Checkpointing Period Optimization of Distributed Fail-Operational Automotive Applications

Philipp Weiss*, Emil Daporta*, Andreas Weichslgartner† and Sebastian Steinhorst*
*Technical University of Munich, Germany; firstname.lastname@tum.de
†AUDI AG, Germany; andreas.weichslgartner@audi.de

***Abstract*—Achieving a cost-efficient fail-operational behavior of safety-critical software is crucial for autonomous systems. However, most applications hold a state such that a checkpoint is required to enable a safe recovery. Here, the challenge is to find the maximum possible checkpointing period while minimizing network and computational overhead. For this purpose, we present an approach to analytically derive the maximum checkpointing period by giving an upper bound on the number of missed computational steps due to failure effects. Worst-case results of our case study using a SLAM application are consistent with our analytically derived exact bound. Overall, by using our approach, a maximum achievable checkpointing period can be determined to reduce network overhead in order to achieve a cost-efficient and safe behavior of autonomous systems.**

I. Introduction

With the rapid development of new functionalities to achieve autonomous driving, the automotive industry sees itself confronted with increased safety requirements. Ensuring a fail-operational behavior is indispensable to achieve a safe behavior of autonomous systems. With a dynamic resource management, safety-critical applications can be restarted after the malfunction on a still functional electronic control unit (ECU). Here, graceful degradation can be used by shutting down non-critical applications on this ECU to free resources for the restarting safety-critical application. Instead of adding costly additional hardware resources, the existing resources can be repurposed [1].

However, a major challenge is that most applications have a state that might get lost during a failure such that a recovery might be impossible. Thus, in a system with passive redundancy, periodic checkpointing with rollback recovery can be used to send the state to another ECU, where the application can be restarted after a failure. Here, the challenge is to find a suitable checkpointing period such that application-specific constraints on the state data age are met and the network and processing overhead caused by sending the checkpoint is minimized. However, as it might take the system some time to recover from the failure, the state data also ages during the downtime such that the worst-case state data age can not only be determined by the checkpointing period.

As the implications of the state data age are application-specific, we use a Simultaneous Mapping and Localization (SLAM) algorithm, an application commonly used in autonomous systems such as robots or self-driving cars, as a real-world example to determine the effects on the quality of the application. Using the SLAM algorithm, a moving object creates a global map of its current environment and uses this map to navigate or deduce its position and orientation. In the following, we refer to automotive terminology, although the work is also applicable for other safety-critical autonomous systems. The work presented in this paper can be used to determine the maximum checkpointing period at which a safe operation within the bounds of the maximum allowed state data age is still possible. Therefore, we make the following contributions:

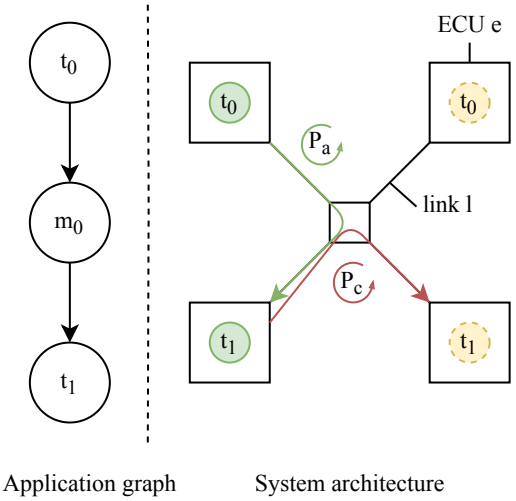


Figure 1: Representation of our system model with an example application and system architecture. The green circles indicate the active bindings of tasks t_0 and t_1, while the yellow circles indicate the passive task bindings. The message m_0 is sent periodically with the period P_a. A checkpoint is sent from the active task instance of t_1 to its passive task instance with the checkpointing period P_c.

- We analyze related work in the fields of fail-operational systems and checkpointing approaches in Section II.
- After introducing our application and failover model in Section III, we present our checkpointing approach in Section IV. Here, we formally introduce the checkpointing period and the achievable overhead reduction. By identifying components that influence the data age of a task we derive an upper bound for the worst-case data age. Using a state data age constraint we then derive the maximum possible checkpointing period that minimizes our network and processing overhead.
- In a case study we show the dependency between state data age and accuracy of an SLAM algorithm as a real-world example in Section V-B. Furthermore, we show

the applicability of our formal analysis by conducting failover experiments on our demonstrator setup with an agent-based software platform using the fail-operational distributed SLAM algorithm. Worst-case results of our randomized experiments are consistent with our analytically derived exact bound.

II. Related Work

The authors in [1] present an agent-based approach to find mappings of active and passive tasks at run-time to achieve a gracefully degrading system behavior. We use a similar platform to conduct our experiments but consider the mapping approach as given. Our work could be used complementary to evaluate different mappings at run-time or at design-time in order to find a mapping where the checkpointing period can be maximized.

The authors in [2] propose a predictable task migration mechanism by implementing a migration timing analysis and a feasibility check for real-time applications. Here, the goal is to enable a dynamic resource management to adapt the mapping of tasks at run-time. In our paper, we do not migrate the tasks to optimize the mapping, but periodically send a checkpoint containing state information.

The work of [3] proposes a lightweight architecture in order to avoid an overly redundant setup and addresses the challenge of a verifiable, fail-safe safety implementation for trajectory planning. In contrast to their example with low-level prediction models applied to a real-world situation, we show an EKF-SLAM algorithm integrated in our platform.

The authors in [4] investigate the approach of dynamical reconfiguration and propose a hardware extension integrated in the architecture to prevent the system from loss of state when communicating with peripherals. However, this approach does not cover restarting entire backup tasks on other ECUs. Furthermore, we assume that the applications tolerate a deviation in state data age such that the checkpointing period can be varied and optimized.

In [5], a solution to minimize checkpoint overhead is presented. This can be reached by tailoring the checkpoint interval on the failure probability instead of taking a checkpoint periodically. The authors in [6] propose a solution to reduce lost messages due to server failures by reducing the cost of checkpointing, cost of rollback and total time cost of overheads due to the fixed checkpointing intervals. However, none of the two methods performs the checkpoint interval optimization based on a worst-case analysis of the recovery time. By contrast, our method ensures that a maximum given data age of the checkpoint data is never exceeded and optimizes the checkpoint accordingly. Violating this data age constraint would lead to an unsafe behavior of the system.

The work of [7] presents an approach of the synthesis of fault-tolerant hard real-time systems for safety-critical applications based on checkpointing with rollback recovery and active replication. However, the approach does not consider alternating the checkpoint period such that our method could be used complementary.

The authors in [8] present a formal analysis for deriving the worst-case failover time of distributed applications. We use parts of this analysis for deriving the missed number of computational steps to calculate the optimal checkpointing period.

In summary, although there is literature that has considered alternating the checkpointing period, none has used a worst-case analysis of the failure and recovery time to obtain an optimal checkpointing period. Thus, existing approaches are unsuitable to ensure a fail-operational behavior within safe bounds when optimizing the checkpointing period.

III. System Model

A. Application Model

Our system architecture consists of a set of ECUs $e \in E$ which are interconnected via switches and Ethernet links $l \in L$. Our system software consists of a set of safety-critical applications $a \in A$, where each application a can be modeled by an acyclic, directed, bipartite application graph. Each application a consists of a set of tasks $t \in T$ and a set of messages $m \in M$. For our analysis we assume that each node has at maximum one predecessor and one successor such that the application graph builds a task chain. Tasks are communicating using a service-oriented middleware with a publish/subscribe pattern, which allows a dynamic adaptation of the system e.g. in the event of ECU or task failures.

We assume that a valid binding is given for both the active and passive redundant task instances. Furthermore, we assume that a routing is given which assigns each message $m \in M$ to a set of links. As the routing will also change after a failover, up to three passive routes are required of which one will become activated depending on which tasks are affected by the failover. Figure 1 depicts the application and system model with an exemplary application consisting of a task chain with two tasks.

We use the notation $\mathcal{L}_i(a)$ to define the end-to-end application latency of a single iteration i of the application a. Using composable task and communication scheduling, the interference between tasks and messages can be bounded, such that a worst-case $\mathcal{L}_{wc}(a)$ and best-case application latency $\mathcal{L}_{bc}(a)$ can be calculated even at run-time [9], [10]. Similar, we define $\mathcal{L}_{bc}(t)$ and $\mathcal{L}_{wc}(t)$ as the best-case and worst-case latency from the application start until task t has finished execution. We assume that the application is periodically executed with the period P_a.

B. Failover Model

We define a failure $f \in F$ with $F \subseteq E$, where f identifies the failed ECU. A failover is required once an ECU e fails to which at least one active task instance of a safety-critical application a has a binding. In a failover scenario we assume that affected task instances are lost and that tasks are restarted using the passive task instances. Similarly, one of the passive routing paths between the new active task instances has to be activated.

IV. Checkpoint Optimization

Since distributed systems are susceptible to failure, techniques to add reliability and high availability were developed. The usage of checkpoints is a technique to ensure

the fail-operational behavior of safety-critical applications in autonomous vehicles. By sending checkpoints over the network in distributed systems, states of the process can be saved during the failure-free execution. These checkpoints can be used after a failure and reloaded on an other ECU to continue the execution of safety-critical tasks. Restarting the computation of tasks from an older saved state is called rollback recovery. A system recovers correctly when its internal state is consistent with its observable behavior before the failure [11]. In the following we use the notation $P_c(t)$ to describe the checkpointing period of a task t that is periodically transferring its state from an active task instance to a passive task instance residing on another ECU.

The proper checkpoint period is a trade off between network and processing overload and fail-safe recovery. Generating a checkpoint after every computational step ensures a higher probability of being able to recover from a failure properly. On the other hand, this would not only cause an overload of the network by forwarding the checkpoints to the passive tasks scheduled on the associated ECUs, but also occupies processing resources. Therefore, increasing the checkpointing period is desirable to minimize both network and processing overhead. In order to obtain the maximum achievable checkpointing period $P_{c,max}(t)$ we are first introducing a formal definition of the checkpointing period and the achievable overhead reduction in Subsection IV-A. Afterwards, we introduce the data age in Subsection IV-B. By analyzing components that influence the data age we derive an analytical bound for the worst-case data age in Subsection IV-C. Based on this upper bound we present the formula to derive the maximum checkpointing period $P_{c,max}(t)$ in Subsection IV-D.

A. Checkpointing Period

We define the default checkpointing period as $P_{c,d} = P_a$ in which case a checkpoint is taken after every single computation and sent to a passive backup task. Furthermore, we define that the checkpointing period P_c can only be a multiple $n \in \mathbb{N}$ of the default checkpointing period $P_c = n \cdot P_{c,d}$, which corresponds to a checkpoint taken after each n-th computation. Increasing the checkpointing period as much as possible is desired as this would minimize the computational and networking effort spent on taking and sending the checkpoint to the passive backup task. In general, increasing the checkpointing period by the factor n results in an overhead reduction of $\frac{n-1}{n}$ as the checkpoint is only sent every n-th application period P_a. As an example, when increasing the default checkpointing period from $P_c = P_{c,d}$ to $P_c = 5 \cdot P_{c,d}$, a reduction of 80% of the networking and processing overhead caused by checkpointing can be achieved.

B. Data Age

However, the checkpointing period can not be increased indefinitely, otherwise no checkpoint would be required. If a failure occurs in the system and a passive backup task starts computing with the checkpoint data, the state data has already aged. We define the data age $d(t) \in \mathbb{N}$ as the number of computational steps that the data is behind when computing the next output. Under normal operating conditions a task would compute the next output using its internal state data, which is getting updated after every execution such that $d(t) = 1$. Using a checkpoint instead, the data is in the worst case behind by n computational steps, resulting in a data age of $d(t) = n$.

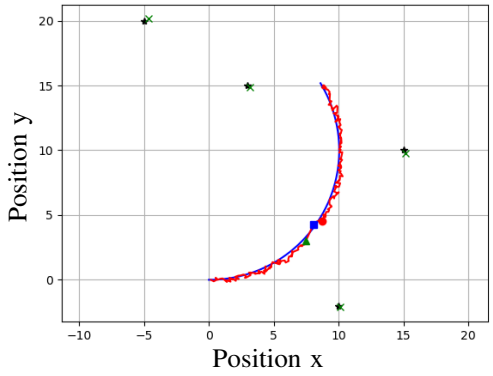
C. Worst-Case Data Age

The state data age $d(t)$ can not only increase with an increasing checkpointing period, but also due to failure and recovery time effects. After the failure of an active task, the passive backup task can not immediately start the next computation. The failure detection takes time and additionally the task might have to perform recovery steps until it is able to continue computing as usual. In our system setup we use heartbeats combined with timeouts to detect failures. Here, we assume that the worst-case failure detection time τ_d is equal for every ECU failure. For the remaining recovery the tasks might have to re-publish their data and re-subscribe to preceding tasks in the application tree using the service-oriented middleware. An upper bound for the detection or subscription time can be found by using a composable system, which is well-described in related work [9], [10]. In the following we refer to $\tau_r(t, f)$ as the worst-case recovery time that it takes from the occurrence of a failure f until a task t is ready to receive and process the next input after recovery, which includes the worst-case failure detection time and publish and subscription times as described in more detail in [8].

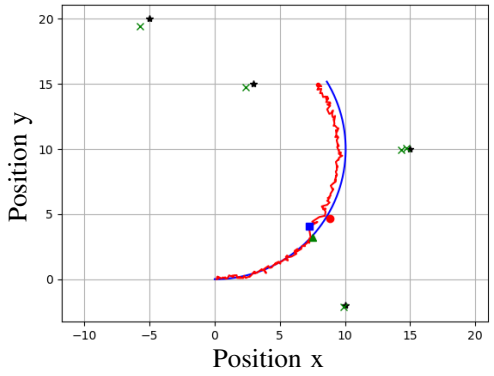
During the downtime of a task represented by the recovery time, it might miss one or multiple periodic inputs. In the worst case the active task instance just finished processing but is not able to send out the checkpoint in time, which causes the data at the passive task instance to be behind by an additional computational step. Afterwards, it might take some time for the passive task instance to recover until it is able to receive and process the next input. During this time it might miss additional inputs such that the data ages even further. Looking at the worst case, there will be always at least one additional computational step missed when the active task instance just finished the execution and was about to send the next checkpoint. The authors in [8] present an analysis to derive the worst-case failover time for distributed applications. The failover time itself does not play a role for designing the checkpointing period. However, the work also includes an upper bound $N_t(a, f)$ for the missed computational steps for a scenario where multiple tasks of an application a are affected at once by the failure f of an ECU, which is calculated as

$$N_t(a, f) = \lfloor \frac{\tau_r(t, f) + \mathcal{L}_{wc}(t_l) - \mathcal{L}_{bc}(p(t_f))}{P_a} \rfloor + 1. \quad (1)$$

This bound considers the worst-case recovery time $\tau(r, f)$, but also the worst-case latencies $\mathcal{L}_{wc}(t_l)$ and best-case latencies $\mathcal{L}_{bc}(p(t_f))$ of latencies of the affected tasks. The formula also directly reflects that in the worst case the loss of at least one iteration is unavoidable. By repurposing this analytical bound



(a) With an ideal recovery, no large impact on the deviation of the estimated trajectory to the real trajectory can be observed.



(b) A faulty re-detection of an already recognized landmark after recovery leads to a high deviation of the estimated trajectory to the real trajectory.

Figure 2: Ideal and non-ideal recovery after a failure in a simulation run with the EKF-SLAM algorithm. The blue curve is the real trajectory starting at (0,0), while the red curve is the trajectory estimated by the EKF-SLAM algorithm. Black stars represent real obstacle positions while green crosses represent the estimated object position. The green triangle corresponds to the last checkpoint taken before the failure, the red circle to the failure time and the blue square to the first computation after recovery using the checkpoint as the last recorded position. As the first computation after recovery has a larger deviation, the algorithm jumps a bit backwards.

we can calculate the worst-case data age of a task after a failover as follows

$$d_{wc}(t) = \frac{P_c}{P_{c,d}} + N_t(a, f) = n + N_t(a, f). \tag{2}$$

We are using this formula to evaluate the effect of the checkpointing period and the application period on the data age of a SLAM applicaton in Subsection V-C. We can observe that the factor n linearly influences the worst-case data age, while $N_t(a, f)$ acts as an offset for a given system architecture and given task mappings.

D. Maximum Checkpointing Period

The question that arises and which plays a major role in finding an optimal checkpointing period is which worst-case data age $d_{wc}(t)$ can be tolerated by a task t. In the following we refer to $d_{max}(t)$ as the maximum data age that a task t can tolerate. This maximum data age is highly application-dependent and has to be found individually for each application and task as it directly influences the functionality of the underlying algorithms. We will derive such an exemplary data age constraint $d_{max}(t)$ in our case study using application-specific metrics in Subsection V-B.

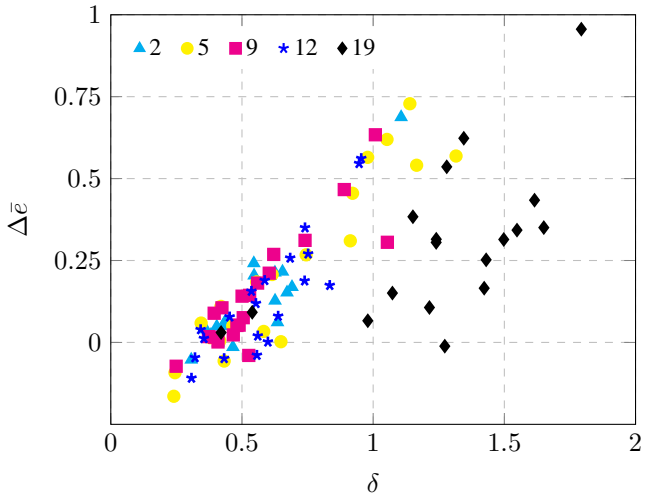
Using the number of computational steps $N_t(a, f)$ that will be missed in the worst-case due to the failure and recovery effects together with a data age constraint $d_{max}(t)$ we can obtain the maximum achievable checkpointing period $P_{c,max}(t)$, which is the goal of this paper, as

$$P_{c,max}(t) = (d_{max}(t) - N_t(a, f)) \cdot P_{c,d}. \tag{3}$$

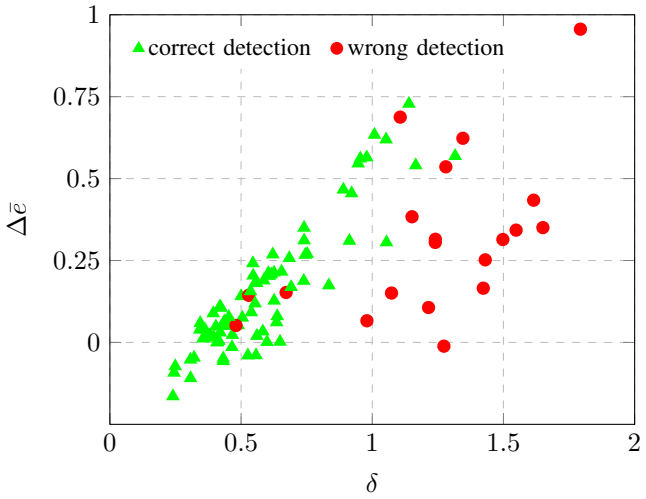
This period minimizes the network and processing overhead caused by the checkpoint messages as it exhausts the available data age limit $d_{max}(t)$ under consideration of the failure and recovery time effects. It can be observed that an increase of the checkpointing period is only possible if the tasks can tolerate the missed computational steps due to failure and recovery time effects. In case $d_{max}(t) < N_t(a, f)$, the system setup would already violate the data age constraint with the default checkpointing period. Therefore, a fast failure detection and recovery is a critical aspect when optimizing the checkpointing period. We are using Equation 3 to calculate the maximum achievable checkpointing period on our demonstrator setup in Subsection V-C for the data age constraint $d_{max}(t)$ obtained in Subsection V-B.

V. Case Study: SLAM Application

To evaluate the applicability of our checkpoint optimization approach introduced in Section IV, we present a case study using a SLAM application as a representative real-world application from the autonomous systems domain. Determining a suitable upper bound for the data age is always application-dependent and should also consider the impact on the safety of the application e.g. by considering application-specific metrics. We first introduce the SLAM application in Subsection V-A and present the effect of a successful and a faulty recovery. Afterwards, we examine the dependency between state data age and accuracy by defining our metrics in Subsection V-B. Here, using a quality analysis we define an exemplary upper bound for the data age $d_{max}(t)$ that should not be exceeded. Furthermore, we perform failover experiments and calculate the corresponding worst-case data age $d_{wc}(t)$ as an upper bound in



(a) Simulation results for the data ages 2, 5, 9, 13 and 19 showing the average deviation $\Delta\bar{e}$ and the worst-case deviation δ.



(b) Red marks correspond to a faulty landmark detection, while green marks indicate that all landmarks have been detected correctly.

Figure 3: Simulated failover results of the EKF-SLAM application with a discrete time step $\tau_{step} = 2.0s$ and varying data ages. With a few exceptions, the data points of data ages 2, 5, 9 and 12 lie within the same cluster, while the data points of data age 19 are clearly separated with a direct negative effect on the average deviation $\Delta\bar{e}$ and the worst-case deviation δ. Furthermore, the probability for a wrong landmark detection is drastically increased.

Subsection V-C. Using $d_{max}(t)$ as a constraint we analytically derive the maximum achievable checkpointing period $P_{c,max}(t)$ on our demonstrator setup.

A. SLAM Application

Simultaneous Mapping and Localization (SLAM) is the process where a moving object creates a global map of its current environment and uses this map to navigate and deduce its position and orientation at any point in time. A SLAM algorithm consists of a prediction step, which is based on a motion model and an update step, which is based on an observation model. A learning process between the states and measurements ensure an accurate representation. The most common learning method for SLAM is the Kalman Filter, which is based on the assumption of a uni-modal distribution. We use a simulation of an Extended Kalman Filter (EKF) SLAM application which can be found in [12].

Figure 2a presents simulation results, which compares the curve the algorithm is trying to follow (blue) with the curve calculated by the EKF-SLAM approach (red). The simulation advances with discrete time steps τ_{step}. The plot also shows a successful recovery where the green triangle represents the last checkpoint taken before a failure. The red data point represents the time step at which the failure occurs operation and the blue square represents the first computation after the recovery using the old checkpoint data. It can be observed that this first data point after the recovery estimates the position further behind than the actual position is. However, with further operations the red curve quickly swings back into normal operation.

Figure 2b presents an example with a faulty recovery as a landmark is re-detected after recovery. The algorithm uses a threshold for distinguishing points in the environment. During a recovery from a failure it may occur that already detected landmarks are lying outside of this threshold radius which leads to a false re-detection as a new landmark. This leads clearly visible to a permanent deviation from the actual curve as the algorithm believes that at position (15, 30) are two obstacles instead of one . This occurs only in the case where the recovered position is far away from the actual position, which leads to a temporal difference between the position of the car derived from the landmarks and the old saved position.

B. Quality Analysis

To measure the influence of the data age on the quality of the algorithm we introduce two metrics. The error e_i reflects the euclidean distance of the computed points of the two curves in the computational step i. The average of these errors can be computed before and after a simulated failure. To define the metrics we use the following definitions:

$$\bar{e}^{(bf)} = \frac{1}{n}\sum_{i=1}^{n} e_i^{(bf)}, \tag{4}$$

$$\bar{e}^{(af)} = \frac{1}{n}\sum_{i=1}^{n} e_i^{(af)}, \tag{5}$$

$$e_{wc}^{(af)} = \max_{\forall i \in [1\ldots n]} e_i^{(af)}, \tag{6}$$

with $\bar{e}^{(bf)}$ being the average error before the failure, $\bar{e}^{(af)}$ being the average error after the failure and $e_{wc}^{(af)}$ being the worst-case error after the failure. We define $\Delta\bar{e}$ as the difference of the average of all calculated points before $\bar{e}^{(bf)}$ and after $\bar{e}^{(af)}$ a simulated failure occurred:

$$\Delta\bar{e} = \bar{e}^{(af)} - \bar{e}^{(bf)}. \tag{7}$$

We define δ as the worst-case deviation $e_{wc}^{(af)}$ compared to the average $\bar{e}^{(bf)}$ as follows

$$\delta = e_{wc}^{(af)} - \bar{e}^{(bf)}. \tag{8}$$

The data shown in Figure 3 is the result of a simulation with a discrete time step of $\tau_{step} = 2.0s$ and fixed landmark positions. To evaluate the effect of the data age we performed experiments with the data ages 2, 5, 9, 12 or 19. Smaller values mean that the checkpoint has been taken more recently to the occurrence of the failure such that at the point of recovery a more recent data point can be used for calculating the position. Some outliers of the data age 19 could not be shown in the plot. Figure 3b marks the simulation runs where a recovery could be performed successfully and the simulation runs where a landmark was detected wrongly. It can be observed that an older checkpoint leads to bigger average errors but especially to bigger worst-case errors. Considering the influence of the data age on both the average deviation $\Delta\bar{e}$ and the worst-case deviation δ as well as on the number of wrong detections, we define the data age $d_{max}(t) = 12$ as an exemplary upper bound with acceptable deviations. We use the presented EKF-SLAM application in the following subsection to further perform failover experiments in order to show the applicability of our analysis from Section IV. Furthermore, we use the upper bound $d_{max}(t) = 12$ as an example to derive a maximum achievable checkpointing period.

C. Failover Experiments

For our failover experiments we use a setup consisting of 3 Raspberry-Pi 4B devices, with 8GB RAM and a quad-core Cortex-A72 with maximum frequency of 1.5 GHz. We use a star-topology network using a switch and Ethernet links for the communication between the devices. Our framework running on each Raspberry Pi uses a service-oriented communication middleware with a publish/subscribe pattern and allows to simulate ECU failures by shutting down the framework instances. Failures are detected via heartbeats and timeouts. On top we use an agent-based platform, characterized by automatic mapping and checkpoint subscription. In addition to the active tasks, safety-critical tasks have a passive redundancy on another ECU. Passive tasks subscribe to the checkpoints of the active tasks. In the event of an active task failure, the passive task is restarted using the last successfully transmitted checkpoint. We divided the complete application into four tasks:

- t_1: Simulation task
- t_2: Pre-processing task
- t_3: Observation task
- t_4: EKF-SLAM algorithm task

For the failover measurements presented in Figure 4 the tasks t_1, t_2 and t_3 are active on e_1 and t_4 is active on e_2, while all passive tasks are mapped to e_3. In the experiments we shut down e_2 at random points in time and measured the data age $d(t_4)$ after the recovery.

Figure 4a presents the data age obtained by our experiments carried out with an application period of $P_a = 2s$ and a varying checkpoint period $P_c(t_4)$. For each variable set-up of $P_c(t_4)$ we carried out ten measurements. The red curve corresponds to the upper bound $d_{wc}(t_4)$ obtained through Equation 2 for a given checkpointing period $P_c(t_4)$. For the experiments depicted in Figure 4b, a fixed checkpoint period of $P_c = 5 \cdot P_{c,d}$ was selected, while the application period P_a was varied.

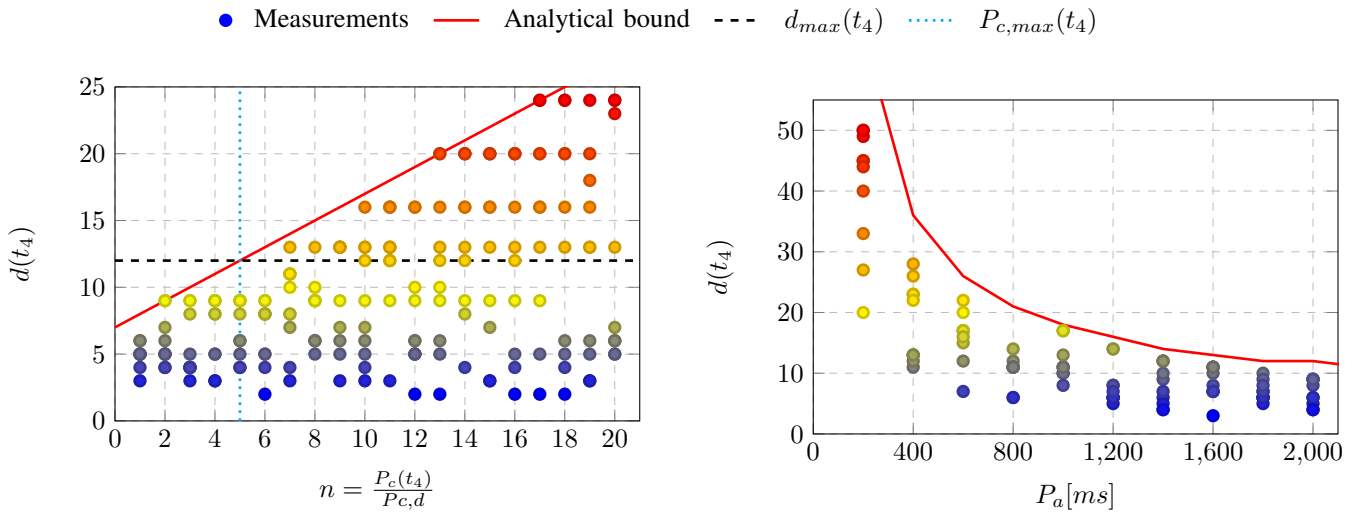
Most importantly, it can be observed that none of the measurements exceed the upper bound obtained through Equation 2. As some of the worst-case measurements lie directly on the bound, the experiments are consistent with our analytically derived exact bound. In Figure 4a the worst-case data age increases linearly with the checkpointing period $P_c(t_4)$, while the worst-case number of lost iterations $N_t(a, f)$ adds a constant offset. In Figure 4b the worst-case data age decreases exponentially with an increasing application period P_a and an offset of the checkpointing period $P_c(t_4) = 5 \cdot \cdot P_{c,d}$. Once P_a doubles, about half of the iterations will be lost as the worst-case recovery time stays the same. Note that we define the data age in iterations and not in absolute time, which could be obtained by multiplying the data age $d(t_4)$ with the application period P_a.

Following the quality analysis from Subsection V-B we use an exemplary data age constraint $d_{max}(t_4) = 12$, which is depicted as a black dashed line in Figure 4a. The resulting maximum achievable checkpointing period of $P_{c,max} = 5 \cdot P_{c,d}$ is marked with a green dotted line. By increasing the default checkpointing period from $P_c(t_4) = 1 \cdot P_{c,d}$ to $P_{c,max}(t_4) = 5 \cdot P_{c,d}$, the networking and processing overhead caused by checkpointing can already be reduced by 80% in this example.

In summary, our case study and experiments have confirmed the applicability of our approach. We have shown the application-specific dependency between state data age and the accuracy of an EKF-SLAM application. Using a data age constraint obtained by the quality analysis, we analytically derived the maximum achievable checkpointing period. By performing additional failover experiments we have confirmed that worst-case results of our randomized experiments are consistent with our analytically derived exact bound. To apply our approach to other applications, a safe upper bound for the data age has to be found by using application-specific metrics and safety considerations. This maximum data age constraint can then be used for our application-independent analysis to find a suitable maximum achievable checkpointing period. Overall, when designing the checkpointing period a trade-off between the impact on algorithmic behavior (in our example on the average deviation $\Delta\bar{e}$ and the worst-case deviation δ) and the overhead reduction has to be made.

VI. Conclusion

In this paper we have presented an analysis to calculate a maximum achievable checkpointing period for distributed fail-operational automotive applications. Here, we analyzed the effects of the recovery time on the data age and determined a worst-case number of computational steps that will be missed during the downtime in order to obtain the maximum possible checkpointing period. Furthermore, we presented a detailed case study of a SLAM application as a representative real-world



(a) Results with an application period $P_a = 2s$ and a varying checkpointing period $P_c(t_4)$. For a given data age constraint the maximum checkpointing period can be obtained graphically by finding the intersection point with the red curve. The black dashed line marks the data age constraint $d_{max}(t_4)$ obtained through the quality analysis in Subsection V-B. The cyan dotted line marks the corresponding maximum checkpointing period $P_{c,max}(t_4)$ obtained through Equation 3.

(b) Results for a fixed checkpointing period of $P_c(t_4) = 5 \cdot P_{c,d}$ and a varying application period P_a.

Figure 4: Results of the random failover experiments presenting the data age $d(t_4)$ at the backup task instance of t_4 after successful recovery with $\tau_r(t, f) = 11.82s$, $\mathcal{L}_{bc}(p(t_4)) = 0.0548s$ and $\mathcal{L}_{wc}(t_4) = 0.238s$. None of the measurements exceed the upper bound obtained through Equation 2, here marked as the red curve. Worst-case results of our experiments are consistent with our analytically derived exact bound as some measurements lie directly on the curve.

application from the automotive domain. Here, we analyzed the dependency between state data age and accuracy and derived an exemplary state data age constraint. In addition, we conducted failover experiments on our demonstrator setup using an agent-based software platform to show the applicability of our worst-case analysis. Worst-case results of our randomized experiments are consistent with our analytically derived exact bound. In summary, our approach can be used to reduce network and processing overhead caused by sending checkpoints in order to achieve a cost-efficient and safe behavior of stateful distributed fail-operational applications.

References

[1] P. Weiss, A. Weichslgartner, F. Reimann, and S. Steinhorst, "Fail-operational automotive software design using agent-based graceful degradation," in *Proceedings of the Conference on Design, Automation and Test in Europe (DATE)*, 2020.

[2] B. Pourmohseni, F. Smirnov, S. Wildermann, and J. Teich, "Real-Time Task Migration for Dynamic Resource Management in Many-Core Systems," in *Workshop on Next Generation Real-Time Embedded Systems (NG-RES 2020)*, ser. OpenAccess Series in Informatics (OASIcs), vol. 77. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2020, pp. 5:1–5:14.

[3] S. v. Dorff, B. Boddeker, M. Kneissl, and M. Franzle, "A fail-safe architecture for automated driving," in *2020 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, pp. 828–833. [Online]. Available: https://ieeexplore.ieee.org/document/9116283/

[4] F. Oszwald, P. Obergfell, M. Traub, and J. Becker, "Reliable fail-operational automotive e/e-architectures by dynamic redundancy and reconfiguration," in *2019 32nd IEEE International System-on-Chip Conference (SOCC)*. IEEE, pp. 203–208. [Online]. Available: https://ieeexplore.ieee.org/document/9088090/

[5] S. Muhammad Abrar Akber, H. Chen, Y. Wang, and H. Jin, "Minimizing overheads of checkpoints in distributed stream processing systems," in *2018 IEEE 7th International Conference on Cloud Networking (CloudNet)*. IEEE, pp. 1–4. [Online]. Available: https://ieeexplore.ieee.org/document/8549548/

[6] T. Aung, H. Y. Min, and A. H. Maw, "Coordinate checkpoint mechanism on real-time messaging system in kafka pipeline architecture," in *2019 International Conference on Advanced Information Technologies (ICAIT)*. IEEE, pp. 37–42. [Online]. Available: https://ieeexplore.ieee.org/document/8921392/

[7] P. Pop, V. Izosimov, P. Eles, and Zebo Peng, "Design optimization of time- and cost-constrained fault-tolerant embedded systems with checkpointing and replication," vol. 17, no. 3, pp. 389–402. [Online]. Available: http://ieeexplore.ieee.org/document/4757196/

[8] P. Weiss, S. Elsabbahy, A. Weichslgartner, and S. Steinhorst, "Worst-case failover timing analysis of distributed fail-operational automotive applications," in *Proceedings of the Conference on Design, Automationand Test in Europe (DATE)*, 2021. [Online]. Available: https://tum-esi.github.io/publications-list/PDF/2021-DATE-Worst-Case%20Failover%20Timing%20Analysis%20of%20Distributed%20Fail-Operational%20Automotive%20Applications.pdf

[9] B. Akesson, A. Molnos, A. Hansson, J. A. Angelo, and K. Goossens, *Composability and Predictability for Independent Application Development,Verification, and Execution*. New York, NY: Springer New York, 2011, pp. 25–56.

[10] A. Weichslgartner, S. Wildermann, D. Gangadharan, M. Glaß, and J. Teich, "A design-time/run-time application mapping methodology for predictable execution time in mpsocs," *ACM Trans. Embed. Comput. Syst.*, vol. 17, no. 5, 2018.

[11] A. D. Kshemkalyani and M. Singhal, "Distributed computing: Principles, algorithms, and systems," pp. 456–507, 2008.

[12] AtsushiSakai, "Ekf-slam," https://github.com/AtsushiSakai/PythonRobotics/blob/master/Localization/extended_kalman_filter/extended_kalman_filter_localization.ipynb.