SHARPE at the Age of 25

Huawei, Shenzhen
April 24, 2012

Kishor S. Trivedi
Department of Electrical and Computer Engineering
Duke University
Durham, NC
Email: kst@ee.duke.edu
URL: www.ee.duke.edu/~kst
Outline

- An Introduction to SHARPE software package
- Reliability/Availability models in practice
- Performance models in practice
- Performability models in practice
- Possible outputs
- Case studies
Overview of SHARPE

- SHARPE: Symbolic-Hierarchical Automated Reliability and Performance Evaluator
- First version was released in Sept. 1986
- Well-known modeling software package (Installed at over 500 Sites; companies and universities)
- Combines flexibility of Markov models and efficiency of non-state-space models
- Ported to most architectures and operating systems
- Used for Education, Research, Engineering Practice
- Users: Boeing, 3Com, EMC, DEC, AT & T, Lucent, IBM, NEC, Motorola, Siemens, GE, HP, …
Overview of SHARPE (cont.)

- Graphical as well as Textual User Interface
- Used for assessment of performance, dependability, performability, security and survivability
- Computer, software, telco, automotive, power, aerospace and other industries
- Combines flexibility of Markov models and efficiency of non-state-space models
- Hierarchy facilitates largeness & stiffness avoidance
- Steady-state, transient, cumulative transient analysis
- Written in C language
- Used as an engine by other tools such as Boeing’s IRAP
Some Recent Uses

- Scalable Performance, availability and power models of IBM Research Cloud
- Boeing 787 Current Return Network Reliability model
- Motorola Cable Modem Termination system
- IBM Bladecenter availability model
- IBM SIP on Websphere availability/DPM model
- Survivability quantification of networks and PSTN
- Resiliency quantification
- Many previous models at DEC (vaxcluster), SUN, HP, 3com
# Architecture of SHARPE

<table>
<thead>
<tr>
<th>Model Type</th>
<th>Reliability/Availability</th>
<th>Performance</th>
<th>Performability</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fault tree</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Multistate fault tree</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Reliability block diagram</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Reliability graph</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Phased-mission systems</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Markov chain</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Semi-Markov chain</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GSPN</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Stochastic reward net</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MRGP</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PFQN</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MPFQN</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Task Graph</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- **Reliability/Availability**: Indicates the focus on reliability and availability.
- **Performance**: Indicates the focus on performance.
- **Performability**: Indicates the focus on performability.
Architecture of SHARPE interface

Reliability Block Diagrams

Fault tree

SMP/MRGP

Markov chain

Hierarchical & Hybrid Compositions

Reliability graph

Task graph

Petri net (GSPN & SRN)

Pfqn, Mpfqn

Reliability/Availability

Performance

Performability
Recently added Features

- Equivalent mean time to system failure and equivalent mean time to system repair implemented for Markov chains, RBDs, ftrees
- Steady-state computation of MRGP models
- Stochastic reward net is available as a model type
- Bounding algorithm for relgraphs
- Weibull and other distributions for combinatorial models
- Derivative computation for Markov chain measures
- Importance measures for fault tree models
- Loops and if statements in model evaluations
- Loops in the definition of Markov chains
- Large Fault trees with global repeat events
Evaluation Methods

Quantitative Evaluation

Measurement-based

Model-based

Discrete-event simulation

Hybrid

Analytic Solution

Close-form solution

Numerical solution via a tool

Numerical solution of analytic models
Not as well utilized;
Unnecessarily excessive
Use of simulation
Evaluation Methods

- Measurement-Based
  - More Accurate, most expensive
  - Not always possible or cost effective during system design.
  - Statistical techniques are very important here

- Model-Based
  - Combined approach where measurements are made at the subsystem level and models are built for the system as a whole
MODELER'S DILEMMA

Should I Use Discrete-Event Simulation?

- Point estimates and Confidence intervals.
- How many simulation runs are sufficient?
- What Specification Language to use?
  - C, SIMULA, SIMSCRIPT, MODSIM, GPSS, RESQ, SPNP v6, Bones, SES workbench, ns 2, opnet, Blocksim
MODELER'S DILEMMA (cont’d)

- Simulation (Pros and Cons):
  + Detailed System Behavior including non-exponential behavior.
  + Performance, Dependability and Performability Modeling Possible.
  - Long Execution Time (Variance Reduction Possible)
    ◆ Importance Sampling, importance splitting, regenerative simulation.
    ◆ Parallel and Distributed Simulation.
  - Many users in practice do not realize the need to calculate confidence intervals.

In practice we often combine simulation and analytic models
Analytic Modeling Taxonomy

- Quantitative Dependability Evaluation
- Discrete-event simulation
- Hybrid
- Analytic Solution
  - Non-state-space models
  - State-space models
  - Hierarchical composition
  - Fixed point iterative solution
Non-state-space Models: Taxonomy

Non-state space model types

- SP reliability block diagrams (RBD)
- Non-SP reliability block diagrams (relgraph)
- Fault trees (FT)
- Fault trees with repeated events

Extensions such as multi-state systems, phased-mission systems etc.
MODELER'S DILEMMA (Contd.)

Should I Use Non-State-Space Methods?

- Also called Combinatorial Models.
- Model Solved Without Generating State Space
- Use: Order Statistics, Mixing, Convolution
- Common Dependability Model Types:
  - Series-Parallel Reliability Block Diagrams
  - Non-Series-Parallel Block Diagrams (or Reliability Graphs).
  - Fault-Trees Without Repeated Events
  - Fault-Trees With Repeated Events
Non-state-space models

- Reliability block diagrams, Fault trees and Reliability graphs
  - Commonly used for Reliability, Availability, Security, Safety
  - These model types are similar in that they capture conditions that make a system fail in terms of the structural relationships between the system components.
Non-state-space models (cont’d)

- Non-state-space methods for RBDs, relgraphs and FTs are easy to use and assuming statistical independence solve for system reliability, system availability and system MTTF; can find bottlenecks

- Relatively good algorithms are known for solving medium sized systems

- Each component can have attached to it
  - A probability of failure
  - A failure rate
  - A distribution of time to failure
  - Steady-state or instantaneous (un)availability
SHARPE Model Types

- Non-state space
- State space
SHARPE Model Types

- SHARPE
  - Non-state space
  - State space
Fault tree example
Fault Tree Model of GE Steam Turbine Control System
Fault Tree Model of GE Equipment Ventilation System
FAULT TREE MODEL, Motorola Bedrock System

Yin, Lanus, Trivedi, *IEEE TR*, 2003
SHARPE

Non-state space

RBD

Sahner 86

FT

FTRE

Multi state FT (BDD)

PMS FT (BDD)

Sahner 86

BDD ZST 99

Factoring

SDP

BDD ZST 99

VT 89

State space
RBD example

File Server

Computer Network

Workstation 1

Workstation 2
Two Router Reliability Block Diagrams

RBD of Cisco 12000 GSR

RBD of Juniper M20

Non-state space

- State space
  - SHARPE
  - Ftree
  - RBD
  - Rel. Graph
    - Sahner 86
    - BDDZST 99
    - SDP
  - FT
  - FTRE
  - Multi state FT (BDD)
  - PMS FT (BDD)
    - Sahner 86
    - BDD ZST 99
    - Factoring
    - SDP
    - BDD ZST 99
  - VT 89
Avionics

- Reliability analysis of each major subsystem of a commercial airplane needs to be carried out and presented to Federal Aviation Administration (FAA) for certification.

Real world example from Boeing Commercial Airplane Company.
Reliability Analysis of Boeing 787

- Current Return Network Modeled as a Reliability Graph
  - Consists of a set of nodes and edges
  - Edges represent components that can fail
  - Source and target nodes
  - System fails when no path from source to target
  - Compute probability of a path from source to target
Reliability Analysis of Boeing 787 (cont’d)

- Known solution methods for Relgraph
  - Find all minpaths followed by SDP (sum of disjoint products)
  - BDD (binary decision diagrams)-based method
- The above two methods implemented in our SHARPE software package
- Boeing tried to use SHARPE for this problem but it was too large to solve
Reliability Analysis of Boeing 787 (cont’d)

- Too many minpaths

- Compute reliability bounds instead of exact reliability

Number of paths from source to target

<table>
<thead>
<tr>
<th>node</th>
<th>#paths</th>
</tr>
</thead>
<tbody>
<tr>
<td>$E_7 \rightarrow$ target</td>
<td>40</td>
</tr>
<tr>
<td>$D_{12} \rightarrow$ target</td>
<td>143140</td>
</tr>
<tr>
<td>$C_4 \rightarrow$ target</td>
<td>308055</td>
</tr>
<tr>
<td>$B_y \rightarrow$ target</td>
<td>21054950355</td>
</tr>
<tr>
<td>$A_8 \rightarrow$ target</td>
<td>461604232201</td>
</tr>
<tr>
<td>source $\rightarrow$ target</td>
<td>$4248274506778 \approx 4 \times 10^{12}$</td>
</tr>
</tbody>
</table>
Reliability Analysis of Boeing 787 (cont’d)

- Developed a new efficient algorithm for (un)reliability bounds computation and incorporated in SHARPE

<table>
<thead>
<tr>
<th>runtime</th>
<th>20 seconds</th>
<th>120 seconds</th>
<th>900 seconds</th>
</tr>
</thead>
<tbody>
<tr>
<td>upper bound</td>
<td>1.1460365721e-008</td>
<td>1.0814324701e-008</td>
<td>1.0255197263e-008</td>
</tr>
<tr>
<td>lower bound</td>
<td>1.0199959877e-008</td>
<td>1.0199959877e-008</td>
<td>1.0199959877e-008</td>
</tr>
</tbody>
</table>

- Boeing has patented the algorithm jointly with Duke
- Satisfying FAA that SHARPE development used DO-178 B software standard was the hardest part
Non-state-space Methods (cont’d)

- Following are possible to compute (given component failure/repair rates)
  - System Reliability
  - System Availability (Steady-state, instantaneous)
  - Downtime
  - System MTTF
Non-state-space Methods (cont’d)

- Assuming:
  - Failures are statistically independent
  - As many repair units are assumed available, as needed.
- Relatively good algorithms are available for solving such models so that we can easily solve models with 100’s of components
  - Easy specification, fast computation, no distributional assumptions
Non-state-space methods (cont’d)

- Non-state-space approaches have relatively fast algorithms assuming stochastic independence between system components
  - Sum of Disjoint Products (SDP) algorithms.
  - Binary Decision Diagrams (BDD) algorithms.
  - Factoring (conditioning) algorithms.
  - Series-parallel composition algorithms.
  - Bounding algorithm for relgraphs

- All of the above implemented in SHARPE

- Solving a fault tree of a whole plane such as Boeing 787 is still a challenge

- Failure/Repair Dependencies are often present; RBDs, relgraphs, FTREEs cannot easily handle these (e.g., shared repair, warm/cold spares, imperfect coverage, non-zero switching time, travel time of repair person, reliability with repair).
Analytic Modeling Taxonomy

Analytic solution

Non-state-space methods

State-space methods
State-Space model taxonomy

Can relax the assumption of exponential distributions

- Markovian models
  - discrete-time Markov chains (DTMC)
  - continuous-time Markov chains (CTMC)
  - Markov reward models (MRM)
  - Semi-Markov process (SMP)
  - Markov regenerative process
- (discrete) State space models
- non-Markovian models
- Non-Homogeneous Markov
State-space models: Markov chains

- To model complex interactions between components, use models like Markov chains or more generally state space models.
- Markov reliability models will have one or more absorbing states; Markov availability models will have no absorbing states.
- Many examples of dependencies among system components have been observed in practice and captured by continuous-time Markov chains (CTMCs).
- Extension to Markov reward models makes computation of measures of interest relatively easy.
Should I Use Markov Models?

State-Space-Based Methods

+ Model Fault-Tolerance and Recovery/Repair
+ Model Dependencies
+ Model Contention for Resources
+ Model Concurrency and Timeliness
+ Generalize to Markov Reward Models for Modeling Degradable Performance
Generalize to Markov Regenerative Models for allowing Generally Distributed event times.

Generalize to Non-Homogeneous Markov Chains for allowing Weibull failure distributions

Performance, Availability and Performability Modeling Possible

- Large State Space (exponential in number of components.)
Markov chain model of a multiprocessor system
Modeling Hardware/Software Faults

Avaya Swift system

Availability model with passive redundancy
(warm replication) of application; Operational phase;
Mandelbugs or hardware transients

Assumptions

- A web server software, that fails at the rate $\gamma_p$ running on a machine that fails at the rate $\gamma_m$
- Mean time to detect server process failure $\delta_{1p}$ and the mean time to detect machine failure $\delta_{1m}$
- The mean restart time of a machine $\tau_{1m}$
- The mean restart time of a server $\tau_{1p}$

---

Performance and Reliability Evaluation of Passive Replication Schemes in Application Level Fault Tolerance
S. Garg, Y. Huang, C. Kintala, K. S. Trivedi and S. Yagnik
Markov reward model
Semi-Markov model; hot standby with periodic diagnostics for latent faults

Time to diagnostic is uniformly distributed over (0,T)
GSPN model
IN ORDER TO FULFILL OUR GOALS

- Modeling Reliability, Availability and Performability
- For Modeling Complex Systems
  
  We Need

- Automated Generation and Solution of Large Markov (Reward) Models
IN ORDER TO FULFILL OUR GOALS

- Facility for State Truncation, Hierarchical composition of Non-State-Space and State-Space Models, Fixed-Point Iteration
- There are Two of our software packages that potentially meet these goals
  - Stochastic Petri Net Package (SPNP)
  - Symbolic Hierarchical Automated Reliability and Performance Evaluator (SHARPE)
State Space Explosion

- State space explosion can be handled in two ways:
  - Large model tolerance must apply to specification, storage and solution of the model. If the storage and solution problems can be solved, the specification problem can be solved by using more concise (and smaller) model specifications that can be automatically transformed into Markov models (GSPN and SRN models).
  - Large models can be avoided by using hierarchical model composition.

- Ability of SHARPE to combine results from different kinds of models
  - Possibility to use state-space methods for those parts of a system that require them, and use non-state-space methods for the more “well-behaved” parts of the system.
Analytic Solution Taxonomy

Analytic solution

Non-state-space methods
Efficiency, simplicity

State-space methods
Dependency capture

Hierarchical composition
To avoid largeness
Availability model of SIP on IBM WebSphere

- Real problem from IBM
- SIP: Session Initiation Protocol
- Hardware platform: IBM BladeCenter
- Software platform: IBM WebSphere

- Subsystems modeled using Markov chains to capture dependence within the subsystem
- Fault tree used at higher levels as independence across subsystems can be assumed
- This is an example of hierarchical composition
  - A single monolithic model is not constructed/stored/solved
  - Each submodel is built and solved separately and results are propagated up to the higher level model
  - SHARPE facilitates such hierarchical model composition
Architecture of SIP on IBM WebSphere

<table>
<thead>
<tr>
<th>Replication domain</th>
<th>Nodes</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>A, D</td>
</tr>
<tr>
<td>2</td>
<td>A, E</td>
</tr>
<tr>
<td>3</td>
<td>B, F</td>
</tr>
<tr>
<td>4</td>
<td>B, D</td>
</tr>
<tr>
<td>5</td>
<td>C, E</td>
</tr>
<tr>
<td>6</td>
<td>C, F</td>
</tr>
</tbody>
</table>
Software Fault Tolerance

- Identical copies of SIP proxy used as backups (hot spares)
- Identical copies of WebSphere Applications Server (WAS) used as backups (hot spares)

- Traditional: Design diversity
- New Thinking
  - Use identical software copies as spares or backups
    - Does it help? If yes, why?
  - Recovery method for software failures is to
    - Restart a process, reboot a node
    - Does it help in dealing with failures caused by software bugs? If yes, why?
Adopted SW Fault Classification

Bohrbug
- An easily isolated fault that always manifests consistently under a well-defined set of conditions, because its activation and error propagation lack “complexity” as defined above. Bohrbug is the complementary antonym of Mandelbug.

Mandelbug
- A fault whose activation and/or error propagation are complex (e.g. a long time lag between the fault activation and the occurrence of a failure; interactions among hardware, operating system, other applications, timing and sequencing effects, etc.)

Aging-related bug
- A fault that causes an increased failure rate and/or degraded performance. The fault causes the accumulation of errors either inside the running application or in its system-internal environment.
Software Faults: Mitigation

Software (OS, recovery s/w, applications)

Bohrbugs
Aging-related bugs
Mandelbugs

Test/Debug
Des./Data Diversity
Rejuvenate
Restart app.
Reboot node
Failover to standby
Retry opn.

Design/Development
Operational
Failures Incorporated in Models

- **Physical faults**
  - Power faults
  - Cooling faults
  - Blade faults
  - midplane faults
  - Network faults

- **Software failures**
  - OS
  - Application
    - WAS
    - Proxy
      - Process hang
      - Process die

- **Memory faults**
  - NIC faults
  - CPU faults
  - base faults
  - I/O (RAID) faults
Availability model of SIP on IBM WebSphere

- Hierarchical model (ftree at top level; Markov submodels)
  - SIP Application Server top level availability model (GUI)

\( i_x \): \( i \)th appserver on node \( X \)

\( P_i \): \( i \)th proxy server

\( BS_X \): node \( X \) hardware

\( CM_i \): chassis \( i \) hardware
Availability model of SIP on IBM WebSphere

- Availability models of a Blade Server and Common BladeCenter Hardware

Chassis failure

SHARPE GUI representation

SHARPE textual representation

ftree CM
repeat MP prob(1-exrss(midplane))
repeat Cool prob(1-exrss(cooling))
repeat Pwr prob(1-exrss(power))
or CM MP Cool Pwr
end
Availability model of SIP on IBM WebSphere

- Availability models of a Blade Server and Common BladeCenter Hardware

Blade server failure

SHARPE GUI representation

SHARPE textual representation

```
ftree BLADE
basic Base prob(1-exrss(base))
basic CPU prob(1-exrss(processor))
basic Mem prob(1-exrss(memory))
basic RAID prob(1-exrss(disk))
basic OS prob(1-exrss(OS))
basic nic1 prob(1-exrss(nic))
basic nic2 prob(1-exrss(nic))
repeat esw1 prob(1-exrss(switch))
repeat esw2 prob(1-exrss(switch))
or eth1 nic1 esw1
or eth2 nic2 esw2
and eth eth1 eth2
or BS Base CPU Mem RAID eth OS
end
```
Availability model of SIP on IBM WebSphere

- Markov Availability Models of Midplane Subsystem

```
markov midplane
UP DN  c_mp*lambda_mp
UP U1  (1-c_mp)*lambda_mp
U1 RP  alpha_sp
RP UP  mu_mp
DN RP  alpha_sp
reward
  UP 1
  U1 1
end
end
```

SHARPE GUI representation

SHARPE textual representation
Availability model of SIP on IBM WebSphere

- Markov Availability Models of Cooling Subsystem

markov cooling
UP U1 2*lambda_c
U1 RP alpha_sp
U1 DN lambda_c
DN DW alpha_sp
RP UP mu_c
RP DW lambda_c
DW UP mu_2c
reward
UP 1
U1 1
RP 1
end
end

SHARPE GUI representation

SHARPE textual representation
Availability model of SIP on IBM WebSphere

- Markov Availability Models of CPU Subsystem

```
markov processor
UP D1 2*lambda_cpu
D1 RP alpha_sp
RP UP mu_cpu
reward
  UP 1
end
end
```
Availability model of SIP on IBM WebSphere

- Markov Availability Models of Memory Subsystem

**SHARPE GUI representation**

**SHARPE textual representation**

markov memory
UP D1 4*lambda_mem
D1 RP alpha_sp
RP UP mu_mem
reward
  UP 1
end
end
Availability model of SIP on IBM WebSphere

- Markov Availability Models of Power Domain Subsystem

![Markov Availability Model Diagram]

**SHARPE GUI representation**

**SHARPE textual representation**

```
markov power
UP U1 2*c_ps*lambda_ps
UP DN 2*(1-c_ps)*lambda_ps
U1 RP alpha_sp
U1 DN lambda_ps
DN DW alpha_sp
RP UP mu_ps
RP DW lambda_ps
DW UP mu_2ps
reward
UP 1
U1 1
RP 1
end
end
```
Availability model of SIP on IBM WebSphere

- Markov Availability Models of Base, Switch, NIC Subsystem

SHARPE GUI representation

<table>
<thead>
<tr>
<th></th>
<th>λ_base</th>
<th>α_sp</th>
<th>μ_base</th>
</tr>
</thead>
<tbody>
<tr>
<td>UP</td>
<td>D1</td>
<td>RP</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>λ_esw</th>
<th>α_sp</th>
<th>μ_esw</th>
</tr>
</thead>
<tbody>
<tr>
<td>UP</td>
<td>D1</td>
<td>RP</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>λ_nic</th>
<th>α_sp</th>
<th>μ_nic</th>
</tr>
</thead>
<tbody>
<tr>
<td>UP</td>
<td>D1</td>
<td>RP</td>
<td></td>
</tr>
</tbody>
</table>

SHARPE textual representation

markov base
UP DN lambda_base
DN RP alpha_sp
RP UP mu_base
reward
   UP 1
end
end

markov switch
UP DN lambda_esw
DN RP alpha_sp
RP UP mu_esw
reward
   UP 1
end
end

markov nic
UP DN lambda_nic
DN RP alpha_sp
RP UP mu_nic
reward
   UP 1
end
end
Availability model of SIP on IBM WebSphere

- Markov Availability Models of RAID Subsystem

**SHARPE GUI representation**

**SHARPE textual representation**

markov disk
UP U1 2*lambda_hd
U1 RP alpha_sp
RP CP mu_hd
CP DW lambda_hd
U1 DN lambda_hd
DN DW alpha_sp
CP UP chi_hd
DW UP mu_2hd
reward
UP 1
U1 1
CP 1
end
end
Availability model of SIP on IBM WebSphere

- Markov Availability Models of OS Subsystem

```
markov OS
UP DN lambda_os
DN DT delta_os
DT UP b_os*beta_os
DT DW (1-b_os)*beta_os
DW RP alpha_sp
RP UP mu_os
reward
  UP 1
end
end
```
Availability model of SIP on IBM WebSphere

- Markov Availability Models of AP Server

- Application server and proxy server (with escalated levels of recovery)
- Delay and imperfect coverage in each step of recovery modeled
- Note the use of restart and reboot as a method of mitigation

- Failure detection
  - By WLM
  - By Node Agent
  - Manual detection

- Recovery
  - Node Agent
    - Auto process restart
  - Manual recovery
    - Process restart
    - Node reboot
    - Repair
Availability model of SIP on IBM WebSphere

- Markov Availability Models of AP Server

Application server

<table>
<thead>
<tr>
<th>markov appserver</th>
<th>markov proxy</th>
</tr>
</thead>
<tbody>
<tr>
<td>UP UO gamma</td>
<td>UP UO pxy_gamma</td>
</tr>
<tr>
<td>UO UA e*delta2</td>
<td>UO UA pxy_e*pxy_delta</td>
</tr>
<tr>
<td>UO 1D d*delta1</td>
<td>UO 1N (1-pxy_d)*pxy_delta</td>
</tr>
<tr>
<td>UO 1N (1-d)*delta1</td>
<td>UO 2N (1-pxy_e)*pxy_delta</td>
</tr>
<tr>
<td>UO 2N (1-e)*delta2</td>
<td>1N UA pxy_e*pxy_delta</td>
</tr>
<tr>
<td>1N UA e*delta2</td>
<td>1N UN (1-pxy_e)*pxy_delta</td>
</tr>
<tr>
<td>1N UN (1-e)*delta2</td>
<td>2N UR pxy_d*pxy_delta</td>
</tr>
<tr>
<td>2N UR d*delta1</td>
<td>2N UN (1-pxy_d)*pxy_delta</td>
</tr>
<tr>
<td>2N UN (1-d)*delta1</td>
<td>UA UR (1-pxy_q)*pxy_rou_a</td>
</tr>
<tr>
<td>1D UA e*delta2</td>
<td>UR UB (1-pxy_r)*pxy_rou_m</td>
</tr>
<tr>
<td>1D UR (1-e)*delta2</td>
<td>UB RE (1-pxy_b)*pxy_beta_m</td>
</tr>
<tr>
<td>UA UR (1-q)*rou_a</td>
<td>UA UR pxy_d*pxy_delta</td>
</tr>
<tr>
<td>UR UB (1-r)*rou_m</td>
<td>1D UA pxy_e*pxy_delta</td>
</tr>
<tr>
<td>UB RE (1-b)*beta_m</td>
<td>1D UR (1-pxy_e)*pxy_delta</td>
</tr>
<tr>
<td>UN UR delta_m</td>
<td>UN UR pxy_delta</td>
</tr>
<tr>
<td>UA UP q*rou_a</td>
<td>UA UP pxy_q*pxy_rou_a</td>
</tr>
<tr>
<td>UR UP r*rou_m</td>
<td>UR UP pxy_r*pxy_rou_m</td>
</tr>
<tr>
<td>UB UP b*beta_m</td>
<td>UB UP pxy_b*pxy_beta_m</td>
</tr>
<tr>
<td>RE UP mu reward</td>
<td>RE UP pxy_mu reward</td>
</tr>
<tr>
<td>UP 1</td>
<td>UP 1</td>
</tr>
<tr>
<td>end</td>
<td>end</td>
</tr>
<tr>
<td>end</td>
<td>end</td>
</tr>
</tbody>
</table>

Proxy server
Hierarchical Composition

A single monolithic Markov model will have too many states
Top Level Fault Tree (cont’)

repeat Node_F prob(sysprob(BLADE))
repeat nic1_F prob(1-exrss(nic))
repeat nic2_F prob(1-exrss(nic))
or eth1_F nic1_F esw3
or eth2_F nic2_F esw4
and eth_F eth1_F eth2_F
or BS_F Node_F eth_F
repeat Node_G prob(sysprob(BLADE))
repeat nic1_G prob(1-exrss(nic))
repeat nic2_G prob(1-exrss(nic))
or eth1_G nic1_G esw1
or eth2_G nic2_G esw2
and eth_G eth1_G eth2_G
or BS_G Node_G eth_G
repeat Node_H prob(sysprob(BLADE))
repeat nic1_H prob(1-exrss(nic))
repeat nic2_H prob(1-exrss(nic))
or eth1_H nic1_H esw3
or eth2_H nic2_H esw4
and eth_H eth1_H eth2_H
or BS_H Node_H eth_H

*==========Application Servers==========*
or AS1 SW BS_A CM1
or AS2 SW BS_A CM1
or AS3 SW BS_B CM1
or AS4 SW BS_B CM1
or AS5 SW BS_C CM1
or AS6 SW BS_C CM1
or AS7 SW BS_D CM2
or AS8 SW BS_D CM2
or AS9 SW BS_E CM2
or AS10 SW BS_E CM2
or AS11 SW BS_F CM2
or AS12 SW BS_F CM2
kofn apps 6, 12, AS1 AS2 AS3 AS4 AS5 AS6 AS7 AS8 AS9 AS10 AS11 AS12
*==========Proxy Servers==========*
or PX1 SWP BS_G CM1
or PX2 SWP BS_H CM2
and pxys PX1 PX2
or top apps pxys
end
Availability model of SIP on IBM WebSphere

- Sharpe textual representation to obtain output measures

```
format 8
include param_sharpe.txt
include models.txt
bind
  gamma        gamma_hang + gamma_die
  pxy_gamma    pxy_gamma_hang + pxy_gamma_die
end
expr sysprob(CLUSTER)
end
```

- Output measure obtained

```
C:\Sharpe-Gui\sharpe>sharpe avail.txt
sysprob<CLUSTER>:  1.96059916e-006
```
Our Contributions (1)

- Developed a very comprehensive availability model
  - “Discovered the Software failure/recovery architecture
  - Hardware and software failures
  - Hardware and Software failure-detection delays
  - Software Detection/Failover/Restart/Reboot delay
  - Escalated levels of recovery
    - Automated and manual restart, failover, reboot, repair
  - Imperfect coverage (detection, failover, restart, reboot)
Our Contributions (2)

- Developed a new method for calculating DPM (defects per million calls) (IBM is filing for a patent on this algorithm)
  - Taking into account interactions between call flow and failure/recovery & Retry of messages
- Many of the parameters collected from experiments
- Detailed sensitivity analysis to find bottlenecks and give feedback to designers
- This model made the sale of this system to the Telco customer
Parameterization

- Hardware/Software Configuration parameters
- Hardware component MTTFs
- Hardware/Software Detection/Failover/Restart/Reboot times
- Repair time
  - Hot swap, multiple components at once, field service travel time
- Software component MTTFs (experiments have started for this)
  - OS, WAS, SIP/Proxy
- Coverage (Success) probabilities
  - Detection, restart, failover, reboot, repair
- Validation
Outline

- An Introduction to SHARPE software tool
  - Reliability/Availability models in practice
- Performance models in practice
- Performability models in practice
- Possible outputs
- Case studies
Reliability models in practice

Creation of a model

Analysis

Finding bottlenecks
Breakdown of unreliability by causes

Reliability
Mean time to failure
Availability models in practice

Creation of a model

Analysis

Availability
Unavailability
Downtime
Cost of Downtime

Breakdown of Downtime
- Hardware
- Software
- Software Upgrades
- Preventive Maintenance
MTTSF, MTTSR

Expected interval availability
Outline

- An Introduction to SHARPE software tool
- Reliability/Availability models in practice
  - Performance models in practice
- Performability models in practice
- Possible outputs
- Case studies
Performance models in practice

Creation of a model

Analysis

Steady-state probability of a node
Expected steady-state reward rate

Throughput
Average response time
Average queue length
Utilization
Outline

- An Introduction to SHARPE software tool
- Reliability/Availability models in practice
- Performance models in practice
  - Performability models in practice
- Possible outputs
- Case studies
Performability models in practice

Creation of a model

Analysis

Total loss probability
Capacity oriented availability
Throughput oriented availability
Outline

- An Introduction to SHARPE software tool
- Reliability/Availability models in practice
- Performance models in practice
- Performability models in practice
  - Possible outputs
- Case studies
Possible outputs

- Availability, Unavailability and Downtime
- Cost of downtime
- Mean Time to System Failure, Mean Time to System Repair
- Downtime breakdown into Hardware, Software & Upgrade
- Breakdown of downtime by states for Markov chain models, by blocks for Reliability block diagram models.
- Sensitivity Analysis, Strategy to improve the availability of the systems.
Outline

- An Introduction to SHARPE software tool
- Reliability/Availability models in practice
- Performance models in practice
- Performability models in practice
- Possible outputs
  - Case studies
Some Recent Uses

- Scalable Performance, availability and power models of IBM Research Cloud
- Boeing 787 Current Return Network Reliability model
- Motorola Cable Modem Termination system
- IBM Bladecenter availability model
- IBM SIP on Websphere availability/DPM model