An Adaptive Software Fault-Tolerant Framework for Ubiquitous Vehicular Technologies

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

***Abstract*—Probability for the occurrence of faults increases manifolds when program Lines of Code (LoC) exceeds a few thousand in ubiquitous applications. Faults 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, that 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 paper, current replication-based SFT techniques have been analyzed and classified with respect to their diversity, adjudication, and adaptivity. Essential parameters (such as Reliability, Time, Variance, etc) for adjudication, diversity, and adaptiveness were recorded. The identified parameters were mapped to different techniques (such as AFTRC, SCOP, VFT, etc) for observing their shortcomings. Consequently, a generic framework named "Diverse Parallel Adjudication for Software Fault Tolerance (DPA-SFT)" has been 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 was 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.**

***Index Terms*—Software Fault Tolerance, Vehicular Technologies, Ubiquity, Adjudication, N-Version Programming, Safety-Critical Systems**

I. Introduction

The ubiquitous applications in human life demand a higher degree of reliability due to their safety-critical nature such as Autonomous Vehicles, Air Traffic Control, Auto-Pilots and UAV Drones etc. In these applications, the desired level of reliability ranges between 10^{-8} to 10^{-9} failures per hour [1]. However, this mark of reliability has not been achieved yet. Consequently, countless accidents were encountered, incurring the loss of human lives, environmental calamities and property wreckages. These potential threats emphasize the imperative need for mitigating application faults in safety-critical applications (especially Autonomous and Ubiquitous Technologies). So that degree of reliability in stated systems can be enhanced for preventing all the damages.

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 assure the aspects of safety, reliability and efficiency. These techniques [2] include: correctness of vehicular coordination problems by satisfiability module theories (SMT), automatic formal verification tool on distributed coordinates, manual proof strategies to avoid collision and 2-3 theorem for fault safety [3]. The safety of 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. SFT). The immediate functional impact is usually achieved by fault prevention, removal, and tolerance. When there are more than few KLoC, the probability for the occurrence of faults increases despite employing the formal methods for fault prevention [5]. Moreover, the idea of performing exhaustive testing for fault removal is not practical due to time/computation constraints. Therefore, opting SFT is the best choice to prevent consequences of residual faults in VTs for timely reconfiguration, maintenance or graceful degradation. Here, lesser critical operations are terminated for resource allocation to critical ones to assure 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 assuring 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 resources and modules of VT applications may be catered only through adaptive techniques. So, the proposed framework is focused exclusively on adaptive SFT techniques. Some of the Adaptive SFTs have been detailed as follows: Most prevalent Adaptive SFT technique widely used for VTs are: Self-configuring optimal programming (SCOP), Virtualization and Fault Tolerance (VFT), Adaptive N-Version Programming (A-NVP) and Adaptive Fault Tolerance in Real-Time Cloud Computing (AFTRC). SCOP was the first known SFT technique that selects, executes variants and then adjudicates the result of the variants dynamically. However, it is affected adversely by undetected similar errors (addressed by VFT and AFTRC). Later on, both VFT 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 a conduciveness of architecture for parallel execution of adjudication (details om section E). 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 due to the aspects of lesser time, more reliability and time-resource optimization. Details of DPA-SFT have been furnished in section III. Rest of the paper is structured as follows: Section II comprises the exploration and brief rationale of SFT in VTs, existing SFT techniques along with their pros and cons. Section III is dedicated to the detailed description of DPA-SFT: modules of proposed framework DPA-SFT and some of its exclusive features. Empirical comparison is furnished in section IV by designing different experiments. Section V concludes the work with potential future directions.

II. Related Work

A. Self-Configuring Optimistic Programming scheme (SCOP)

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

B. Adaptive N-version programming (A-NVP)

N-version programming (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. The A-NVP considers application-specific information and configures the redundancy related dimensions by means of user-defined parameters. It is designed to meet the application-specific requirements regarding time and resources. Keeping in the view of the application domain, A-NVP may raise the "Failure" flag to proceed with the sub-optimal redundancy.

C. Adaptive Fault Tolerance in Real-time Cloud computing (AFTRC)

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

D. 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 assurance for SFT). So model checking is widely used technique to perform formal verification of these applications. Another research [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 vehicles missions by a single operator, which can execute a set of tasks to multiple UAV vehicles. The research by [10, 11] focuses on 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, collision freedom verification during the process of local lane control, global lane control and local highway control using formal techniques. So a thorough mechanism is desired to measure and manage SFT in these modules (related to lane control) of autonomous VTs.

E. 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 have been taken based upon their applicability in our scope or any general application i.e. modules of lane control in VTs. Besides, 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 adjudicator is very specific to an application and cannot be generalized. This leads to the additional cost of 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 $+$ and $-$. Moreover, AFTRC showed better performance so it has been selected as a baseline technique for comparative evaluation.

III. DPA-SFT - The Proposed Framework

A framework named "DPA-SFT" **D**iverse **P**arallel **A**djudication based SFT (DPA-SFT) has been proposed to overcome the deficiencies of existing techniques while maintaining their strengths. Table I shows the minimum software and hardware requirements: DPA-SFT constitutes multiple adjudicators, which work in parallel. However, DPA-SFT can work even with only 1 variant and one adjudicator (i.e. *AT*) but for exploiting benefits of its complete functionality, there is a need to have at least 6 variants, and 7 processors with all the adjudicators.

TABLE I
Software and Hardware Requirements of DPA-SFT

Requirements	Minimum functionality	Moderate functionality	Full functionality
Resource	2× processors	3× processors	$\geq$ 7× processors
	1× variants	2× variants	$\geq$ 6× variants
	1× AT	1× comparator	1× AT
			1× comparator
			1× voter
	1× controller	1× controller	1× controller
Configuration	Experimentally calculated reliabilities of variants	Experimentally calculated reliabilities of variants	Experimentally calculated reliabilities of variants

A. Structure of DPA-SFT

DPA-SFT has 9 components as shown in Figure 1. 2-phases[1] (single border without gray filling), 4-stages[2] (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 doted border). The block diagram shows two types of transitions. First represents a transfer of control, marked with a dotted line having double arrow and tail, second is the transfer of data, marked by bold line with a filled arrow. Then there is an arrow coming from watchdog timer going through result repository to the ending node. This denotes the completion of the algorithm while checking the value in the resulting repository.

[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

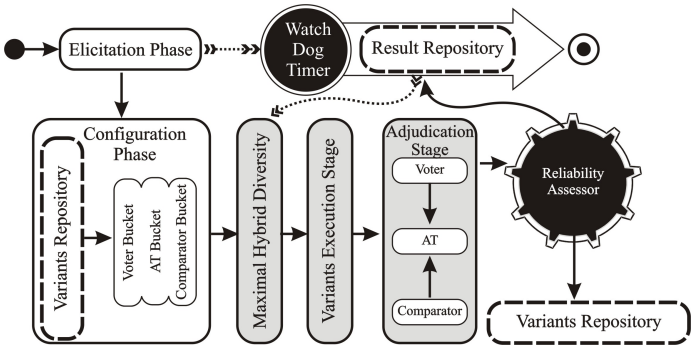

Fig. 1. Proposed Framework: DPA-SFT

B. Phases of DPA-SFT

1) 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:

1) Number of available Processors, denoted by $|R|$
2) Minimum available Time, denoted by T_{min}
3) Maximum available Time, denoted by T_{max}
4) Minimum required Reliability, denoted by R_{min}
5) Maximum required Reliability, denoted by R_{max}

2) Configuration Phase: Configuration phase is responsible for the selection of variants and their dissemination in buckets. This phase makes a decision based upon information collected in the elicitation phase, and data read from "Variants Repository". The selection of variants has been made over following rules:

1) Select from available variants having execution time lesser than $(T_{max} + \alpha + \beta)$ and named it as InTimeV. Where α is a time taken by configuration, adjudication and result transformation, etc, and β is an additional time taken by certain hardware.
2) Select the variant from InTimeV having reliability (upon success) is greater than or equal to R_{min} and named it as ReliableV
3) Select the most reliable variants to form ReliableV that can be run on $(|R| - 1)$ resources and named it as SV (Selected Variants). 1 has been subtracted because one processor would be reserved for timing.

Dissemination has been done on following rulings:

1) Variants having reliability less than or equal to Low-level (LL) are subjected to AT Bucket or simply (AB)
2) Variants having reliability greater than Low-level are examined in such a way that: If any pair of variants exist having a reliability difference greater than or equal to "Degree of Diversity" (RD) is sent to comparator Bucket (CB). Otherwise, variant(s) would be sent to Voters Bucket (VB).
3) If VB has less than two variants, these variants would also be sent to AB

C. Stages of DPA-SFT

1) Variants Execution Stage: In this stage, selected variants are executed in a multiprocessing environment. Every variant is expected to have the capability to terminate its execution if its execution time becomes greater than (TimeV + β). Such variants send a null result as an output. In 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.

2) Adjudication Stage: The adjudication stage has three modules:

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.

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.

AT Module consists of AT and AT-assessor. AT-assessor waits as long as it gets the result from any variant of AB. When all the resulting variants are adjudicated, it is checked if there is any result left for adjudication in CB or VB. It is because of those results that can be sent to AB upon failure to prevent the occurrence of MCR.

3) 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_{max}, so this phase re-expresses the input. Successful variants and re-expressed ones are not sent to MHD. Such variant are considered as fail variants and their reliability gets decremented. MHD is done only for the fail variants.

D. Memory Components

1) Variants' Repository: 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 $RelV_i$
- Trails faced by a variant, denoted by $TrialV_i$
- Time of execution of a variant $TimeV_i$

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

2) Result Repository: Result repository is a storage component that contains following data from a variant having reliability greater than or equal to the R_{min} and is denoted by V_x;

- Result of V_x, denoted by $ResultV_x$
- Reliability of V_x, denoted by $RelV_x$
- Time of execution of V_x, denoted by $TimeV_x$

If any other variant achieves reliability greater than RelVx, then V_x is replaced by that variant.

E. Assistive Components

1) Reliability Assessor: Its responsibility is to calculate reliability and then store it in Variants repository. Success variants reliability is increased and fail variants reliability is decreased. Reliability is a real number which is always between 0 to 1.

2) Watch Dog Timer: It is an independent component responsible for checking the elapsed time (Telapsed). It starts right after the elicitation phase. The working of the watchdog timer is shown in Flowchart below.

F. 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.

1) 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 min available time, the system either awaits either highly reliable result or wait until min. available time equalizes the elapsed time. When: $T_{min} > T_{elapsed}$ Then Seek more reliable result until $T_{elapsed} = T_{min}$

2) Consideration of maximum required results reliability: This is the upper limit of reliability that user/environment desires. The program tries to achieve maximum reliability as long as time factor permits to do so. When: $R_{min} \leq RelV_x \leq R_{max}$ $T_{elapsed} \leq T_{max}$ Then DPA-SFT seeks for more reliable result by re-expressing the input until either $RelVx \geq R_{max}$ Or $T_{elapsed} = T_{max}$

3) Initial variant's Reliability: AFTRC assigned reliability score of '1' to all nodes. Nevertheless, if it has not been run before. Likewise, VFT gives '0.5' reliability to all nodes. DPA-SFT considers experimentally calculated reliability, so it possesses mature figures of reliability. If reliability has not been calculated at the time of testing, it is considered as 0.5 for all variants.

4) Available Variants ($|V|$) and Available Processors ($|R|$): AFTRC and VFT supposed $|V| \approx \infty$ so it cannot be generalized nor configured in case of limited resources. Whereas DPA-SFT selects most reliable variants when $|V| \geq |R|$and all variants when $|V| \leq |R|$.

5) Application-Independent adjudication: This is incorporated by introducing comparator and voter. This significantly reduces development cost, saves time and provides deliverance from dependent adjudication. DPA-SFT can effectively work in the absence of *AT*.

6) Guard against MCR : This is accomplished by AT, which evaluates every result that is failed by comparator and voter.

TABLE II
TIME (μs) TAKEN BY DIFFERENT ACTIVITIES IN DPA-SFT

Activity	Tasks	1^{st} Trial	2^{nd} Trial	3^{rd} Trial	4^{th} Trial	5^{th} 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
Variants' execution	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	1,126.3	1,197.8	1102.4	1099.4	1078.8	1,120.94
	Distance to leading vehicle	736.5	723.3	719.6	717.8	716.6	722.8
	Percentage Traffic Light	127.4	126.8	125	119.5	117.1	123.2
Adjudication	1^{st} condition check	81.5	79.7	79.1	76.7	76.1	78.6
	2^{nd} condition check	54.9	53.7	50.7	50.1	49.5	51.8
	3^{rd} condition check	246.9	202.2	200.4	198.6	176.3	204.9
	AT	383.4	335.7	330.2	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
Reliability calcuation	Success variant	26917.5	28640.5	26681.8	27831.9	27472.2	27508.8
	Fail variants	834.9	1015.5	846.4	975	843.3	903

7) Provision of parallel adjudication mechanism: This feature is provided by either multiple copies of *AT* or by using comparator for diverse and voter for the most reliable variants.

8) Minimizing chance Similar Errors' occurrence: The chance for occurrence of similar errors are minimized by adjudicating the most diverse variants through comparator and most reliable variants by voter

9) Optimal configuration: Optimal configuration before execution for selection and dissemination of variants is important. 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 Voter and most diverse variants by Comparator. As detailed implementation information of the variants is unavailable, their diversity has been assessed using reliability information.

10) 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.

11) Least Resource Demanding: DPA-SFT provides an architecture that can execute with as minimal resources as two processors. The first processor to be used for the execution of variants and AT; other for WatchDogTimer. Once there are as many numbers of processors as the variants, DPA-SFT attempts to invoke all adjudicators and variants to run in parallel. Consideration

12) Selection of Quality Variants: The selection of quality variants is a key to success in SFT. Such selection is dependent upon 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 that helps to further enrich the configuration phase.

IV. EVALUATION OF DPA-SFT

Prototype implementation and execution has been done on the system having specifications: Intel Core (TM) i5-3317U CPU @ 1.70Ghz, 4GB RAM, 64 Bit Operating system on surface pro -1 with Windows 10.

A. Time computed by different components

1) Time taken by Configuration: The configuration phase works in three steps. In first step, variants are filtered that may be executed within max. available time. In second step, variants are filtered from inTime variants to acquire the required level of reliability. In 3^{rd} step those variants are selected form reliable results that can run on available resources optimally. Time (in μs) taken by these steps is calculated in 5-trials and average time taken in microseconds (μs) is considered. Average of the completion of tasks is considered in an average environment.

2) 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 thousand randomly generated element. The average time taken was recorded in microseconds (μs).

3) Time taken by Adjudication: In order to measure the system tolerance, three adjudicators were developed i.e. AT, Comparator and Voter. Adjudication time was recorded first, by sending all the variants to AT, then to Comparator and then to Voter. It was assumed that all results have been passed by AT and got 100% consensus (when sent for comparison and voting).

4) Time taken by Reliability Assessment: Reliability has been calculated (as per the formula shown in the respective flowchart) for both the pass and failed variants. Five trials were developed to re-calculate the reliability of variants and came up with the following results.

B. 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 with AFTRC in addressing the adjudication anomalies. For this comparison, following configurations were assumed:

1) R = 9; $|V|$ = 8;
2) 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 the table 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 upon the reliability of variants.

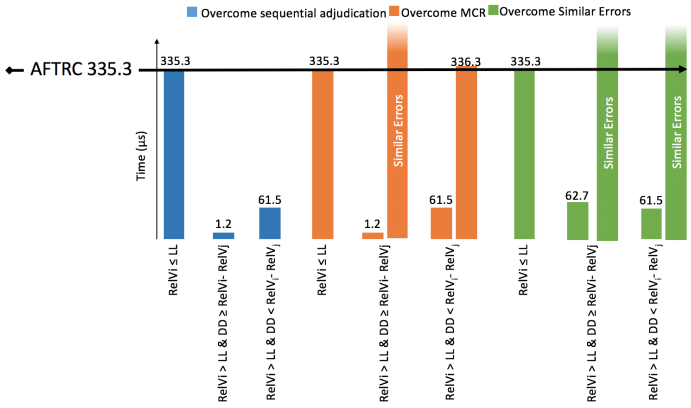

Fig. 2. Comparison of time (μs) utilization

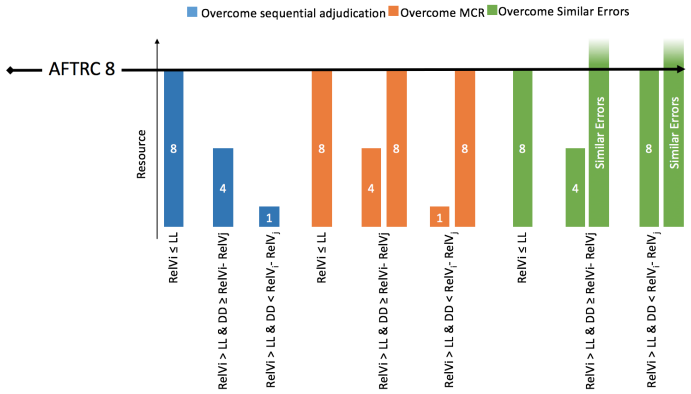

Fig. 3. Comparison of Resource Utilization

1) DPA-SFT saves more time and resources when variants are either “reliable” or “reliable and diverse”.
2) When reliable and diverse variants fail to tolerate MCR, DPA-SFT takes more time under the same resources to prevent system from MCR.
3) When variants are less reliable, resource and time consumption in DPA-SFT for adjudication is the same as in AFTRC.
4) Similar error usually occurs when either variant is not reliable or is not sufficiently diverse. To ensure prevention from similar errors DPA-SFT transmits only the most reliable variants to voters, then most reliable but diverse variants to the comparator. So 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.
5) DPA-SFT, a novel approach to prevent the occurrence of MCR and to minimize the occurrence of similar errors with maximum time and resource-saving.
6) DPA-SFT can work even in the absence of ’AT’, so significantly minimizes the development cost and deliverance from application dependent adjudicator i.e. AT.

C. Comparison in Efficient Configuration

For the comparison, following environmental variables have been assumed:

1) $|V|$=8; R=9
2) T_{max} < Time of execution of all tasks (for best case)
3) T_{max} > Time of execution of all tasks (for worst case)

Time and resource utilization in ”Fail” and ”Pass” configuration have been plotted in the Table IV. Best case and Worst case were considered: The best case means variants have been successfully declared unfit because either variant could not produce results within time or required reliable results. Time and resources will vary in both conditions. Table III furnishes

TABLE III
COMPARISON IN EFFICIENT CONFIGURATION UTILIZATION

DPA-SFT					*Other*	*Conclusion*
Time Resource Utilization		*Time Check*	*Reliability Check*	*Remaining execution*	*All Tasks*	
Result vs Available time						
Time (µs)	Best	18.6	0		29,247.3	DPA-SFT saves **29,228.7 µs**
	Worst		12.3	29,264.8		DPA-SFT takes 48.4 **µs more time**
Resources	Best	1	0		8	DPA-SFT saves **7 processors**
	Worst		1	8		DPA-SFT = other techniques
Required Reliability vs Acquired Reliability						
Time (µs)	Best	18.6	12.3	0	29,247.3	DPA-SFT saves **29,216.4 µs**
	Worst			29,264.8		DPA-SFT takes 48.4 **µs more time**
Resources	Best	1		0	8	DPA-SFT saves **7 processors**
	Worst			8		DPA-SFT = other techniques

that the resource and respective utilization for selecting the most optimal solution.

1) When none among 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.
2) When required reliability is not achieved by the available variants, DPA-SFT may be terminated at the time of configuration and sends failure message 29,216.4 μs before all other adaptive techniques.

V. Conclusion And Future Direction

The indispensability of reliable autonomous vehicles demands an extreme fault-tolerant system. Failures of any component in VT applications imply the failure of existing SFT techniques. It is caused by lack of any of the aspects: adjudication, diversity or in 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. But still, it is a feasible choice to prevent the consequences of failures. We look forward to further refine DPA-SFT by considering following potential areas: The proposed technique is based upon the presumption of available min and maximum required reliability, without being discussed the computation of these reliabilities. The effort may be put to quantity the initial reliability of the variant before put into the proposed framework. Data diversity is cost-effective relative to design diversity, yet it can achieve 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 upon the reliability of adjudicators (AT and Voter). So there is a need to make more reliable adjudicators that are our potential future target.

REFERENCES

[1] X. M. Z. e. a. L. Zhou, J. H. She, "Performance enhancement of repetitive-control systems and application to tracking control of chuck-workpiece systems," *IEEE Transactions on Industrial Electronics*, vol. 17, no. 6, pp. 1–18, 2019.

[2] M. C. G. Ali, N. Rahim, "Analysis and improvement of reliability through coding for safety message broadcasting in urban vehicular networks," *IEEE Trans. Veh. Technol*, vol. 67, no. 3, pp. 6774—6787, 2018.

[3] M. B. M. Asplund, A. Manzoor, "A formal approach to autonomous vehicle coordination," in *International Symposium on Formal Methods*, 2012.

[4] M. F. A. Zita, S. Mohajerani, "Application of formal verification to the lane change module of an autonomous vehicle," in *13th IEEE Conference on Automation Science and Engineering (CASE)*, 2017.

[5] M. A. J. Almeida, J. Rufino, "A survey on fault tolerance techniques for wireless vehicular networks," *MDPI Journal of Electronics*, vol. 8, no. 11, pp. 1358–1371, 2019.

[6] A. F. S Seelem, "Effective fault-tolerant control paradigm for path tracking in autonomous vehicles," *Systems Science and Control Engineering*, vol. 3, no. 1, pp. 177–188, 2015.

[7] B. N. A. Mihaly, P. Gaspar, "Multiple fault-tolerant in-wheel vehicle control based on high-level control reconfiguration," *IFAC-PapersOnLine*, vol. 50, no. 1, pp. 8606–8611, 2017.

[8] R. Y. X. Liu, Y. Shang, "A hybrid fault-tolerant control for nonlinear active suspension systems subjected to actuator faults and road disturbances," *Hindawi Journal of Vehicular Complexity, ISSN: 1076-2787 (Print)*, vol. 20, no. 12, pp. 15–29, 2020.

[9] U. T. L. Humphrey, "Formal specification and synthesis of mission plans for unmanned aerial vehicles," *AAAI Spring Symposium - Technical Report ER*, vol. 12, no. 2, pp. 112–126, 2014.

[10] L. N. M. Sarah, A. Platzer, "Adaptive cruise control: Hybrid, distributed, and now formally verified," *Formal Methods, Lecture Notes in Computer Science*, pp. 20–25, 2011.

[11] F. Z. W. Huang, K. Wang, "Autonomous vehicles testing methods review," in *IEEE 19th International Conference on Intelligent Transportation Systems (ITSC)*, 2016, pp. 121–129.

[12] M. Rizwan, A. Nadeem, and M. B. Khan, "An evaluation of software fault tolerance techniques for optimality," in *Emerging Technologies (ICET), 2015 International Conference on*. IEEE, 2015, pp. 1–6.

[13] [Online]. Available: https://github.com/udacity/self-driving-car/tree/master/vehicle-detection

[14] P. N. M. Sarah, L. Andre, "Adaptive cruise control hybrid, distributed, and now formally verified," in *Carnegie Mellon University*, 2011.

[15] Z. Q. M. S. S. Sarwar, Saad Zia, "Context aware ontology based hybrid intelligent framework for vehicle driver categorization," *Transaction on Emerging Telecommunication Technologies*, 2019.