Evaluating the Reliability of NAND Multiplexing with PRISM

Gethin Norman, David Parker, Marta Kwiatkowska and Sandeep Shukla

***Abstract*— Probabilistic model checking is a formal verification technique for analysing the reliability and performance of systems exhibiting stochastic behaviour. In this paper, we demonstrate the applicability of this approach and, in particular, the probabilistic model checking tool PRISM to the evaluation of reliability and redundancy of defect-tolerant systems in the field of computer-aided design. We illustrate the technique with an example due to von Neumann, namely NAND multiplexing. We show how, having constructed a model of a defect-tolerant system incorporating probabilistic assumptions about its defects, it is straightforward to compute a range of reliability measures and investigate how they are affected by slight variations in the behaviour of the system. This allows a designer to evaluate, for example, the trade-off between redundancy and reliability in the design. We also highlight errors in analytically computed reliability bounds, recently published for the same case study.**

***Index Terms*— Probabilistic model checking, reliability, defect-tolerant architectures, multiplexing**

I. Introduction

PROBABILISTIC MODEL checking is a formal verification technique which has already been successfully used to analyse the performance and reliability of a wide range of real-life systems, including dynamic power management schemes [1], embedded systems [2], computer networks, queueing systems and manufacturing processes. It has also been used to study "quality of service" properties of real-time probabilistic communication protocols, such as IEEE 1394 FireWire [3], IEEE 802.3 CSMA/CD [4], Zeroconf [5], IEEE 802.11 wireless LANs [6] and Bluetooth [7], and to verify both probabilistic security protocols (e.g. [8]) and randomised distributed algorithms (e.g. [9]).

In this paper we present results which demonstrate the advantages of using probabilistic model checking and, in particular, the probabilistic model checking tool PRISM [10], to model and analyse defect-tolerant systems. We have chosen to investigate the reliability of NAND multiplexing [11], but this approach can be applied to other defect-tolerant systems such as R-fold Modular Redundancy [11] and Reconfiguration [12], [13]. This work differs from the standard approaches in the literature to analysing multiplexing in that we evaluate the reliability of specific cases as opposed to considering the general framework, and hence are not necessarily restricted by the analytical bounds of reliability of, for example, von Neumann [11] and Pippenger [14].

Our results demonstrate that, by applying probabilistic model checking, it is straightforward to investigate the effect on reliability of slight variations in the behaviour of the system's components, for example the change in reliability as the probability of gate failure varies. In addition, the construction of a formal specification of a NAND multiplexing system, a step required in the probabilistic model checking approach, enabled us to find a flaw in the the analytical approach of Han and Jonker [15]. We must note here that the flaw in the analysis in [15] does not invalidate their results on the suitability of NAND multiplexing, but does change the characteristic curves presented slightly. However, we use the error to illustrate that analytical modelling for such a complex combinatorial system with probabilistic quantification is error prone, and hence automating the construction of the probabilistic model and its analysis is desirable to obtain accurate results. Furthermore, using PRISM, we show that this flaw can lead to both an under- and over-approximation of reliability.

In the next section, we introduce the basic concepts of NAND multiplexing, probabilistic model checking and the PRISM tool. In Section III, we describe how we use the PRISM framework to model NAND multiplexing. Section IV reports on the results obtained for this case study and Section V concludes the paper.

II. Background

A. NAND Multiplexing

In 1952, von Neumann studied the problem of performing reliable computations with unreliable devices (due to the unreliable valve-based computers in use at that time), introducing a redundancy technique called *multiplexing* [11]. Today, such methods are again gaining significance, for example in the field of nanotechnology, where manufacturing devices at an extremely small scale suffers from unavoidable problems of defects in their components.

The basic technique of multiplexing is to replace a single processing unit by a *multiplexing unit*, which has N copies of every input and output of the original unit. The multiplexing unit, using multiple instances of the original unit, processes the N inputs in parallel, giving N outputs which represent the output of the original processing unit. If the inputs and devices are reliable, then each element of the output set should be identical and equal to the output of the corresponding processing unit. However, in the case where there are errors,

This work was supported by NSF grants CCR-0098335 and CCR-0340740, EPSRC grants GR/N22960 and GR/S11107, FORWARD, SRC, and DARPA/ITO supported PADS project under the PAC/C program.

Gethin Norman, David Parker and Marta Kwiatkowska are with the School of Computer Science, University of Birmingham, Birmingham, B15 2TT, UK.

Sandeep Shukla is with the Bradley School of Electrical & Computer Engineering, Virginia Tech, Blacksburg, VA 24060, USA.

either in the inputs or in the processing devices, the outputs will not all take the same value. Instead, after defining some critical level $\Delta \in (0, 0.5)$, the output of the multiplexing unit is considered to be *stimulated* (taking logical value `true` or '1') if at least $(1-\Delta)\cdot N$ of the outputs are stimulated, and is said to be *non-stimulated* if no more than $\Delta\cdot N$ lines are stimulated. In cases where the number of stimulated outputs does not meet either of these criteria, i.e. the number of stimulated outputs is in the interval $(\Delta\cdot N, (1-\Delta)\cdot N)$, the output is undecided, and a malfunction occurs.

The design of a multiplexing unit consists of two stages: the *executive* stage and the *restorative* stage. The executive stage carries out the basic function of the unit to be replaced, while the restorative stage is used to reduce the degradation in the executive stage caused by errors in the inputs and faulty devices.

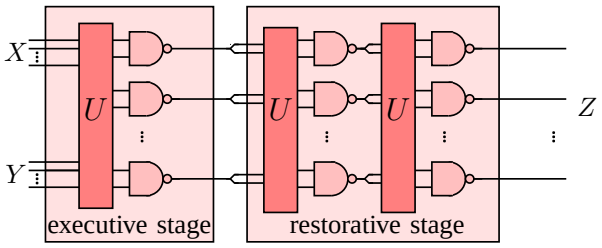


Fig. 1. A NAND multiplexing unit

The specific instance of *NAND multiplexing*, i.e. when the original processing unit is a NAND gate, is illustrated in Figure 1. In this case, the executive stage consists of N copies of a NAND gate and a unit U which perform a *random permutation* of the input signals, that is, each signal of the first input bundle (X) is randomly paired with a signal from the second input bundle (Y) to form an input pair for one of the copies of the NAND gate. Also shown in Figure 1 is the restorative stage which is derived using the same technique as the executive stage, duplicating the outputs of the executive stage to use as its inputs. Note that applying this approach only once will invert the result, and therefore two steps are required. To give a more effective restoration mechanism the restorative stage can be iterated [11]. A more detailed description of NAND multiplexing can be found in [15].

In [11], von Neumann concluded that, for extremely large N, the number of stimulated outputs of the executive stage is a stochastic variable, approximately normally distributed, and he gave an upper bound of 0.0107 for the probability of gate failure that can be tolerated given the value of Δ equal to 0.07, which is most favourable to restoration. Recently, it was shown that, if each NAND gate fails independently, the tolerable threshold probability of each gate will be 0.08856 [16]. However, this particular result is independent of the NAND multiplexing construction, applying to general Boolean functions. It is shown in [15] that, for smaller N, the number of outputs of the executive stage is theoretically a binomial distribution. The authors then go on to demonstrate how additional restorative stages improve fault-tolerance and that the error distribution of the system evolves as a stochastic homogeneous Markov chain.

B. Probabilistic Model Checking and PRISM

Probabilistic model checking is a formal verification technique for analysing reliability and performance measures of systems exhibiting randomised behaviour. For example, using this approach, one can establish properties such as "shutdown eventually occurs with probability at most 0.01" and "with probability 0.95 or greater, the process will successfully complete within 200 steps and without requiring any repairs".

The process of probabilistic model checking involves construction of a formal model of the real-life system which is to be analysed. This is usually a labelled state transition system enriched with probabilistic information, which represents all the possible configurations which the system can be in and all the transitions which can occur between them. Three types of probabilistic models commonly used are discrete-time Markov chains (DTMCs), continuous-time Markov chains (CTMCs) and Markov decision processes (MDPs). The models used in this paper are DTMCs, which comprise a set of states S and a transition probability matrix $\mathbf{P} : S \times S \to [0, 1]$ such that $\sum_{s' \in S} \mathbf{P}(s, s') = 1$ for all $s \in S$. Each element $\mathbf{P}(s, s')$ of the transition probability matrix gives the probability of making a transition from state s to state s'.

Properties of probabilistic models to be analysed are specified formally. Traditionally, in the model checking paradigm, this is done using temporal logic, which provides a concise and unambiguous specification. For DTMCs and MDPs, an appropriate logic is PCTL (Probabilistic Computation Tree Logic) [17], and for CTMCs, the logic CSL (Continuous Stochastic Logic) [18], [19] is often used. Since we focus on DTMCs, we write properties in PCTL. Some example specifications in this logic are as follows:

- $\mathcal{P}_{\leq 0.01}[\Diamond\ shutdown]$ – "shutdown eventually occurs with probability at most 0.01";
- $\mathcal{P}_{\geq 0.95}[\neg repair\ \mathcal{U}^{\leq 200}\ complete]$ – "with probability 0.95 or greater, the process will successfully complete within 200 steps and without requiring any repairs".

The labels such as $shutdown$ and $repair$ are atomic propositions which are assigned to states of the model at the time of its construction. The use of probability bounds (≤ 0.01, ≥ 0.95) ensures that the properties above constitute questions which can be verified either to be true or false, as is traditionally the case in formal verification. In practise, though, it is often more useful to request the actual values by writing for example:

- $\mathcal{P}_{=?}[\Diamond\ shutdown]$ – "what is the probability that the system shuts down?"

A probabilistic model checker applies algorithmic techniques to construct and analyse a probabilistic model and determine whether the given specifications are satisfied. We use the probabilistic model checker PRISM [10], [20] developed at the University of Birmingham. This provides support for the three types of models (DTMCs, MDPs and CTMCs) and the two logics (PCTL and CSL) described above. Probabilistic models are specified in the high-level PRISM modelling language, a variant of Reactive Modules [21], which is based on guarded commands. More details on this language are given in Section III-A.

For the specific scenario of verifying DTMCs against PCTL specifications, as is the case in this paper, probabilistic model checking constitutes a combination of graph-based analysis and solving linear equation systems. In PRISM, the latter is performed using iterative numerical solution techniques, for example Jacobi and Gauss-Seidel (see e.g. [22]). Furthermore, for both types of analysis, PRISM incorporates sophisticated *symbolic* implementation techniques, using Binary Decision Diagrams (BDDs) and related data structures [23], [24]. The principal advantage of these methods is that that they allow efficient compact storage and efficient manipulation of extremely large probabilistic models, by exploiting high-level regularity from the PRISM language description.

For a detailed overview of probabilistic model checking, see for example [25]. For more information about the PRISM tool and the range of case studies to which it has been applied, see the tool website [20].

III. NAND Multiplexing and PRISM

In this section, we explain how probabilistic model checking, and in particular PRISM, can be used to model and analyse the performance of von Neumann's NAND multiplexing system. We have chosen the NAND multiplexing system as it is a typical example of a fault-tolerant architecture from the literature. Note, however, that it is straightforward to construct models where the NAND gates have different faults (in our analysis we restricted our attention to von Neumann faults, where the output of a gate is inverted with a given probability) or to consider different architectures for reliable computing with unreliable devices, for example R-fold modular redundancy, Cascaded Triple Modular redundancy and Reconfiguration [26].

We obtain results that not only uncover a bug in the previous computations of [15], based on analytical methods, but also reveal an interesting trend as the probability of gate failure varies. More precisely, our results show that, for large probabilities of gate failure, increasing the number of restorative stages decreases reliability, while, for small probabilities of gate failure, it has the opposite effect. This demonstrates that, with our framework, we can not only quickly evaluate these measures, but can also easily find counter-intuitive phenomena.

It is important to note that the results presented here differ from the theoretical results which give a lower bound of the failure rates required for correctness (for example [11], [14]), since our results are with respect to a multiplexing system with a fixed configuration. The advantage of this approach is that we obtain *exact* values for the configuration under study. The disadvantage is that the results do not mechanically carry over to the performance of an architecture with a different configuration. However, one can simply construct a PRISM model for the different configuration and then re-run experiments on this model. For example, as shown in Section IV, we have considered configurations with varying gate failures probabilities and varying numbers of restorative stages.

A. Model Construction

In this section we explain our PRISM model. We first note that it was during this phase that we noticed the error made by [15] in modelling the random permutation made by the unit U. In the analysis technique of [15], the random permutation made by U is instead modelled by a random choice *with* replacement. More precisely, in the approach of [15], if in the previous stage there are k stimulated outputs, then after passing through the unit U the probability that any one of the resulting inputs of the current stage is stimulated is k/N.

To illustrate the difference that this modelling error can cause to the behaviour of the system, consider the case when k outputs from the previous stage are stimulated for some $0 < k < N$. In the approach used by [15], the probability, in the current stage after passing through the unit U, of all inputs being stimulated is $(k/N)^N$, and of no inputs being stimulated is $((N-k)/N)^N$. On the other hand, since there are k stimulated outputs to begin with, if we suppose that the unit U does in fact perform a random permutation, then there will be k of the inputs stimulated, and hence the probability of either all or none of inputs are stimulated is 0.

As our analysis will demonstrate, these two approaches to modelling the unit U can lead to different conclusions about the reliability of the system under study. We will also illustrate, however, that, as the bundle size increases, the results obtained by these modelling approaches converge. In fact, if the bundle sizes were infinite, then the models of U would be equivalent.

Note that, unlike in the case when the unit U is modelled as a random choice with replacement, when (correctly) modelling U as a random permutation the inputs of each of the NAND gates in a stage are dependent on one another; for example, if one NAND has a stimulated input, then the probability of another having the same input stimulated decreases. Therefore, in this scenario, it is not as straightforward to calculate the reliability of a NAND multiplexing unit by means of analytical techniques as, for example, in [15]. However, as far as a probabilistic model checker is concerned, there is no difference between the two approaches; the only requirement is that the user correctly specifies the behaviour of the unit U.

We now explain the main steps in the construction of our PRISM model of a NAND multiplexing system. The basic components of PRISM's input language are *modules* and *variables*. A system is described as the parallel composition of a number of interacting modules. Each module contains a number of variables which represent its state. Its behaviour is given by a set of guarded commands of the form:

$$[]\ \texttt{<guard>} \rightarrow \texttt{<command>};$$

The guard is a predicate over the variables of the system and the command describes a transition which the module can make if the guard is true (using primed variables to denote the next values of variables). If a transition is probabilistic, then the command is specified as:

$$\texttt{<prob>} : \texttt{<update>} + \cdots + \texttt{<prob>} : \texttt{<update>}$$

See Figure 2 for examples of this notation.

The first approach was to directly model the system as given in Figure 1: for each stage construct a PRISM module for each

```
const int N=20; // number of inputs in each bundle
const int M=3; // number of restorative stages equals (M−1)/2
const double p_err=0.01; // probability gate has von Neumann error
const double p_in=0.9; // probability an input is stimulated

module multiplex_system

    u : [1..M] init 1; // current stage (initially 1 - start with first stage)
    c : [0..N] init N; // counter: number of gates to perform in current stage (initially N - no gates performed yet)
    s : [0..3] init 0; // local state (initially 0 - ready to choose inputs to first stage)
    n_x : [0..N]; // number of stimulated X inputs (value not instantiated initially)
    n_y : [0..N]; // number of stimulated Y inputs (value not instantiated initially)
    x : [0..1]; // value of current X input (value not instantiated initially)
    y : [0..1]; // value of current Y input (value not instantiated initially)
    z : [0..N] init 0; // number of stimulated outputs (initially 0 - no ouputs determined)

    // move to next NAND gate of current stage
    [] (s=0) ∧ (c>0) → (s′=1);
    // move onto next stage (copy output to inputs, update the stage and reset other variables)
    [] (s=0) ∧ (c=0) ∧ (u<M) → (s′=1) ∧ (n_x′=z) ∧ (n_y′=z) ∧ (z′=0) ∧ (u′=u+1) ∧ (c′=N);
    // initial choice of x and y: random choice
    [] (s=1) ∧ (u=1) → p_in : (x′=1) ∧ (s′=2) + (1−p_in) : (x′=0) ∧ (s′=2);
    [] (s=2) ∧ (u=1) → p_in : (y′=1) ∧ (s′=3) + (1−p_in) : (y′=0) ∧ (s′=3);
    // select x performed by U: random permutation (randomly pick inputs from those available)
    [] (s=1) ∧ (u>1) → n_x/c : (x′=1) ∧ (s′=2) ∧ (n_x′=n_x−1) // select stimulated input
                     + (c−n_x)/c : (x′=0) ∧ (s′=2); // select non-stimulated input
    // select y performed by U: random permutation (randomly pick inputs from those available)
    [] (s=2) ∧ (u>1) → n_y/c : (y′=1) ∧ (s′=3) ∧ (n_y′=n_y−1) // select stimulated input
                     + (c−n_y)/c : (y′=0) ∧ (s′=3); // select non-stimulated input
    // NAND gate (update number of simulated outputs)
    [] (s=3) → (1−p_err) : (z′=z+!(x ∧ y)) ∧ (s′=0) ∧ (c′=c−1) // gate behaves correct
                + p_err : (z′=z+(x∧y)) ∧ (s′=0) ∧ (c′=c−1); // gate suffers von Neumann error

endmodule
```

Fig. 2. PRISM description of a NAND multiplexing unit

of the N NAND gates in the stage, and then combine these modules through synchronous parallel composition. However, this leads to the well known state space explosion problem, where the size of the probabilistic model constructed grows to an unfeasible level. For example, in the case when the I/O bundle size equals 20, modelling the executive stage of the NAND multiplexing unit required more than 10^{14} states.

An important observation, which allowed us to overcome this problem, was that the actual value of each input and output is not important: instead one need only store the total number of stimulated (and non-stimulated) inputs and outputs. This observation is a result of the following two facts:

1) Since each unit U performs a random permutation, the output of a unit U depends only on the number of stimulated (and non-stimulated) inputs in the bundle which U takes as input, and not on the actual values of each input in the bundle.
2) Reliability concerns only the number of stimulated (and non-stimulated) outputs in the final bundle.

Taking this approach, we replace, in each stage, the set of N NAND gates working in *parallel* with N NAND gates working in *sequence* and keep track of the number of stimulated outputs generated by these gates. Furthermore, one can apply the same methodology to the stages of the system, i.e. reuse the same module for each stage after recording the number of stimulated outputs from the previous stage. This allows us to fold space into time, or in other words reuse the same gate/stage over time, rather than modelling explicit redundancy over space. Note that taking this approach does not influence the performance of the system since each NAND gate works independently, and the probability of each NAND gate failing is also independent.

Following these observations, the PRISM description of the NAND multiplexing system, for the case when the bundle size equals 20, is given in Figure 2. In this model we have assumed that the inputs X and Y are identically distributed (each having probability 0.9 of being stimulated), and the NAND gates suffer from a von Neumann fault (inverted output) with probability 0.01. We use the variable u to record the current stage and the variable c to keep track of the number of gates that still need to be performed in the current stage. The variable s represents the current step in the process of completing a stage. The actions performed in each step are detailed in the comments (prefixed "//") included in Figure 2. For example, consider the second guarded command in the model description. This command corresponds to the case where all the gates have been completed in the current stage ($c=0$) but there remain stages to perform ($u<M$). In this command, we proceed to the next stage ($u'=u+1$), reset the

number of gates that need to be performed to N ($c'{=}N$) and move onto the next step (i.e. the variable s changes from 0 to 1). The remaining updates correspond to the fact that the outputs of the "old" stage become the inputs of the "new" stage and the outputs of the "new" stage are as yet uncomputed.

This model is a DTMC and has 78,311 states. In the case when the bundle size is 40 (the constant N is set equal to 40), the number of states equals 1,004,821, and when the bundle size is 60 the model has 4,717,531 states. Note that, in the executive stage, a random permutation of the inputs cannot be performed as we only know the probability of each individual input being stimulated. However, for a given a distribution over the initial inputs, one can easily modify the PRISM model so that the system performs a random permutation of the initial inputs. To change the number of restorative stages, bundle size, input probabilities or probability of the NAND gates failing requires only modification of the constants given at the start of the description. Furthermore, since PRISM can also represent nondeterministic behaviour, one can set upper and lower bounds on the probability of gate failure and then obtain (best and worst case) reliability characteristics for the system under these bounds. Lastly we note that, to model the units U performing a random permutation with replacement (as in [15]), the only modifications that need to be made are to the probabilities with which the variables x and y are set (when the local state s equals 1 and 2 respectively).

IV. Experimental Results

In this section we study the performance of NAND multiplexing systems when the I/O bundles are of size 20, 40 and 60. In all the experiments, we assume that the inputs X and Y are identical (this is often true in circuits containing similar devices) and that initially 90% of the inputs are stimulated (correct). We suppose that the gate failure in each NAND is a von Neumann fault, i.e. when a gate fails, the value of its output is inverted.

The properties we consider are the probability of there being k stimulated outputs (which, in terms of the PRISM model presented in Figure 2, corresponds to verifying the PCTL formula $\mathcal{P}_{=?}[\Diamond\ (z{=}k \wedge u{=}M \wedge c{=}N]$), for $k = 0, \ldots, N$ where N is the system's I/O bundle size. By performing this analysis we have in fact computed the output distribution of the system, and hence any measure of reliability can be calculated from these results. Note that PRISM can be used directly for computing these measures of reliability, for example, "the probability of errors being less than than K%" and "the expected number of incorrect outputs of the system".

All experiments were run on a PC running Linux, with a 1400 MHz processor and 512 MB of RAM, using PRISM's hybrid engine [23]. In the case where the bundle size equals 20 (number of states: 78,311 states), PRISM requires 1.37 seconds to construct the model and 3.29 seconds to calculate the probability of there being 0 stimulated outputs. When the bundle size is 40 and 60 (number of states: 1,004,821 states and 4,717,531 respectively), model construction requires 5.42 and 11.7 seconds, while calculating the probability of there being 0 stimulated outputs requires 28.4 and 48.7 seconds respectively. Further details relating to the construction and verification statistics are available from the PRISM web page [20].

Our analysis of the reliability of the NAND multiplexing system using probabilistic model checking concentrates on the effects of the failure probabilities of the NAND gates and of the number of restorative stages. Recall that, to change either of these in the PRISM language description of the system (see Figure 2), one needs only to change the parameter p_{err} or the parameter M. The results we present show:

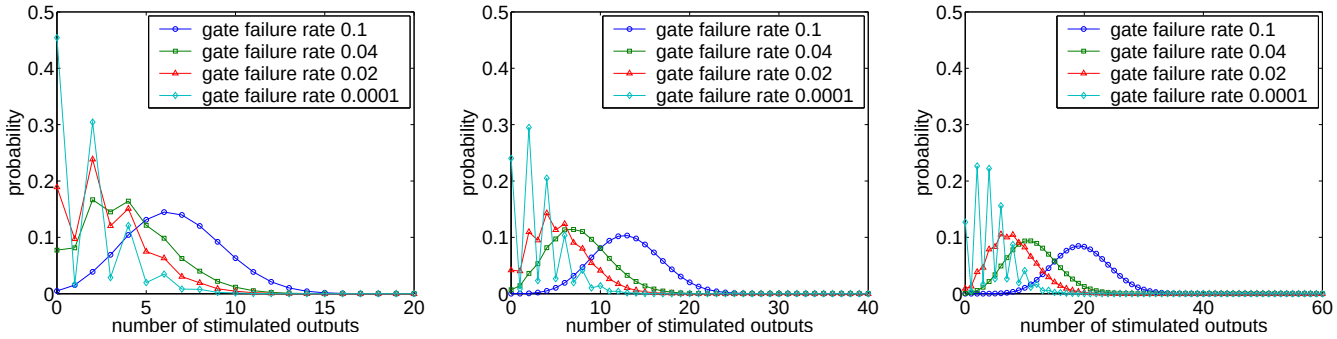
- the shape of the output distribution as the probability of gate failure varies (Figure 3);
- for different probabilities of gate failure, the resulting change in shape of the output distribution when additional restorative stages are added (Figure 4);
- an analysis of reliability, in terms of the probability that at most 10% of the outputs are incorrect, as the probability of gate failure varies (Figure 5);
- how, in the case when the probability of gate failure is very small, the reliability can be improved by increasing the number of restorative stages (Figure 5);
- by comparing the probability that at most 10% of the outputs are incorrect and the expected percentage of incorrect outputs for different numbers of restorative stages, the maximum probability of gate failure allowed for the system to function reliably (Figures 6 and 7).

Where appropriate, we also compare these results with those obtained when the random permutation performed by the unit U is replaced by a random choice with replacement as used by [15]. The results corresponding to this case are referenced 'UR'.

A. Basic System Reliability

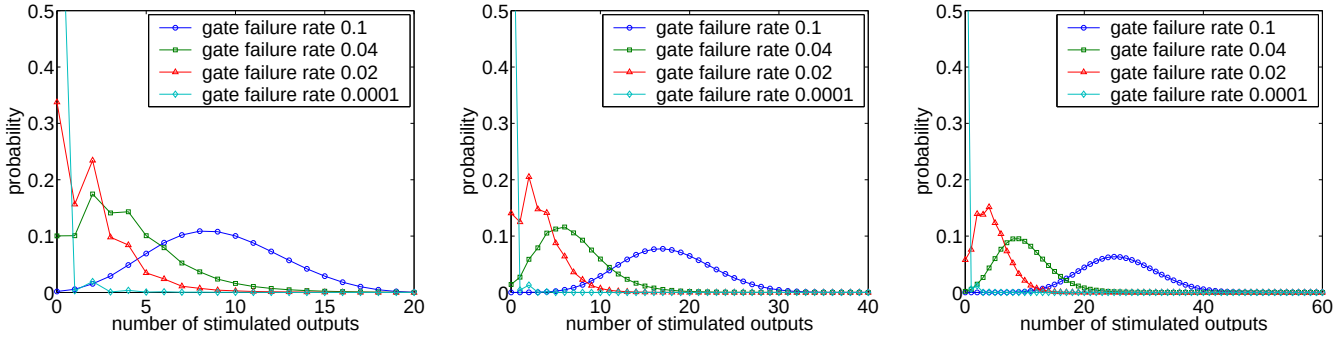
Initially, we consider a basic version of the NAND multiplexing system, as shown in Figure 1 (i.e. with a single restorative stage), where the I/O bundle size equals 20, 40 and 60. We first investigate the effect that changing the failure probabilities of the NAND gates has on the reliability of the system. In Figure 3 we present the output distribution of the system in the cases when the probability of gate failure equals 0.1, 0.04, 0.02 and 0.0001. Note that the system is supposed to model a correctly functioning NAND gate and we assume that the inputs are correct when stimulated. Hence, the less outputs that are stimulated, the greater the reliability of the system.

As can be seen in Figure 3, when the probability of gate failure is 0.0001 (one sees a similar pattern whenever the probability of gate failure is very small or even 0) there is a sharp oscillation in the distribution, with the probabilities for even numbers of stimulated outputs being higher. This phenomenon is due to the random permutation performed by the second unit U of the restorative stage and the fact that, because the probability of gate failure is very low, the probability of any input to this second unit being non-stimulated is very low. More precisely, supposing that k (where k is small) inputs to the second unit of the restorative stage are non-stimulated, then, when this unit performs the random permutation, there is a very high probability that no non-stimulated inputs are



(a) I/O bundle size equals 20 (b) I/O bundle size equals 40 (c) I/O bundle size equals 60

Fig. 3. Output distribution of NAND multiplexing unit with 1 restorative stage under different gate failure rates and I/O bundle sizes



(a) I/O bundle size equals 20 (b) I/O bundle size equals 40 (c) I/O bundle size equals 60

Fig. 4. Output distributions of NAND multiplexing unit with 4 restorative stages under different gate failure rates and I/O bundle sizes

paired together (because k is small), and hence there is a very high probability that there are $2{\cdot}k$ stimulated outputs (since the probability of any gate functioning incorrectly is very small).

Using these output distributions, in Figure 5 (1 restorative stage), we have plotted the probability that less than 10% of the outputs are incorrect against the probability of gate failure. We also plot the same results for the case where the behaviour of the unit U is replaced with a random choice with replacement (denoted 'UR').

As expected, the output distributions given in Figure 3 and the results presented in Figure 5 show that, as the probability of gate failure decreases, the reliability of the multiplexing system increases, i.e. the chance of the system returning incorrect results diminishes. Furthermore, Figure 5 demonstrates that increasing the bundle size leads to a decrease in the probability of error, i.e. an increase in the reliability of the system, and that the rate of increase decreases as the bundle size increases (compare the difference between the results for bundle sizes of 20 and 40 with those for sizes of 40 and 60).

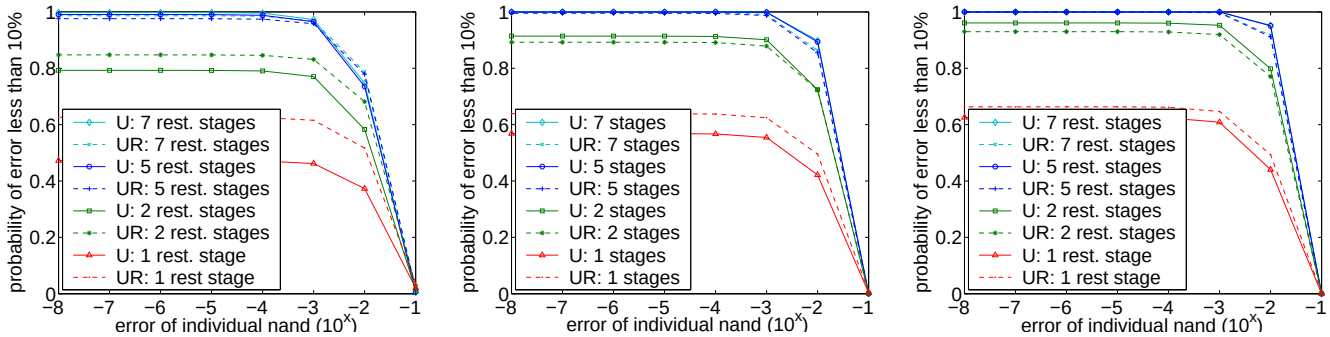
Considering the results given in Figure 5 (1 restorative stage) for the case when the behaviour of the unit U is replaced with a random choice with replacement (denoted 'UR'), we see that this leads to an over-approximation of the reliability of the multiplexing system: the chance of correct outputs is higher than when the unit U is modelled correctly. As mentioned in Section III-A, the difference between the results obtained with the two approaches decreases as the bundle size increases. Note that, as our later results will demonstrate, modelling U in this way does not always result in upper bounds on the reliability of the system.

B. Adding Restorative Stages

Next, we investigate the change in reliability of a NAND multiplexing system as the number of multiplexing stages increases, i.e. when additional restorative units are added to the system. In Figure 4 we present the output distribution of the system with 4 restorative stages. The gate failure probabilities are as in Figure 3. To improve readability, the y axes in these graphs have been truncated, which has removed the probability of 0 outputs being stimulated when the gate failure rate is 0.0001. This value is approximately 0.969 when the bundle size equals 20, 0.981 when the bundle size is 40 and 0.981 when the bundle size is 60.

Comparing these output distributions with those presented in Figure 3, we see that, when the NAND gate failure probability is sufficiently small (e.g. 0.0001), adding additional stages results in a much more reliable system (the probability of any outputs being stimulated is very small). On the other hand, in the cases when the probability of gate failure is sufficiently large, adding additional stages does not increase reliability and, in fact, can actually decrease the reliability of the system (compare the distributions when the failure probability equals 0.1 for each bundle size).

1) Small Probabilities of Gate Failure: To emphasise the first observation in the previous paragraph, in Figure 5, which shows the probability that at most 10% of the outputs of the



(a) I/O bundle size equals 20 (b) I/O bundle size equals 40 (c) I/O bundle size equals 60

Fig. 5. Probability that at most 10% of the outputs of the overall system are incorrect for different I/O bundle sizes

overall system are incorrect (stimulated) against the failure probability of the gates, we have also plotted results as the number of restorative stages varies between 2 and 7.

These results again demonstrate that, for small probabilities of gate failure, increasing the number of stages can greatly enhance the reliability of the system. However, the results also show that the rate of increase in reliability decreases as more restorative stages are added to the system. Moreover, there is a limit to the reliability which can be gained by adding additional stages: compare, for example the plots presented in Figure 5 when the number of restorative stages equals 5 and 7. We should also mention that this result corresponds to the observation made in [15] that, as the number of stages increases, the output distribution of the system will eventually become stable and independent of the number of stages.

In Figure 5, we have also included the statistics obtained when the unit U performs a random choice with replacement instead of a random permutation. In this case, unlike in the case of 1 restorative stage (discussed previously), the approach can now give either an over-approximation or an under-approximation of reliability.

2) Large Probabilities of Gate Failure: We now consider in more detail the case when the probability of gate failure of the NAND gates becomes too large for the multiplexing system to function reliably. In Figure 6 we have plotted the probability that system error is less than 10% against the number of restorative stages, for the cases when the failure rate of the NAND gates is between 0.04 and 0.01. In Figure 7, we have plotted the expected percentage of incorrect inputs for the same configurations.

As can be seen from the results, especially in Figure 7(a), when the bundle size equals 20 and the probability of gate failure equals 0.04, even increasing the number of restorative stages cannot make the computation reliable. In fact, in this case, if one keeps increasing the number of stages, the system's reliability will actually start to decrease. This anomalous behaviour can be understood as follows: when the failure rate is 0.04 (or higher), each restorative stage is sufficiently affected by the probability of gate failure as to actually increase the error, and hence increasing the number of stages in this case makes the system more unreliable.

From these results, we therefore conclude that, in the case of a bundle size equal to 20, if the gate failure probability of the gates is greater than or equal to 0.04, then the system cannot be made reliable. On the other hand, in the case where the failure probability is 0.01, for certain criteria of reliability, the results demonstrate that the system can be made reliable once a sufficient number of restorative stages have been added.

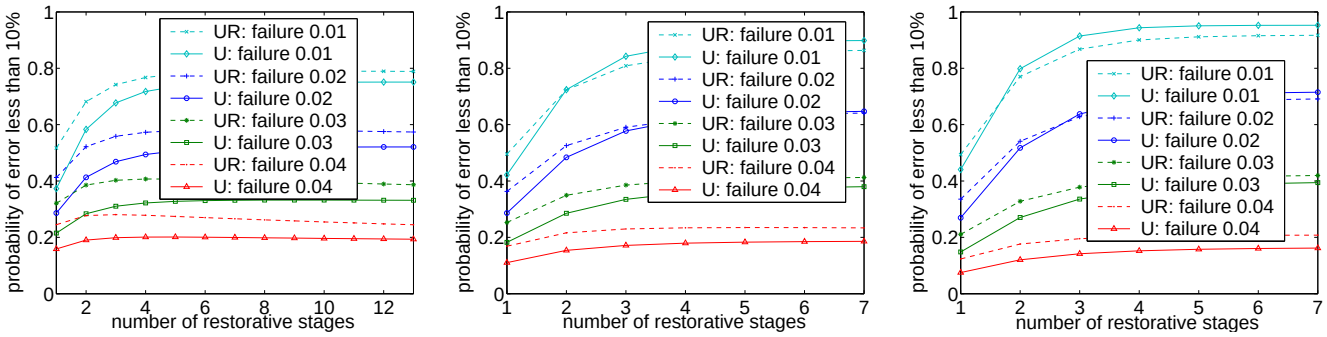
When the bundle size equals 40 or 60, the results presented in Figures 6(b–c) and 7(b–c) show that, if the gate failure probability is 0.04, then adding even large numbers of restorative stages has little effect on the reliability. However, when the gate failure probability equals 0.01 (and for certain criteria, when it equals 0.02), the system can be considered as reliable once a sufficient number of restorative stages have been added.

It is important to note that there is a difference between the bounds on the probability of gate failure required here for reliable computation and the theoretical bounds presented in the literature. This difference is to be expected: in this paper we evaluate the performance of the system under a fixed configuration (bundle size and number of restorative stages), whereas the bounds presented in the literature correspond to the scenario where the bundle size or number of restorative stages can be increased arbitrarily in order to achieve a reliable system.

In Figures 6 and 7, statistics for the case where when the unit U performs a random choice with replacement (denoted 'UR') are again included. These results show that using random choice with replacement can lead to very different results. For example, in Figure 7(a), for a gate failure probability equal to 0.03, when the unit U is modelled by a random choice with replacement, adding additional restorative stages decreases reliability, whereas if U is modelled (correctly) as a random permutation, this actually increases reliability. Furthermore, if we consider the results presented in Figure 6(b–c), when the probability of gate failure equals 0.01 or 0.02 and the number of stages is small, modelling U as a random choice with replacement leads one to assume that the system is more reliable than it actually is. However, as the number of restorative stages increases, the converse holds: one would believe the system to be less reliable than it actually is.

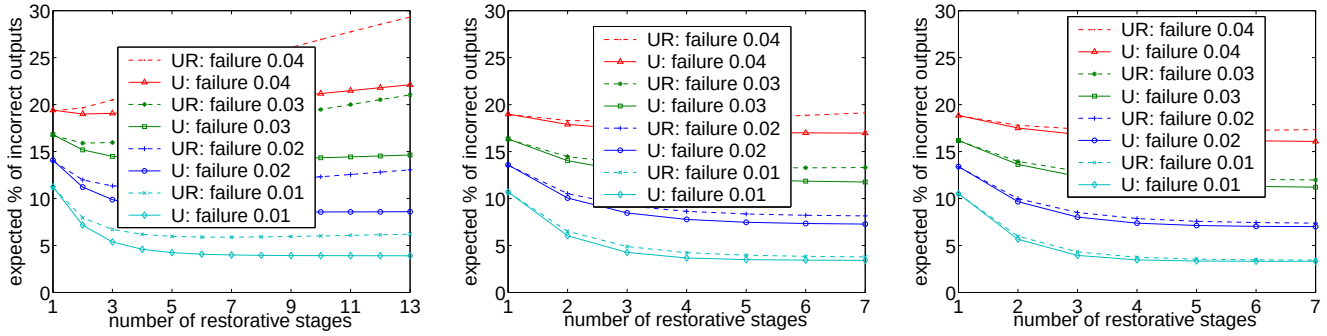
V. Conclusion and Future Work

In this work, we have demonstrated how probabilistic model checking can be used for an evaluation of the redundancy and reliability trade-off for defect-tolerant systems. In particular,



(a) I/O bundle size equals 20 (b) I/O bundle size equals 40 (c) I/O bundle size equals 60

Fig. 6. Probability that at most 10% of the outputs of the overall system are incorrect for different I/O bundle sizes (large probability of failure)



(a) I/O bundle size equals 20 (b) I/O bundle size equals 40 (c) I/O bundle size equals 60

Fig. 7. Expected percentage of incorrect outputs of the overall system for different I/O bundle sizes (large probability of failure)

we have shown how, for a given probability of gate failure (or bound on the probability of gate failure), probabilistic model checking can find the minimum level of redundancy (I/O bundle size and, in the case of multiplexing units, the number of restorative stages) which enables reliable computation.

We have reported on the results obtained for a NAND multiplexing system and investigated the performance of the system as the error rate of the individual NAND gates and the number of stages varies. The first step in this approach involved constructing a system model in PRISM's model description language, and we described an approach used to allow for the analysis of large configurations. In the analysis using PRISM, we were able to compute the exact output distribution of the system, and hence construct a complete picture of the reliability of the system under study for a range of bundle sizes, restorative stages and gate failure rates. Note that the results included here are only a representative selection to demonstrate the applicability of this approach.

We chose to analyse von Neumann's NAND multiplexing approach since it is a canonical example used in the literature, and it therefore enabled us to compare the techniques and results with those of others. One can argue that the amount of redundancy in the NAND multiplexing technique would need to be extremely high to be of use for realistic designs. Nonetheless, this remains a good example for the demonstration of our approach. Furthermore, our methodology is not limited to this particular redundancy architecture, but is equally applicable to alternative fault-tolerant architectures.

In conclusion, this paper shows how the probabilistic model checking framework offers a complementary approach to the theoretical results present in the literature. More precisely, our analysis technique based on probabilistic model checking allows us to obtain sharp bounds and study probabilistic anomalies for a fixed architecture that is relevant in practise, as opposed to establishing general bounds which are independent of the configuration.

REFERENCES

[1] G. Norman, D. Parker, M. Kwiatkowska, S. Shukla, and R. Gupta, "Formal analysis and validation of continuous time Markov chain based system level power management strategies," in *Proc. 7th Annual IEEE International Workshop on High Level Design Validation and Test (HLDVT'02)*, W. Rosenstiel, Ed. IEEE Computer Society Press, 2002, pp. 45–50.

[2] M. Kwiatkowska, G. Norman, and D. Parker, "Controller dependability analysis by probabilistic model checking," in *Proc. 11th IFAC Symposium on Information Control Problems in Manufacturing (INCOM'04)*, 2004.

[3] M. Kwiatkowska, G. Norman, and J. Sproston, "Probabilistic model checking of deadline properties in the IEEE 1394 FireWire root contention protocol," *Formal Aspects of Computing*, vol. 14, pp. 295–318, 2003.

[4] M. Kwiatkowska, G. Norman, J. Sproston, and F. Wang, "Symbolic model checking for probabilistic timed automata," in *Joint Conference on Formal Modelling and Analysis of Timed Systems (FORMATS) and Formal Techniques in Real-Time and Fault Tolerant Systems (FTRTFT)*, ser. Lecture Notes in Computer Science, Y. Lakhnech and S. Yovine, Eds., vol. 3253. Springer Verlag, 2004.

[5] M. Kwiatkowska, G. Norman, D.Parker, and J. Sproston, "Performance analysis of probabilistic timed automata using digital clocks," in *Proc. Formal Modeling and Analysis of Timed Systems (FORMATS'03)*, ser. LNCS, K. Larsen and P. Niebert, Eds., vol. 2791. Springer-Verlag, 2003.

[6] M. Kwiatkowska, G. Norman, and J. Sproston, "Probabilistic model checking of the IEEE 802.11 wireless local area network protocol," in *Proc. 2nd Joint International Workshop on Process Algebra and Probabilistic Methods, Performance Modeling and Verification (PAPM/PROBMIV'02)*, ser. LNCS, H. Hermanns and R. Segala, Eds., vol. 2399. Springer, 2002, pp. 169–187.

[7] M. Duflot, M. Kwiatkowska, G. Norman, and D. Parker, "A formal analysis of bluetooth device discovery," in *Proc. 1st International Symposium on Leveraging Applications of Formal Methods (ISOLA'04)*, 2004, to appear.

[8] V. Shmatikov, "Probabilistic model checking of an anonymity system," *Journal of Computer Security*, vol. 12, no. 3/4, pp. 355–377, 2004.

[9] M. Kwiatkowska and G. Norman, "Verifying randomized Byzantine agreement," in *Proc. Formal Techniques for Networked and Distributed Systems (FORTE'02)*, ser. LNCS, D. Peled and M. Vardi, Eds., vol. 2529. Springer, 2002, pp. 194–209.

[10] M. Kwiatkowska, G. Norman, and D. Parker, "PRISM 2.0: A tool for probabilistic model checking," in *Proc. 1st International Conference on Quantitative Evaluation of Systems (QEST'04)*. IEEE Computer Society Press, 2004, pp. 322–323.

[11] J. von Neumann, "Probabilistic logics and synthesis of reliable organisms from unreliable components," *Automata Studies*, pp. 43–98, 1956.

[12] J. Heath, G. S. P. Kuekes, and R. Williams, "A defect tolerant computer architecture: Opportunities for nanotechnology," *Science*, vol. 80, pp. 1716–1721, 1998.

[13] D. Mange, M. Sipper, A. Stauffer, and G. Tempesti, "Towards robust integrated circuits: The embryonics approach," *Proc. IEEE*, vol. 88, no. 4, pp. 516–541, 2000.

[14] N. Pippenger, "Reliable computation by formulas in the presence of noise," *IEEE Transactions on Information Theory*, vol. 34, no. 2, pp. 194–197, 1988.

[15] J. Han and P. Jonker, "A system architecture solution for unreliable nanoelectronic devices," *IEEE Transactions on Nanotechnology*, vol. 1, pp. 201–208, 2002.

[16] W. Evans and N. Pippenger, "On the maximum tolerable noise for reliable computation by formulas," *IEEE Transactions on Information Theory*, vol. 44, no. 3, pp. 1299–1305, 1998.

[17] H. Hansson and B. Jonsson, "A logic for reasoning about time and probability," *Formal Aspects of Computing*, vol. 6, no. 5, pp. 512–535, 1994.

[18] A. Aziz, K. Sanwal, V. Singhal, and R. Brayton, "Verifying continuous time Markov chains," in *Proc. 8th International Conference on Computer Aided Verification (CAV'96)*, ser. Lecture Notes in Computer Science, R. Alur and T. Henzinger, Eds., vol. 1102. Springer-Verlag, 1996, pp. 269–276.

[19] C. Baier, J.-P. Katoen, and H. Hermanns, "Approximate symbolic model checking of continuous-time Markov chains," in *Proc. 10th International Conference on Concurrency Theory (CONCUR'99)*, ser. Lecture Notes in Computer Science, J. Baeten and S. Mauw, Eds., vol. 1664. Springer-Verlag, 1999, pp. 146–161.

[20] "PRISM web page: www.cs.bham.ac.uk/∼dxp/prism."

[21] R. Alur and T. Henzinger, "Reactive modules," *Formal Methods in System Design*, vol. 15, no. 1, pp. 7–48, 1999.

[22] W. J. Stewart, *Introduction to the Numerical Solution of Markov Chains*. Princeton, 1994.

[23] M. Kwiatkowska, G. Norman, and D. Parker, "Probabilistic symbolic model checking with PRISM: A hybrid approach," *International Journal on Software Tools for Technology Transfer (STTT)*, vol. 6, no. 2, pp. 128–142, 2004.

[24] D. Parker, "Implementation of symbolic model checking for probabilistic systems," Ph.D. dissertation, University of Birmingham, 2002.

[25] J. Rutten, M. Kwiatkowska, G. Norman, and D. Parker, *Mathematical Techniques for Analyzing Concurrent and Probabilistic Systems*, P. Panangaden and F. van Breugel (eds.), ser. CRM Monograph Series. American Mathematical Society, 2004, vol. 23.

[26] K. Nikolic, A. Sadek, and M. Forshaw, "Architectures for reliable computing with unreliable nanodevices," in *Proc. 1st IEEE Conference on Nanotechnology (IEEE-NANO'01)*. IEEE Computer Society Press, 2001, pp. 254–259.