Reliability and Availability Analysis in Practice

Kishor Trivedi

Andrea Bobbio

Kishor Trivedi
Duke University
Electrical and Computer Engineering
Durham, NC, 27708, USA
e-mail: ktrivedi@duke.edu

Andrea Bobbio
Università del Piemonte Orientale
DiSit – Sezione Informatica
15131 Alessandria (Italy)
e-mail: andrea.bobbio@uniupo.it

SUMMARY AND PURPOSE

Reliability and Availability are key attributes of technical systems. Methods of quantifying these attributes are thus essential during all phases of system lifecycle. Data (measurement)-driven methods are suitable for components or subsystems but, for the system as a whole, model-driven methods are more desirable. Simulative solution or analytic-numeric solution of the models are two major alternatives. In this tutorial, we explore model-driven methods with analytic-numeric solution. Non-state-space, state space, hierarchical and fixed-point iterative methods are explored using real-world examples. Challenges faced by such modeling endeavors and potential solutions are described as also one of the software packages used for such modeling exercises.

Kishor Trivedi

Dr. Kishor Trivedi is a Professor of ECE and Computer Science at Duke University and a Life Fellow of IEEE. He is the author of a well-known text entitled, *Probability and Statistics with Reliability, Queuing and Computer Science Applications*. His latest book, co-authored with Andrea Bobbio, Reliability and Availability Engineering, is published by Cambridge University Press in 2017. He has supervised 46 Ph.D. dissertations and has h-index 98 as per google scholar. Recipient of IEEE Computer Society's Technical Achievement Award for research on Software Aging and Rejuvenation, he works closely with industry in carrying out reliability/availability analysis and in the development and dissemination of modeling software packages such as SHARPE and SPNP.

Andrea Bobbio

Dr. Andrea Bobbio is a Professor of Computer Science at Università del Piemonte Orientale in Italy and Senior Member of IEEE. He is co-author of the book, *Reliability and Availability Engineering*, published by Cambridge University Press in 2017. His academic and professional activity has been mainly directed to the area of reliability engineering and system reliability. He contributed to the study of heterogeneous modeling techniques for dependable systems, ranging from combinatorial techniques to Bayesian belief networks, to state-space based techniques, and fluid models. He has visited several important institutions and is the author of 200 papers in international journals, conferences and workshops.

Table of contents

1. INTRODUCTION

Modern life heavily relies on man-made systems that are expected to be reliable. Many high-tech cyber systems are found wanting since their failures are not so uncommon. Such failures and consequent downtimes lead to economic losses, to a loss of reputation and to even loss of lives. To ameliorate the situation many methods are designed to reduce failure occurrences and resultant downtimes. In order to gauge the effectiveness of these improvement methods, scalable and high-fidelity techniques of reliability and availability assessment are needed.

This tutorial discusses the techniques that have been utilized for reliability and availability assessment in practice. Assessment methods can be divided into measurement-driven (or data-driven) vs. model-driven methods (Figure 1). Data-driven methods are suitable for small subsystems while model-driven methods are appropriate for large systems. Using model-driven methods, we can derive the dynamic behavior of a system consisting of many components using first principles (of probability theory) rather than from measurements. In practice, these two approaches are combined together so that subsystem or component behavior is derived using data-driven methods while the system behavior is derived using model-driven methods .

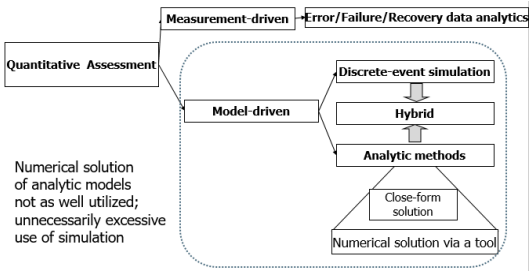

Figure 1. Reliability/Availability assessment methods

This tutorial focuses on model-driven methods. Models can be

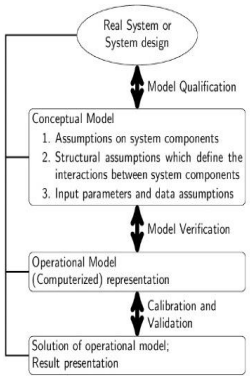

Figure 2. The overall modeling process

solved using discrete-event simulation or using analytic-numeric techniques. Some simple models can be solved analytically in a closed-form while a much larger set of models can be dealt with by a numerical solution of their underlying equations. Distinction between analytic-numeric solution vs. discrete-event simulation-based solution ought to be noted. We believe that simulative solution and analytic-numeric solutions should be judiciously combined in order to solve complex system models. This tutorial is on analytic-numeric methods providing an overview of a recently published book by the authors [1].

Our approach to exposing the methods is example-based. The examples are chosen to be those of real systems that we have ourselves analyzed for some companies. Overall modeling process that could be used is depicted in Figure 2.

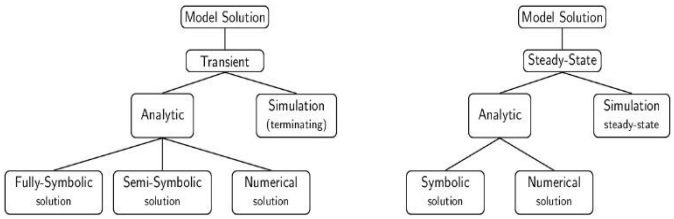

Figure 3. Solution techniques

Model types that we discuss include non-state-space ones such as Reliability Block Diagrams (RBD), Fault Trees (with and without repeated events) and reliability graphs. State space model types, specifically continuous-time Markov chains and Markov reward models are commonly utilized. Multi-level models that judiciously combine state space and non-state-space methods will be seen to have the scalability and fidelity needed for capturing real system behavior. Depending on the application, a model may be solved for its long-term (steady-state) behavior or its time-dependent or transient behavior. Solution types for such models are classified in Figure 3 [1,12]. Software packages that are used in solving the examples of this tutorial are SHARPE [2,3] and SPNP [4,5].

2. NON-STATE-SPACE METHODS

Several traditional methods for the analysis of system reliability and availability can be classified under the umbrella of non-state-space (sometimes called combinatorial) methods:

- Reliability Block Diagrams (RBD);
- Network reliability or Reliability graphs (RelGraph);
- Fault Trees

The simplest paradigm for reliability/availability is the (series-parallel) reliability block diagram (RBD). These are commonly used in computer and communications industry, are easy to use and assuming statistical independence, simple algorithms are available to solve very large RBDs. Reliabilities (availabilities) multiply for blocks in series while un-reliabilities (un-availabilities) multiply for blocks in parallel. Efficient algorithms for *k-out-of-n* blocks are also available, both in the case of statistically identical blocks and non-identical blocks [1]. Besides system reliability at time *t*, system mean time to failure, and system availability (steady-state and instantaneous), importance measures can also be computed so as to point out critical components (bottlenecks) [1].

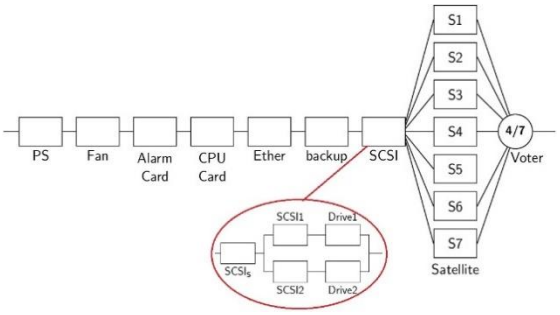

Figure 4. High Availability Platform from Sun Microsystems

High availability requirement in telecommunication systems is usually more stringent than most other sectors of industry. The carrier grade platform from SUN Microsystems requires a "five nines and better" availability. From the availability point of view, the top-level architecture of a typical carrier grade platform was modeled in [6] as a reliability block diagram consisting of series, parallel, and *k-out-of-n* subsystems, as shown in Figure 4. The SCSI series block is expanded as in the inset of Figure 4.

Series-parallel structure is often violated in practice. Non-series-parallel block diagrams are often cast as s-t connectedness problem or as network reliability problem or just relgraph in SHARPE. The price to be paid for this additional modeling power is the increased complexity of solution methods. Known solution methods are: factoring (or conditioning), finding all minpaths followed by the use of one of many sum-of-disjoint-products (SDP) algorithms, the use of binary decision diagrams (BDD) or the use of Monte-Carlo simulation. SDP and BDD based algorithms have been implemented in the SHARPE software package [2,3]. Nevertheless, real systems pose a challenge to these algorithms. For instance, the reliability of the current return network subsystem of Boeing 787 was modeled as a relgraph shown in Figure 5. However, the number of minpaths was estimated to be over 4 trillion.

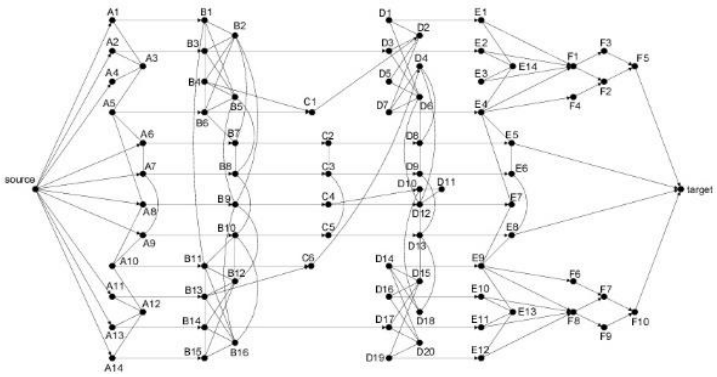

Figure 5. Boeing Relgraph Example

To solve the model, for the purpose of FAA certification, a new bounding algorithm was developed, was patented and published in a journal [7]. Table 1 reports the results showing that the upper and lower bounds to the s-t reliability were close enough, with a very small number of minpaths and mincuts selected for the computation. The computation time was very short for this otherwise intractable problem.

Table 1 - Unreliability upper/lower bounds

runtime	*20 seconds*	*120 seconds*	*900 seconds*
up bound	$1.146036\ 10^{-8}$	$1.081432\ 10^{-8}$	$1.025519\ 10^{-8}$
low bound	$1.019995\ 10^{-8}$	$1.019995\ 10^{-8}$	$1.019995\ 10^{-8}$
#minpaths	*28*	*29*	*33*
#mincuts	*113*	*113*	*113*

Table 2 shows a comparison of SDP and BDD methods for various benchmark networks of increasing complexity. The different BDD columns show the effect of node ordering on the computational time. Note that benchmark networks used here are not shown for the sake of brevity. Note also that the bounding method is not utilized in the comparison table.

Table 2 – Comparison of SDP and BDD with various orderings

	SDP		BDD (O1)		BDD (O2)		BDD (O3)		BDD (O4)	
Example	# DP	Time(s)	Size	Time(s)	Size	Time(s)	Size	Time(s)	Size	Time(s)
relex1	7	0.03	15	0.04	15	0.04	19	0.04	17	0.04
relex2	11	0.04	27	0.05	19	0.05	27	0.05	27	0.05
relex3	16	0.04	21	0.05	28	0.05	32	0.05	28	0.05
relex4	40	0.05	28	0.05	57	0.05	54	0.05	33	0.05
relex5	78	0.06	64	0.06	65	0.06	85	0.06	67	0.06
relex6	150	0.10	291	0.08	347	0.08	277	0.08	277	0.08
relex7	402	0.31	120	0.06	187	0.07	1178	0.12	444	0.09
relex8	2294	6.46	966	0.16	505	0.19	4865	0.38	743	0.15
relex9	47312	148.45	2083	0.41	14821	2.26	12277	1.25	3360	0.46

In aerospace, chemical and nuclear industry, engineers use fault trees to capture the conditions under which system fails. These Boolean conditions are encoded into a tree with AND gates, OR gates and *k-out-of-n* gates as internal nodes while leaf nodes represent component failures and the top node indicates system failure.

Fault trees without repeated events are equivalent to RBDs while those with repeated events are more powerful [1,2,8]. Solution techniques for fault trees with repeated events are the same as those for the network reliability problem discussed in the previous paragraph [1]. Fault trees with several thousand components can be solved with relative ease. Figure 6 shows a FT for a GE Equipment Ventilation System. Notice that leaves drawn as circles are basic events, while inverted triangles represent repeated events. Assuming that all the events have a failure probability equal to *q=0.001*, the SHARPE input file and the SHARPE output file are reported in Figure 7 on the left and on the right-hand side, respectively. In this example, SHARPE is asked to compute the Top Event probability (QTE = 1.0945e-02) as well as the list of the mincuts. We could ask for importance measures as well as a closed form expression of top event probability [1,3]. By assigning failure rates for each event,

we could ask for the time dependent failure probability of the system. Many other possibilities exist.

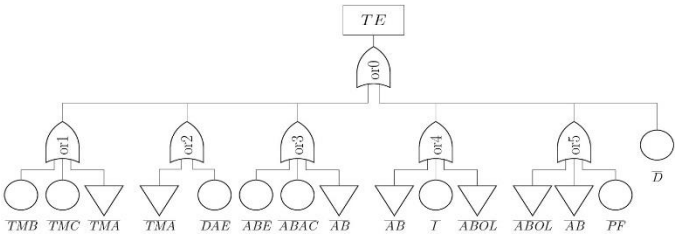

Figure 6. Fault Tree Model Equipment Ventilation System

```
echo Ventilation System

ftree vent
basic TMB prob(lp)
basic TMC prob(lp)
repeat TMA prob(lp)
basic DAE prob(lp)
basic ABE prob(lp)
basic ABAC prob(lp)
repeat AB prob(lp)
basic I prob(lp)
basic PF prob(lp)
basic D prob(lp)
repeat ABOL prob(lp)
or orr1 TMB TMC TMA
or orr2 TMA DAE
or orr3 ABE ABAC AB
or orr4 AB I ABOL
or orr5 AB PF ABOL
or te orr1 orr2 orr3 orr4 orr5 D
end
bind
lp 0.001
end
var sysunrel sysprob(vent)
expr sysunrel

mincuts(vent)
end
```

```
Ventilation System
sysunrel:   1.0945e-002

Mincuts for system vent:

[(TMC)],
[(TMB)],
[(PF)],
[(ABOL)],
[(I)],
[(AB)],
[(ABAC)],
[(ABE)],
[(TMA))],
[(DAE)],
[(D)]
```

Figure 7. SHARPE Input/Output file for Ventilation System

Table 3 presents the computation time for FT models with repeated events as the total number of nodes increases assuming a specific fault tree that we do not show here.

Table 3 – Complexity comparison of 3 computation techniques

Number of total leaves	Computation Time (second)		
	factoring	BDD	SDP
10000	0.98	2.06	11.12
20000	2.63	7.11	23.07
30000	5.58	14.89	37.97
40000	10.23	26.69	52.46
50000	13.94	42.04	69.37
60000	19.53	59.33	85.90
70000	26.48	—	102.09
80000	34.49	—	122.04
90000	41.25	—	141.35
100000	52.41	—	162.84

By assigning failure rates to components, system reliability at time t and the mean time to system failure can be computed. Time to failure distribution other than exponential (e.g., Weibull) can be used. Furthermore, by assigning failure rate and repair rate to each component, steady state and instantaneous availability can be computed (assuming independence in repair besides failure independence).

Fault trees have been extended to non-coherent gates such as NOT gates, to multi-state components [9], phased mission systems [10] and with dynamic gates [11]. SHARPE fault trees allow NOT gate, multi-state components and phased-mission systems. Dynamic gates are not explicitly included in SHARPE but can easily implemented since fault trees, Markov chains and hierarchical modeling are provided [1].

3. STATE SPACE METHODS

As stated in the last Section, non-state-space models with thousands of components can be solved without generating their underlying state space by making the independence assumption. But in practice, dependencies do exist among components. We then need to resort to state space models such as (homogeneous) continuous-time homogeneous Markov chains (CTMC). Markov models have been used to capture dynamic redundancy, imperfect coverage, escalated levels of recovery, concurrency, contention for resources, combined performance and reliability/availability and survivability [1,12]. Markov availability model will have no absorbing states (Figure 8) while Markov reliability models will have one or more absorbing states (Figure 10). Markov Models can be solved for steady state, transient and cumulative transient behavior according to the following equations [12,1]:

Steady state	$\boldsymbol{\pi Q} = 0$ with $\sum \pi = 1$
Transient	$d\boldsymbol{\pi}(t)/dt = \boldsymbol{\pi}(t)\ \boldsymbol{Q}$ given $\boldsymbol{\pi}(0)$
Cumulative Transient	$d\boldsymbol{b}(t)/dt = \boldsymbol{b}(t)\ \boldsymbol{Q} + \boldsymbol{\pi}(0)$

where $\boldsymbol{Q}$ is the infinitesimal generator matrix of the CTMC, $\boldsymbol{\pi}(t)$ is the state probability vector at time *t*, $\boldsymbol{\pi}(0)$ is the initial state probability vector, $\boldsymbol{\pi} = \mathbf{lim}_{t\to\infty}\, \boldsymbol{\pi}(t)$ is the steady state probability vector and $\boldsymbol{b}(t) = \int_0^t \boldsymbol{\pi}(u)\, du$ is the vector of the expected state occupancy times in the interval from 0 and *t*.

3.1 CTMC Availability models

The system availability (or instantaneous, point, or transient availability) is defined as the probability that at time *t* the system is in an up state:

$$A(t) = P\{system\ working\ at\ t\}$$

Steady-state availability (A_{ss}) or just availability is the long-term probability that the system is available when requested:

$$A_{ss} = \lim_{t\to\infty} A(t) = \frac{MTTF}{MTTF + MTTR}$$

Where MTTF is the system mean time to failure and MTTR is the system mean time to recovery. When applied to a single component, the above equation holds without a distributional assumption. For a complex system with redundancy, the equation holds if we use "equivalent" MTTF and "equivalent" MTTR [1,12].

The availability model of the Linux operating system used in the IBM SIP WebSphere was presented in [13] and is shown in Figure 8. From the UP state, the model enters the down state

DN with failure rate λ_{OS}. After failure detection, with a mean time of $1/\delta_{OS}$, the system enters the failure detected state DT.

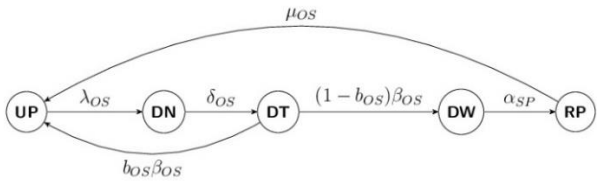

Figure 8. CTMC availability model of Linux OS

The OS is then rebooted with the mean time to reboot given by $1/\beta_{OS}$.With probability b_{OS} the reboot is successful, and system returns to UP state. However, with probability $1 - b_{OS}$ the reboot is unsuccessful, and the system enters the DW state where a repairperson is summoned. The travel time of the repairperson is assumed to be exponentially distributed with rate α_{SP}. The system then moves to state RP. The repair takes a mean time of $1/\mu_{OS}$, and after its completion, the system returns to the UP state.

```
echo Linux OS in IBM WebSphere Model
markov LinuxOS
1 2 los
2 3 dos
3 1 bos*beta
3 4 (1-bos)*beta
4 5 asp
5 1 mos

Parameter values
```

```
bind
los 1/4000
dos 1
beta 6
bos 0.9
asp 1/2
mos 1
end
echo  Steady-state availability equal
echo  to probability of state 1
expr prob (LinuxOS,1)
end
```

Figure 9. SHARPE Input file for the CTMC of Figure 8

Solving the steady state balance equations, a closed-form solution for the steady state availability of the OS is obtained in this case due to the simplicity of the Markov chain. Thus, $A_{ss} = \pi_{UP}$:

$$A_{ss} = \frac{1}{\lambda_{OS}} \left[\frac{1}{\lambda_{OS}} + \frac{1}{\delta_{OS}} + \frac{1}{\beta_{OS}} + (1 - b_{OS})\left(\frac{1}{\alpha_{SP}} + \frac{1}{\mu_{OS}}\right)\right]^{-1}$$

We can alternatively obtain a numerical solution of the underlying equations by using a software package such as SHARPE. Either graphical or textual input can be employed. The SHARPE textual input file modeling the CTMC of Figure 8 is shown in Figure 9. The Steady state availability is computed using the command `expr prob (LinuxOS,1)` and with the assigned numerical values for parameters, the result is $A_{ss} = 0.99963$.

3.2 CTMC Reliability models

CTMC for reliability models have one or more absorbing states and the reliability at time t is defined as the probability that the system is continuously working during the interval *(0-t]*. Further, since in a reliability model the system down state is an absorbing state, the MTTF can be calculated as the mean time to absorption in the corresponding CTMC model [1,2,12].

The reliability model for the Linux operating system used in the IBM is shown in Figure 10. The repair transition from state RP to state UP and the transition from state DW are removed, that is, the down state reached from the UP state, is made absorbing.

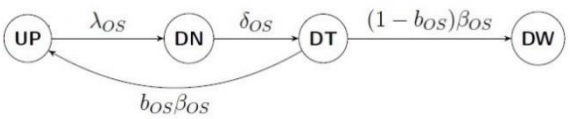

Figure 10. CTMC reliability model of Linux OS

In this case, the model is simple enough so that a closed-form solution can be obtained by hand (or using Mathematica) by setting up and solving the underlying Kolmogorov differential equations. Alternatively, a numerical solution of the underlying equations can be obtained using SHARPE. The SHARPE textual input file for the reliability model of Figure 10 is shown in Figure 11. Note that in this case, since the CTMC is not irreducible, an initial probability vector must be specified.

```
markov LinuxOS
1 2 los
2 3 dos
3 1 bos*beta
3 4 (1-bos)*beta
end

* initial state probabilities
1 1.0
2 0.0
3 0.0
4 0.0
end

* Parameter values
bind
los 1/4000
dos 1
beta 6
bos 0.9
end
```

```
echo  Reliability vs time equal to probability
echo  of state 1  at time t
echo System reliability at times 0 thru 10000
echo in steps of 2000
func rel(t) tvalue(t;LinuxOS,1)
loop t,0,  10000, 2000
expr rel(t)
end
expr mean(LinuxOS)
end
```

Figure 11. SHARPE Input file for the CTMC of Figure 9

The system reliability at time t is defined in this case as $R(t) = \pi_{UP}(t)$ and is computed from *t=0* to *t=10000* in steps of *2000.* As noted earlier, the MTTF is defined as the mean time to absorption and is computed using the SHARPE command `expr mean(LinuxOS)`. With the assigned numerical values the result is *MTTF=40012 hr.*

The CTMC of a reliability model can be, but need not be, acyclic, as in the case of Figure 10. If there is no component level repair (recovery) then the CTMC will be acyclic but if there is component level repair (but no repair after system failure) then the CTMC will have cycles. However, it will one or more absorbing states. System down states will be absorbing states.

Reliability modeling techniques have wide applications in different technological fields and have been proposed to provide new frontiers in predicting health care outcomes. With the rise in quantifiable approaches to health care, lessons from reliability modeling may well provide new ways of improving patient healthcare. Describing the development of conditions leading to organ system failure provides motivation for quantifying disease progression. As an example, a simple model for progressive kidney diseases leading to renal failure is reported in Figure 12 [14] where five discrete conditions are enumerated in keeping with clinical classification of kidney function:

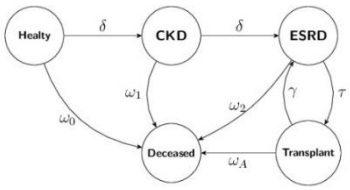

Figure 12. Markov model of renal disease progression [14]

1 Healthy: Normal renal function,
2 CKD: Chronic Kidney Disease without renal failure,
3 ESRD: End-Stage Renal Disease
4 Transplant: patients who have received a transplant,
5 Deceased

The parameter values, used in solving the model of Figure 12, are reported in Table 4. These values are estimated for a 65-year old Medicare patient, and are based on the latest available statistics from United States Renal Data System (USRDS) annual report [15].

Table 4 - Parameter estimates for a 65-year medicare patient

Description	Symbol	Value *(event/year)*
Decline	δ	0.1887
Transplant	τ	0.1786
Graft Rejection	γ	0.0050
Prognosis-Healthy	ω_0	0.0645
Prognosis-CKD	ω_1	0.1013
Prognosis-ESRD	ω_2	0.2174
Prognosis-Transplant	ω_T	0.0775

The model of Figure 12 is solved for measures such as survival rate and expected cost incurred by a patient in a one-year interval [14].

Efficient algorithms are available for solving Markov chains with several million states [16,17,18], both in steady-state and in transient regime. Furthermore, measures of interest such as reliability, availability, performability, survivability etc. can be computed by means of reward rate assignments to the states of the CTMC [1,12]. Derivatives (sensitivity functions) of the measures of interest with respect to input parameters can also be computed to help detect bottlenecks [19,20,21]. Nevertheless, the generation, storage and solution of real-life-system Markov models still poses challenges. Higher level formalisms such as those based on stochastic Petri nets and their variants [22,4,23] have been used to automate the generation, storage and the solution of large state spaces [24].

An example of the use stochastic reward net to model the availability of an Infrastructure-as-a-Service (IaaS) cloud is shown in Figure 13 [25]. To reduce power usage costs, physical machines (PMs) are divided into three pools: Hot pool (high performance and high power usage), Warm pool (medium performance and medium power usage) and Cold pool (lowest performance and lowest power usage). PMs may fail and get repaired. A minimum number of operational hot PMs is required for the system to function but PMs in other pools may be temporarily assigned to the hot pool in order to maintain system operation. Upon repair, PMs migrate back to their original pool. Migration creates dependence among the pools.

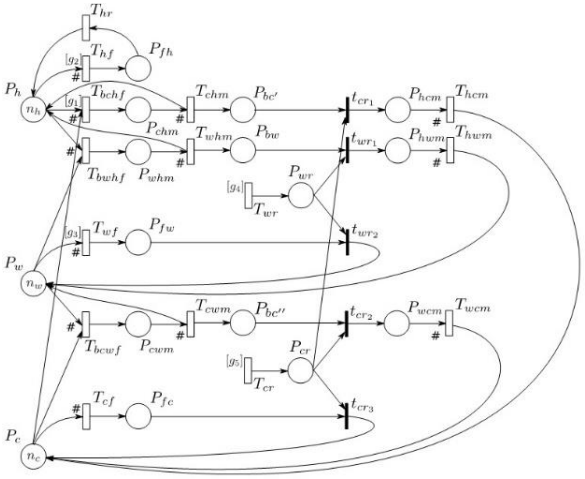

Figure 13. SRN availability model of IaaS Cloud

A monolithic CTMC is too large to construct by hand. We use a high-level formalism of stochastic Petri net known as Stochastic Reward Net (SRN) [23,24]. An SRN model can be automatically converted into an underlying Markov (reward) model that is solved numerically for the measures of interest such as expected downtime, steady-state availability, reliability, sensitivities of these measures. In Figure 13, place P_h initially contains n_h PMs of the hot pool, P_w contains n_w PMs of the warm pool and P_c contains n_c PMs of the cold pool. Assuming the number of PMs in each pool is identical and equal to n, the number of states for the monolithic model of Figure 13, is reported in the second column of Table 5. From this table, it is clear that this approach based merely on SRN, that we call largeness tolerance, soon reaches its limits as the time needed for the generation and storage of the state space becomes prohibitively large for real systems.

4. HIERARCHY & FIXED-POINT ITERATION

In order to avoid large models as is the case in a monolithic Markov (or generally state space) model, we advocate the use of multi-level models in which the modeling power of state space models and efficiency of non-state-space models are combined together (Figure 14).

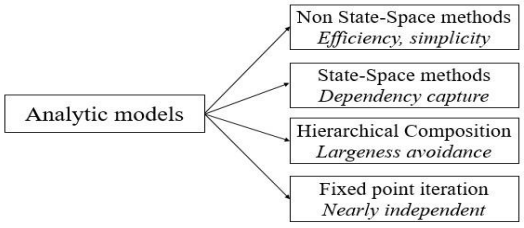

Figure 14. Analytic Modeling Taxonomy

Since a single monolithic model is never generated and stored in this approach, this is largeness avoidance in contrast with the use of largeness tolerance (recall SRN and related modeling paradigms) wherein the underlying large model is generated and stored. In multi-level modeling each of the model is solved and results are conveyed to other relevant models to use as their input parameters. This transmission of results of one sub-model as input parameters to other sub-models is depicted as a graph that has been called an import graph.

Consider for instance, the availability model of the SUN Microsystem whose top-level RBD availability model was shown in Figure 4. Each block of the RBD of Figure 4 is a complex sub-system that was modeled separately using the appropriate formalism in order to compute the block steady-state availability. In the present case, the subsystems were modeled as Markov chains to cater for dependence within each subsystem. The block availability is then rolled up to the higher level RBD model to compute the system steady state availability. The import graph for this model is shown in Figure 15.

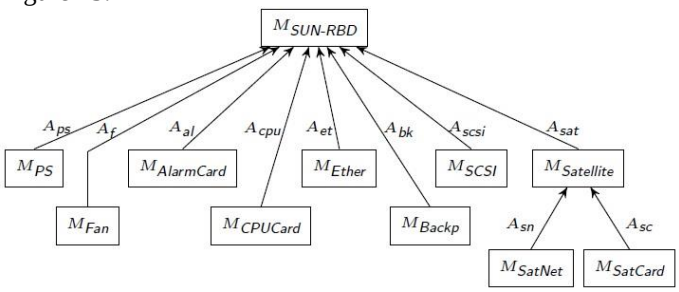

Figure 15. Import graph for the High Availability Platform from Sun Microsystems [6]

The import graph in this case is acyclic. We can then carry out a topological sort of the graph resulting in a linear order specifying the order in which the sub-models are to be solved and the result rolled up to the hierarchy.

As the next example we return to the IaaS cloud availability model and improve its scalability. The monolithic SRN model of Figure 13 is decomposed into three sub-models to describe separately the behavior of three pools [25] while taking into account their mutual dependencies by means parameter passing. The three sub-models are shown in Figure 16. Its import graph is shown in Figure 17 indicating the input parameters and output measures that are exchanged among sub-models to obtain the overall model solution. Import graphs such as the one of Figure 17 are not acyclic and hence the solution to the overall problem can be set up as a fixed-point problem. Such problems can be solved iteratively by successive substitution with some initial starting point.

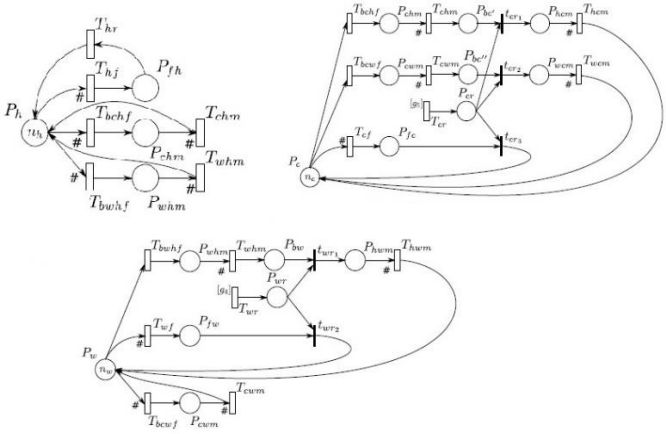

Figure 16. Decomposed SRN Availability model of IaaS Cloud

Many mathematical issues arise such as the existence of the fixed point, the uniqueness of the fixed-point, the rate of convergence, accuracy and scalability. Except for the existence of the fixed-point [26], all other issues are open for investigation. Nevertheless, the method has been successfully utilized on many real problems [1].

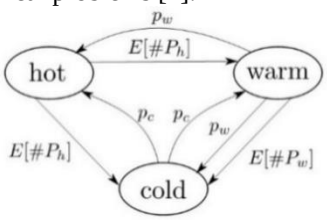

Figure 17. Import graph describing sub-model interactions

Table 5 shows the effect of the decomposition method (which is also known as interacting sub-models method), comparing the number of states of the monolithic model (column 2) with the number of states of the interacting sub-models (column 3).

Table 5 – Comparison of monolithic vs decomposed model

n	Monolithic model #states	Interacting sub-models #states
3	10,272	196
4	67,075	491
5	334,948	1,100
6	1,371,436	2,262
7	4,816,252	3,770
8	Memory overflow	6,939
10	-	20,460
20	-	21,273
40	-	271,543
60	-	1,270,813
80	-	3,859,083
100	-	9,196,353

n is the initial #PM in each pool.

There are many more examples of this type of multi-level models in [1,2,13, 27,28,29,30,31].

5. RELAXING THE EXPONENTIAL ASSUMPTION

One standard complain about the use of homogeneous Markov chains is the ubiquitous assumption of all event times being exponentially distributed. There are several known paradigms that can remove this assumption: non-homogeneous Markov chains, semi Markov and Markov regenerative process, and the use of phase-type expansions. All these techniques have been used and many examples are illustrated in [1].

Nevertheless, there is additional complexity in using non-exponential techniques in practice, partly because the analytical solution is more complex but also because additional information about the non-exponential distributions that is then needed is often hard to come by.

A flowchart comparing the modelling power of the different state space model-types is given in Figure 18 [1].

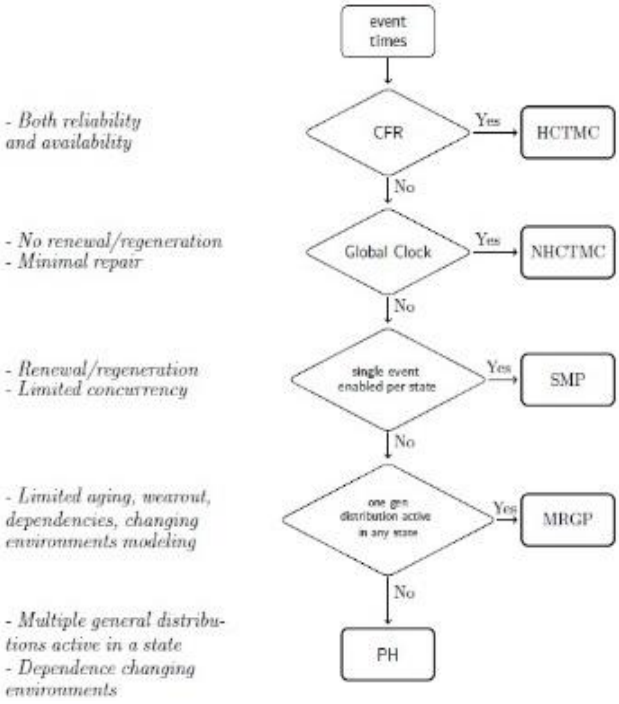

Figure 18 - Flow chart comparing the modeling power of the different state space model types [1].

6. CONCLUSIONS

We have tried to provide an overview of known modeling techniques for the reliability and availability of complex systems. We believe that techniques and tools do exist to capture the behavior of current-day systems of moderate complexity. Nevertheless, higher and higher complexity is being designed into systems and hence the techniques must continue to evolve. Together with the largeness problem, the need for higher fidelity will require increasing use of non-exponential distributions, the need to properly combine performance, power and other measures of system effectiveness together with failure and recovery. Parameterization and validation of the models need to be further emphasized and aided. Tighter connection between data-driven and model-driven methods on the one hand, and, combining simulative solution with analytic-numeric solution on the other hand is desired. Validated models need to be maintained throughout the life of a system so that they can be used for tuning at operational time as well. Uncertainty in model parameters, so-called epistemic uncertainty, as opposed to aleatory uncertainty already incorporated in the models discussed here, needs to be accounted for in a high-fidelity assessment of reliability and availability [32]. For further discussion on these topics, we encourage the tutorial attendees to consult our book [1].

Part I – Introduction (Chapters 1:3)

Part II - Non-state-space models (Chapters 4:8)

Part III - State-space Models with Exponential Distributions (Chapters 9:12)

Part IV - State-space Models with Non-Exponential Distributions (Chapters 13:15)

Part V - Multi-Level Models (Chapters 16:17)

Part VI - Case Studies (Chapter 18)

Figure 19 – Chapters overview of the book

In Figure 19 we provide the chapters overview of the book [1], and in Figure 20 a classification of the considered modeling formalisms. We are currently preparing a solution manual and set of power point slides as education material to accompany the book.

7. REFERENCES

1. K. Trivedi and A. Bobbio, Reliability and Availability Engineering, Cambridge University Press, 2017
2. R. Sahner, K. Trivedi, and A. Puliafito, Performance and Reliability Analysis of Computer Systems: An Example-based Approach Using the SHARPE Software Package. Kluwer Academic Publishers, 1996.
3. K. Trivedi and R. A. Sahner, "SHARPE at the Age of Twenty Two," ACM Performance Evaluation Review, vol. 36, issue 4, March 2009.
4. G. Ciardo, J. Muppala, and K. S. Trivedi, "SPNP: Stochastic petri net package," in Proc. Third Int. Workshop on Petri Nets and Performance Models, 1989, pp. 142–151.
5. C. Hirel, B. Tuffin, and K. S. Trivedi, "SPNP: Stochastic Petri Nets. Version 6," in International Conference on Computer Performance Evaluation: Modelling Techniques and Tools (TOOLS 2000), B. Haverkort, H. Bohnenkamp (eds.), LNCS 1786, Springer Verlag, 2000, pp. 354 – 357.
6. K. Trivedi, R. Vasireddy, D. Trindade, S. Nathan, and R. Castro, "Modeling high availability systems," in Proc. IEEE Pacific Rim International Symposium on Dependable Computing (PRDC), 2006.

7. S. Sebastio, K. Trivedi, D.Wang, X. Yin, Fast computation of bounds for two-terminal network reliability, European Journal of Operational Research, vol. 238, no. 3, pp. 810–823, 2014.
8. M. Malhotra and K. Trivedi, "Power-hierarchy among dependability model types," IEEE Transactions on Reliability, vol. R-43, pp. 493–502, 1994.
9. X. Zang, D. Wang, H. Sun, and K. Trivedi, "A BDD-based algorithm for analysis of multistate systems with multistate components," IEEE Transactions on Computers, vol. 52, no. 12, pp. 1608–1618, 2003.
10. X. Zang, H. Sun, and K. Trivedi, "A BDD-based algorithm for reliability analysis of phased mission systems", IEEE Trans. on Reliability, vol. 48, no. 1, pp. 50 – 60, March 1999.
11. G. Merle, J. Roussel, J. Lesage, and A. Bobbio, "Probabilistic algebraic analysis of fault trees with priority dynamic gates and repeated events," IEEE Transactions on Reliability, vol. 59, no. 1, pp. 250–261, 2010.
12. K. Trivedi, Probability & Statistics with Reliability, Queueing & Computer Science applications, 2nd ed. John Wiley & Sons, 2001.
13. K. S. Trivedi, D. Wang, J. Hunt, A. Rindos, W. E. Smith, and B. Vashaw, "Availability modeling of SIP protocol on IBM © Websphere ©," in Proc. Pacific Rim International Symposium on Dependable Computing (PRDC), 2008, pp. 323–330.
14. R. Fricks, A. Bobbio, and K. Trivedi, "Reliability models of chronic kidney disease," in Proceedings IEEE Annual Reliability and Maintainability Symposium, 2016, pp. 1–6.
15. United States Renal Data System, "2014 annual data report: An overview of the epidemiology of kidney disease in the United States," National Institutes of Health - National Institute of Diabetes and Digestive and Kidney Diseases, Tech. Rep., 2014.
16. W. Stewart, Introduction to the Numerical Solution of Markov Chains. Princeton University Press, 1994.
17. A. Reibman and K. Trivedi, "Numerical transient analysis of Markov models," Computers and Operations Research, vol. 15, pp. 19–36, 1988.
18. A. Reibman, R. Smith, and K. Trivedi, "Markov and Markov reward model transient analysis: an overview of numerical approaches," European Journal of Operational Research, vol. 40, pp. 257–267, 1989.
19. A. Bobbio and A. Premoli, "Fast algorithm for unavailability and sensitivity analysis of series-parallel systems," IEEE Transactions on Reliability, vol. R-31, pp. 359–361, 1982.
20. J. Blake, A. Reibman, and K. Trivedi, "Sensitivity analysis of reliability and performability measures for multiprocessor systems," ACM SIGMETRICS Perform. Eval. Rev., vol. 16, no. 1, pp. 177–186, May 1988.
21. R. Matos, P. Maciel, F. Machida, D. S. Kim, and K. Trivedi, "Sensitivity analysis of server virtualized system availability," IEEE Transactions on Reliability, vol. 61, no., pp. 994–1006, Dec 2012.
22. A. Bobbio, "System modelling with Petri nets," in System Reliability Assessment, A. Colombo and A. de Bustamante, Eds. Kluwer Academic P.G., 1990, pp. 103–143.
23. G. Ciardo, J. Muppala, and K. Trivedi, "On the solution of GSPN reward models," Performance Evaluation, vol. 12, pp. 237–253, 1991.
24. G. Ciardo, A. Blakemore, P. F. Chimento, J. K. Muppala, and K. Trivedi, "Automated generation and analysis of Markov reward models using stochastic reward nets," in Linear Algebra, Markov Chains, and Queueing Models,. The IMA Vol in Mathematics and its Applications, C. Meyer and R. Plemmons, Eds. Springer, 1993, vol. 48, pp. 145–191.
25. R. Ghosh, F. Longo, L. Frattini, S. Russo & K. Trivedi, "Scalable Analytics for IaaS Cloud Availability", IEEE Trans. on Cloud Computing, 2014.
26. V. Mainkar and K. Trivedi, "Sufficient Conditions for Existence of a Fixed Point in Stochastic Reward Net-Based Iterative Models," IEEE Trans. Software Eng., vol. 22, no. 9, pp. 640-653, 1996.
27. H. Sukhwani, A. Bobbio, and K. Trivedi, "Largeness avoidance in availability modeling using hierarchical and fixed-point iterative techniques," International Journal of Performability Engineering, vol. 11, no. 4, pp. 305–319, 2015.
28. R. Ghosh, F. Longo, V. Naik, and K. Trivedi, "Modeling and performance analysis of large scale IaaS Clouds," Future Generation Comp. Systems, vol. 29, no. 5, pp. 1216-1234, 2013.
29. R. Ghosh, F. Longo, R. Xia, V. Naik, and K. Trivedi, "Stochastic Model Driven Capacity Planning for an Infrastructure-as-a-Service Cloud," IEEE Trans. Services Computing, vol. 7, no. 4, pp. 667-680, 2014.
30. K. Trivedi, D. Wang, and J. Hunt, "Computing the Number of Calls Dropped Due to Failures," ISSRE, pp. 11-20, 2010.
31. S. Mondal, X. Yin, J. Muppala, J. Alonso Lopez, and K. Trivedi, "Defects per Million Computation in Service-Oriented Environments," IEEE Trans. Services Computing, vol. 8, no. 1, pp. 32-46, 2015.
32. K. Mishra and K. Trivedi, "Closed-form approach for epistemic uncertainty propagation in analytic models," Stochastic Reliability and Maintenance Modeling, vol. 9. Springer Series in Reliability Engineering, 2013, pp. 315–332.

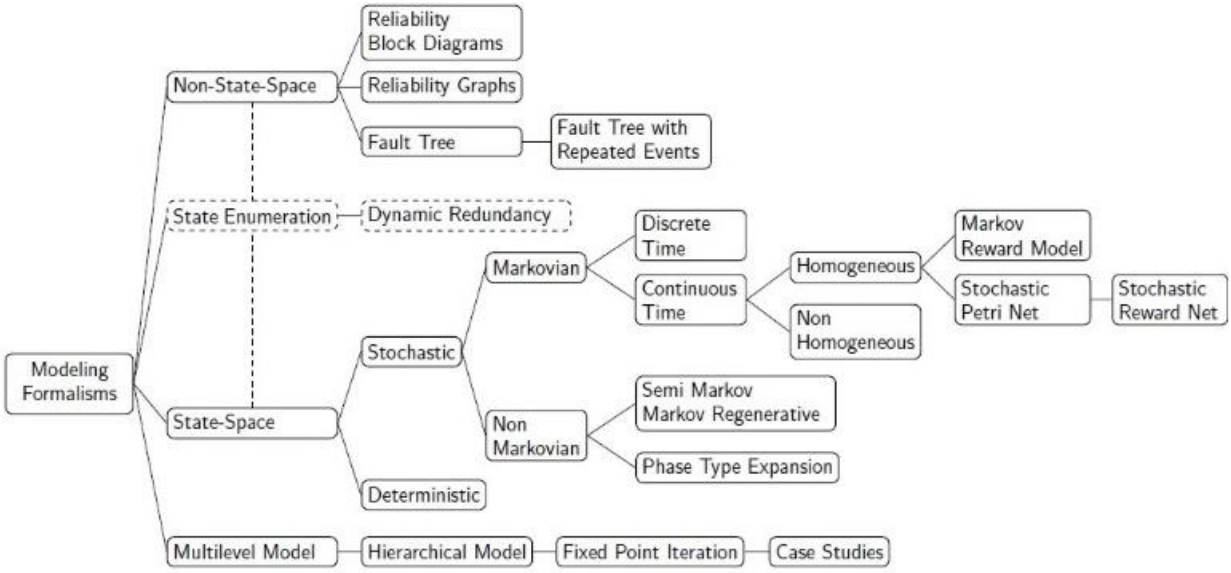

Figure 20 – modeling formalisms