Fault-Tolerant Workstation Cluster

AVACS S3 Benchmark*

[1] Albert-Ludwigs-Universität Freiburg, Fahnenbergplatz, 79085 Freiburg, Germany
[2] Carl von Ossietzky Universität Oldenburg, 26111 Oldenburg, Germany
[3] Universität des Saarlandes, 66041 Saarbrücken, Germany

1 Introduction

In [3] we have briefly introduced a trajectory that leads from labelled transition systems (LTSs) to interactive Markov chains (IMCs) [6], and subsequently transforms these IMCs to continuous-time Markov decision processes (CTMDPs) as, for instance, given in [11]. These CTMDPs are ensured to be analysable by the algorithm proposed in [1]. More detailed descriptions of this trajectory can be found in [9,7], but we like to highlight, that the models that we generate must ensure a property called *uniformity*. Intuitively, the time spent in a state of a uniform model is on average the same for all states.

Here, we apply the trajectory to the fault-tolerant workstation cluster (FTWC) that has been proposed in [5]. The general design of the workstation cluster is shown on the top of Figure 1. It consists of two sub-clusters which are connected via a backbone. There are N workstations in each sub-cluster which are connected together in a star topology with a switch as central node. The switches provide additionally the interface to the backbone.

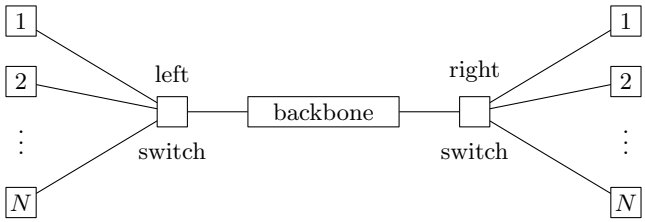
Each of the components in the fault-tolerant workstation cluster can break down (fail) and then needs to be repaired before becoming available again. The mean time to failure and the mean repair time for each component in isolation are depicted on the bottom of Figure 1. They correspond to mean durations of exponential distributions.

There is a single repairunit for the entire cluster, not depicted in Figure 1, which is only capable of repairing one failed component at a time. Essentially, this means that when multiple components are failed, they must be handled in sequence, and there is a decision to be taken which of the failed components the repairunit is assigned to first (or next). This inherent nondeterminism in the specification has been ignored in the original model [5] and in subsequent work, e. g., the FTWC model used and generated by the model checker Prism [8]. The nondeterministic decision has been approximated by using a very fast, but probabilistic decision, encoded via the use of very high rates (of exponential distributions) assigned to the decisive transitions. These high rates are absent in the original problem statement where the repairunit is assigned nondeterministically.

2 Modelling of components

In this section we describe how the singular components of the FTWC are modelled. We start by describing the LTSs representing the untimed behaviour of the workstations, switches, the backbone and the repairunit. Afterwards, we describe the *time constraints*, which are used to weave timed behaviour into the model [9].

* `http://www.avacs.org`



	fail	*repair*
Workstation	500 h	0.5 h
Switch	4000 h	4 h
Backbone	5000 h	8 h

Fig. 1. FTWC with mean fail and repair times.

Labelled transition systems. There are six basic ingredients, namely the workstations (left and right), the switches (left and right), the backbone and the repairunit. Their behaviour is modelled as simple LTSs. In Figure 2 we depict two different LTSs, where the initial states are indicated by the short incoming arrows. The LTS of the repairunit is shown on the left hand side, where we depicted for the sake of readability only two transitions. In fact, there are five parallel transitions emanating the initial state labelled with, e. g., *g_wsL*, and five transitions ending in the initial state labelled with, e. g., *r_wsL*. For example, action *g_wsL* indicates that the repairunit is grabbed by one of the failed left workstations, and *r_wsL* indicates that it is again released by that workstation once it is repaired.

The LTSs for workstations, switches and the backbone are very similar in nature, their general structure is given on the right hand side of Figure 2. Each of them can *fail* and has to *grab* the repairunit afterwards. Only when the repairunit is assigned to that particular component, a *repair* can be performed. Once the component is repaired, the repairunit will be released and can be assigned to another failed component. For a particular component, e. g., the left workstations, the actions *grab* and *repair* in Figure 2 have to be replaced by the according actions, e. g., by *g_wsL* and *r_wsL*, respectively (this is an instance of process algebraic relabelling).

Time constraints. For each of the components the occurrence of *fail* and *repair* is governed by delays. These delays have to be incorporated in the model by composition.

We exemplify the construction of time constraints and the incorporation of these time constraints into the LTS for the (left) workstations as follows: The LTS of a workstation is shown leftmost in Figure 3. In the middle of Figure 3 we have depicted the time constraint for action *fail*,

Fig. 2. Repairunit, workstation, switch and backbone

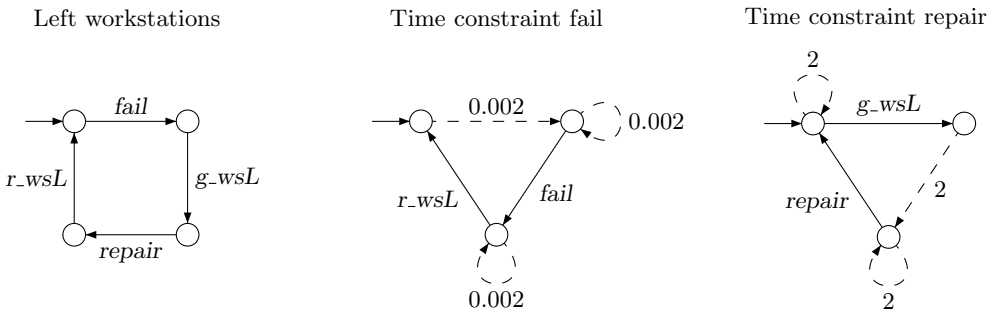


Fig. 3. Left workstation and time constraints

and rightmost in Figure 3 the time constraint of action *repair* is shown. For example, the time constraint for *fail* is started in the initial state of the complete system description. Once the delay, governed by a negative exponential distribution with rate $0.002 = \frac{1}{500}$, is finished, the left workstation fails and awaits to be repaired. When the repairunit is released, the delay is started over again. On the other hand, the delay of action *repair*, governed by a negative exponential distribution with rate $2 = \frac{1}{0.5}$, is started once the repairunit has been grabbed by the failed left workstation.

Technically, the time constraints are generated from a *phase-type* distribution and sets of actions characterising the synchronisation potential that is used to weave the time constraint into the model. We refer the interested reader to [9] where more details can be found.

3 Compositional generation

Components. All of the components, including timed behaviour, are constructed in fairly the same way. We exemplify this for the (left) workstations. Consider again the workstation and the time constraints depicted in Figure 3. The overall description of a left workstation is obtained by parallel composition of the LTS with both time constraints. The synchronisation set is given by {*r_wsL, g_wsL, repair, fail*}. The result of this composition is depicted in Figure 4 (left). In the initial state the workstation is working, and the delay until a failure occurs is started. Once this delay has finished, action *fail* indicates the failure itself, and the workstation tries to grab the repairunit (*g_wsL*). Now, the delay until repair is started, and once it has finished, the workstation is repaired (*repair*) and releases the repairunit (*r_wsL*). However, for the complete system description the actions *fail* and *repair* are not of importance anymore and can thus be hidden. The resulting uniform IMC (uIMC), after a minimisation step, is depicted on the right hand side of Figure 3.

System model. The FTWC is composed out of the components for the workstations, switches, the backbone and the repairunit. First, the workstations, switches and the backbone are constructed as exemplified above and, after hiding actions *fail, repair* they are composed in parallel (with empty synchronisation set). Note, that uniformity is ensured [9].

The resulting uIMCs are then minimised, and composed with the repairunit on the synchronisation alphabet {*g_i, r_i*}, for $i \in \{wsL, wsR, swL, swR, bb\}$, which yields the overall system

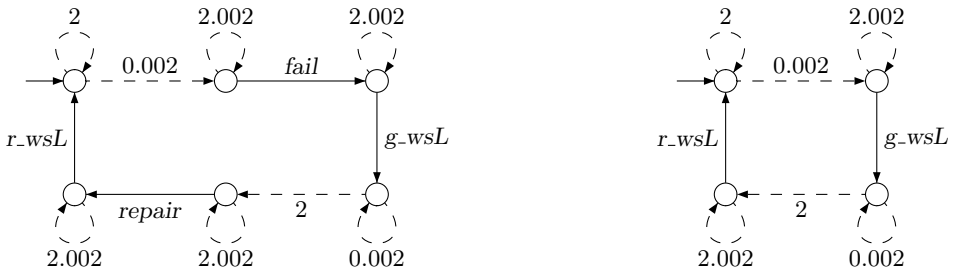


Fig. 4. (Left) workstation with timed behaviour

description of the FTWC (modulo another hiding and minimisation step) as an IMC. From [9] it is ensured that this IMC is uniform, too.

Technicalities. The compositional construction of the FTWC has been carried out using the CADP toolbox [4], and here especially the SVL scripting language [12] and the BCG_MIN tool [2]. For $N = 14$ we obtained an intermediate state space with $5 \cdot 10^6$ states and $6 \cdot 10^7$ transitions. This model then reduces to a uIMC with $6 \cdot 10^4$ states and $5 \cdot 10^5$ transitions. For $N = 16$ we were not able to construct the FTWC in a compositional way. The intermediate state space generation stopped with an incomplete system description that already took 2 GB of hard disk memory.

Therefore we have generated larger models with PRISM [8]. PRISM generates a CTMC of the FTWC example in which the nondeterminism is replaced by uniform probability distributions using a fixed large rate Γ. In order to retain the original nondeterminism, we replaced this particular Γ by an interactive transition and applied afterwards the transformation. We made sure that for $N \leq 14$ equivalent models were obtained via CADP and PRISM – up to uniformity.

4 Results

In this section we report on the results and statistics we obtained when analysing the FTWC. We focus on the performance of the transformation and of the analysis algorithm.

As in [5] we say that our system operates in *premium* quality when there are at least N workstations operational. These workstations have to be connected to each other via operational switches. When the number of operational workstations in one sub-cluster is below N the premium quality can be ensured by an operational backbone under the condition that there are N operational workstations in total. We are interested in the following property: *"What is the worst case probability to hit a state in which premium service is not guaranteed within t time units?"* for which we report results and statistics.

In Table 1 we have collected different statistics of the transformation for different N. Columns 2-6 display the number of states, number of transitions and the memory usage of the CTMDP representation of the FTWC. The depicted numbers are given for the *strictly alternating* IMCs (which comprises precisely what needs to be stored for the corresponding CTMDP), and thus are differentiated in interactive states/transitions and Markov states/transitions, see [9] for details. In column 7 the time for the transformation (from uIMC to uCTMDP) is shown.

In Table 2 we collect statistics about the implementation of the timed reachability algorithm on uniform CTMDPs [1]. For example, column three shows the statistics for a time bound of

N	# States Inter.	Markov	# Transitions Inter.	Markov	Mem	Transf. time (s)
1	110	81	155	324	7.4 KB	0.21
2	274	205	403	920	24 KB	0.27
4	818	621	1235	3000	79 KB	0.30
8	2770	2125	4243	10712	296 KB	0.39
16	10130	7821	15635	40344	1.2 MB	0.83
32	38674	29965	59923	156440	4.6 MB	2.61
64	151058	117261	234515	615960	19 MB	10.19
128	597010	463885	927763	2444312	78 MB	42.51
256	2373650	1845261	3690515	9738264	325 MB	164.28

Table 1. Model sizes, memory usage and runtime of the transformation

100 h. In the second row of column three the computation time for $N = 1$ is shown, in row three we display the number of iterations, and the fourth row shows the corresponding reachability probability. We have generated the statistics by our implementation of the algorithm (which is integrated into the MRMC model checker [10]). Even for large models ($N = 256$) and large time bounds ($t = 50000$ h), the computation is with 27 minutes remarkably fast.

In Figure 5 we depict the reachability probabilities for different N and varying time bounds. As can be observed, for large N the probability that premium service is ensured decreases faster as for small N.

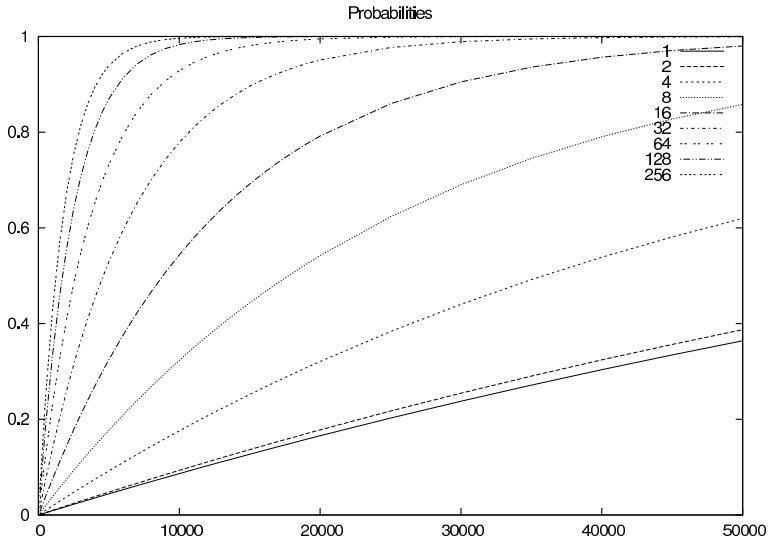


Fig. 5. Reachability probabilities of the FTWC

CTMDP and CTMC. Finally, we show in Figure 6 graphs for $N = 4$ and $N = 128$ in which we compare the worst case probabilities obtained by the CTMDP algorithm with the probabilities obtained from CTMC analysis. As evident in the plot, the CTMC analysis consistently overestimates the true probabilities (computed with MRMC, confirmed with CADP). This is quite

N	Memory	100 h	1000 h	2000 h	5000 h	10000 h	30000 h	50000 h	
		0s	0s	0s	0s	0s	0s	0s	Time
1	744 KB	343	2320	4455	10723	21030	61816	102375	Iter.
		0.00	0.01	0.02	0.04	0.09	0.24	0.36	Prob.
		0s	0s	0s	0s	0s	0s	0s	Time
2	776 KB	343	2325	4463	10743	21071	61938	102577	Iter.
		0.00	0.01	0.02	0.05	0.09	0.25	0.39	Prob.
		0s	0s	0s	0s	0s	0s	1s	Time
4	888 KB	344	2333	4480	10785	21153	62181	102981	Iter.
		0.00	0.02	0.04	0.09	0.18	0.44	0.62	Prob.
		0s	0s	0s	0s	0s	1s	2s	Time
8	1.22 MB	346	2351	4514	10868	21317	62668	103790	Iter.
		0.00	0.04	0.07	0.18	0.32	0.69	0.86	Prob.
		0s	0s	0s	1s	1s	3s	5s	Time
16	2.56 MB	349	2385	4581	11033	21645	63642	105408	Iter.
		0.01	0.08	0.14	0.32	0.54	0.91	0.98	Prob.
		0s	0s	1s	2s	3s	10s	16s	Time
32	7.75 MB	355	2454	4716	11364	22300	65589	108643	Iter.
		0.01	0.14	0.26	0.53	0.78	0.99	1.00	Prob.
		0s	2s	4s	9s	17s	51s	1m 23s	Time
64	27.38 MB	368	2592	4986	12026	23611	69482	115111	Iter.
		0.03	0.23	0.41	0.73	0.93	1.00	1.00	Prob.
		1s	8s	16s	38s	1m 15s	3m 39s	6m 4s	Time
128	105.32 MB	394	2866	5524	13347	26229	77264	128042	Iter.
		0.04	0.33	0.56	0.87	0.98	1.00	1.00	Prob.
		7s	38s	1m 12s	2m 53s	5m 39s	16m 35s	27m 17s	Time
256	415.68 MB	445	3413	6597	15984	31458	92812	153886	Iter.
		0.05	0.43	0.67	0.94	1.00	1.00	1.00	Prob.

Table 2. Timed reachability analysis of the FTWC

remarkable, because the CTMDP algorithm accounts for the worst-case. Nothing worse is possible in the model, and we would thus expect, that this probability will be higher than in a corresponding CTMC model of the system. This overestimation, which indicates a modelling flaw in the CTMC approach, can be explained as follows. When replacing a nondeterministic selection by high rates, certain paths become possible (though with low probability), that in a nondeterministic interpretation would be absent, and thus not contribute to the reachability probability. For a more detailed explanation of this phenomenon we refer the interested reader to [9].

References

1. C. Baier, B. R. Haverkort, H. Hermanns, and J.-P. Katoen. Efficient Computation of Time-Bounded Reachability Probabilities in Uniform Continuous-Time Markov Decision Processes. *TCS: Journal on Theoretical Computer Science*, 345(1):2–26, 2005.
2. BCG_MIN. Project Website, March 2006. http://www.inrialpes.fr/vasy/cadp/man/bcg_min.html.
3. E. Böde, M. Herbstritt, H. Hermanns S. Johr, T. Peikenkamp, R. Pulungan R. Wimmer, and B. Becker. Compositional Performability Evaluation for Statemate. In *QEST: Conference on Quantitative Evaluation of SysTems*, pages 167–176. IEEE Computer Society, 2006.
4. CADP. Project Website, Aug 2006. http://www.inrialpes.fr/vasy/cadp/demos.html.

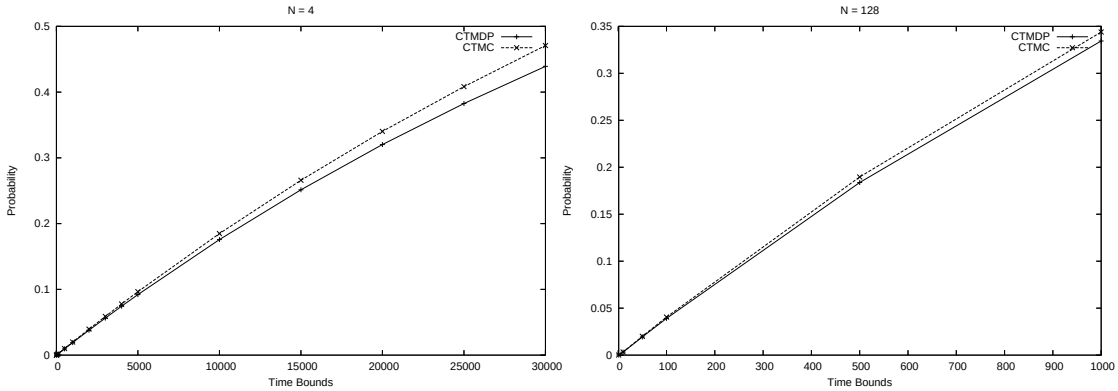


Fig. 6. Comparing CTMC and CTMDP probabilities

5. B. R. Haverkort, H. Hermanns, and J.-P. Katoen. On the Use of Model Checking Techniques for Dependability Evaluation. In *SRDS: Symposium on Reliable Distributed Systems*, pages 228–237. IEEE Computer Society, 2000.
6. H. Hermanns. *Interactive Markov Chains and the Quest for Quantified Quality*, volume 2428 of *LNCS*. Springer, 2002.
7. H. Hermanns and S. Johr. Uniformity by Construction in the Analysis of Nondeterministic Stochastic Systems. In *DSN: International Conference on Dependable Systems and Networks*, pages 718–728, 2007.
8. A. Hinton, M. Kwiatkowska, G. Norman, and D. Parker. PRISM: A Tool for Automatic Verification of Probabilistic Systems. In *TACAS: International Workshop on Tools and Algorithms for the Construction and Analysis of Systems*, LNCS, pages 441–444. Springer, 2006.
9. S. Johr. *Model Checking Compositional Markov Systems*. PhD thesis, Universität des Saarlandes, 2007. submitted.
10. J.-P. Katoen, M. Khattri, and I. S. Zapreev. A Markov reward model checker. In *QEST: Conference on Quantitative Evaluation of SysTems*, pages 243–244. IEEE Computer Society, 2005.
11. M. L. Puterman. *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. Wiley, 1994.
12. SVL. Project Website, March 2006. `http://www.inrialpes.fr/vasy/cadp/man/svl.html`.