Deriving a Good Trade-off Between System Availability and Time Redundancy

Nils Müllner, Abhishek Dhama, Oliver Theel
Carl von Ossietzky Universität Oldenburg
Department of Computer Science
D-26111 Oldenburg, Germany
Email: {nils.muellner|abhishek.dhama|oliver.theel}@informatik.uni-oldenburg.de

Abstract

What to do if at a given time a service dearly required is unavailable? Is it a good strategy to simply invoke the service again (and again)? How many times should one retry in such a situation in order to get a the service delivered with a reasonably high probability but without "losing too much time?" In this paper, we explore the relation between time redundancy that a system can utilize to cope with faults and the increase of system availability. We propose a generalization of instantaneous availability called instantaneous window availability to systematize our approach. We then present two methods for deriving trade-off solutions in terms of average instantaneous window availability, namely Markov model analysis and discrete-time simulation. We apply these methods to two instances of a self-stabilizing system and discuss the outcome.

1. Introduction

Widely used measures to quantify the fault-tolerance property of a system under a given fault model are *reliability*, *instantaneous availability*, and *limiting availability* [1]. Reliability is defined as the "continuity of correct service" of a system [2] and is used for systems where failed components are not repaired. Thus, the system's "lifetime" consists of a single, initial "up" phase followed by an eternal "down" phase. In an "up" phase, the system is expected to correctly deliver a service the system user is interested in, whereas in a "down" phase the system does not. Reliability is the probability that at time $t > 0$ the system is still in its only "up" phase. In contrast, availability analysis is suited for systems with repairs of failed components. Thus, a system toggles between "up" and "down" phases throughout its "lifetime." Availability therefore can be regarded as the "readiness for correct service" [2]. Instantaneous availability is the probability that the system is in an "up" phase at time $t > 0$ under the condition that it was in an "up" phase at time $t = 0$. Note that instantaneous availability is equivalent to reliability in the absence of component repairs. Availability analysis often is concerned with the probability that a system is in an "up" phase after a "sufficiently long time" after system start when looking at it at an *arbitrary* point in time $t > 0$. The precise value of t, in this case, is irrelevant. For this purpose, limiting availability is used. It is defined as the limiting value of instantaneous availability as the observation point t approaches infinity. It is interesting to observe, that for limiting availability, the requirement of the system to start its "lifetime" in an "up" phase can be dropped. In this paper, we investigate the following situation. Imagine, a system user requests a service of a system that has already been running for a long time at a point t in time with $0 \ll t < \infty$. With a certain probability, the system is able to perform the required service at time t. Clearly, this probability corresponds to instantaneous availability at time t and, as t is chosen very large, approaches limiting availability of the system with $t \to \infty$. If the system is available at time t (i.e., it is in an "up" phase) then the service is delivered to the system user. But what should the system user do in case the system is unavailable? A possible strategy might be to simply request the service again, "some time later" hoping that the system has been repaired in the meantime. But how long should the system user wait and how many times should he or she resubmit the service request in case the repeated requests could not be serviced? Obviously, there is a trade-off relation between the "time" the user is willing to spend and the increase in probability that the system delivers a correct service to the system user within the allowed time-frame.

978-0-7695-3737-5/09 $25.00 © 2009 IEEE
DOI 10.1109/UIC-ATC.2009.88

Analyzing this trade-off relation for a given system followed by an exploitation of particularly suited points is expected to be a promising optimization strategy for highly available systems: as we will show, it basically transforms a "certain amount of time redundancy" into a generalized form of instantaneous availability of a system. We call this generalized form *instantaneous window availability*.

In the scope of this paper, we restrict the investigation of the trade-off between time redundancy and instantaneous window availability to discrete-time systems that execute in computational steps, i.e., systems that are only active at discrete points $k = 0, 1, \ldots$ in time. Furthermore, the "time to wait" until a re-try of service invocation is performed, is assumed being exactly one step. For example, if the system was unavailable at step $k = 4711$ then a re-try is triggered at step $k = 4712$. At most w service invocations, including the first invocation, are attempted. This maximal number of overall attempts w is referred to as *window (size)*. Since the particular realization of the trade-off relation is obviously fault model- and system-specific, we show the exploitation by two instances of an example discrete-time system, namely a self-stabilizing implementation of the breadth first search algorithm [3] subject to an arbitrary number of transient faults. The example system's instances comprise three and five processes, respectively. They are analyzed using two methods: 1) by analysis employing the probabilistic model checker PRISM [4] and 2) by event-based simulation using the simulation framework SiSSDA [5]. Although the analysis is able to derive the precise trade-off, state-space explosion may easily render this method inutile for relatively small problem sizes. Contrarily, event-based simulation is able to cope with larger problem sizes often at the cost of less precise results. The paper is structured as follows. In Section 2, we state necessary definitions and our system model followed by a certain way of system embedding we subsequently built upon. In Section 3, we propose the notion of instantaneous window availability and describe our approaches of how to quantify the trade-off using an analytic as well as a simulation method. In Section 4, we present two instances of an example system on which we subsequently apply the two methods. Furthermore, we present the results and interpret them. In Section 5, we give a brief overview over related work. Finally, we conclude with an outlook on future work in Section 6.

2. System Model and System Embedding

We now give general definitions and explain the system model that specifies a system whose availability is to be increased by adding time redundancy. As we will see, in the scope of this paper, we chose a so-called self-stabilizing system for this purpose. This system is subsequently embedded into a conceptional model we use for transforming time redundancy into instantaneous window availability. In particular, using this conceptional model, there is no need to alter the original way, system user and system interact.

2.1. Definitions and System Model

A *distributed system* S consists of a finite set of processes $V = \{v_0, ..., v_{n-1}\}$ along with an underlying communication infrastructure. The communications network of the distributed system is represented by an *undirected* graph $G = (V, E)$ that consists of a set of processes V and a set of edges $E \subseteq V \times V$. Two nodes v_i and v_j are *neighbors* iff $\{(v_i, v_j), (v_j, v_i)\} \subseteq E$.

We assume a *shared memory* model implying that each process has access to two sets of communication registers: *write* and *read* communication registers. Write communication registers can be read and written by the process they belong to whereas read communication registers can only be read. For example, process v_i may write into write communication register $w_{i,j}$. By reading read communication register $r_{j,i}$, process v_j may subsequently access the value written by process v_i. There is assumed to exists one such pair of read and write communication registers per element in E.

The *(local) state* $c(v_i)$ of a process v_i is given by a vector consisting of the values of this process' local variables and write communication registers (read communication register values are not part of the local state) according to some arbitrary but fixed total order of variables and communication registers. The *global state* C of a system S is analogously given by a vector of local states of the constituent processes, i.e. $C := \langle c(v_0), ..., c(v_n)\rangle$.

Each process is specified in terms of a set of *guarded commands* of the form $\langle guard\rangle \rightarrow \langle assignment\rangle$[6]. A guarded command consists of a *guard* and an *assignment statement*. A guard is a boolean expression over the local variables and the read and the write communication registers of a process. A guarded command is said to be *enabled* if the boolean expression in the guard holds true. An assignment statement may involve the particular process' read communication registers

for reading and writing communication registers as well as local variables for reading and writing.

An *execution* $\mathcal{E}$ of a distributed system S is a maximal sequence of global states $\mathcal{E} = \langle C_0, ..., C_{i-1}, C_i, ... \rangle$ such that C_i is reached from C_{i-1} via execution of the assignment statement of one enabled guarded command. In each step, a scheduler chooses one enabled guarded command to execute in a non-deterministic fashion. In other words, we assume *serialized execution semantics* in the scope of this paper [7]. We also assume that the scheduler is *weakly fair* ensuring that a *continuously* enabled guarded command is chosen *eventually* [8].

The *self-stabilization* property of a distributed system S is defined with respect to a predicate $\mathcal{P}$, where $\mathcal{P}$ specifies the set of *legal* states. A distributed system S is self-stabilizing with respect to a predicate $\mathcal{P}$ iff it satisfies the following conditions for all possible execution sequences [9]:

1) *Convergence*: $\exists i : C_i \models \mathcal{P}$, guaranteeing that, irrespective of the starting state, the system reaches the set of legal states within a finite number of steps
2) *Closure*: $C_i \models \mathcal{P} \Rightarrow C_{i+1} \models \mathcal{P}$, implying that the system does not leave the set of legal states voluntarily.

The distributed system is assumed to execute in an environment subject to an arbitrary number of *transient faults* potentially impacting the distributed system's global state. The manifestation of a fault is an *error* in the global state. If the error is such that predicate $\mathcal{P}$ is falsified then this situation constitutes a *failure* of the service provided by S: the service in *unavailable* (or "down" synonymously). Otherwise, if $\mathcal{P}$ holds, the distributed system is said to be *available* (or "up"). Note that – due to the self-stabilizing nature – the self-stabilizing system always tries to "repair failures" in case $\mathcal{P}$ does not hold and new failures do not occur.

In the remainder of the paper, we use the term "distributed system" and "system" synonymously.

2.2. System Embedding

For ease of system usage, we delegate the burden of re-requesting service invocation by the system user to a level beneath the user as shown in Figure 1(a). We call this level and the component implementing it *fault masker*. Thus, the system user simply specifies how long (in terms of service re-invocation rounds and – synonymously – computation steps) he or she is *at most* willing to wait for a successful service deliverance and calls the fault masker. In the figure,

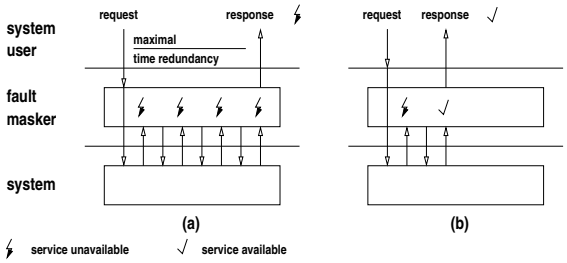


Figure 1. System Embedding

this value is set to four rounds, corresponding to a window size $w = 3$. The fault masker, subsequently, invokes the service and, if the service can be correctly delivered, conveys the result to the system user, who is – in this case – unblocked and can enjoy the service. If the system could not correctly be delivered then the fault masker perfectly detects this mishap and re-invokes the service in the next round. Only if the maximum number of rounds has been exceeded then the fault masker eventually informs the system user of the unavailability of the service, thereby unblocking the system user. This situation is depicted in Figure 1(a), where all four invocations fail. In Figure 1(b), contrarily, the second invocation was successful and lead to a deblocking of the system user.

Due to the masking nature of the fault masker wrt. failed service invocations, one can also regard our approach as a method for increasing the *degree of masking* experienced by the system user. Thus, when regarding the resultant system after the embedding, the degree of masking is traded against availability.

In case of our self-stabilizing example, the fault masker, for detecting the unavailability of the requested service, simply checks whether predicate $\mathcal{P}$ is falsified. Contrarily, if $\mathcal{P}$ holds then the service is available and can be delivered to the system user. If a system other than a self-stabilizing system is used, for example a masking fault-tolerant system such as a data replication system based on majority voting then fault detection might become as easy as checking for a certain "error return value" delivered by the system.

3. Concepts and Methods for Trade-off Quantification

In this section, we propose a generalization of instantaneous availability notions called *instantaneous window availability*. We subsequently motivate the usefulness of instantaneous window availability as fault-tolerance measure for long-running systems. Then, we describe how to evaluate limiting window availability for a given self-stabilizing system, 1) by

analysis and 2) by simulation.

3.1. Instantaneous Window Availability

In order to quantify the probability that the system user of the embedded system depicted in Figure 1 can successfully use the service, we propose a generalization of the instantaneous availability fault-tolerance measure. Formally, instantaneous availability $A(t)$ of a system S is the probability that system S is in an "up" phase at time t under the constraint that it was correct at time $t = 0$ (i.e., $A(0) = 1$). As proposed in [1], for self-stabilizing systems wrt. predicate $\mathcal{P}$ – which are discrete-time systems where execution evolves step by step – instantaneous availability is defined as

$$A(k) := P(\mathcal{P}_k \mid \mathcal{P}_0) \tag{1}$$

with $\mathcal{P}_i$ being that predicate $\mathcal{P}$ holds true at step i. Based on this, limiting availability of a self-stabilizing system is defined as

$$A := \lim_{k \to \infty} A(k) \tag{2}$$

Note that limiting availability is independent of the starting state of the system at $k = 0$. For steps k with $0 << k < \infty$, the system "behaves more and more" independent of the particular starting state.

Definition 1: Given a system S that self-stabilizes wrt. predicate $\mathcal{P}$. Then, the probability $A(k, w)$, $k, w \in \mathcal{N}_0$, with

$$A(k, w) := A(k) + P\left(\overline{\mathcal{P}}_k \cdot \mathcal{P}_{k+1} + \ldots + \overline{\mathcal{P}}_k \cdot \ldots \cdot \overline{\mathcal{P}}_{k+w-1} \cdot \mathcal{P}_{k+w}\right)$$

is called *instantaneous window availability* of the system at step k with *window (size)* w. □

Clearly, instantaneous window availability with window size $w = 0$ is identical to the instantaneous availability of the system at step k. Furthermore, it is obvious that an increase in the window size results in either an unchanged or increased instantaneous window availability.

We are now interested in figuring out a particular window size w for a given system and fault model that leads – on the average – to an instantaneous window availability increase "worth spending" up to $w - 1$ additional service invocation attempts.

3.2. An Analytical Method

The crucial part of the analytical method is the representation of a self-stabilizing system as a *discrete-time Markov chain* (DTMC). A stochastic process is a DTMC if it has the following property: $P(C_k = x_k \mid C_{k-1} = x_{k-1} \cdot C_{k-2} = x_{k-2} \cdot \cdots \cdot C_1 = x_{k-1}) = P(C_k = x_k \mid C_{k-1} = x_{k-1})$, implying that the probability of the transition between the states C_k and C_{k-1} depends only on the state C_{k-1} [10].

The states of a DTMC representing a self-stabilizing system are the same states as the ones of the self-stabilizing system. Thus, there is a one-to-one mapping between the states of a self-stabilizing system and its resulting DTMC. As the resultant model – i.e., DTMC – is a probabilistic system devoid of non-determinism, the transition probabilities must encode the behavior of the scheduler. In the following, we explain how to calculate the transition probabilities between any two states of the system.

The behavior of a non-deterministic fair scheduler is instantiated by assigning a probability to the event that the scheduler chooses a process v_i. We assume that $p_{\mathcal{S}i}$ is the probability that the scheduler chooses the process v_i such that $\sum_{i=0}^{n-1} p_{\mathcal{S}i} = 1$. Choosing a process suffices in the scope of our example system since every process at every step has exactly one enabled guarded command. Since the fault model assumes that transient faults can occur intermittently, the probability that a fault occurs instead of a computation step is $p_{\mathcal{F}}$. A transient fault most likely leads to a state transition. Thus, the probability of the state transition between the states C_i and C_j – due to a fault – is assumed to be $p_{\mathcal{F}_{ij}}$. The transition between any two states, C_m and C_n, may be caused by a computation step or by a transient fault. In case the transition is exclusively due to a transient fault then the probability of such a transition is $p_{\mathcal{F}} p_{\mathcal{F}_{mn}}$. However, if the execution of a guarded command in the state C_m leads to the state C_n then the probability of the transition is the sum of the probability that the process is selected by the scheduler and executes its enabled guarded command, and the probability that a transient fault led to the transition. The probability that process v_i executes its enabled guarded command is $p_{\mathcal{S}i}(1-p_{\mathcal{F}})$. Thus, the probability of such a transition is $p_{\mathcal{S}i}(1 - p_{\mathcal{F}}) + p_{\mathcal{F}} p_{\mathcal{F}_{mn}}$. The transition probabilities for each pair of states can be constructed in the fashion described above to derive the DTMC representing a self-stabilizing system.

A DTMC derived as described above can be analyzed using a probabilistic model checker such as PRISM [4]. Properties such as the "probability to reach the legal set of states in k steps" or the "average number of steps to reach a safe state" can be computed using PRISM.

However, due to the fact that the set of initial states is equal to the set of states itself, probabilistic model checking suffers from state space explosion. This problem can be circumvented for larger systems by employing simulation to derive values of fault

tolerance measures. We next describe a simulation framework that is suitable for this task.

3.3. A Simulation Method

The simulation framework for self-stabilizing distributed algorithms (SiSSDA) [5] is a tool to acquire empirical results. Scenarios consisting of a scheduler, a system definition (i.e., the distributed system's topology), a fault model, and an algorithm to be executed can be specified to measure fault tolerance properties.

SiSSDA starts a given system in an arbitrary state, executes at least a predefined number j of steps ("swing-in time") plus an additional, randomly chosen number i of steps due to a homogeneous distribution over some interval. After executing $i + j$ steps, it is checked whether the current system state satisfies $\mathcal{P}$ or not. If the system does not then simulation continues until $\mathcal{P}$ is eventually satisfied. The number of additional steps w from step $i+j$ to step $i+j+w$ at which $S \models \mathcal{P}$ holds first since step $i + j$, is the experiment's result, we are interested in. After a sufficiently large number of experiments m, SiSSDA prompts the relative frequency of the system requiring $w = 0, 1, \ldots$ (additional) steps to first fulfill $\mathcal{P}$ when starting to ask for it at step $i + j$, i.e., the absolute values are divided by m.

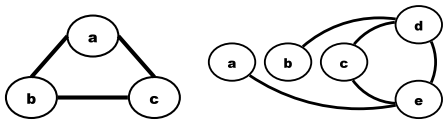
4. Example

After we introduced the system model, the methods of analytic derivation and derivation by simulation, we apply those methods on an example in order to obtain average values of $A(k, w)$ with $k \gg 0$ being chosen from some homogeneously distributed interval. Among others, this will give guidance in answering the question of how large the window size should be in order to obtain an instantaneous window availability of a certain amount on the average.

4.1. System Specification

As an example, we chose the popular distributed self-stabilizing *breadth first search* algorithm (BFS) [3]. The algorithm consists of one sub-algorithm for the root process and one sub-algorithm for all other process. It builds in a self-stabilizing fashion a spanning tree among the process participating in the algorithm. System S_1 consists of three process, a, b, and c, (as shorthands for precesses v_0, v_1, and v_2) such that there is a communication link between all the process as shown in Figure 2(a). System S_2 is built accordingly as shown in Figure 2(b). On these topologies we execute the self-stabilizing distributed BFS algorithm [3].

In three instances, we employ fault probabilities of 0.01, 0.05, and 0.1. We measure the average instantaneous window availability wrt. $k \gg 0$ in some interval. We derive results using both methods for system S_1. For system S_2, unfortunately but not too astonishingly, due to state space explosion, analysis was not feasible. But by showing similarity of results derived through both methods for system S_1, we feel to have established believe in the validity of results derived by simulation for system S_2.



(a) Three Process System (b) Five Process System

Figure 2. Topology of Systems S_1 and S_2

4.2. Analysis Settings

We analyzed the performance of the self-stabilizing BFS spanning tree algorithm on systems comprising of three process. We assumed that the probabilistic scheduler follows uniform probability distribution while selecting the next process to execute. Hence, $p_{\mathcal{S}i}$, the probability of the scheduler selecting a process is $0.\overline{3}$. We also assumed that the transitions caused by transient faults follow a uniform probability distribution over the system states. The probability of state transitions between any two pairs of states due to transient faults in the three process system is 0.000152439. The probabilities of transitions between the pairs of state C_m and C_n, such that C_n is reachable from C_m via a guarded command execution, is given by the expressions $0.\overline{3}(1 - p_{\mathcal{F}}) + 0.000152439$. The DTMC derived in this fashion was model checked with PRISM.

The system was evaluated in two steps. In order to derive the probability distribution over the system states for the long running system, we determined the probability of the system being in a specific state after 100 steps irrespective of the starting state. This probability was determined for all possible system states. In the next step, the probability of reaching the legal state within w steps was determined for a specific initial state and w was varied from 1 to 30. In a fashion similar to the first step, the probability of reaching the legal state within w steps was also determined for all possible system states. The average increase compared to instantaneous availability (i.e., $A(k, w)$) of the three-process system (for each w) was then derived by computing the weighted mean

of the probabilities derived in the second step. The probability distribution over the system states in a long run was used to determine the weighted mean.

4.3. Simulation Settings

For both systems S_1 and S_2 and every single experiment, we chose a swing-in phase of $j = 1000$ steps starting from an arbitrary system state. Then, at step $i+j$ with i being homogeneously distributed over the $[1, 1000]$ interval, a service request was first issued and the particular number w of steps until the first successful service deliverance recorded. The number m of experiments executed for any given fault probability was set to $30,000$.

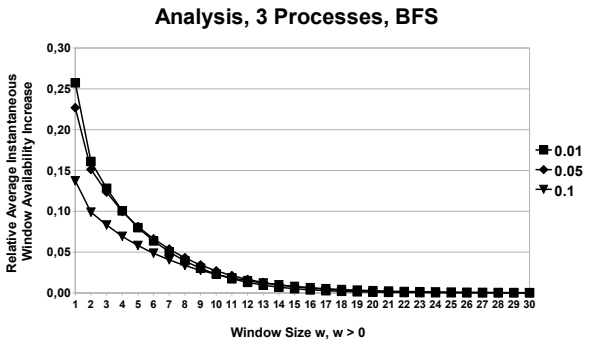
4.4. Results

Please note that the graphs presented in this section show continuous lines. We have connected the discrete points of each step only for a better visibility. It is also worthwhile to mention that smoother graphs for simulation can be obtained by running a larger number of experiments.

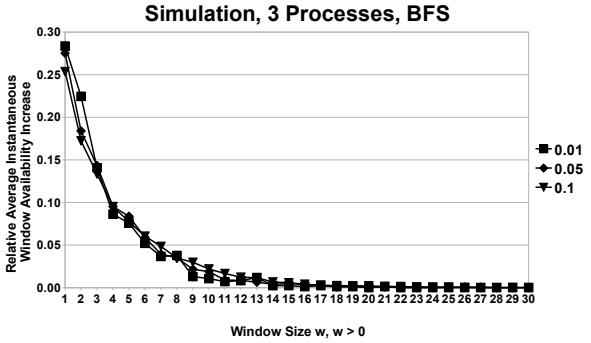
First we present the results from the analysis for the three process system S_1 in Figure 3(a). Then we present the results from the simulation for S_1 in Figure 3(b) and the results of the simulation of the five process system in Figure 3(c). Finally we discuss and interpret the results.

The plots of relative increase in instantaneous window availability with respect to instantaneous window availability at $w = 0$, derived via analysis and simulation, for the three process system demonstrate similar characteristics. One can observe that as the fault probability increases, the plot becomes less steep. This is due to the fact that increased fault probability implies smaller temporal separation between the transient faults. As the temporal separation decreases, the repairing actions of a self-stabilizing system are disrupted more often. Thus, the system requires, on an average, longer time to stabilize and thereby slowing the rate of relative increase of instantaneous window availability. A similar plot derived for the five process system via analysis also shows the dependence of the rate of relative increase of instantaneous window availability over the failure probability.

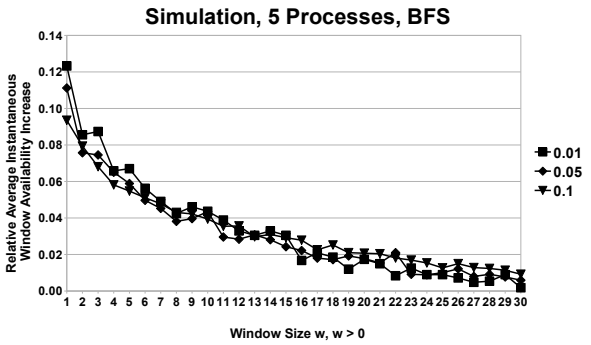
Another interesting observation is the relation between the rate of relative increase of instantaneous window availability over the size of the system. As one can see in Figure 3(c), the rate is evidently lower than that of the three process system. As the system grows larger, the repairing actions and the required information flow takes longer thus the stabilizing time decreases.



(a) Analysis Results of System S_1



(b) Simulation Results of System S_1



(c) Simulation Results of System S_2

Figure 3. Results

These plots can be used to determine the waiting time necessary for the desired related increase in instantaneous window availability. For instance, consider the the plot of the five process system in Figure 3(c). A relative increase of 66.67% instantaneous window availability can be obtained by waiting for 10 steps if the fault probability is 0.01. However, one has to wait for 15 steps to obtain a similar if the fault probability is 0.10.

5. Related Work

The analytic and simulation methods employed in this paper are *per se* described in [1], [5]. In the context of this paper, these methods are used to comparatively quantify time redundancy vs. availability trade-off of a given self-stabilizing example system under a prominent, realistic fault model. Where necessary, the basic methods have been refined to suite the current purpose.

We are not aware of any work that previously explored the cited trade-off. Although, when regarding the "increased degree of masking" introduced by our method as described in Section 2.2 then the following literature is related.

Arora and Kulkarni [11] presented a method to transform a fault-intolerant program into a non-masking fault-tolerant program and subsequently into a masking fault-tolerant program. Although Arora and Kulkarni introduced a method to add masking to a non-masking fault tolerant system they do not analyze the trade-off between system availability, neither vs. time redundancy nor vs. space redundancy.

In [12], Kulkarni and Arora, again, focus on automating the addition of fault tolerance without considering trade-off solutions.

Kulkarni and Ebnenasir [13] focus on automated techniques to enhance fault tolerance of a non-masking fault-tolerant program into a masking fault-tolerant program. Yet, trade-off solutions were not investigated.

6. Conclusion

We presented a new notion of system availability, namely instantaneous window availability, and showed how to use two methods – probabilistic model checking and discrete-time simulation – for quantifying the trade-off between time redundancy and system availability in terms of this new notion. Furthermore, we applied these methods on two instances of a self-stabilizing example system implementing a breadth first search algorithm, and demonstrated the exploitation of the trade-off for the benefit of the system user.

Although we focused on an example self-stabilizing system, the approach is not restricted to self-stabilization. Any system, initially fault-tolerant or not, can be equipped with the embedding technique adopted in order to subsequently exploit the described trade-off.

An interesting research question in this context is whether – given a particular system – a certain level of system availability can be "cheaper" (in terms of some cost metrics) obtained by using additional time or space redundancy or a particular combination of the two.

Acknowledgment

This work was partly supported by the German Research Foundation (DFG) under grants GRK 1076/1 "TrustSoft" and SFB/TR 14/2 "AVACS".

References

[1] A. Dhama, O. E. Theel, and T. Warns, "Reliability and Availability Analysis of Self-Stabilizing Systems," in *SSS '06*, 2006, pp. 244–261.

[2] A. Avizienis, J.-C. Laprie, B. Randell, and V. M. U, "Fundamental Concepts of Dependability," in *Proceedings of the Third Information Survivability Workshop*, 2000.

[3] S. Dolev, *Self-Stabilization*. Cambridge, MA, USA: MIT Press, 2000.

[4] M. Kwiatkowska, G. Norman, and D. Parker, "PRISM: Probabilistic Model Checking for Performance and Reliability Analysis," *ACM SIGMETRICS Performance Evaluation Review*, vol. To Appear, 2009.

[5] N. Müllner, A. Dhama, and O. Theel, "Derivation of Fault Tolerance Measures of Self-Stabilizing Algorithms by Simulation," in *ANSS '08*, Ottawa, Ontario, Canada, 2008, pp. 183–192.

[6] E. W. Dijkstra, "Guarded Commands, Nondeterminacy and Formal Derivation of Programs," *Comm. ACM*, vol. 18, no. 8, pp. 453–457, 1975. [Online]. Available: http://dx.doi.org/10.1145/360933.360975

[7] M. G. Gouda and T. Herman, "Stabilizing Unison," *Inf. Proc. Lt.*, vol. 35, no. 4, pp. 171–175, 1990.

[8] P. C. Attie, N. Francez, and O. Grumberg, "A Distributed Abstract Data Type Implemented by a Probabilistic Communication Scheme," in *SFCS '80*, 1980, pp. 373–379.

[9] M. Schneider, "Self-stabilization," *ACM Comp. Sur.*, vol. 25, no. 1, pp. 45–67, 1993.

[10] P. V. Mieghem, *Performance Analysis of Communications Networks and Systems*. Cambridge University Press, 2006.

[11] A. Arora and S. Kulkarni, "Designing Masking Fault-Tolerance via Nonmasking Fault-Tolerance," *IEEE Trans. Soft. Eng.*, vol. 24, no. 6, pp. 435–450, 1998.

[12] S. Kulkarni and A. Arora, "Automating the Addition of Fault-Tolerance," in *FTRTFT*, London, UK, 2000, pp. 82–93.

[13] S. Kulkarni and A. Ebnenasir, "Enhancing The Fault-Tolerance of Nonmasking Programs," *ICDCS*, vol. 00, p. 441, 2003.