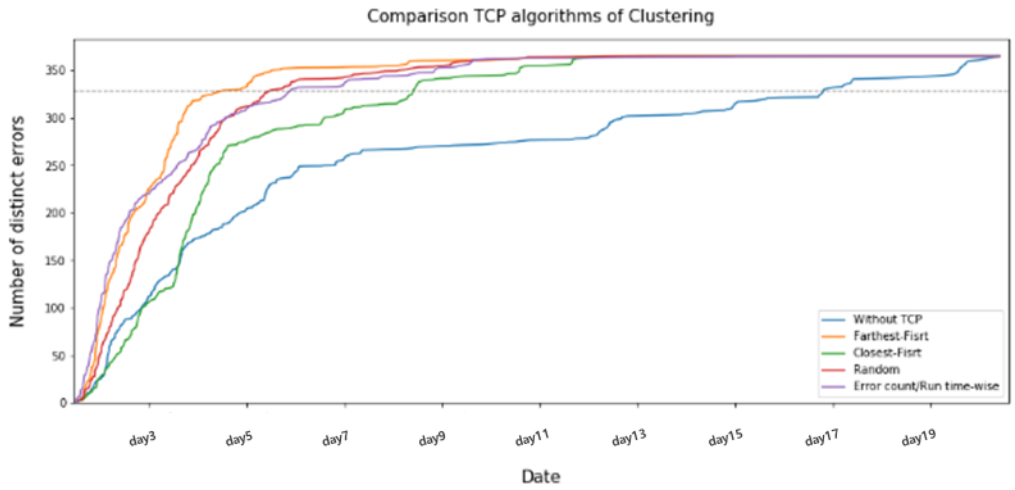


Figure 8. Comparison of Test Case Prioritization (TCP) algorithms of Clustering.

Table 1. Time for detecting 90% errors of Test Case Prioritization algorithms.

Test Case Prioritization	Time for detecting 90% errors / Total run time (%)	Ratio of test cases for detecting 90% errors (%)
Without-TCP	368.57 hours / 459.52 hours (80.2 %)	100 %
Farthest-First	73.08 hours / 315.37 hours (23.2 %)	76.1 %
Closest-First	166.89 hours / 288.22 hours (57.9 %)	97.7 %
Random	96.7 hours / 325.25 hours (29.7 %)	87.5 %
Error-count/Runtime-wise	106.79 hours / 386.02 hours (27.7 %)	90.6 %

For weighted clustering, critical errors such as accessing registers were given weights. To evaluate the effect of weighting, we compared the first occurrence of specific errors. In weighted clustering, specific errors that are given weights were found in only 19 hours after starting regressions including simulation runtime, where the simple clustering algorithm took 45 hours to find the errors. According to the result, time for detecting 90% errors was delayed by up to three days compared to the simple clustering algorithm. Since the purpose of weighted clustering is to find specific errors in the earlier stage, there is trade-off between detecting different errors and finding specific errors. Therefore, the proposed algorithm can be configurable depending on purposes.

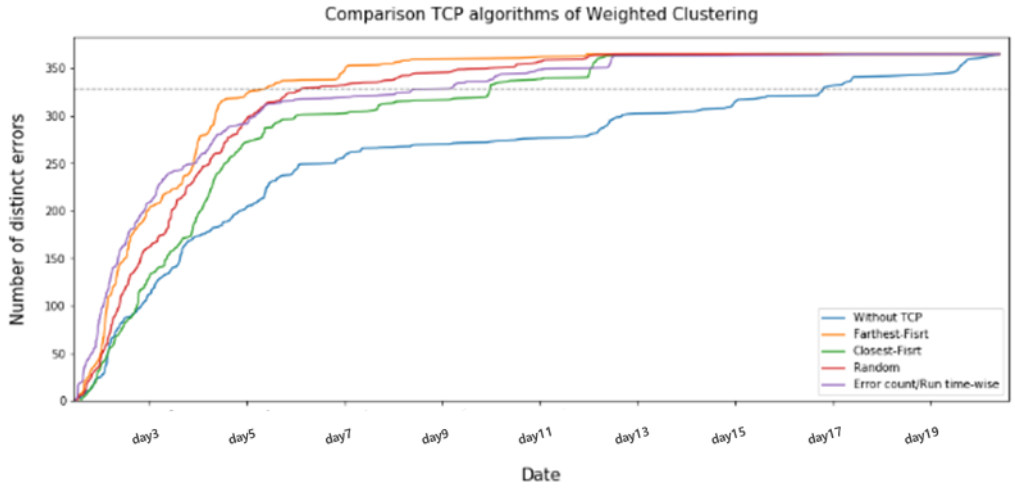


Figure 9. Comparison of Test Case Prioritization (TCP) algorithms of Weighted Clustering.

VI. Conclusions

In design verification, it is almost impossible to review terabytes of simulation logs and debug errors one by one. This paper proposes the test case prioritization algorithm based on clustering. The proposed prioritization algorithm finds different errors or important errors as early as possible. The clustering algorithm uses components as features of test cases, such as tasks, functions and errors. After clustering, test cases are prioritized in each cluster considering weights of components or how early the algorithm reaches a specific rate of error detection. We also propose robust and scalable system architecture handling terabytes of data for practical use in design verification industry. The proposed system is configurable in that weights can be given depending on purposes and appropriate prioritization algorithm is selected depending on data. All pipelines run automatically without human intervention.

To validate the proposed algorithm and system, hundreds of thousands test cases were used as datasets. Compared to the conventional system without any prioritization, the proposed prioritization algorithm showed remarkable decrease in design verification time. This indicates that the algorithm and system enable verification engineers to help to reduce repetitive and tedious debugging task and to focus on finding different or critical errors in digital design.

Even if the proposed algorithm showed improvements in debugging process, there still are many challenges. First, it is possible that the name of components can be updated even if there is no change in actions, and this may reduce correlation accuracy between clustering results and test cases. Instead of considering the literal meaning of components, the context such as functions should be considered to solve this challenge. Second, this paper uses only K-means clustering algorithm, as it is suitable for our datasets. Other clustering algorithms can be used in the future algorithm. Finally, the algorithm is based on a specific version of the project, but it can be improved by considering overall history of projects. Such challenges are under consideration in our future works.

REFERENCES

[1] Chen, Wen & Ray, Sandip & Abadir, Magdy & Bhadra, Jayanta & Wang, Li-C. (2017). Challenges and Trends in Modern SoC Design Verification. IEEE Design & Test. PP. 1-1. 10.1109/MDAT.2017.2735383.

[2] Harry D. Foster. 2015. Trends in functional verification: a 2014 industry study. In Proceedings of the 52nd Annual Design Automation Conference (DAC '15). Association for Computing Machinery, New York, NY, USA, Article 48, 1–6.

[3] B. Miranda, E. Cruciani, R. Verdecchia and A. Bertolino, "FAST Approaches to Scalable Similarity-Based Test Case Prioritization," 2018 IEEE/ACM 40th International Conference on Software Engineering (ICSE), 2018, pp. 222-232, doi: 10.1145/3180155.3180210.

[4] A. Wahba, J. Hohnerlein and F. Rahman, "Expediting Design Bug Discovery in Regressions of x86 Processors Using Machine Learning," 2019 20th International Workshop on Microprocessor/SoC Test, Security and Verification (MTV), 2019, pp. 1-6.

[5] Fang, C., Chen, Z., Wu, K. et al. Similarity-based test case prioritization using ordered sequences of program entities. Software Qual J 22, 335–361 (2014).

[6] T. B. Noor and H. Hemmati, "A similarity-based approach for test case prioritization using historical failure data," 2015 IEEE 26th International Symposium on Software Reliability Engineering (ISSRE), 2015, pp. 58-68, doi: 10.1109/ISSRE.2015.7381799.

[7] Flemström, D., Potena, P., Sundmark, D. et al. "Similarity-based prioritization of test case automation". Software Qual J 26, 1421–1449 (2018).

[8] S. K. Gonugondla, M. Kang and N. R. Shanbhag, "A Variation-Tolerant In-Memory Machine Learning Classifier via On-Chip Training," in IEEE Journal of Solid-State Circuits, vol. 53, no. 11, pp. 3163-3173, Nov. 2018, doi: 10.1109/JSSC.2018.2867275.

[9] Zaharia, Matei, et al. "Spark: Cluster computing with working sets." HotCloud 10.10-10 (2010): 95.

[10] Seonghee Yim, Hanna Jang, Sunchang Choi and Seonil Brian Choi, "Accelerating SOC Verification using Process Automation and Integration" Design Verification Conference (DVCon), 2020.