AN ADAPTIVE SOFTWARE FAULT-TOLERANT FRAMEWORK FOR UBQUITOUS VEHICULAR TECHNOLOGIES

Muhammad Rizwan, Aamer Nadeem, Muddesar Iqbal, Sohail Sarwar, Muhammad Safyan, and Zia Ul Qayyum

ABSTRACT

The probability of the occurrence of faults increases manifolds when program lines of code exceed a few thousand in ubiquitous applications. Fault mitigation in ubiquitous applications, such as those of autonomous vehicular technologies (VTs), has not been effective even with the use of formal methods. Faults in such applications require exhaustive testing for a timely fix, which seems infeasible computationally. This emphasizes the imperative role of software fault tolerance (SFT) for autonomous applications. Several SFT techniques have been proposed, but failures revealed in VT applications imply that existing SFT techniques need to be fine-tuned. In this article, current replication-based SFT techniques are analyzed and classified with respect to their diversity, adjudication, and adaptivity. Essential parameters (reliability, time, variance, etc.) for adjudication, diversity, and adaptiveness are recorded. The identified parameters are mapped to different techniques (e.g., AFTRC, SCOP, VFT) for observing their shortcomings. Consequently, a generic framework named Diverse Parallel Adjudication for Software Fault Tolerance (DPA-SFT) is proposed. DPA-SFT addresses the shortcomings of existing SFT techniques for VTs with the added value of parallel and diverse adjudication. A prototype implementation of the proposed framework has been developed for assessing the viability of DPA-SFT over modules of VT. An empirical comparison of the proposed framework is performed with prevalent techniques (AFTRC, SCOP, VFT, etc.). A thorough evaluation suggests that DPA-SFT performs better than contemporary SFT techniques in VTs due to its parallel and diverse adjudication.

INTRODUCTION

The ubiquitous applications in human life demand a higher degree of reliability due to their safety-critical nature, including autonomous vehicles, air traffic control, auto-pilots, unmanned aerial vehicles (UAVs/drones), and so on. In these applications, the desired level of reliability ranges between $10^{-9}$ to $10^{-9}$ failures per hour [1]. However, this mark of reliability has not been achieved yet. Consequently, countless accidents have occurred, incurring the loss of human lives, environmental calamities, and property wreckage.

These potential threats emphasize the imperative need for mitigating application faults in safety-critical applications (especially autonomous and ubiquitous technologies) so that the degree of reliability in the mentioned systems can be enhanced to prevent all the damage.

Nearly 1.2 million lives are lost every year due to traffic accidents in urban vicinities [1]. This factor has been given special consideration while developing autonomous vehicular applications (for ground, aerial, and underwater vehicles) to ensure the aspects of safety, reliability, and efficiency. These techniques [2] include correctness of vehicular coordination problems by satisfiability module theories (SMT), an automatic formal verification tool on distributed coordinates, manual proof strategies to avoid collision, and the 2-3 theorem for fault safety [3]. The safety of vehicular technology (VT) has been improved through formal verification over the decision control module of “lane change” using lateral state managers [4].

Generally, VTs may use four approaches to cope with application faults: fault forecasting, fault prevention, fault removal, and fault tolerance (i.e., software fault tolerance, SFT). The immediate functional impact is usually achieved by fault prevention, removal, and tolerance. When there are more than a few lines of code (LoCs), the probability of the occurrence of faults increases despite employing formal methods of fault prevention [5]. Moreover, the idea of performing exhaustive testing for fault removal is not practical due to time/computation constraints. Therefore, opting for SFT is the best choice to prevent consequences of residual faults in VTs for timely reconfiguration, maintenance, and graceful degradation. Here, less critical operations are terminated for resource allocation to critical ones to ensure availability.

SFT techniques have been exploited to circumvent asserted failures in VTs with the presence of application faults. A variety of approaches have been proposed to complement the SFT process augmented with VTs [6]. These approaches can be categorized broadly into adaptive and non-adaptive techniques [7]. Contrary to non-adaptive ones, adaptive techniques dynamically maneuver themselves for ensuring quality of service (QoS) in VTs (availability time, required reliability, degree of fault tolerance, etc.) and operating environment (number of processes, storage, etc.) [8]. Extreme diversity in the available

Digital Object Identifier: 10.1109/MCOMSTD.001.2000012

Author affiliations

2471-2825/20/$25.00 © 2020 IEEE

IEEE Wireless Communications • December 2020
resources and modules of VT applications may be catered only through adaptive techniques. Thus, the proposed framework is focused exclusively on adaptive SFT techniques. Some of the adaptive SFTs are detailed as follows. The most prevalent adaptive SFT techniques widely used for VTs are self-configuring optimal programming (SCOP), virtualization and fault tolerance (VT), adaptive N-version programming (ANVP), and adaptive fault tolerance in real-time cloud computing (AFTRC). SCOP was the first known SFT technique that selects and executes variants, and then adjudicates the result of the variants dynamically. However, it is affected adversely by undetected similar errors (addressed by VT and AFTRC). Later on, both VT and AFTRC were proposed to tolerate faults but could be generalized to other software components safely. A qualitative comparison of these approaches was carried out, asserting AFTRC as a more effective approach. A novel framework named diverse parallel adjudication-based SFT (DPA-SFT) has been proposed to address the limitations of existing SFT techniques. Here, adjudication is a process for determining if the correct output is produced by a technique. Parallelism is the conduciveness of the architecture to parallel execution of adjudication. DPA-SFT selects the variants and adjudication mechanism to address the limitations of existing techniques when applied with VTs. It constitutes efficient configuration in the selection of variants and then in adjudication, with the added feature of parallel and diverse adjudication. Empirical and descriptive validation of DPA-SFT advocates the validity of the proposed framework for VTs. This superiority of DPA-SFT is signified by the aspects of less time, more reliability, and time-resource optimization. Details of DPA-SFT are furnished. The rest of the article is structured as follows. We present exploration and a brief rationale of SFT in VTs, and existing SFT techniques along with their pros and cons. We dedicate a section to the detailed description of DPA-SFT: modules of the proposed framework and some of its exclusive features. Empirical comparison is furnished by designing different experiments. We then conclude the work with potential future directions.

**Related Work**

**Self-Configuring Optimistic Programming Scheme**

SCOP aims to run a minimum number of variants enough to achieve maximum reliability. Each phase of SCOP considers a subset of variants, presented for adjudication by using the information collector syndrome. When results are releasable, verification stops. Selection may lead to successful adjudication by the adjudicator. The selection of variant is based on the current one, not yet executed. Finally, the remaining variants are executed. “Success” is flagged upon successfully obtaining the required probability; otherwise, “Failure” is flagged even if all the yet unused variants were executed.

**Adaptive N-Version Programming**

The NVP approach is generally defined for a fixed amount of duplicate and an immutable set of versions. This limitation is addressed by an adaptive NVP-based algorithm, A-NVP, where configurations are dynamically constructed. A-NVP considers application-specific information and configures the redundancy related dimensions by means of user-defined parameters. It is designed to meet application-specific requirements regarding time and resources. Keeping in view the application domain, A-NVP may raise the “Failure” flag to proceed with suboptimal redundancy.

**Adaptive Fault Tolerance in Real-Time Cloud Computing**

AFTRC is a fault tolerance technique for real-time applications running on cloud infrastructure. This scheme tolerates faults based on the reliability of each virtual machine running on the cloud. The selection and removal of a virtual node are based on its reliability. Two nodes are considered: virtual machine and adjudication. A virtual machine contains the real-time application along with an acceptance test that validates its logic. This scheme provides both forward recovery as well as optional backward recovery in the context of VTs.

**Adaptive Fault Tolerance in Autonomous Vehicular Technologies**

A fault-tolerant vehicular application gives a sufficient amount of time for counter actions if a critical fault occurs that enables stopping for vehicles safely. The complexity of autonomous vehicles cannot be measured easily due to the state explosion problem (hence the ensuring of SFT). Thus, model checking is a widely used technique to perform formal verification of these applications. Another research work [9] focuses on robot control and multi-agent planning in VT via formal methods in UAV surveillance, intelligence, and reconnaissance missions. LTL was used as the sole specification language. The major purpose of this research is to plan multiple-vehicle missions by a single operator, which can execute a set of tasks to multiple UAVs. The research in [10, 11] focuses on a distributed car control system, which can help to measure car safety hazards effectively by coordinating their control actions. Reliability issues have been faced by employing these models such as distributed car control with hybrid systems. It makes the verification of safety objectives challenging, such as collision freedom verification during the process of local lane control, global lane control, and local highway control using formal techniques. Thus, a thorough mechanism is desired to measure and manage SFT in these modules (related to lane control) of autonomous VTs.

**Evaluation Criteria**

There are usually three main stages in all adaptive SFT techniques: configuration, adjudication, and diversity. Every stage requires some features to be addressed. In [12], 25 parameters are enumerated, from which 19 parameters are taken based on their applicability in our scope or any general application (i.e., modules of lane control in VTs). Furthermore, three new evaluation parameters have been introduced: application-independent adjudication, parallelism, and configuration before first execution.

**Application-independent adjudication** is an adjudication-related property. It occurs when an
more than once.
node that can be traversed only once.
node that has been traversed
28
Stage 2
FIGURe 1. Proposed framework: DPA-SFT.
Table 1. Software and hardware requirements of DPA-SFT.
| Requirements          | Minimum functionality | Moderate functionality | Full functionality |
|-----------------------|-----------------------|------------------------|--------------------|
| Resource              | 2 × processors        | 3 × processors         | ≥ 7 × processors   |
|                       | 1 × variants           | 2 × variants           | ≥ 6 × variants      |
|                       | 1 × AT                 | 1 × comparator          | 1 × AT             |
|                       | 1 × controller         | 1 × controller          | 1 × controller      |
| Configuration         | Experimentally calculated reliabilities of variants | Experimentally calculated reliabilities of variants | Experimentally calculated reliabilities of variants |

Table 1. Software and hardware requirements of DPA-SFT.

FIGURE 1. Proposed framework: DPA-SFT.

adjudicator is very specific to an application and cannot be generalized. This leads to the additional cost of the adjudicator’s development. Application-specific dependability disregards universality and brings additional development cost. Thus, it is an adjudication anomaly and is associated with AT.

Parallelism is a conduciveness of architecture for parallel execution of variants or adjudication. This greatly saves time overhead, which is there in sequential variants’ execution and adjudication.

Configuration before first execution assists in optimum resource utilization, even on the first run. This could be done by keeping in view the software/hardware against user time and reliability requirements. Therefore, the selection of variants and efficient utilization events before and after (even the first) execution of the program can prevent system failure.

A variety of adaptive SFT parameters have been identified, but 22 evaluation parameters were selected from [12]. R and P are based on their resource and performance impact. This influence may be presented as positive or negative, + or –. Moreover, AFTRC showed better performance, so it has been selected as a baseline technique for comparative evaluation.

DPA-SFT: The Proposed Framework

The DPA-SFT framework has been proposed to overcome the deficiencies of existing techniques while maintaining their strengths. Table 1 shows the minimum software and hardware require-ments: DPA-SFT constitutes multiple adjudicators, which work in parallel. However, DPA-SFT can work even with only one variant and one adjudicator (i.e., AT), but for exploiting the benefits of its complete functionality, there is a need to have at least six variants, and seven processors with all the adjudicators.

Structure of DPA-SFT

DPA-SFT has 9 components, as shown in Fig. 1: 2 phases1 (single border without gray filling), 4 stages2 (marked with gray filling), 2 assistive components (marked with black filling and double border), and 1 permanent but editable data structure V-repository (marked by an empty circle and dotted border). The block diagram shows two types of transitions. The first represents a transfer of control, marked by a dotted line with a double arrow and tail, and the second is the transfer of data, marked by a bold line with a filled arrow. Then there is an arrow coming from the watchdog timer going through the result repository to the ending node. This denotes the completion of the algorithm while checking the value in the resulting repository.

Phases of DPA-SFT

Elicitation Phase: In this phase, all the required information has been elicited, which is necessary for effective configuration, efficient reliability measurement, and resource awareness. This information includes:

• Number of available processors, denoted by |R|
• Minimum available time, denoted by Tmin
• Maximum available time, denoted by Tmax
• Minimum required reliability, denoted by Rmin
• Maximum required reliability, denoted by Rmax

Configuration Phase: The configuration phase is responsible for the selection of variants and their dissemination in buckets. This phase makes a decision based on information collected in the elicitation phase, and data read from the variants repository.

The selection of variants has been made under the following rules:

• Select from available variants having execution time less than (Tmax + α + β), named as InTimeV, where α is the time taken by configuration, adjudication, result transformation, and so on, and β is the additional time taken by certain hardware.
• Select the variant from InTimeV having reliability (upon success) greater than or equal to Rmax named ReliableV.
• Select the most reliable variants to form ReliableV that can be run on (|R| − 1) resources and named selected variants (SV). One has been subtracted because one processor is reserved for timing.

Dissemination has been done under the following rules:

• Variants having reliability less than or equal to low-level (LL) are subjected to AT bucket (AB).
• Variants having reliability greater than low-level are examined in such a way that: If any pair of variants exist with a reliability

1  Phase is a term used for a node that has been traversed only once.
2  Stage is a term used for a node that can be traversed more than once.
difference greater than or equal to degree of diversity (RD), it is sent to the comparator bucket (CB). Otherwise, variant(s) would be sent to the voters bucket (VB).

- If VB has less than two variants, these variants would also be sent to AB.

**STAGES OF DPA-SFT**

**Variants Execution Stage:** In this stage, selected variants are executed in a multiprocessor environment. Every variant is expected to have the capability to terminate its execution if its execution time becomes greater than \((\text{Time}_V + \beta)\). Such variants send a null result as an output.

In the case of involvement of MHD, a variant once executed with the original and re-expressed input may not be re-executed; only its result is used for adjudication.

**Adjudication Stage:** The adjudication stage has three modules.

- The **voter module** contains the “Majority voter” and “voter result assessor.” Voter result assessor halts until results from all the variants placed in VB provide a result. After all the results are received, they are subjected to “ResultVB” and then to voters for adjudication. If the majority is reached then the “Reliability assessor” is invoked. In case of a lack of majority, all the results are sent to ResultAB for preventing the occurrence of MCR.

- The **comparator module** contains “Comparator” and “Comparator result assessor.” Comparator result assessor is halted as long as both variants in a pair produce results. Once both the variants give results, it is placed in “ResultCB” and checked by a comparator. If the comparison stage passes then the “Reliability assessor” is invoked. But in case of failure, both results are sent to “ResultAB” to prevent the occurrence of MCR.

- The **AT module** consists of AT and AT-assessor. AT-assessor waits until it gets the result from any variant of AB. When all the resulting variants are adjudicated, it checks if there is any result left for adjudication in CB or VB. Those results, which can be sent to AB upon failure, prevent the occurrence of MCR.

**Maximal Hybrid Diversity Stage:** It is the input generator module. The input is produced for the first time. All the variants may not produce results with reliability at least \(R_{\text{max}}\), so this phase re-expresses the input.

Successful variants and re-expressed ones are not sent to MHD. Such variants are considered as fail variants, and their reliability gets decremented. MHD is done only for the fail variants.

**MEMORY COMPONENTS**

**Variants Repository:** The variants repository is a storage component that contains the following information for all the available variants:

- Identity of a variant usually denoted by \(V_i\) where \(i\) is an index of a variant
- Reliability of a variant, denoted by \(\text{Rel}_V_i\)
- Trails faced by a variant, denoted by \(\text{Trial}_V_i\)
- Time of execution of a variant, \(\text{Time}_V_i\)

It is used by the configuration stage and reliability assessor. The variants repository is updated after every trial for all participating variants, regardless of getting a pass or fail in adjudication.

**RESULT REPOSITORY:** The result repository is a storage component that contains the following data from a variant having reliability greater than or equal to the \(R_{\text{min}}\) and is denoted by \(V_x\):

- Result of \(V_x\), denoted by \(\text{Result}_V_x\)
- Reliability of \(V_x\), denoted by \(\text{Rel}_V_x\)
- Time of execution of \(V_x\), denoted by \(\text{Time}_V_x\)

If any other variant achieves reliability greater than \(\text{Rel}_V_x\), \(V_x\) is replaced by that variant.

**ASSISTIVE COMPONENTS**

- **Reliability Assessor:** Its responsibility is to calculate reliability and then store it in the variants repository. Successful variants’ reliability is increased, and failed variants’ reliability is decreased. Reliability is a real number that is always between 0 to 1.

- **Watch Dog Timer:** It is an independent component responsible for checking the elapsed time \(T_{\text{elapsed}}\). It starts right after the elicitation phase.

**EXCLUSIVE FEATURES OF DPA-SFT**

DPA-SFT offers a few important features that have not been considered so far to the best of our knowledge. This section briefly discusses these features.

**Consideration of Minimum Available Time:** This is very beneficial in scenarios where prior results are not effective enough for acceptance and considered as performance failure. In DPA-SFT, once the elapsed time is less than minimum...
The DPA-SFT disseminates variants, to remove chances of occurrence of similar errors by adjudicating the most reliable variants by the voter and the most diverse variants by the comparator. As detailed implementation information of the variants is unavailable, their diversity has been assessed using reliability information.

### Provision of a Parallel Adjudication Mechanism
This feature is provided by either multiple copies of AT or by using the comparator for diverse and the voter for the most reliable variants.

### Minimizing the Chance of Similar Errors’ Occurrence
The chance of occurrence of similar errors is minimized by adjudicating the most diverse variants through the comparator and the most reliable variants by the voter.

### Optimal Configuration
Optimal configuration before execution for selection and dissemination of variants is important. This entails selection of quality variants that could give results in time and are reliable enough to meet reliability requirements and could be run in available resources. The DPA-SFT disseminates variants to remove chances of occurrence of similar errors by adjudicating the most reliable variants by the voter and the most diverse variants by the comparator. As detailed implementation information of the variants is unavailable, their diversity has been assessed using reliability information.

### Incorporation of Data and Design Diversity
Both types of diversity are used to cope with design and input related faults together with the added feature of MHD.

### Least Resource Demanding
DPA-SFT provides an architecture that can execute with as minimal resources as two processors. The first processor is used for the execution of variants and AT; the other is used as the WatchDogTimer. Once there are as many processors as variants, DPA-SFT can effectively work in the absence of AT.

### Adjudication
This is accomplished by AT, which evaluates every result that is failed by the comparator and voter.

### Table 2: Time (μs) taken by different activities in DPA-SFT.

| Activity            | Tasks                      | 1st trial | 2nd trial | 3rd trial | 4th trial | 5th trial | Average |
|---------------------|----------------------------|-----------|-----------|-----------|-----------|-----------|---------|
| Configuration       | Timer filtration           | 20        | 18.7      | 17.5      | 19.3      | 17.5      | 18.6    |
|                     | Reliability filtration     | 13.6      | 12.1      | 12.1      | 11.5      | 12.1      | 12.3    |
|                     | Resource filtration        | 16.3      | 13.9      | 16.9      | 20.5      | 19.9      | 17.5    |
|                     | Local lane control         | 1427.2    | 1422.4    | 1400.6    | 1386.1    | 1383.7    | 1404    |
|                     | Global lane control        | 185.3     | 182.3     | 179.3     | 176.9     | 170.3     | 178.8   |
|                     | Cruise control module      | 266.8     | 264.4     | 260.8     | 254.2     | 254.2     | 260.1   |
|                     | Overtaking module          | 297       | 295.2     | 294       | 288       | 285.6     | 292     |
|                     | California PATH tracking   | 298.2     | 296.4     | 292.8     | 286.8     | 285.6     | 292     |
|                     | CICAS                      | 1,126.5   | 1,197.8   | 1,102.4   | 1,099.4   | 1,078.8   | 1,120.94 |
|                     | Distance to leading vehicle| 736.5     | 723.3     | 719.6     | 717.8     | 716.6     | 722.8   |
|                     | Percentage traffic light   | 1274.8    | 126.8     | 125       | 119.5     | 117.1     | 123.2   |
| Variants’ execution | 1st condition check        | 81.5      | 79.7      | 79.1      | 76.7      | 76.1      | 78.6    |
|                     | 2nd condition check        | 54.9      | 53.7      | 50.7      | 50.1      | 49.5      | 51.8    |
|                     | 3rd condition check        | 246.9     | 202.2     | 200.4     | 198.6     | 176.3     | 204.9   |
|                     | AT                         | 383.4     | 355.7     | 350.2     | 325.4     | 301.9     | 335.3   |
|                     | Comparator                 | 1.2       | 1.2       | 1.2       | 1.2       | 1.2       | 1.2     |
|                     | Voter                      | 63.4      | 61.6      | 61.6      | 60.4      | 60.4      | 61.5    |
|                     | Success variant            | 20175     | 28640.5   | 26681.8   | 27831.9   | 27472.2   | 27508.8 |
|                     | Fail variants              | 834.9     | 1015.5    | 846.4     | 975       | 843.3     | 903     |

![Image](SARWAR_LAYOUT.indd 30)
The selection of quality variants is key to success in SFT. Such selection is dependent on the reliability and execution time of variants. DPA-SFT continuously monitors and updates its information in every trial. The variants’ repository is updated every time, which helps to further enrich the configuration phase.

### Evaluation of DPA-SFT

Prototype implementation and execution has been done on the system with Intel Core™ i5-3317U CPU @ 1.70 GHz, 4 GB RAM, 64 bit operating system on Surface Pro –1 with Windows 10.

### Time Computed by Different Components

#### Time Taken by Configuration

The configuration phase works in three steps. In the first step, variants are filtered that may be executed within maximum available time. In the second step, variants are filtered that can run on available resources optimally. Time (in microseconds) taken by these steps is calculated in five trials, and average time taken in microseconds is considered. Average of the completion of tasks is considered in an average environment.

#### Time Taken by Variants Execution

Eight different algorithms of VT modules were acquired from Github [13] and other sources [14, 15] as variants. Then every algorithm was run 5 times to sort an array of 10,000 randomly generated elements. The average time taken was recorded in microseconds.

#### Time Taken by Adjudication

In order to measure the system tolerance, three adjudicators were developed: AT, comparator, and voter. Adjudication time was recorded first, by sending all the variants to AT, then to the comparator and then to the voter. It was assumed that all results have been passed by AT and got 100 percent consensus (when sent for comparison and voting).

### Consideration

#### Selection of Quality Variants

The selection of quality variants is key to success in SFT. Such selection is dependent on the reliability and execution time of variants. DPA-SFT continuously monitors and updates its information in every trial. The variants’ repository is updated every time, which helps to further enrich the configuration phase.

### Comparison to Overcome Adjudication Anomalies

As far as overcoming the 4-adjudication anomalies is concerned, AFTRC is the best (as shown in Table 2). So DPA-SFT was compared to AFTRC in addressing the adjudication anomalies. For this comparison, the following configurations were assumed:

- $R = 9$; $|V| = 8$
- $t_{max}$ = Total time taken for execution of all tasks

#### Overview of time and resource was taken by DPA-SFT, and then AFTRC was mapped to Tables 2 and 3, respectively, in adjudication (exclusively).

AFTRC has only AT to adjudicate results, whereas DPA-SFT has AT, comparator, and voter for adjudication based on the reliability of variants:

- DPA-SFT saves more time and resources when variants are either “reliable” or “reliable and diverse.”
- When reliable and diverse variants fail to tolerate MCR, DPA-SFT takes more time under the same resources to prevent the system from MCR.
- When variants are less reliable, resource and time consumption in DPA-SFT for adjudication is the same as in AFTRC.
- Similar error usually occurs when either variant is not reliable or is not sufficiently diverse. To ensure prevention of similar errors, DPA-SFT transmits only the most reliable variants to voters, then the most reliable but diverse variants to the comparator. Thus, in the best case scenario, there are a handful of resources and time saved. In the worst case, there could be an occurrence of similar errors.
- DPA-SFT is a novel approach to prevent the occurrence of MCR and to minimize the occurrence of similar errors with maximum time and resource savings.
The proposed technique is based on the presumption of available minimum and maximum required reliability, without discussing the computation of these reliabilities. The effort may be made to quantify the initial reliability of the variant before putting it into the proposed framework.

- DPA-SFT can work even in the absence of AT, and hence significantly minimizes the development cost and deliverance from an application-dependent adjudicator (i.e., AT).

**COMPARISON IN EFFICIENT CONFIGURATION**

For the comparison, the following environmental variables have been assumed:

- $|V| = 8; R = 9$
- $t_{max} < \text{Time of execution of all tasks (for best case)}$
- $t_{max} > \text{Time of execution of all tasks (for worst case)}$

Best case and worst case were considered. The best case means variants have been successfully declared unfit because a variant could not produce either results within time or required reliable results. Time and resources will vary in both conditions. Table 3 indicates the resource and respective utilization for selecting the most optimal solution.

- When none of the available variants could produce the results within the available time, the system is terminated, and a failure message is transmitted 29,228.7 μs before the adaptive techniques.
- When required reliability is not achieved by the available variants, DPA-SFT may be terminated at the time of configuration and sends a failure message 29,216.4 μs before all other adaptive techniques.

**CONCLUSION AND FUTURE DIRECTION**

The indispensability of reliable autonomous vehicles demands an extremely fault-tolerant system. The failure of any component in VT applications implies the failure of existing SFT techniques. It is caused by the lack of any of three aspects: adjudication, diversity, and adaptiveness. These shortcomings have been precisely highlighted and addressed in this research over a variety of test-beds from VTs. A framework named DPA-SFT has been proposed with optimum parameters. Comparative analysis of DPA-SFT with the prevalent approaches asserts the viability of the proposed framework. The prototype implementation of DPA-SFT enlists the associated overheads. Still, it is a feasible choice to prevent the consequences of failures. We look forward to further refining DPA-SFT by considering the following potential areas.

The proposed technique is based on the presumption of available minimum and maximum required reliability, without the computation of these reliabilities being discussed. The effort may be made to quantify the initial reliability of the variant before it is put into the proposed framework.

Data diversity is cost-effective relative to design diversity, but it has achieved sporadic attention by the research community. Therefore, an effort in the effectiveness of data diversity can be a good addition.

All the SFT techniques are heavily dependent on the reliability of adjudicators (AT and voter). Thus, there is a need to make more reliable adjudicators that are our potential future target.