Formal Parameter Synthesis for Energy-Utility-Optimal Fault Tolerance

Linda Herrmann, Christel Baier, Christof Fetzer,
Sascha Klüppelholz, and Markus Napierkowski *

Technische Universität Dresden
{linda.herrmann1,christel.baier,christof.fetzer,
sascha.klueppelholz,markus.napierkowski}@tu-dresden.de

Abstract. Fault-tolerance techniques are widely used to improve the resiliency of hardware/software systems. An important step for the deployment of such techniques in a concrete setting is to find reasonable configurations balancing the tradeoff between resiliency and energy. The paper reports on a case study where we employ probabilistic model checking to synthesize values for tunable system parameters of a redo-based fault-tolerance mechanism. We consider discrete parameters of a finite range (as the number of redos) as well as continuous parameters to encode the error detection rates of the underlying control- and data-flow checkers. To tackle the state-explosion problem, we exploit structural properties of redo-based protocols. The parameter synthesis approach combines probabilistic model checking for Markov chains with parametric transition probabilities and reward values and computer-algebra techniques to determine parameter valuations that minimize the expected overhead given constraints on the utility, depending on a given error probability.

1 Introduction

The paper reports on a case study that addresses the synthesis problem for redo-based fault-tolerance mechanisms. We assume environmental parameters that are part of the input (e.g., error probabilities and energy costs) and configurable parameters (e.g., detection probabilities and the number of redos). The latter are controllable and hence part of the output. The goal is to search for configurations (i.e., values for the tunable parameters) that allow balancing the tradeoff between resiliency, energy and performance.

The considered fault-tolerance mechanism is inspired by the hardware-assisted fault-tolerance protocol HAFT [1]. HAFT enables error[1] detection and correction for a finite sequence of instructions (in the further named application), that

* The authors are supported by the DFG through the Collaborative Research Center SFB 912 – HAEC, the Excellence Initiative by the German Federal and State Governments (cluster of excellence cfAED and Institutional Strategy), the Research Training Groups QuantLA (GRK 1763) and RoSI (GRK 1907), the DFG-project BA-1679/11-1, and the DFG-project BA-1679/12-1.

[1] Following the taxonomy of [2], we use the term "fault" to describe bit-flips. Errors are caused by faults that affected the applications run. Failures in our sense are

is affected by bit-flips. The operating principle of HAFT at the relevant abstraction level is as follows: The fault-tolerance technique partitions the application into transactions, and performs error detection and correction transaction-wise. Error detection is enabled by replicating instructions, and comparing for differences. Replicated instructions can also be affected by errors, and thus might cause error correction to be invoked although the application would have been correct when not using error detection. An erroneous transaction is corrected by redoing the transaction. Also during a redo errors can occur and further redos might be necessary. If after a pre-defined number of redos still an error is detected, the application is aborted. Error detection and correction cause overhead costs. As a measure, we take the number of instructions executed for error handling. This measure correlates with the energy consumption for error detection and correction, since the additional energy consumption only comes from additional instruction execution. In this paper we build on Markov chains with parameters for transition probabilities and rewards as underlying semantic model. The discrete protocol parameters, e.g., the number of redos and transaction length, induce a family of parametric Markov chains that yield the basis for finding reasonable configurations. Finding optimal solutions in uncountably infinite parameter sets is in general undecidable [3], [4]. We restrict ourselves to families with finitely many members, as the set of reasonable transaction lengths as well as the number of redos can safely be assumed to be finite as well.
Our primary configuration objective is resilience: the probability of terminating without an undetected error shall be very high. Aborting the application is preferred to terminating with a wrong result, thus, the conditional probability of aborting in case of not terminating correctly shall be high. As a subordinate objective, the energy consumption has to be low. This opens the following synthesis problems: 1) What is a good transaction length? Long transactions increase the probability of a transaction to be erroneous and increase the overhead in redos, while short transactions cause error detection to be performed very often and thus increase the overall costs for error detection. 2) What is an optimal number of maximal redos? Redos enable error correction and thus application recovering, but during a redo there is an additional chance of having undetected errors and causing the application to terminate with a wrong result. 3) How many instructions shall be replicated? Increasing this amount increases the chance of detecting an error, but also increases the overhead and the chance of some replica being affected by an error.

Challenges. To address this synthesis problem, we apply variants of probabilistic model checking [5], [6] that take as input parametric Markov chains as described above. These variants yield rational functions for probabilities of reaching some goal state and expected accumulated rewards. We will refer to this variants as parametric probabilistic model checking (PPMC). PPMC yields rational functions, describing system characteristics, instead of single values. The rational functions can then be analyzed for optimality. This comes with several

caused by errors that could not be detected by the fault-tolerance mechanism and thus lead to a wrong computation of the application.

challenges. 1) An application typically consists of billions of instructions, realistic transaction lengths range from several hundred instructions to the extremal case of handling the whole application as one transaction. We need to include both, the details of a transaction, and the execution of millions of transactions, in the model. This results in very large models, which is in contrast to PPMC requiring models to be small. 2) Only the detection probabilities but not the maximal number of redos and the transaction length can be handled as parameters. For the latter, PPMC needs to be invoked once for each considered configuration, which is especially problematic due to the large model sizes. 3) The error probability crucially depends on the hardware and the application scenario. Moreover, tiny error probabilities[2] can lead to numerical instability when treated in a non-parametric fashion. Thus, environmental and tunable parameters are part of the model, but PPMC hardly scales for larger models with multiple parameters [5].

Contribution. In this paper, we report on a case study for the synthesis problem for fault-tolerance mechanisms and show how to overcome the mentioned challenges. The main idea is to exploit the regularity of our model with repeating transaction blocks. Hence, we apply PPMC to families of very small sub-models, modeling only one transaction, but detailed and still parametric in the full set of probability and reward parameters. With this we obtain rational functions for probabilities of successful error detection and aborting as well as energy overhead for each transaction. We present a new, suited factorization approach that allows to combine the transaction-level results and obtain rational functions for the whole application. Our approach allows handling the transaction length as a parameter and PPMC has to be invoked only once for each considered number of maximal redos. Finally, we exemplarily utilize the rational functions to find sweet spots in the tradeoff between resiliency and energy. Here, a surprising result is, that for reasonable error probabilities, increasing the maximal number of redos to more than just one, degrades the resiliency while only introducing additional overhead costs.

2 Related Work

Parameter synthesis for probabilistic systems using formal methods has been done widely before. In, e.g., [10] parametric model checking is applied to synthesize optimal values for transition probabilities in Markov models, as we do for the amount of instructions to be replicated. Rate parameters are synthesized in [11] in a CTMC modelling stochastic biochemical networks, and in [12] for a real-time storage system that is affected by randomly occurring bit-flips. In [13], a parameter synthesis approach is presented and applied to repair systems by tuning transition probabilities. Some instances of adaptive systems with configurable transition probabilities are configured in [14], using stochastic methods like Monte Carlo Sampling and particle swarm optimization. None of these works

[2] [7] gives an error rate of 0.066 FIT (failures in time, the expected number of failures in 10^9 hours) per Mbit. This corresponds to having an error in a single instruction within an hour with probability about $4 \cdot 10^{-15}$. Other error rate estimates from [8] and [9] give error probabilities of $3.7 \cdot 10^{-15}$ and $2.7 \cdot 10^{-15}$.

addresses the parameter synthesis problem for redo-based fault-tolerance mechanisms and exploits the regularity of the model structure to address the imposed scalability challenges. Furthermore, our approach also spans parameters that affect the structure of the Markov model and thus cannot be handled easily with standard parametric model-checking techniques.

3 Preliminaries

We will provide a brief summary of the relevant concepts for Markov chains. For more details, we refer to, e.g., [15]. A *discrete-time Markov chain* (DTMC) is a tuple $\mathcal{M} = (S, P)$ with a finite set of states S and transition probabilities $P : S \times S \to [0,1] \cap \mathbb{Q}$ such that for all $s \in S : \sum_{t \in S} P(s,t) \in \{0,1\}$. A *path* in $\mathcal{M}$ is a finite or infinite sequence of states $\pi = s_0 s_1 \ldots$ such that $P(s_i, s_{i+1}) > 0$ for all $i \geq 0$. We denote the set of all paths in $\mathcal{M}$ by $Paths_{\mathcal{M}}$. For a finite path $\pi = s_0 s_1 \ldots s_n$ we write $P(\pi) = \prod_{i=0}^{n-1} P(s_i, s_{i+1})$. A path π is maximal if it is infinite or π is finite and $\sum_{t \in S} P(s_n, t) = 0$.

Let $G \subseteq S$ and $s_0 \in S$. The probability of eventually reaching some state in G from s_0, denoted by $\Pr_{s_0}(\Diamond G)$, is derived using the standard definition of the induced probability distribution on the set of measurable sets of maximal paths. The event $\Diamond G$ is called a *reachability property*.

A *reward function* $rew : S \to \mathbb{Q}^{\geq 0}$, assigns each state a non-negative value (e.g., energy consumed in that state). The accumulated reward induced by *rew* is a random variable $AccRew : Paths_{\mathcal{M}} \to \mathbb{R}$ assigning each path in $\mathcal{M}$ the sum of its state rewards, i.e. $AccRew(s_0 s_1 \ldots s_n) = \sum_{i=0}^{n} rew(s_i)$, Let $G \subseteq S$ be a set of states such that $\Pr_{s_0}(\Diamond G) = 1$. The *expected accumulated reward* until reaching G from s_0, denoted by $\mathbb{E}_{s_0}(\Diamond\!\!\!\!-\, G)$, is the expectation value of $AccRew$ in $\mathcal{M}$ restricted to the set of paths ending in G, i.e., $\mathbb{E}_{s_0}(\Diamond\!\!\!\!-\, G) = \sum_{\pi \in \Pi} P(\pi) \cdot AccRew(\pi)$, where $\Pi = \{s_0 \ldots s_n \in Paths_{\mathcal{M}} \mid s_n \in G, s_i \notin G \text{ for } 0 \leq i < n\}$.

For $G \subseteq S$ and $\Pr_{s_0}(\Diamond G) > 0$ the *conditional expected accumulated reward* until reaching G from s_0 under the condition of reaching G is, with Π as above:

$$\mathbb{E}_{s_0}(\Diamond\!\!\!\!-\, G \mid \Diamond G) = \sum_{\pi \in \Pi} \frac{P(\pi) \cdot AccRew(\pi)}{\Pr_{s_0}(\Diamond G)}.$$

4 Redo-Based Fault-Tolerance Model

In this section, we introduce the fault-tolerance model. A detailed description can be found in the Appendix A.1. The model contains adjustable *attributes* for, e.g., the error probability. It consists of components for the underlying hardware, the application, and the fault-tolerance protocol. The latter contains a control-flow checker (CFC), a data-flow checker (DFC), and a transaction redo manager (TRM) implementing the redo/abort-schema.

Application. An application performs a fixed number of instructions *inst_num*, each instruction is prone to errors (see paragraph "Errors and Failures" below). The instruction flow is partitioned into transactions, each consisting of *ta_len* instructions. The number of transactions to be performed is *ta_num* = *inst_num*/*ta_len*. We assume three types of instructions: *control-flow instruc-*

tions, where errors affect only the control-flow (e.g., jump), *data-flow instructions*, where errors affect the data-flow (e.g. add), and *transaction management instructions*, implementing the transaction mechanism (e.g., begin-of-transaction and end-of-transaction instructions). Errors in the latter do only affect the control flow. Errors in the data-flow also affect and thus falsify the control-flow. The ratio of control-flow and data-flow instructions can be set via the attribute *cf_df_ratio*. The amount of transaction management instructions is controlled by the attribute *tmi_num*. The application starts in a location "start transaction", performs a transaction and then reaches a "wait" location. Eventually, it either receives an ABORT or a COMMIT from the TRM. An ABORT indicates that an error could not be corrected. Then, the application switches to location "abort". Receiving COMMIT causes the application to complete the transaction. If all instructions are executed, it terminates, reaching location "done". Otherwise, it increases a counter *ta_counter* and starts a new transaction.

Error detection. The TRM initially waits for the application to complete a transaction. Then, it invokes both, CFC and DFC in parallel, and waits for the results. The DFC checks all data-flow instructions for errors. The CFC checks all instructions for errors, since all instructions affect the control flow. For each data-flow-corrupted transaction there is a chance *p_detn_DFC* of the DFC to detect this error. For each transaction with correct data-flow, there is a chance of *p_fp_DFC* to detect an error anyway, i.e., to have a false positive. Analogous attributes *p_detn_CFC* and *p_fp_CFC* are defined for the CFC.

Error correction. After checking, the checkers report their results to the TRM, which switches to location "answers received". If one of the checkers detected an error, a redo is invoked by the TRM, i.e., the transaction is re-executed and a redo-counter *redo_counter* is increased. The re-executed transaction can again be corrupted, thus, error detection is invoked and might result in a further redo. This is repeated until the preset attribute *max_redos* is reached by the redo counter. Then, the TRM sends an ABORT signal to the application. If in the original transaction or in one of its re-executions no error is detected, the TRM sends a COMMIT-signal to the application.

Errors and failures. The hardware model tracks the state of the applications internal memory. When starting the application, the hardware location is "correct". Each instruction can be corrupted with probability *p_e*, which causes the location to switch to "error". When being erroneous, and a redo is invoked by the TRM, the location switches back to "correct". When being erroneous and no redo is invoked, the location is changed to "failure". A failure increases the chance of a subsequent error in an instruction to *p_e_incr*. Once the application has a failure in its internal memory, it will persist until the application either terminates (with this failure) or is aborted due to another error.

Transaction outcomes. After performing, checking, correcting, and committing or aborting a transaction, there are four possible outcomes. 1. The hardware state is "correct" and the application received a commit, i.e., is in a location "transaction completed" (short: cc). This outcome is reached if either no error occurred and also no error was detected, or if an error occurred, was detected

and could be corrected. 2. The hardware is "correct" but the application was aborted (short: ca), arising, if no error occurred in the original transaction, but a false positive triggered redos, and redos failed. 3. If an error occurred in the original application but could not be detected, the hardware model ends up in location "failure" and the application receives a commit, completing the transaction (short: fc). 4. If the hardware after some transaction ends in the previous mentioned outcome fc and in some transaction afterwards an abort signal is sent, then the hardware model is in location "failure" and the application is in location abort (short: fa).

5 State-Space Reduction and Factorization

Semantics and structure of the model. The fault-tolerance model is a Markov decision process (MDP) (see, e.g., [16]), i.e., a discrete-time Markov chain (DTMC) enhanced with nondeterminism. In an MDP there can be several distributions per state. The non-determinism is resolved by a *strategy*. Each strategy $\mathfrak{S}$ induces a DTMC and thus, together with a starting state $s_0 \in S$, a probability distribution $\Pr_{s_0}^{\mathfrak{S}}$ on the set of measurable sets of paths in the induced DTMC. The MDP $\mathcal{M}$ for our model serves as operational model whose states are tuples consisting of the local states of all components. The initial state s_0 is the state where the hardware is "correct", the application's location is "start" and all other modules wait for being invoked. All counters (i.e.,*redo_counter*, *ta_counter*) are set to 0 and variables indicating redos to false. For each state, the outgoing transitions arise from all possible synchronous and asynchronous transitions enabled in the components.

Reduction to Markov Chain The nondeterminism in $\mathcal{M}$ solely arises from interleaving execution of the CFC and DFC. None of the individual components of the model contains nondeterministic choices. Since no variables or locations outside CFC and DFC change during execution of CFC and DFC and no transitions in the CFC and DFC make use of the internal state of the respective other checker component, the state of the MDP after error detection does not depend on the chosen interleaving. Furthermore, none of our configuration criteria in Section 6 distinguishes between states in the non-deterministic part of the MDP, i.e., states that model the error detection process. Thus, we can rely on results that have been established in the context of partial-order reduction for MDPs [17]–[19]: for the chosen configuration criteria it is irrelevant which interleaving is chosen and we can replace $\mathcal{M}$ with, e.g., the DTMC in which first the CFC performs error detection and then the DFC checks the data-flow.

Factorization. The model size increases significantly when increasing *ta_num* (i.e., by decreasing the transaction length). Furthermore, even for small models, PPMC performs badly on our model. When choosing $max\_redos = 1$ and $ta\_num = 5$, the model consists of only 876 states, but computation times of simple probabilistic properties are unfeasible large[3]. Realistic applications con-

[3] Computing the probability of correct termination single threaded on a 2.5Ghz Intel Core Processor did not finish within 2 hours, for $ta\_num = 5$. The model was parametrized in the detection probabilities and the error probabilities.

sist of many more transactions, e.g., in Section 6 we will analyze a model with $ta\_num = 10^{10}$.
We will use the structure of the model to simplify the model checking process. For technical reasons, we slightly modify the model: instead of terminating in abort states, we stepwise increase the transaction counter up to its maximum value (without changing other variable values). This gives us a DTMC with a regular structure with repeating phases (Figure 1). One phase represents one

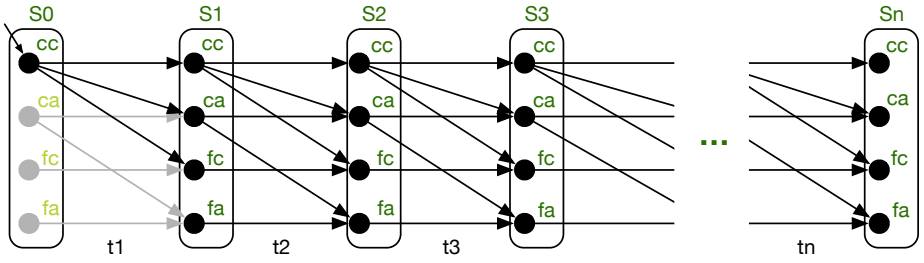

Fig. 1. The structure of the DTMC on transaction level. Each arrow indicates the execution of one transaction, including error detection and correction. The outcome of each transaction is one of the states in {(correct,commit)(cc),(correct,abort)(ca),(failure,commit)(fc),(failure,abort)(fa)}. Light-grey states in S_0 mark states that are not reachable.

transaction and after each phase there are four possible outcomes cc, ca, fc, and fa (cf. Section 4). Internal details of each phase such as program execution, error detection and correction are omitted here.
For the rest of this section we fix $ta\_num = n$. We denote the set of outcome states after the i-th transaction by $S_i = \{(\mathrm{cc}, i), (\mathrm{ca}, i), (\mathrm{fc}, i), (\mathrm{fa}, i)\}$ (cf. Figure 1), and identify the initial state with $s_0 = (\mathrm{cc}, 0) \in S_0$, since the hardware is "correct", the application is not aborted and no transaction is performed yet. Note that for all $0 \leq i < n$ the probabilities of reaching state $t \in S_{i+1}$ from $s \in S_i$ do not depend on the counter value i. Thus, we can choose an arbitrary $0 \leq i < n$ and define a probability matrix $P = (\Pr_s(\Diamond t))_{s \in S_i, t \in S_{i+1}}$. P is a 4×4-matrix that describes the probabilistic effects of a single transaction. Note that the matrix elements are rational functions. The probability of reaching outcome goal $\in S_n$ (thus, after the n-th transaction) can be computed by $\Pr_{s_0}(\Diamond goal) = (P^n)_{s_0, goal}$.

Lemma 1. *Let rew be a reward structure with* $rew(s_n) = 0$ *for all* $s_n \in S_n$, $E = (\mathbb{E}_s(\Diamond\!\!\!\!\cdot\, t | \Diamond t))_{s \in S_i, t \in S_{i+1}}$ *for some arbitrary* $0 \leq i < n$, *and let pe be a* 1×4 *vector with* $pe = \left(\sum_{t \in S_{i+1}} P_{s,t} \cdot E_{s,t}\right)_{s \in S_i}$. *Then we have*

$$\mathbb{E}_{s_0}(\Diamond\!\!\!\!\cdot\, S_n) = \left(\sum_{k=1}^{n} P^{k-1} \cdot pe\right)_{s_0}.$$

The proof relies on Bayesian decomposition for expectation values and can be found in the Appendix A.2. The matrices P and E and the vector pe can be

computed with existing PPMC implementations on very small models (modeling only one transaction). Thus, we can compute rational functions for both, reachability properties and expected accumulated rewards by combining PPMC (on very small models) and a computer algebra system. Furthermore, the structures of the small models do not depend on *ta_num* and *ta_len*, thus *ta_len* can be treated as a parameter.

6 Configuration

In this section, we exemplarily configure an instance of our fault-tolerance model with respect to the configuration parameters *p_detn_CFC*, *p_detn_DFC*, *ta_len* and *max_redos*. During model checking we handle *p_detn_CFC*, *p_detn_DFC* and *ta_len* as parameters, and, using the factorization approach from Section 5, compute rational functions for each $max\_redos \in \{0, 1, 2, 3\}$. We also handle the error probability *p_e* as parameter, and exemplarily configure the fault-tolerance model for all $p\_e \in \{10^{-8}, 10^{-10}, 10^{-15}\}$. The models, rational functions, and additional files can be downloaded[4].

Fault-tolerance setting. The model instance is mainly inspired by HAFT [1], i.e., instructions in our model correspond to instructions on CPU level, and error detection is enabled by duplicating instructions. The amount of duplicated instructions is configurable and represents the detection probability, e.g., replicating 80% of all data-flow instructions means $p\_detn\_DFC = 0.8$. Replicating all instructions gives error detection probabilities of 1, i.e., we neglect the probability of the same error occurring in both, an original instruction and its replication, which would cause the error to be undetectable. Furthermore we assume that transaction management instructions are not replicated.

Model attributes. We fix the following model attributes: The application runs for exactly 10^{12} (*inst_num*) instructions, 10% (*cf_df_ratio*) being control-flow instructions, and two (*tmi_num*) transaction management instructions ("begin of transaction" and "end of transaction") are inserted per transaction. The increased error probability is defined as $p\_e\_incr = (p\_e)^{\frac{8}{10}}$. False positives are errors that occur in replicated instructions or in transaction management instructions, i.e., with $cf\_df\_ratio = 0.1$ we have:

$$p\_fp\_CFC = 1 - \Big((1 - p\_e)^{ta\_len \cdot 0.1 \cdot p\_detn\_CFC} \cdot (1 - p\_e)^{ta\_len \cdot 0.9 \cdot p\_detn\_DFC} \cdot (1 - p\_e)^{tmi\_num}\Big) \quad \text{and}$$

$$p\_fp\_DFC = 1 - \Big((1 - p\_e)^{ta\_len \cdot 0.9 \cdot p\_detn\_DFC}\Big).$$

Reward structures. In the analysis we focus on the expected energy-overhead for error detection and correction. For this we introduce a reward structure that assigns one energy unit each time an instruction is executed for error detection or correction. Formally, we define a reward structure assigning states where the TRM is in location "answers received" the following reward for the energy overhead, depending on whether a transaction or one of its redos was executed:
if *redo_counter* $= 0$: $ta\_len \cdot p\_detn\_CFC + ta\_len \cdot 0.9 \cdot p\_detn\_DFC + tmi\_num$,
if *redo_counter* > 0: $ta\_len \cdot p\_detn\_CFC + ta\_len \cdot 0.9 \cdot p\_detn\_DFC + tmi\_num + ta\_len$.

[4] https://wwwtcs.inf.tu-dresden.de/ALGI/PUB/EPEW18

Configuration criteria. The goal of this section is to exemplarily find good parameter values that optimize the chosen protocol instance with respect to the following criteria: 1) the probability of terminating correctly should be at least 0.9995, 2) the conditional probability of aborting, in case of not terminating correctly, should be greater than 0.15, and 3) from all configurations meeting the conditions above, one with least energy overhead should be chosen.

Finding optimal configuration. Using probabilistic model checking, we compute parametric matrices P and E (as defined in Section 5). For this we use the parametric model checker Storm [20]. The matrices are parameterized over the error probability, detection probabilities, and the transaction length. As P and E depend on the number of redos, this causes separate runs of Storm for $max\_redos \in \{0, 1, 2, 3\}$. To systematically explore the design space, we first fix the error probabilities and replace the parameters with constants within P and E, as these values can be assumed to be given. We will consider three scenarios with $p\_e \in \{10^{-10}, 10^{-12}, 10^{-15}\}$. For each $max\_redos \in \{0, 1, 2, 3\}$ we then consider a discrete number of combinations for the detection probabilities ($p\_detn\_DFC, p\_detn\_DFC \in \{0, 0.001, 0.1, 0.5, 0.75, 0.9, 0.95, 0.99, 0.999\}$) and transaction length ($ta\_len \in \{100, 200, 500, 1000, 2000, 5000, 10^4, 10^6, 10^{10}, 10^{12}\}$) to fill decision tables (or plot the respective rational function). For this step we applied the Python-based computer algebra system SymPy [21][5].

The maximal number of redos. We start with analyzing the effect of the maximal number of redos. Without any fault-tolerance mechanism[6], the probability of terminating correctly is $3 \cdot 10^{-83}$ for $p\_e = 10^{-10}$, 0.15 for $p\_e = 10^{-12}$, and for $p\_e = 10^{-15}$ it is 0.998. Figure 2 shows the probability of terminating correctly, when varying *max_redos*. For all chosen error probabilities and detection probabilities, performing a single redo pays off drastically. For example, when the transaction length is 1000 and the detection probabilities both are set to 0.9, allowing a single redo increases the probability of terminating correctly from 0.027 to 0.827, when $p\_e = 10^{-12}$. When performing error detection without redo-based correction, increasing detection probabilities cause the probability of terminating correctly to shrink, since more instructions are replicated and thus more replicas can be affected by errors. Allowing redos neglect this effect. Also for all transaction lengths, except for the extremal case where the whole application is a single transaction, allowing redos increases the probability of terminating correctly significantly. Allowing more than one redo does only marginally increase the probability of terminating correctly, except for transaction lengths above 10^{10}. Fig.3 shows that each redo decreases the chance of aborting in case of not terminating correctly (Criteria 2). For two redos, this chance is almost zero for all configurations except for extremely large transaction

[5] Computing the matrices took 50 seconds for $max\_redos = 0$, 173 seconds for $max\_redos = 1$, 140 minutes for $max\_redos = 2$, and about one day and 3 hours for $max\_redos = 3$. Evaluating the rational functions to set up decision tables using SymPy [21] took less then a second per evaluation point for $max\_redos = 0$ and about 3 seconds per point for $max\_redos = 3$.

[6] Due to the nature of PPMC, these values need to be computed in a separate run. Applying the factorization approach of Section 5, this took less than three seconds.

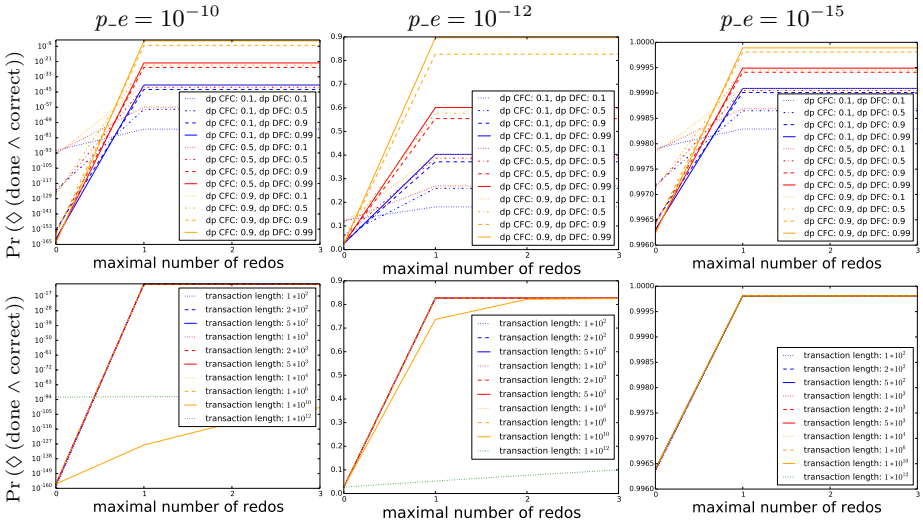

Fig. 2. Probability of terminating correctly in dependence on the maximal number of redos. First row: ranging detection probabilities, $ta\_len = 1000$. Second row: ranging transaction length, $p\_detn\_CFC = p\_detn\_DFC = 0.9$. From left to right: error probability $10^{-10}, 10^{-12}, 10^{-15}$. Note that y-scales in the left column are in log scale.

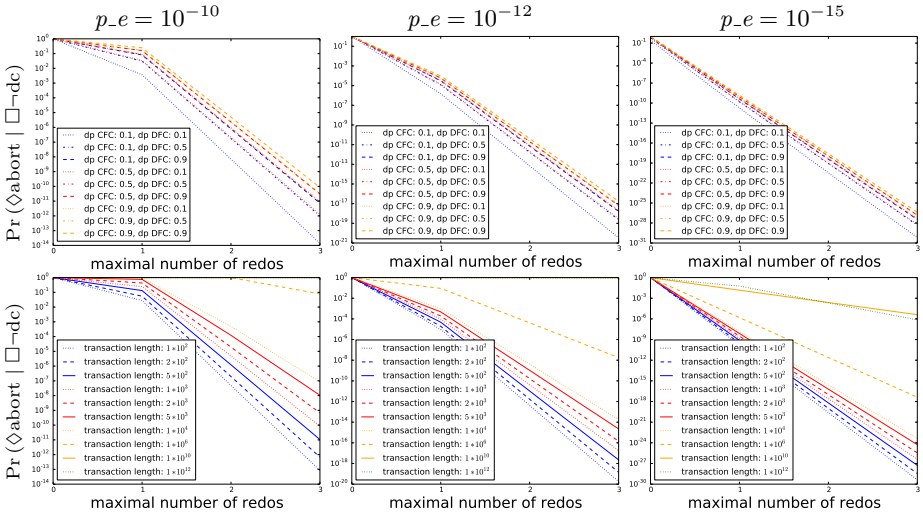

Fig. 3. Conditional probability of aborting in case of not terminating correctly in dependence of the maximal number of redos (dc is short for done ∧ correct.) First row: ranging detection probabilities. Second row: ranging transaction lengths. From left to right: error probability $10^{-10}, 10^{-12}, 10^{-15}$. Note that y-scales are in log scale.

lengths and small error probabilities. The correlation of the energy overhead and

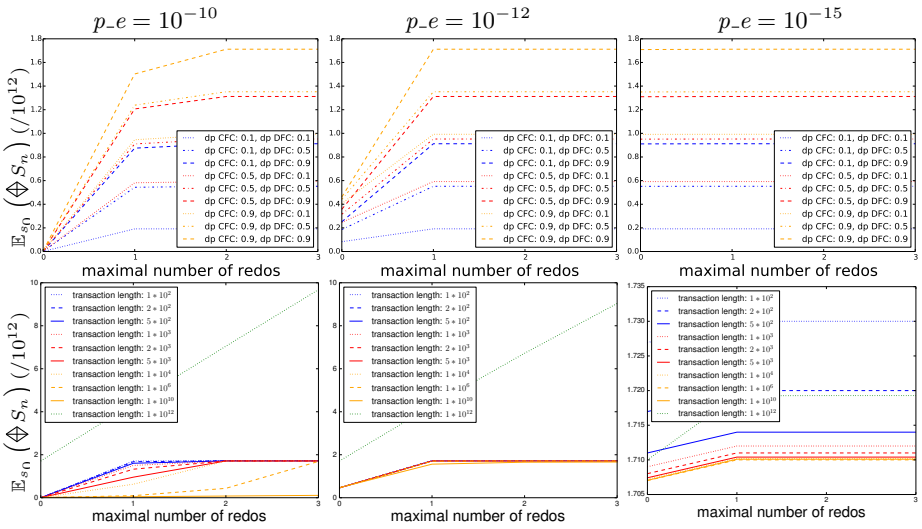

Fig. 4. Expected overhead in dependence of the maximal number of redos. First row: ranging detection probabilities. Second row: ranging transaction length. From left to right: error probability $10^{-10}, 10^{-12}, 10^{-15}$.

the maximal number of redos is depicted in Fig.4. For low error probabilities, the overhead is only marginally affected by the maximal number of redos, since, when errors are unlikely and thus error correction is invoked only seldom, the overhead is mainly defined by the number of replicated instructions executed for error detection. For higher error probabilities, expectably more errors occur and thus more error correction needs to be performed. So the overhead increases, when allowing more redos. Thus, choosing to perform at most one redo increases the probability of correct termination. Allowing another redo does not further increase this probability significantly, but decreases the probability of aborting in case of not terminating correctly without decrease in the overhead. Hence, from now on we fix $max\_redos = 1$.

Optimal transaction lengths. Fig.5 shows from top to bottom results for the three configuration criteria. For $p\_e = 10^{-10}$ the probability of terminating correctly is hardly affected by varying transaction lengths below 10^6. Choosing longer transaction length decreases the probability substantially. Large transaction lengths do increase the probability again, when *p_detn_DFC* is small, but short transaction lengths are in general to be preferred. For lower error probabilities, the turning point moves to the right. For error probability 10^{-15} it is beyond the maximal possible transaction length. The conditional probability of aborting when not terminating correctly (Fig.5, second row) first increases with increasing transaction lengths, then stays on a level near one for middle-large transaction lengths and finally drops again, in the same point as the probability of terminating correctly rises for some detection probabilities. Again, the turn-

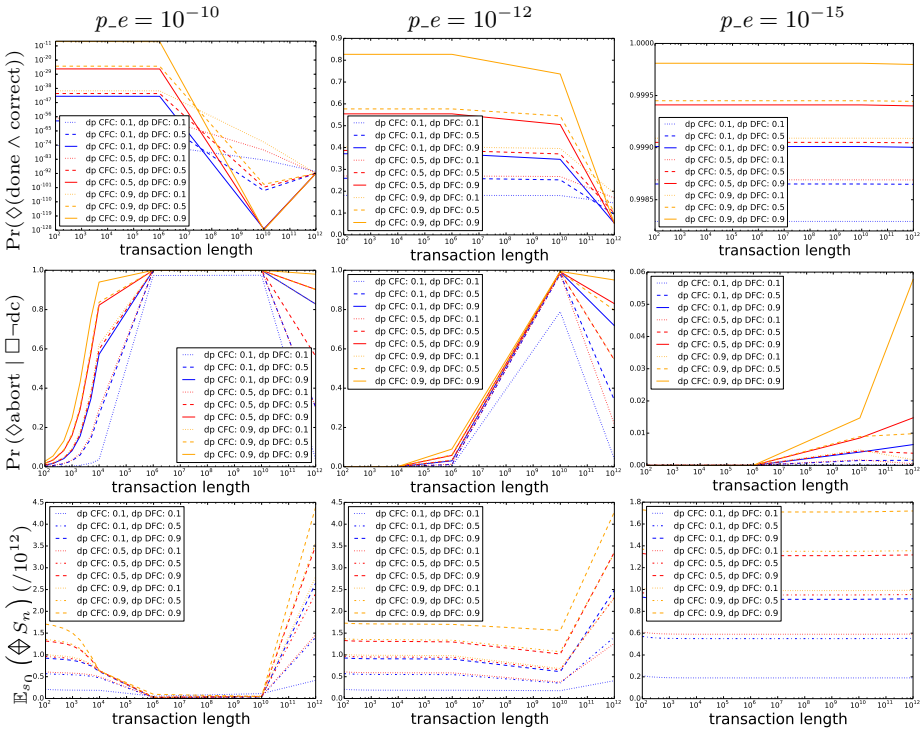

Fig. 5. In dependence of the transaction length for ranging detection probabilities, *max_redos*= 1: First row: Probability of terminating correctly. Second row: Conditional probability of aborting in case of not terminating correctly (dc is short for done $\wedge$ correct). Third row: Expected energy overhead. From left to right: error probability $10^{-10}, 10^{-12}, 10^{-15}$.

ing points move to the right when decreasing error probabilities. Regarding the overhead, small transaction lengths cause less transactions to be erroneous and thus less error correction to be performed. Nevertheless, very small transaction lengths cause error detection to take place very often and thus lead to a higher overhead. This is visible in the third row of Fig.5. For error probabilities 10^{-10} and 10^{-12}, increasing the transaction length up to 10^{10} decreases the overhead, but after this point, the overhead increases significantly. For error probability 10^{-15} the effect is barely visible. The decrease of energy overhead also arises because the probability of aborting increases with higher transaction length (cf. Fig.6): The higher this probability, the sooner the application is likely to be aborted. After an abort no more overhead is produced. Thus the overhead drops with increasing abort rates, and rises, when very large transaction lengths cause a shrinking abort rate. For $p\_e = 10^{-10}$, $ta\_len = 10^6$ is optimal under the investigated lengths. For $p\_e = 10^{-10}$, we choose $ta\_len = 10^{10}$, and for $p\_e = 10^{-15}$, errors are unlikely enough to handle the application as one transaction.

Error detection probabilities. We now fix the remaining configuration parameters, *p_detn_CFC* and *p_detn_DFC*. Fig.7 shows the effect of these param-

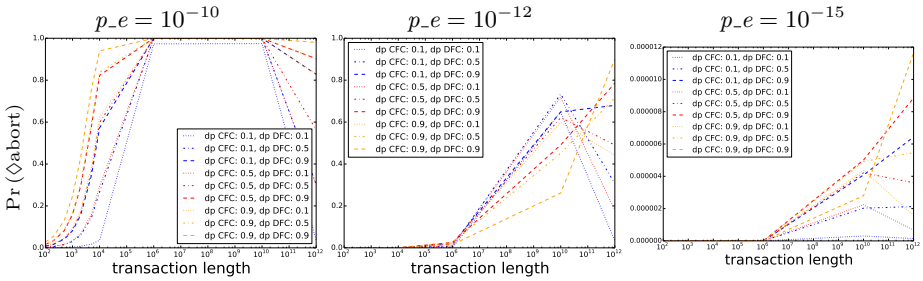

Fig. 6. The (unconditional) probability of aborting in dependence of the transaction length for ranging detection probabilities, *max_redos*= 1: From left to right: error probability $10^{-10}, 10^{-12}, 10^{-15}$.

eters with the previously chosen configurations for *max_redos* and *ta_len*. As expected, all three values increase when increasing the detection probabilities. Increasing only one error detection probability certainly has a visible effect on the probability of terminating correctly, yet it is ineffective to choose one detection probability to be very low and the other one to be very high. To configure the remaining parameters of the fault-tolerance technique, we use a decision table, exemplarily for $p\_e = 10^{-15}$. Table 8 shows results for $max\_redos = 1$ and $ta\_len = 10^{12}$. All depicted lines satisfy the first two configuration criteria. The energy overhead does not differ much, but the conditional probability of aborting when not terminating correctly can be increased significantly when accepting a little more overhead. Thus, it would be worth choosing a configuration that replicates some more instructions, accepting a little more overhead, e.g., $p\_detn\_CFC = p\_detn\_DFC = 0.999$.

7 Conclusion

The purpose of the paper was to illustrate how probabilistic model checking techniques can be employed to determine parameter settings for a redo-based fault-tolerance protocol minimizing the expected overhead subject to resilience constraints. We dealt with discrete parameters that affect the topological structure of the state space (number of redos, transaction length) and continuous parameters (error detection rates). This spans a large family of protocols arising by concrete choices for the parameters. Due to the huge state spaces of these Markov chains, the direct application of standard model checking techniques for probability-parametric Markov chains was not feasible. Instead, we employed a new factorization approach that exploits the repeating phases in the models and used a combination of PPMC and computer-algebra techniques to compute and analyze the rational functions for the relevant probabilities and expectations in these Markov chains, and finally to extract an optimal parameter valuation. While the factorization technique is specific for redo protocols, we argue that the remaining steps are of exemplary character.

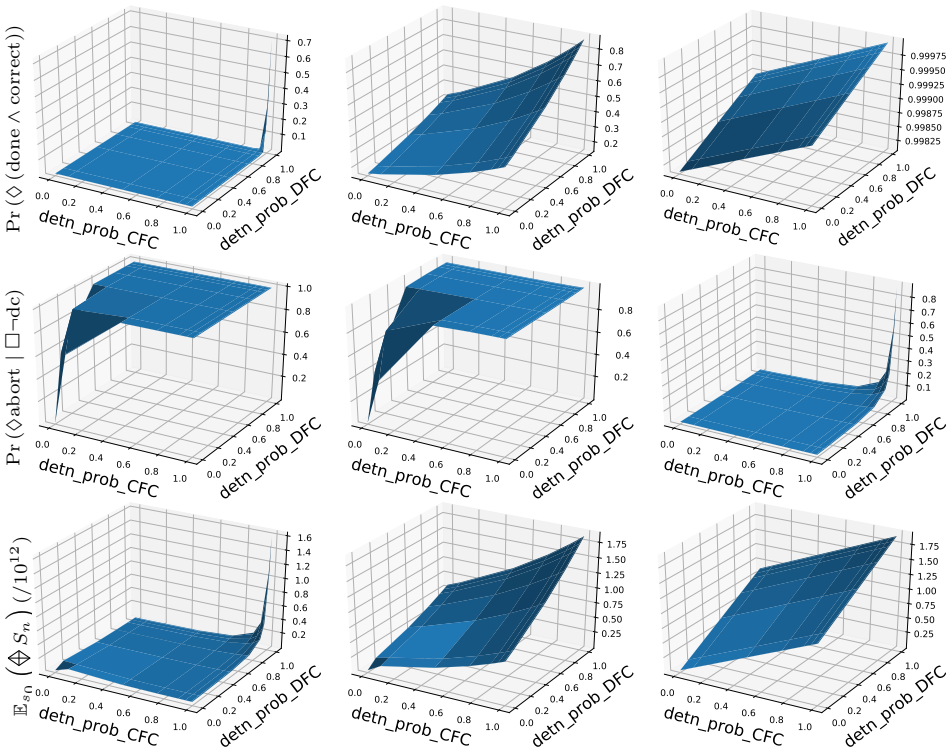

Fig. 7. In dependence of the detection probabilities, for *max_redos*= 1: First row: Probability of terminating correctly. Second row: Conditional probability of aborting in case of not terminating correctly (dc is short for done ∧ correct). Third row: Expected overhead. From left to right: $p\_e = 10^{-10}$ and $ta\_len = 10^6$, $p\_e = 10^{-12}$ and $ta\_len = 10^{10}$, $p\_e = 10^{-15}$ and $ta\_len = 10^{12}$.

p_detn_CFC	*p_detn_DFC*	Criteria 1	Criteria 2	Criteria 3
0.95	0.99	0.99992	0.19	$1.851 \cdot 10^{12}$
0.95	0.999	0.99994	0.21	$1.86 \cdot 10^{12}$
0.99	0.95	0.99993	0.19	$1.855 \cdot 10^{12}$
0.99	0.99	0.99997	0.43	$1.892 \cdot 10^{12}$
0.99	0.999	0.99997	0.57	$1.9 \cdot 10^{12}$
0.999	0.95	0.99994	0.23	$1.864 \cdot 10^{12}$
0.999	0.99	0.99998	0.59	$1.901 \cdot 10^{12}$
0.999	0.999	0.99998	0.88	$1.909 \cdot 10^{12}$

Fig. 8. Decision table for $p\_e = 10^{-15}$, $ta\_len = 10^{12}$ and $max\_redos = 1$.

References

[1] D. Kuvaiskii, R. Faqeh, P. Bhatotia, P. Felber, and C. Fetzer, "HAFT: Hardware-assisted fault tolerance," in *European Conference on Computer Systems (Eurosys)*, ACM, 2016.

[2] G. K. Saha, "Approaches to software based fault tolerance," *Computer Science Journal of Moldova*, vol. 13, no. 2, pp. 193–231, 2005.

[3] K. R. Apt and D. Kozen, "Limits for automatic verification of finite-state concurrent systems," *Information Processing Letters*, vol. 22, no. 6, pp. 307–309, 1986.

[4] R. Lanotte, A. Maggiolo-Schettini, and A. Troina, "Parametric probabilistic transition systems for system design and analysis," *Formal Aspects of Computing*, vol. 19, no. 1, pp. 93–109, Mar. 2007.

[5] E. M. Hahn, H. Hermanns, and L. Zhang, "Probabilistic reachability for parametric Markov models," *Software Tools for Technology Transfer*, vol. 13, no. 1, pp. 3–19, 2011.

[6] T. Quatmann, C. Dehnert, N. Jansen, S. Junges, and J.-P. Katoen, "Parameter synthesis for markov models: Faster than ever," in *Automated Technology for Verification and Analysis*, C. Artho, A. Legay, and D. Peled, Eds., Springer International Publishing, 2016, pp. 50–67.

[7] V. Sridharan and D. Liberty, "A study of dram failures in the field," in *High Performance Computing, Networking, Storage and Analysis (SC), 2012 International Conference for*, Nov. 2012, pp. 1–11.

[8] X. Li, M. C. Huang, K. Shen, and L. Chu, "A realistic evaluation of memory hardware errors and software system susceptibility," in *Proceedings of the 2010 USENIX Conference on USENIX Annual Technical Conference*, ser. USENIXATC'10, USENIX Association, 2010, pp. 6–6.

[9] V. Sridharan, J. Stearley, N. DeBardeleben, S. Blanchard, and S. Gurumurthi, "Feng shui of supercomputer memory positional effects in dram and sram faults," in *2013 SC - International Conference for High Performance Computing, Networking, Storage and Analysis (SC)*, Nov. 2013, pp. 1–11.

[10] M. Cubuktepe, N. Jansen, S. Junges, J. Katoen, I. Papusha, H. A. Poonawala, and U. Topcu, "Sequential convex programming for the efficient verification of parametric MDPs," in *Tools and Algorithms for the Construction and Analysis of Systems (TACAS), Part II*, ser. LNCS, vol. 10206, 2017, pp. 133–150.

[11] M. Češka, F. Dannenberg, N. Paoletti, M. Kwiatkowska, and L. Brim, "Precise parameter synthesis for stochastic biochemical systems," *Acta Informatica*, vol. 54, no. 6, pp. 589–623, 2017.

[12] T. Han, J. Katoen, and A. Mereacre, "Approximate parameter synthesis for probabilistic time-bounded reachability," in *Proceedings of the 29th IEEE Real-Time Systems Symposium, RTSS*, 2008, pp. 173–182.

[13] E. Bartocci, R. Grosu, P. Katsaros, C. R. Ramakrishnan, and S. A. Smolka, "Model repair for probabilistic systems," in *Tools and Algorithms for the Construction and Analysis of Systems*, P. A. Abdulla and K. R. M. Leino, Eds., Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, pp. 326–340.

[14] T. Chen, T. Han, M. Kwiatkowska, and H. Qu, "Efficient probabilistic parameter synthesis for adaptive systems," DCS, Tech. Rep. RR-13-04, 2013, p. 13.

[15] C. Baier and J.-P. Katoen, *Principles of Model Checking*. MIT Press, 2008.

[16] M. L. Puterman, *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. John Wiley & Sons, 1994.

[17] C. Baier, M. Größer, and F. Ciesinski, "Partial order reduction for probabilistic systems," in *Quantitative Evaluation of Systems (QEST)*, IEEE, 2004, pp. 230–239.

[18] P. R. D'Argenio and P. Niebert, "Partial order reduction on concurrent probabilistic programs," in *Quantitative Evaluation of Systems (QEST)*, IEEE, 2004, pp. 240–249.

[19] M. Größer and C. Baier, "Partial order reduction for Markov decision processes: A survey," in *Formal Methods for Components and Objects (FMCO)*, ser. LNCS, vol. 4111, 2005, pp. 408–427.

[20] C. Dehnert, S. Junges, J. Katoen, and M. Volk, "A storm is coming: A modern probabilistic model checker," in *Computer Aided Verification (CAV), Part II*, ser. LNCS, vol. 10427, 2017, pp. 592–600.

[21] A. Meurer, C. P. Smith, M. Paprocki, O. Čertík, S. B. Kirpichev, M. Rocklin, A. Kumar, S. Ivanov, J. K. Moore, S. Singh, T. Rathnayake, S. Vig, B. E. Granger, R. P. Muller, F. Bonazzi, H. Gupta, S. Vats, F. Johansson, F. Pedregosa, M. J. Curry, A. R. Terrel, Š. Roučka, A. Saboo, I. Fernando, S. Kulal, R. Cimrman, and A. Scopatz, "Sympy: Symbolic computing in Python," *PeerJ Computer Science*, vol. 3:e103, 2017.

[22] R. Segala, "Modeling and verification of randomized distributed real-time systems," PhD thesis, MIT, 1995.

A Appendix

A.1 Redo-Based Fault-Tolerance Model

We give a detailed insight in the model we set up for analysis. This model contains adjustable *attributes*, which are summarized in Table 9.

component	attribute	explanation
Application	*cf_df_ratio*	percentage of control-flow instructions of *inst_num*
	inst_num	total number of instructions executed by the application, excluding replications
TRM	*max_redos*	number of transaction redos before application is aborted
	tmi_num	number of instructions added to implement the transaction mechanism
	ta_len	number of instructions per transaction
	p_e_TRM	probability of an error in the TRM
DFC	*p_detn_DFC*	probability of detecting a data-flow error in the data-flow instructions of a transaction
	p_fp_DFC	probability of a false positive in the data-flow of a transaction
CFC	*p_detn_CFC_df*	probability of detecting a control-flow error in the data-flow instructions of a transaction
	p_detn_CFC_cf	probability of detecting a control-flow error in the control-flow instruction of a transaction
	p_detn_CFC_tmi	probability of detecting a control-flow error in the transaction management instructions of a transaction
	p_detn_CFC	combined probability of detecting a control-flow error in a transaction
	p_fp_CFC_df	probability of false positive in control-flow of data-flow instructions of a transaction
	p_fp_CFC_cf	probability of false positive in the control-flow of control-flow instructions of a transaction
	p_fp_CFC_tmi	probability of false positive in transaction management instructions of a transaction
	p_fp_CFC	combined probability of false positive in the control-flow of a transaction
Hardware	*p_e*	probability of an error in a single instruction
	p_e_incr	probability of an error in a single instruction when the hardware is in location "failure"

Fig. 9. Summary of the model attributes.

The formal model consists of components for the underlying hardware, the application, and the fault-tolerance protocol. The latter contains sub-modules for a control-flow checker (CFC), a data-flow checker (DFC), and the transaction redo manager (TRM) implementing the redo/abort-schema (cf. Figure 10).
Application model. An application performs a fixed number on instructions *inst_num*. The instruction flow is separated into transactions, each consisting

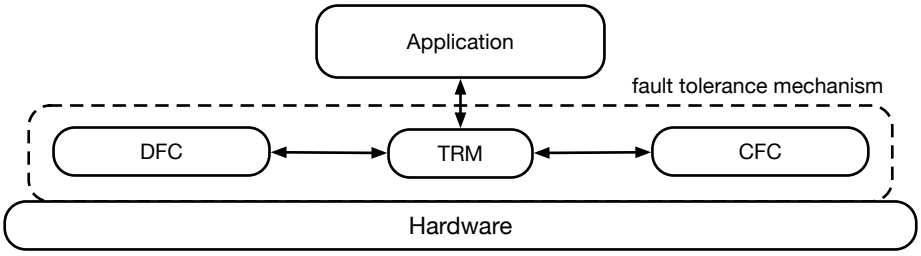

Fig. 10. The structure of our model.

of *ta_len* instructions. These attributes define the number of transactions to be performed, *ta_num* = *inst_num* / *ta_len*. We assume three distinct types of instructions: instructions where errors affect only the control-flow (e.g., jump), named *control-flow instructions*, instructions where errors affect also the data-flow (e.g. add), named *data-flow instructions* and instructions introduced to implement the transaction mechanism (e.g., begin-of-transaction and end-of-transaction instructions), named *transaction management instructions*. For the latter, errors do only affect the control flow. Since data-flow instructions also carry control-flow information, errors in data-flow instructions can also affect and falsify the control-flow. The ratio of control-flow and data-flow instructions can be set via the attribute *cf_df_ratio*. The amount of transaction management instructions is controlled by the attribute *tmi_num*.
Transaction execution is modeled as an atomic step, named PERFORM_TA (see Figure 11), in which the local counter variable *ta_counter* is increased. We use an ! to denote the sender of a synchronization action, where ? is used on the receiver side. In case of PERFORM_TA, the receiver side is played by the hardware component (see paragraph "hardware model" below) that synchronizes on PERFORM_TA.

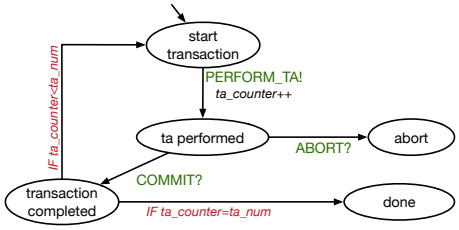

Fig. 11. The program graph of the application.

After performing a transaction, the application is paused. During this pause, error detection and correction is performed by the TRM, DFC, and CFC. After this, the TRM sends either a COMMIT or an ABORT signal to the application, by executing the respective action. The application synchronizes on this action, either ending in an abort location or reaching location "transaction completed". In the latter case, if *ta_counter* did not yet reach *ta_num*, the application's

location is changed to "start" and the next transaction is performed. Otherwise, location "done" is reached and the application stops.

TRM model. The TRM is the coordinative center of the fault tolerance protocol. Its attribute *max_redos* can be set to any natural number including zero. After each transaction performed by the application, it initiates checks on the dataflow and the control flow, launches redos if necessary, and finally sends a COMMIT or ABORT signal to the application. In our model, the TRM (Figure 12) invokes the DFC and CFC with the synchronous action CHECK simultaneously as soon as the transaction has been performed. It then waits, until

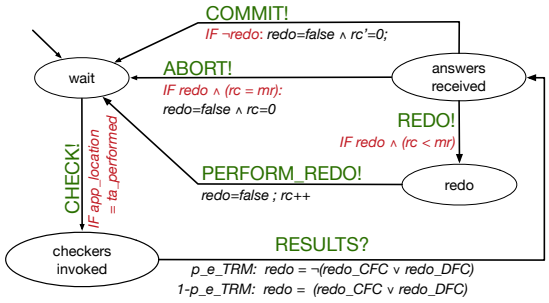

Fig. 12. The program graph of the TRM. rc is short for *redo_counter*, mr abbreviates *max_redos*.

within action RESULTS the results of both checkers are collected and evaluated in location "answer received". The next action depends on whether an error was detected and whether an error in the TRM has occurred. If an error was detected by one of the checkers, and no error in the TRM, e.g., falsified the signal, a redo is initiated (if the maximal number of redos is not yet exceeded). The probability of an error in the TRM can be controlled by the attribute *p_e_TRM*. If either no error was detected, or an error was detected, but another error in the TRM occurred, no redo will be performed but a COMMIT signal will be sent. Whether or not the TRM recommends a redo is recorded in the boolean variable *redo*. If so, and if *redo_counter* $<$ *max_redos*, the transaction will be re-executed by the TRM. If *redo_counter* $=$ *max_redos*, an ABORT signal will be sent to the application. As for the original transaction, the redo is performed in an atomic step, named PERFORM_REDO, and *redo_counter* is increased. After the redo, the TRM falls back to its "wait" location, and immediately leaves it with again invoking the checkers, since the condition "app_location = ta_performed" still is satisfied. The checkers will then check the redo for errors.

DFC and CFC model. Both checkers are invoked via synchronizing on the CHECK signal of the TRM (see Figure 13). The DFC then checks the dataflow in one atomic step. If an error occurred in one of the data-flow instructions, it detects this error with probability *p_detn_DFC*. If no error occurred in the original data-flow instructions, an error might still be detected, i.e., the DFC has a false positive with probability *p_fp_DFC*.

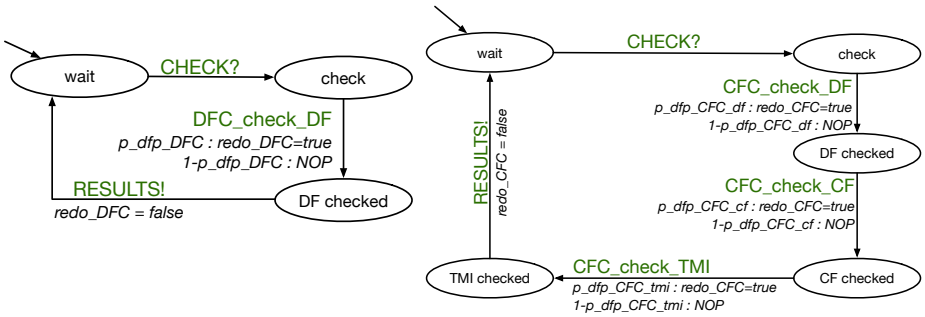

Fig. 13. The program graph of the DFC (left) and CFC (right). If an error occurred, $p\_dfp\_DFC = p\_detn\_DFC$ and $p\_dfp\_CFC\_* = p\_detn\_CFC\_*$. Otherwise $p\_dfp\_DFC = p\_fp\_DFC$ and $p\_dfp\_CFC\_* = p\_fp\_CFC\_*$. NOP means that no variables are changed.

To support a larger class of fault tolerance techniques, we assume the control-flow checker to be able to check each type of instruction (control-flow, data-flow and transaction management) separately. We assume that all control-flow instructions are checked for errors first, followed by all data-flow instructions, and completed by checking all transaction management instructions. In each package errors are detected with $p\_detn\_CFC\_df$, $p\_detn\_CFC\_cf$, and $p\_detn\_CFC\_tmi$, respectively. Analogously, false positives occur with probabilities $p\_fp\_CFC\_df$, $p\_fp\_CFC\_cf$, and $p\_fp\_CFC\_tmi$.

Nevertheless, evaluation criteria used in Section 6 do not distinguish in the actions CFC_check_DF, CFC_check_CF, CFC_check_TMI or the states in-between. Thus for our exemplaric evaluation in Section 6 we can use the results of [22] (Proposition 8.7.1 on probabilistic forward simulations) and replace the sequence of actions CFC_check_DF, CFC_check_CF, CFC_check_TMI by the single action CFC_check, representing the sequential execution of all three steps, see Figure 14.

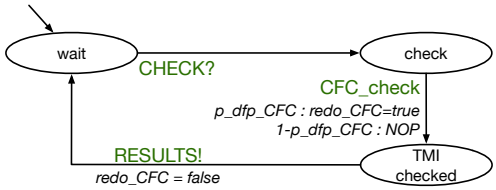

Fig. 14. The program graph of the CFC after simplifying using simulation results. If an error occurred, $p\_\text{dfp\_CFC} = 1 - ((1 - p\_detn\_CFC\_df) \cdot (1 - p\_detn\_CFC\_cf) \cdot (1 - p\_detn\_CFC\_tmi))$ otherwise $p\_\text{dfp\_CFC} = 1 - ((1 - p\_fp\_CFC\_df) \cdot (1 - p\_fp\_CFC\_cf) \cdot (1 - p\_fp\_CFC\_tmi))$

When both checkers are finished, they synchronously send their results to the TRM and fall back to their "wait" location.

Hardware model. The hardware model keeps track of the state of the internal memory occupied by the application. It comes with two attributes: the probability of an error occurring in a single instruction, when the application is correct (p_e) and the increased probability of an error (p_e_incr) occurring in an instruction when the application has a failure, i.e., in a former transaction an undetectable error occurred.

The hardware is characterized by its internal status (operating normally, being erroneous, having a failure, or having a failure and another error occurred, cf Figure 15). Starting in location "correct" and synchronizing on both actions PERFORM_TA and PERFORM_REDO, an error occurs in the transaction with probability $p\_e\_ta = 1 - (1 - p\_e)^{ta\_len}$, leading to location "error". With probability $1 - p\_e\_ta$, the hardware stays in location "correct", when synchronizing on PERFORM_TA or PERFORM_REDO. When the hardware is in location "error", and the TRM initiates a REDO, it falls back to the "correct" location. Being in location "error" and a COMMIT signal is sent, the error was not detected and thus the location is changed to "failure". Being in location "failure", the same behavior is modeled, but with increased error probabilities, and, when being in location "failure and error", both REDO and COMMIT change the location to "failure".

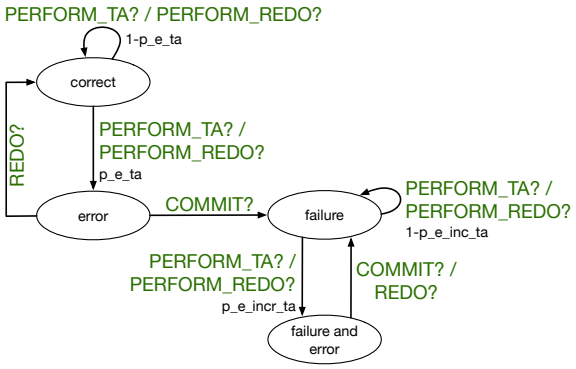

Fig. 15. The program graph of the hardware model.

A.2 Proof of Lemma 1

Proof. For all $0 \leq i < n, s_i \in S_i$ we have

$$\begin{aligned}\mathbb{E}_{s_i}(\Diamond S_n) &= \sum_{s_{i+1} \in S_{i+1}} \Pr_{s_i}(\Diamond s_{i+1}) \cdot \mathbb{E}_{s_i}(\Diamond S_n \mid \Diamond s_{i+1}) \\ &= \sum_{s_{i+1} \in S_{i+1}} \Pr_{s_i}(\Diamond s_{i+1}) \cdot \mathbb{E}_{s_i}(\Diamond s_{i+1} \mid \Diamond s_{i+1}) + \sum_{s_{i+1} \in S_{i+1}} \Pr_{s_i}(\Diamond s_{i+1}) \cdot \mathbb{E}_{s_{i+1}}(\Diamond S_n).\end{aligned}$$

Applying upper equation to $\mathbb{E}_{s_0}(\Diamond S_n)$, then recursively to $\mathbb{E}_{s_i}(\Diamond S_n)$, and using the definition of P and E yields

$$\begin{aligned}\mathbb{E}_{s_0}(\Diamond S_n) &= \sum_{k=1}^{n} \sum_{s_1 \in S_1} \cdots \sum_{s_k \in S_k} \prod_{j=0}^{k-1} \Pr_{s_j}(\Diamond s_{j+1}) \cdot \mathbb{E}_{s_{k-1}}(\Diamond s_k \mid \Diamond s_k) \\ &= \sum_{k=1}^{n} \sum_{s_1 \in S_1} \cdots \sum_{s_k \in S_k} \prod_{j=0}^{k-1} P_{s_j, s_{j+1}} \cdot E_{s_{k-1}, s_k} \\ &= \sum_{k=1}^{n} \sum_{s_{k-1} \in S_{k-1}} \sum_{s_1 \in S_1} \cdots \sum_{s_{k-2} \in S_{k-2}} \prod_{j=0}^{k-2} P_{s_j, s_{j+1}} \cdot \sum_{s_k \in S_k} P_{s_{k-1}, s_k} \cdot E_{s_{k-1}, s_k}.\end{aligned}$$

Now observe that for all $1 \leq k \leq n$

$$\sum_{s_1 \in S_1} \cdots \sum_{s_{k-2} \in S_{k-2}} \prod_{j=0}^{k-2} P_{s_j, s_j+1} = (P^{k-1})_{s_0, s_{k-1}}$$

is the probability of reaching state s_{k-1} from state s_0. This equation and application of the standard definition of scalar multiplication yields

$$\begin{aligned}\mathbb{E}_{s_0}(\Diamond S_n) &= \sum_{k=1}^{n} \sum_{s_{k-1} \in S_{k-1}} (P^{k-1})_{s_0, s_{k-1}} \cdot \sum_{s_k \in S_k} P_{s_{k-1}, s_k} \cdot E_{s_{k-1}, s_k} \\ &= \sum_{k=1}^{n} \sum_{s_{k-1} \in S_{k-1}} \left(P^{k-1}\right)_{s_0, s_{k-1}} \cdot pe_{s_{k-1}} \\ &= \sum_{k=1}^{n} \left(P^{k-1} \cdot pe\right)_{s_0} = \left(\sum_{k=1}^{n} P^{k-1} \cdot pe\right)_{s_0}.\end{aligned}$$

□