TOOLS 2003 Tutorial

September 2, 2003

Urbana, IL, USA

Software Rejuvenation: Modeling and Analysis

Kishor S. Trivedi
Center for Advanced Computing & Communication
Dept. of Electrical & Computer Engineering
Duke University
Durham, NC 27708
kst@ee.duke.edu

Kalyan Vaidyanathan
RAS Computer Analysis Laboratory
Sun Microsystems
San Diego, CA 92121
kalyan.vaidyanathan@sun.com

Outline

- Introduction
  - Software reliability and fault tolerance
- Software Aging and Rejuvenation
- Analytic Models
  - CTMC model
  - MRSPN model
  - SMP model
  - Transaction processing system
  - Cluster systems – SRN model
- Measurement-based Models
  - Time-based
  - Time and workload-based
  - Time series and ARMA
- Software Rejuvenation in a Commercial Server
- Summary and Conclusions

Introduction
Applications

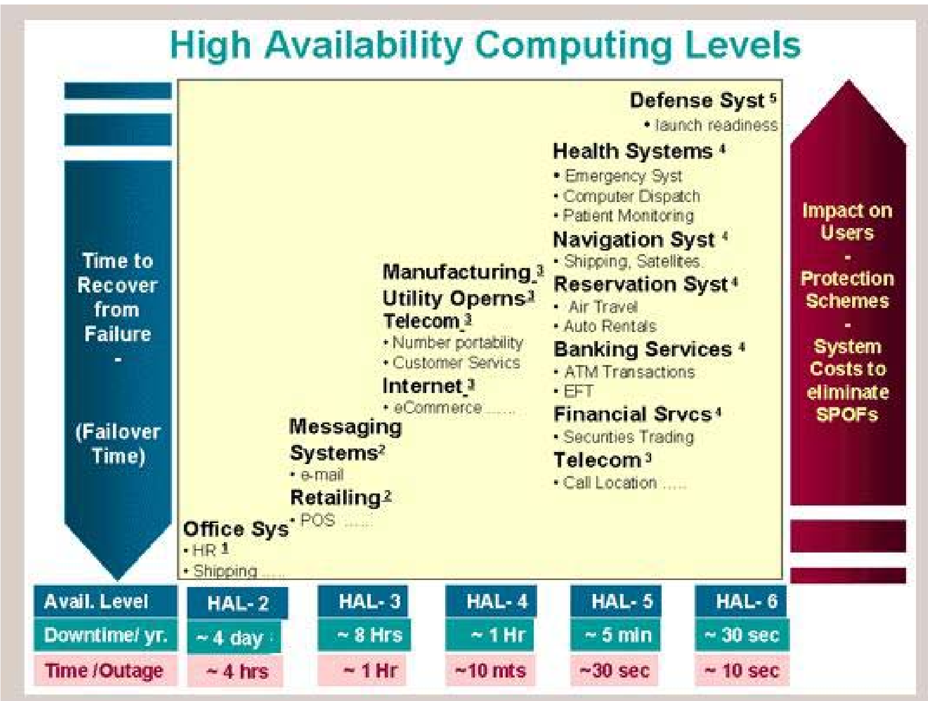
- Traditional applications

  (long-life/life-critical/safety-critical)
  - Space missions, aircraft control, defense, nuclear systems
- New applications

  (non-life-critical/non-safety-critical, business critical)
  - Banking, airline reservation, e-commerce applications, web-hosting, telecommunication
- Scientific applications

  (non-critical)

Introduction
Motivation – High Availability



Industry	Business Operation	Industry Cost Range (per hour)	Average Cost Per Hour of Downtime
Financial	Brokerage Operations	$5.6M to $7.3M	$6.45M
Financial	Credit Card/Sales Authorization	$2.2M to $3.1M	$2.6M
Media	Pay-Per-View	$233K	$150K
Retail	Home Shopping Network (TV)	$87K to $140K	$113K
Retail	Home Catalog Sales	$60K to $120K	$90K
Transportation	Airline Reservations	$67K to $112K	$89.5K
Media	Tee-ticket Sales	$56K to $82K	$69K
Transportation	Package Shipping	$24K to $32K	$28K
Finance	ATM Fees	$12K to $17K	$14.5K

Introduction

Software is the problem

- Hardware FT relatively well developed
- System outages more due to software faults
- Software reliability is one of the weakest links in system reliability
- Fault avoidance through good software engineering practices difficult for large/complex software systems
- Impossible to fully test and verify if software is fault-free
- Stringent requirements for failure-free operation

→ **Software fault tolerance** is a potential solution to improve software reliability in lieu of virtually impossible fault-free software

Software Fault Classification

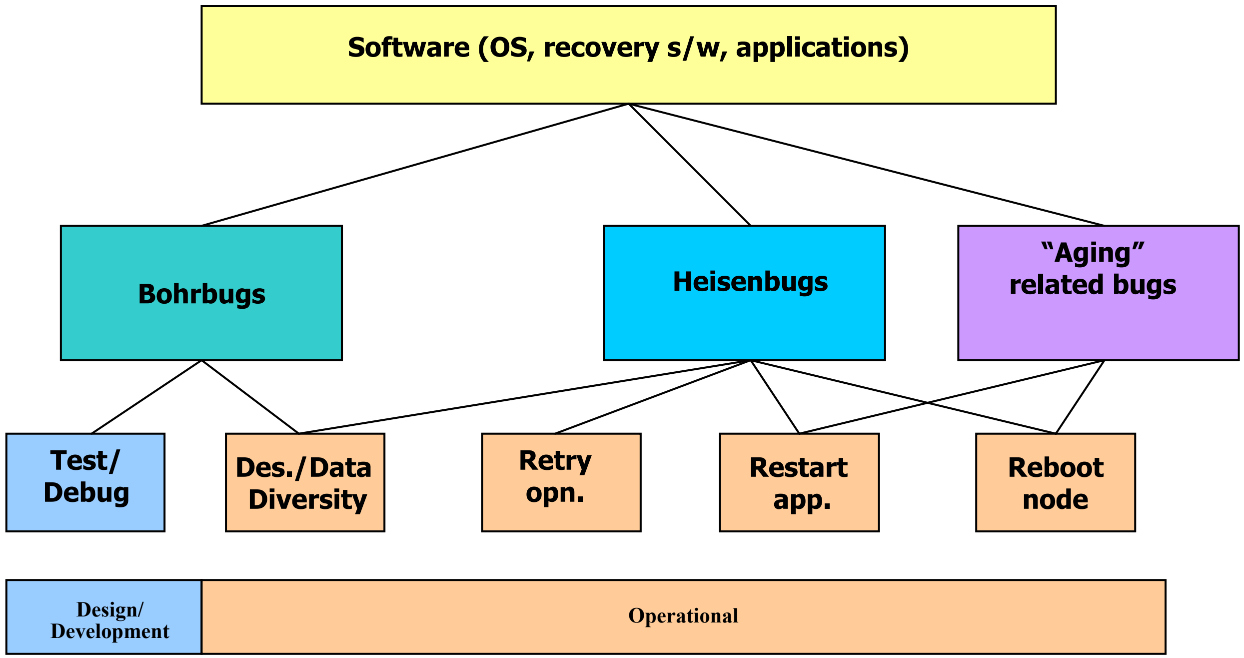
- Many software bugs are reproducible, easily found and fixed during the testing and debugging phase

Bohrbugs

- Other bugs that are hard to find and fix remain in the software during the operational phase
  - may never be fixed, but if the operation is retried or the system is rebooted, the bugs may not manifest themselves as failures
  - manifestation is non-deterministic and dependent on the software reaching very rare states

Heisenbugs

Software Fault Classification

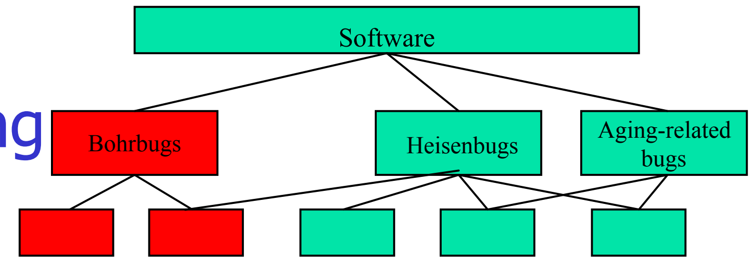


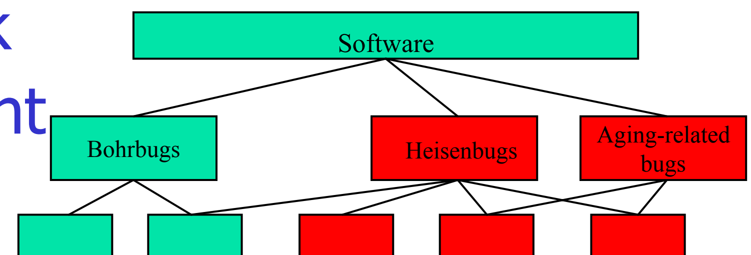
Dimensions of Software Reliability

FAULT	Bohrbugs	Heisenbugs	Aging-related bugs
LAYER	Operating System	Middleware (Recovery Software)	Application Software
REDUNDANCY	No redundancy	Time redundancy	
		Replication	
		Diversity	
		Information redundancy	
PHASE	Testing/Debugging phase	Operational phase	
MODEL	Measurement-based model	Analytical/Simulation model	

Software Fault Tolerance
Techniques

- Design diversity
  - N-version programming
  - Recovery block
  - N-self check programming
- Data diversity
  - N-copy programming
- Environment diversity
  - Checkpointing and rollback
  - Proactive fault management





Software Aging
Definition, causes and manifestation

- Error conditions accumulating over time
  - Leading to performance degradation/crash
- Not related to application program becoming obsolete due to changing requirements/maintenance
- What constitutes aging?
  - Deterioration in the availability of OS resources, data corruption, numerical error accumulation
- How does it manifest itself?
  - Performance degradation, transient failure
- Common examples
  - Memory leaks, data corruption, fragmentation

Software Aging
Examples

- Netscape, xrn, Windows 9x
- File system aging [Smith & Seltzer]
- Crash/hang failures in general purpose applications
- Gradual service degradation in the AT&T transaction processing system [Avritzer et al.]
- Error accumulation in Patriot missile system's software [Marshall]
- Resource exhaustion in Apache [Li et al.]

Environment Diversity

New Approach to S/W FT

- Transient nature of software failures
  - **[Gray] Bohrbugs and Heisenbugs**
  - **[Lee & Iyer] Tandem GUARDIAN – 70% transient faults**
  - **[Sullivan & Chillarege] IBM's system software – most failures caused by peak conditions in workload, timing and exception errors**
- Environmental Diversity
  - Allows the use of time redundancy over expensive design diversity
    - **[Adams] [Grey] [Siewiorek] Restart**
    - **[Jalote et al.] Rollback, rollforward**
    - **[Wang et al.] Progressive retry**
    - **[folklore] Occasional reboot, "switch off and on"**

    Reactive in approach
  - Proactive approach
    - **Software rejuvenation**

Software Rejuvenation

Definition

- Proactive fault management technique aimed at preventing crash failures and/or performance degradation
  - Involves occasionally stopping the running software, "cleaning" its internal state and restarting it
- Counteracts the aging phenomenon
  - Frees up OS resources
  - Removes error accumulation
- Common examples
  - Garbage collection, defragmentation, flushing kernel and file server tables etc

Software Rejuvenation

Examples

- AT&T billing applications [Huang et al.]
- Software capacity restoration [Avritzer et al]
- On-board preventive maintenance for long-life deep space missions (NASA's X2000 Advanced Flight Systems Program) [Tai et al.]
- Patriot missile system software - switch off and on every 8 hours [Marshall]
- IBM Director Software Rejuvenation
- Microsoft IIS 5.0 process recycling tool
- Process restart in Apache

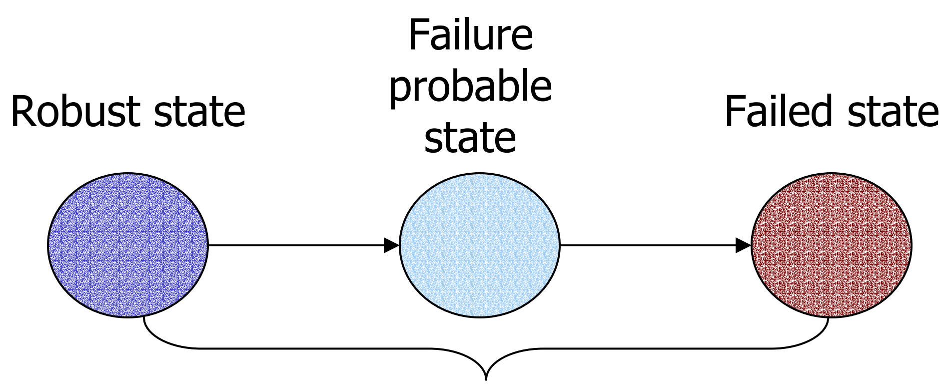
Software Rejuvenation

Research issue and previous work

- Rejuvenation incurs an overhead (downtime, performance, lost transactions etc.)
  - Important research issue to find optimal times to perform rejuvenation
- Two approaches
  - Analytical Modeling
    - Lucent Bell Labs [Huang et al., '95]
    - Duke [IEEE-TC '98, SIGMETRICS '96, ISSRE '95, SIGMETRICS '01, SRDS '02]
    - Others [IPDS '98, PNPM '99]
  - Measurement-based rejuvenation
    - Duke [ISSRE '98, ISSRE '99, IBMJRD '01, ISESE'02]

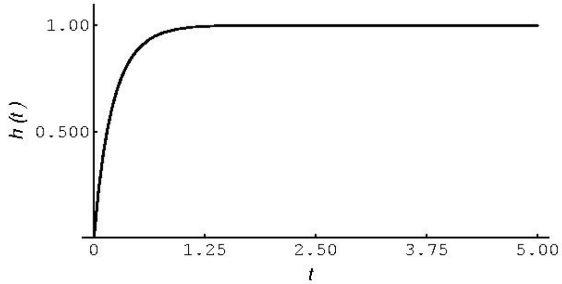
Analytic Models
Software Aging and Rejuvenation

- **Preventive maintenance is useful only if failure rate is increasing**
- **A simple and useful model of increasing failure rate:**

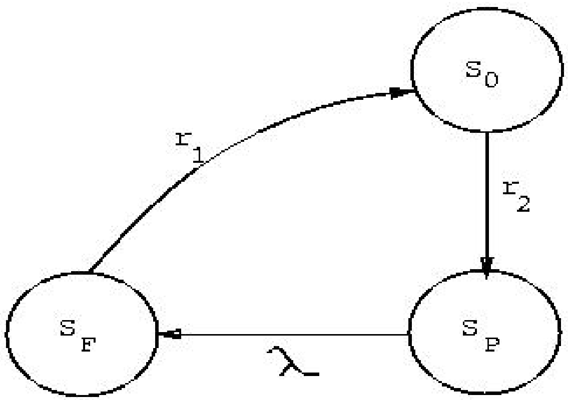


Time to failure: Hypo-exponential distribution

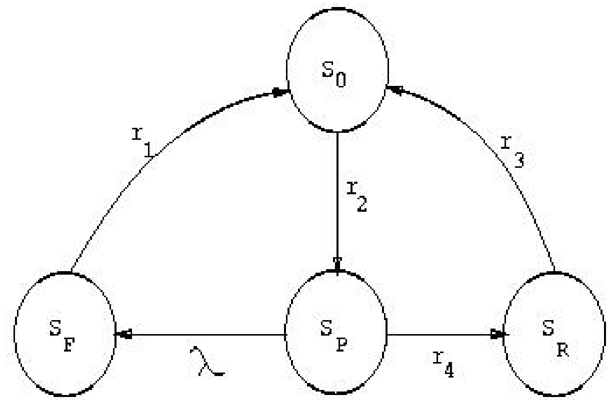
Increasing failure rate → aging



Analytic Models
CTMC model [Huang95]



Model w/o rejuvenation



Model with rejuvenation

- From this Continuous-time Markov chain model
- Can find closed-form expression for the optimal rejuvenation trigger rate (r_4)

Analytic Models

CTMC model (Huang95)

The expected total downtime of the application A with rejuvenation in an interval of L time units is:

$$DownTime_{A^r}(L) = (p_f + p_r) \times L$$

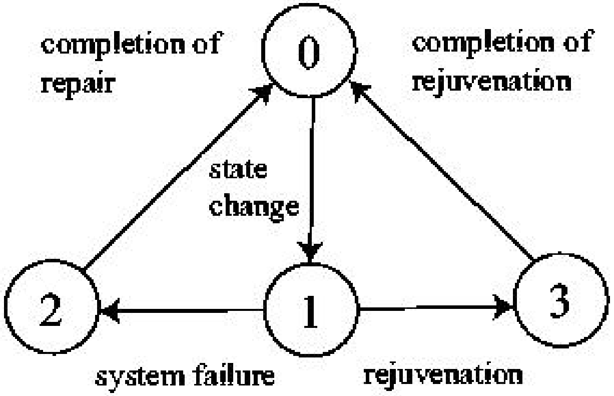
$$DownTime_{A^r}(L) = L \times \frac{\frac{\lambda}{r_1} + \frac{r_4}{r_3}}{1 + \frac{\lambda}{r_1} + \frac{\lambda}{r_2} + \frac{r_4}{r_2} + \frac{r_4}{r_3}}$$

If c_f is the average per unit cost of unscheduled downtime as before and c_r is the average per unit cost of downtime during rejuvenation, then the total expected downtime cost in an interval of L time units is:

$$\begin{aligned} Cost_{A^r}(L) &= (p_f \times c_f + p_r \times c_r) \times L \\ &= \frac{L}{1 + \frac{\lambda}{r_1} + \frac{r_4}{r_3} + \frac{\lambda + r_4}{r_2}} \\ &\quad \times \left(\frac{\lambda}{r_1} \times c_f + \frac{r_4}{r_3} \times c_r\right) \end{aligned}$$

Analytic Models
Semi-Markov model [Dohi00]

- Relax the assumption of exponentially distributed sojourn times (time-independent transition rates)
- Hence have a semi Markov model



- Can find closed-form expression for the optimal (deterministic) time to rejuvenation trigger

Analytic Models
Semi-Markov model (Dohi00)

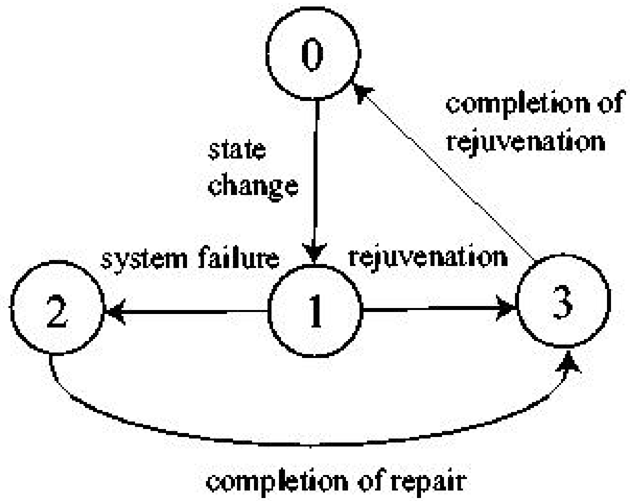


Figure 2. Semi-Markovian diagram of Model 2

$$A_2(t_0) = \frac{\mu_0 + \int_0^{t_0} \overline{F}_f(t)dt}{\mu_0 + \mu_c + \mu_a F_f(t_0) + \int_0^{t_0} \overline{F}_f(t)dt}$$
$$= S_2(t_0)/T_2(t_0).$$

Analytic Models
MRSPN model [Garg95]

- By allowing the rejuvenation trigger clock to start in the robust state, we obtain a Markov Regenerative Process

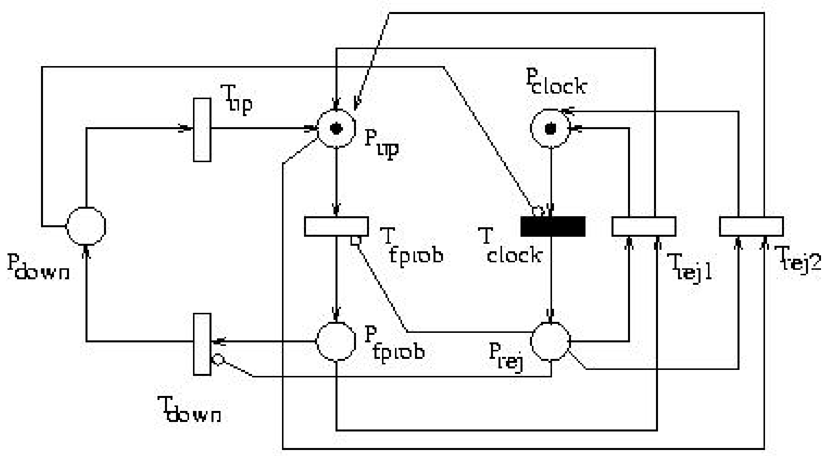


Figure 1: MRSPN Model of Software Rejuvenation

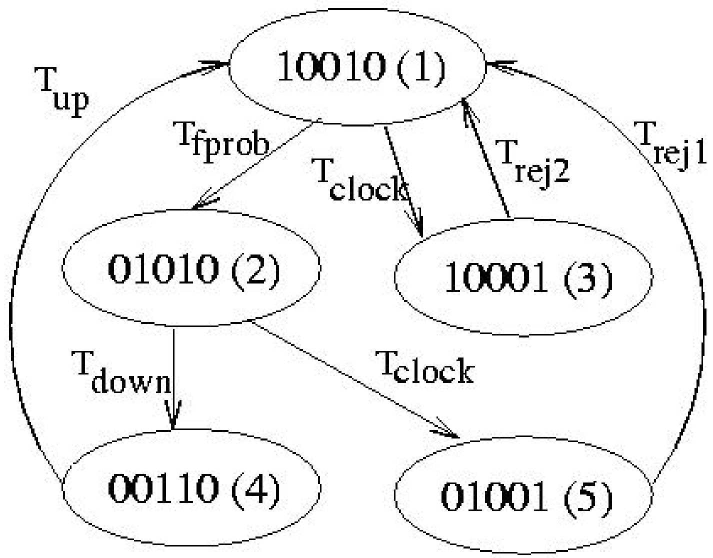


Figure 2: Reachability Graph for the MRSPN Model

Analytic Models
MRSPN model [Garg95]

- Optimal time (deterministic) to rejuvenation trigger is determined numerically

Parameter	*Value*
λ_1^{-1}	240 hrs.
λ_2^{-1}	2160 hrs.
λ_3	6 /hr.
λ_4	2 /hr.
λ_5	6 /hr.

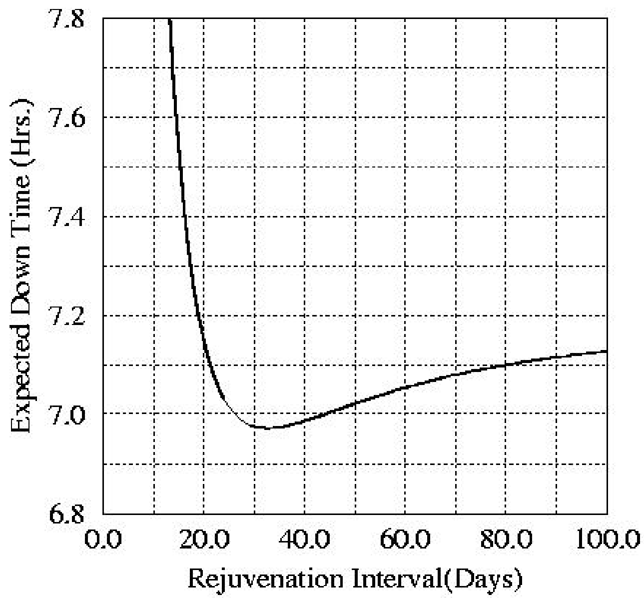
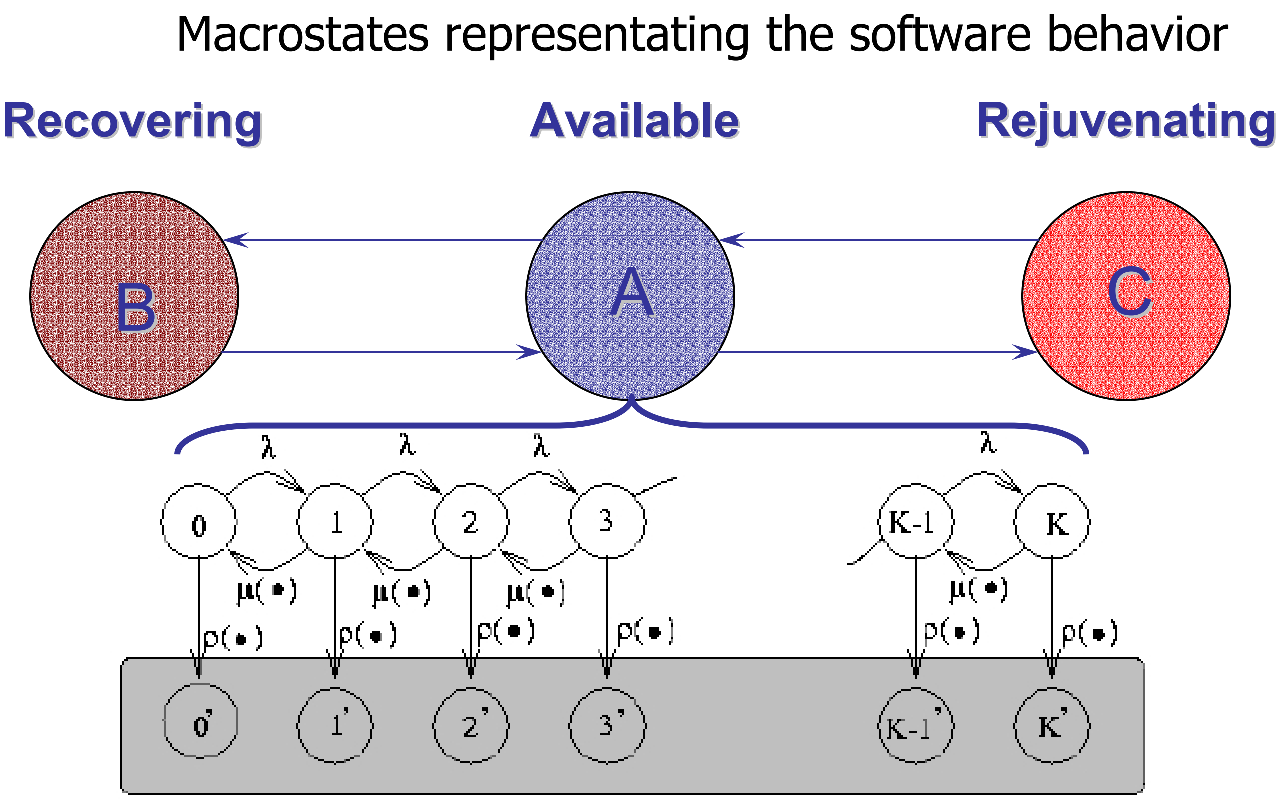


Figure 4: Steady state expected down time versus rejuvenation interval

A Comprehensive MRGP Model

Transaction Based Software System [Garg98]

- Adding details of transaction arrivals and loss



Model Assumptions

- Server type software, transactions arrive at rate λ
- Service rate $\mu(*)$ is arbitrary function measured from the last renewal of the software. Transaction that starts service at time *t* occupies the server for time with distribution

$$1-e^{-\int_{t_1}^{t}\mu(*)dt}$$

- Arriving transactions are queued and the buffer size is K
  - In the absence of failure, the system is an M/M(t)/1/K queue
- Service discipline is FCFS
- Software fails with rate $\rho(*)$. Distribution of time to failure, X, is

$$1-e^{-\int_{0}^{t}\rho(*)dt}$$

Model Assumptions (contd.)

- Time to recover from failure Y_f is generally distributed random variable with expectation

$$\gamma_f = E[Y_f]$$

- Time to perform rejuvenation Y_r is generally distributed random variable with expectation

$$\gamma_r = E[Y_r]$$

- Any transactions in the queue at the time of failure or at the time of initiation of rejuvenation are lost
- Any transactions that arrive while software is recovering or undergoing rejuvenation are also lost

Stochastic Model

- Stochastic process $\{Z(t), t \geq 0\}$ represents the state at time *t*
  - available for service (state A)
  - recovering from failure (state B)
  - undergoing rejuvenation (state C)

- Software behavior as a whole is modeled via $\{Z(t), N(t), t \geq 0\}$
  - If $Z(t) = A$ then $N(t) \in \{0,1,\ldots,K\}$
  - If $Z(t) \in \{B, C\}$ then $N(t) = 0$

- $\{Z(t), N(t), t \geq 0\}$ is a Markov regenerative process,
  - transition to state A from either B or C constitutes regeneration instant

Effects of Aging

- The two effects of aging
  - Performance degradation
  - Crash/hang failure
- can be captured by
  - Decreasing service rate, $\mu(*)$
  - Increasing failure rate, $\rho(*)$

Effects of Aging (Contd.)

- Decrease or increase respectively can be a function of time - $\mu(t)$, $\rho(t)$
  - $\mu(t)=\mu$ software system with no performance degradation
  - $\rho(t)=0$ software system with no crash / hang failures

- instantaneous load - $\mu(N(t))$ and $\rho(N(t))$
  - the value of service and failure rate at time *t* depend on *N(t)*

    the number of transactions in the queue at that time

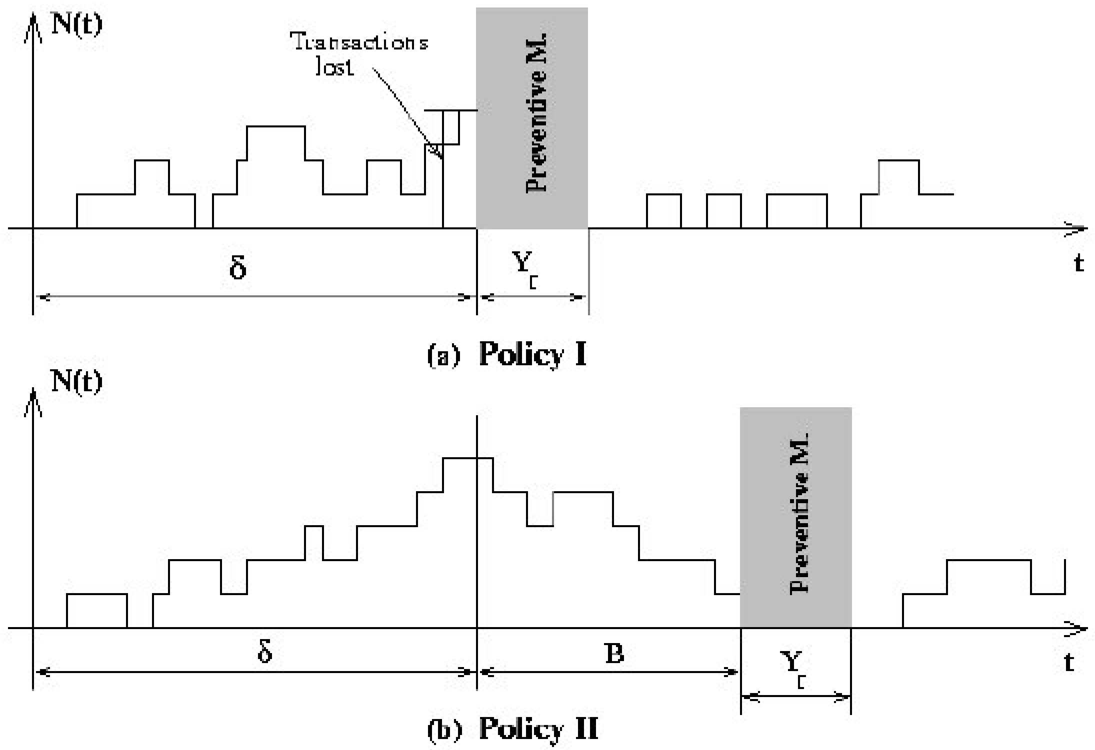
- average time that software is busy - $\mu(L(t))$ and $\rho(L(t))$
  - since idle software is not likely to age, service and failure rates are more realistically modeled as a functions of actual processing time rather than the total available time

- combination of the above

Rejuvenation Policies

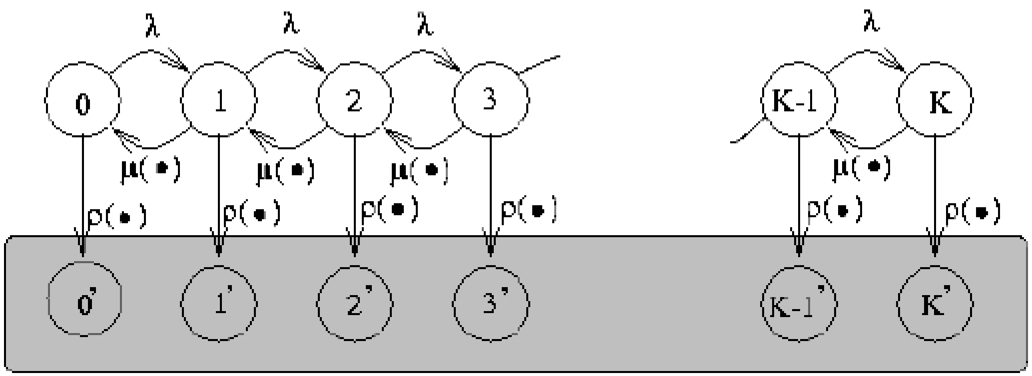
<u>Policy I:</u> Purely time-based

<u>Policy II:</u> Instantaneous load and time-based



Behavior in macro state A under policy I

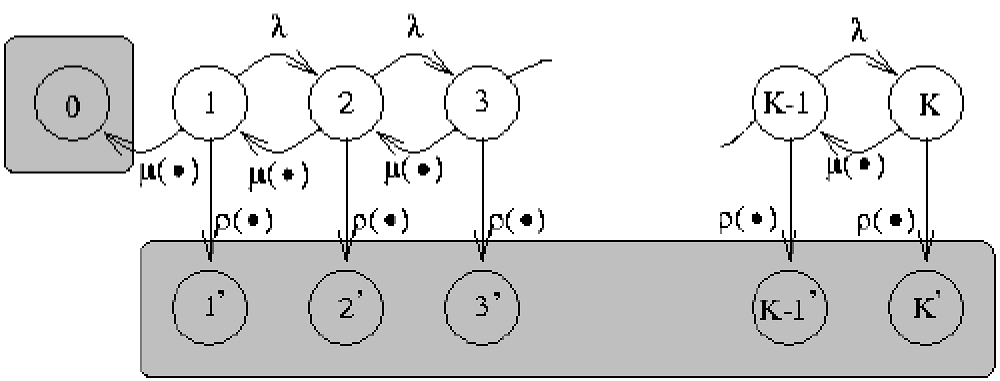
- The process is terminated either by
  - failure (which can happen at any time) or by
  - initiating software rejuvenation at $t = \delta$



Subordinated non-homogeneous CTMC for $t \leq \delta$

Behavior in macro state A under policy II

- Subordinated non-homogeneous CTMC for $t \leq \delta$ is same as under policy I
- Subordinated non-homogeneous CTMC for $t > \delta$



Solution Method

- Transient probabilities $p_i(t)$ and $p_i^{\prime}(t)$ for the subordinated non-homogeneous CTMC can be obtained by solving the system of forward differential-difference equations
- In general they do not have a closed-form analytical solutions and must be evaluated numerically
- Once these probabilities are obtained, compute quantities such as
  - steady state probabilities of the embedded DTMC - $[\pi_A, \pi_B, \pi_C]$
  - expected sojourn time is state A - *E[U]*
  - expected number of transactions in the buffer when the system is exiting state A - $E[N_l]$

Measures obtained

- Steady state availability *A*
- Long run probability of loss of a transaction P_{loss}
- Expected response time of a transaction given that it is successfully served T_{res}
- The goal is to determine optimal values of δ based on constraints on one or more of these measures

Steady State Availability

- Steady state availability obtained using standard formulae from Markov regenerative processes
- A = Pr {software is in state A} =

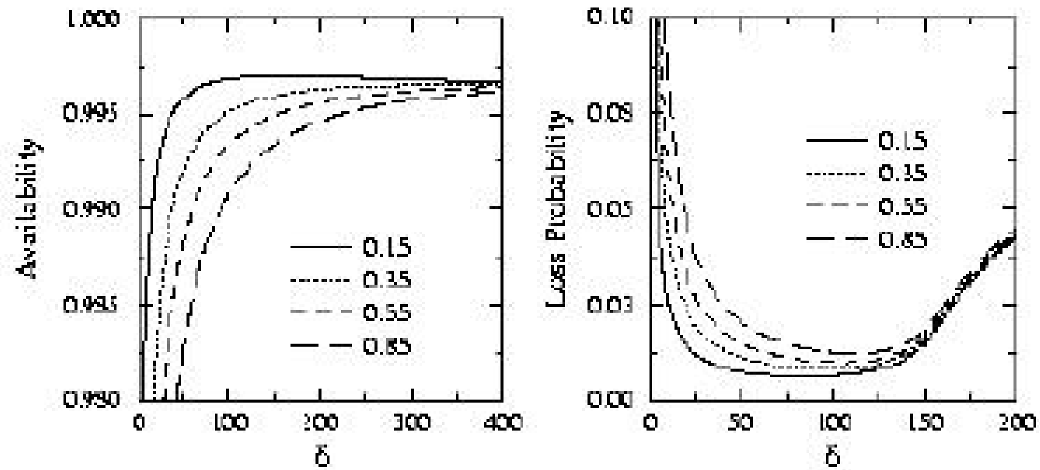
$$\frac{\pi_A E[U]}{\pi_B \gamma_f + \pi_C \gamma_r + \pi_A E[U]}$$

Probability of Loss

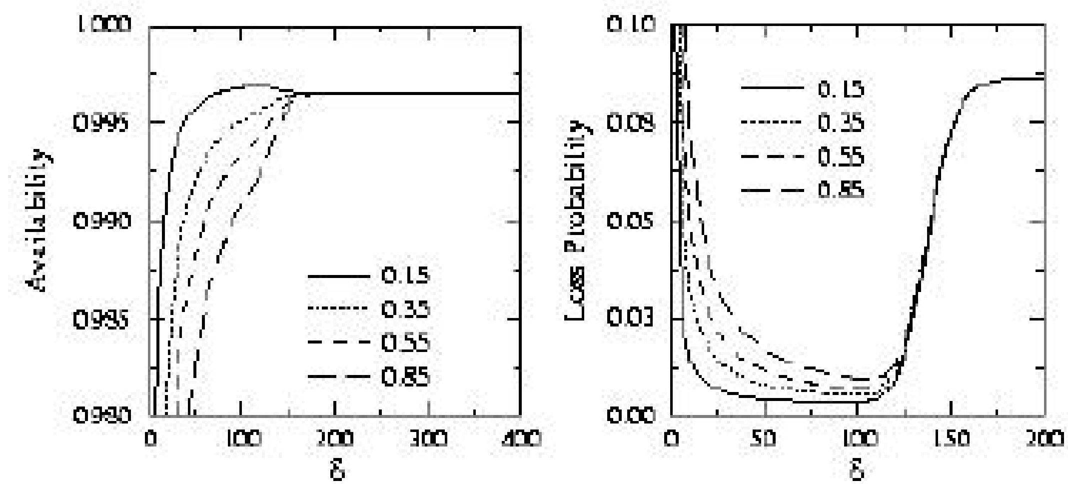
- Probability that transaction is lost is defined as a ratio of the expected number of transactions lost in an interval to the expected total number of transactions which arrive during that interval
- The number of transactions lost is summation of
  - transactions in the queue when the system is exiting state A
  - transactions that arrive while failure recovery or rejuvenation is in progress
  - transactions that are disregarded due to the buffer being full

Numerical Example 1

- Service rate and failure rate are functions of time, $\mu(t)$ and $\rho(t)$
- Vary mean time to perform rejuvenation



Policy I



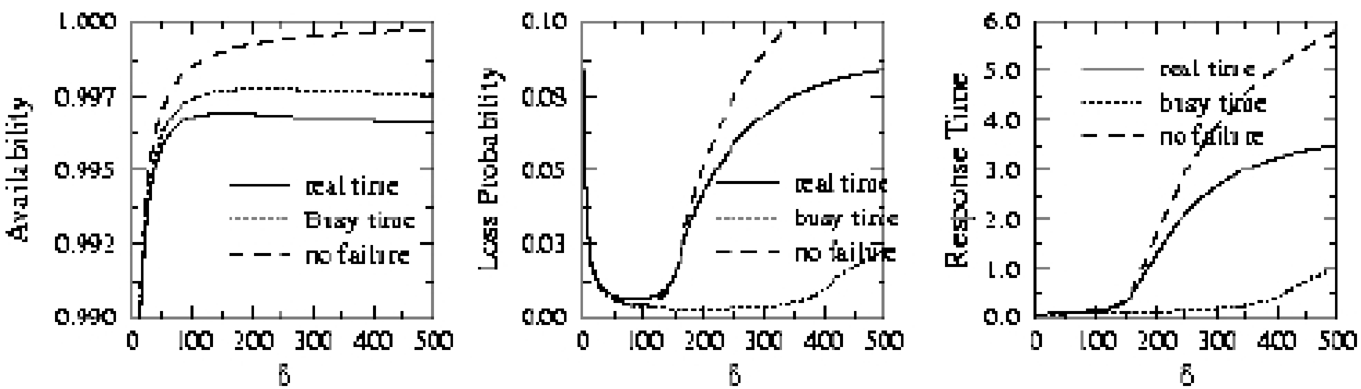
Policy II

Numerical Example 1 (Contd.)

- Service rate and failure rate are functions of time, $\mu(t)$ and $\rho(t)$
- For any particular δ, availability is lower and loss probability is higher for higher values of mean time to perform software rejuvenation γ_r
- Value of δ that minimizes loss probability is much lower than the one that maximizes availability
- If the objective is to maximize availability, it is better not to perform rejuvenation for higher values of γ_r (*0.55 or 0.85*)
- If the objective is to minimize probability of loss, policy II fares better than policy I

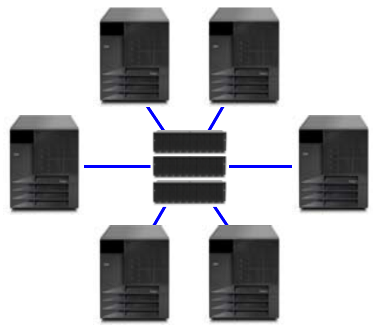
Numerical Example 2

- $\mu(*)$ and $\rho(*)$ are functions of real time $\mu(t)$ and $\rho(t)$
- $\mu(*)$ and $\rho(*)$ are functions of busy time $\mu(L(t))$ and $\rho(L(t))$
- no failures $\rho(*) = 0$, service degradation $\mu(*) = \mu(t)$



- Measures vary in a wide range depending on the forms chosen for the service rate $\mu(*)$ and failure rate $\rho(*)$

Cluster Systems



Cluster System

- [Pfister] Collection of independent, self-contained computer systems working together to provide a more reliable and powerful system than a single node by itself
- Easier scaling to larger systems, high levels of availability/performance and low management costs
- No single point of failure
- Node failures transparent to users
- Graceful repairs, shutdowns, upgrades

Rejuvenation for Cluster Systems

Motivation

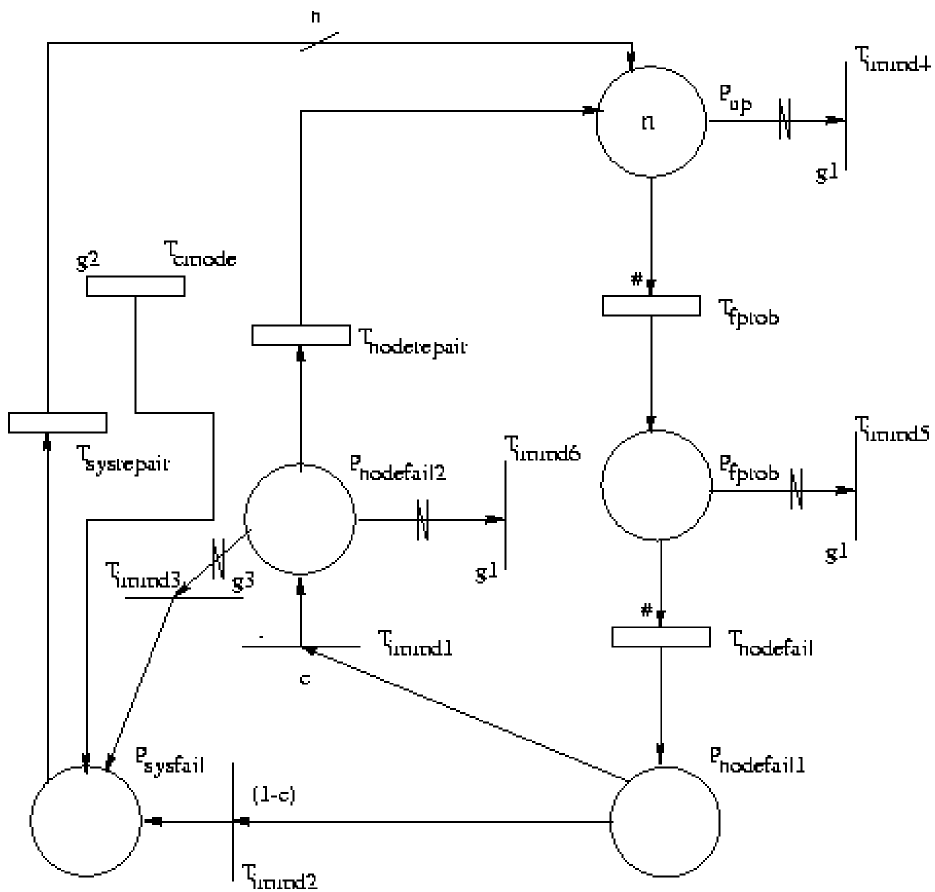
- Rejuvenation using the fail-over mechanisms
- Long-terms benefits in terms of availability/performance
- Continuous operation (possibly at a degraded level)
  - Practically zero downtime
- Less disruptive and lower overhead than unplanned outages
- Transparent to user/application
- Most current industry initiatives *reactive*
- Two approaches
  - Simple time-based (periodic)
  - Condition-based (only from the "failure-impending" state)

Rejuvenation for Cluster Systems
SRN Models

- Rejuvenation using the fail-over mechanisms in a rolling fashion
- Modeling using SRNs (Stochastic Reward Nets)
- Analysis for 2 rejuvenation policies
  - Simple time-based policy
    - All nodes rejuvenated successively at the end of each rejuvenation interval
  - Condition-based policy
    - Nodes rejuvenated only from the "failure-probable" state
- Various configurations
  - a/b: cluster with a nodes that can tolerate at the most b individual node failures, i.e., (a-b)-out-of-a system
- Model solution
  - SPNP (Stochastic Petri Net Package)

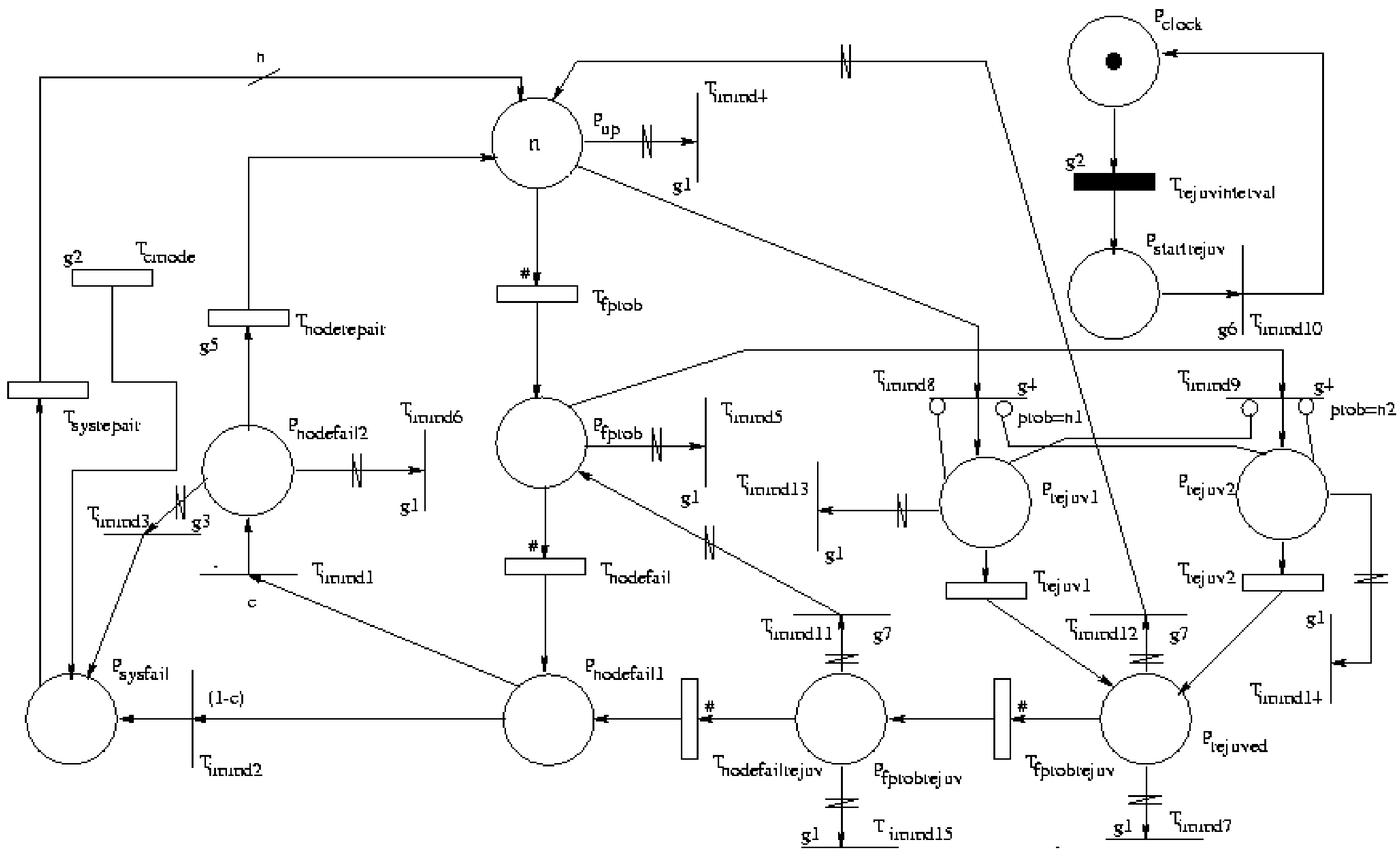
SRN Model

Basic Cluster Model



SRN Model

Simple Time-Based Rejuvenation



Model Parameters

Transition	Mean time
T_{fprob}	240 hours
$T_{nodefail}$	720 hours
$T_{noderepair}$	30 mins
$T_{sysrepair}$	4 hours
T_{rejuv}	10 mins

$cost_{nodefail}$	\$5000/hour
$cost_{noderejuv}$	\$250/hour

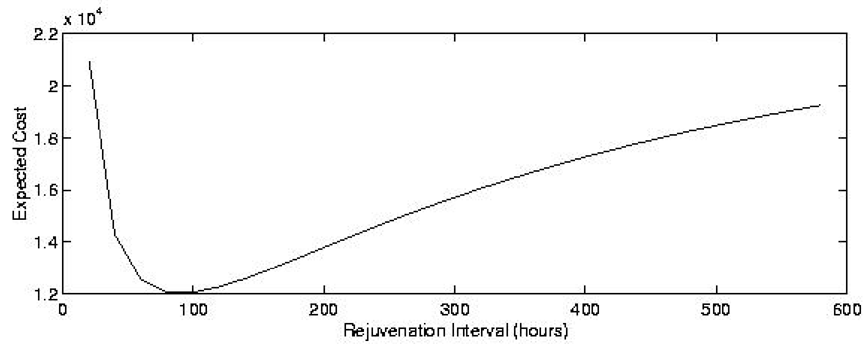
Measures Computed

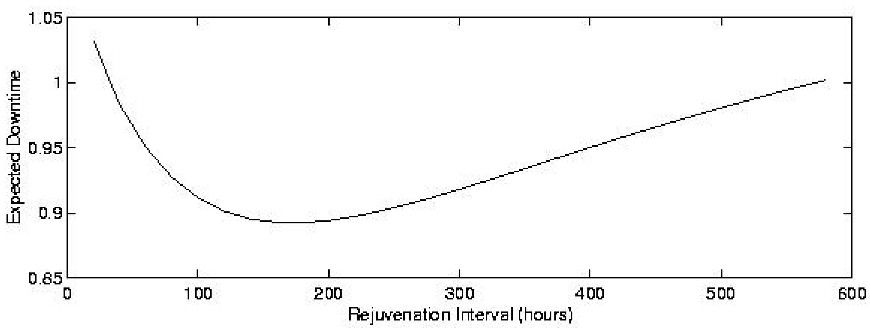
Unavailability	$(\#P_{sysfail} == 1)\ ?\ 1 : 0$
Cost	$\#P_{rejuv} * cost_{rejuv} + \#P_{nodefail} * cost_{nodefail} + \#P_{sysfail} * cost_{sysfail}$

Results

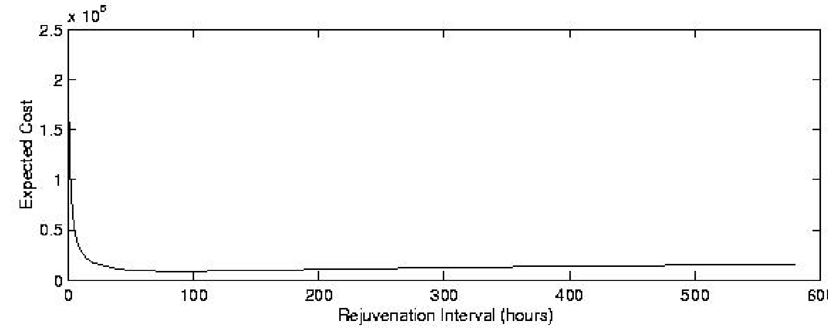
Simple Time-Based Rejuvenation

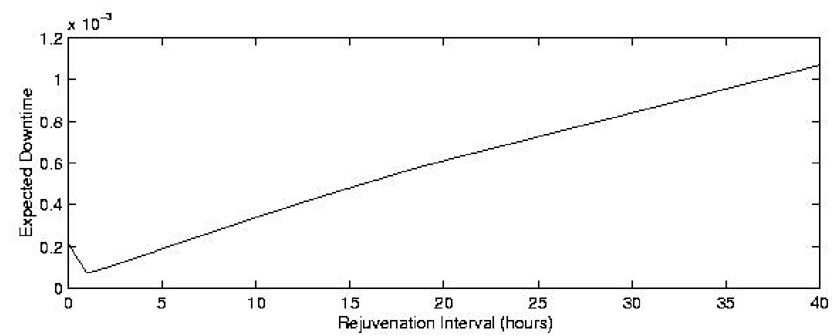
8/1 configuration





8/2 configuration



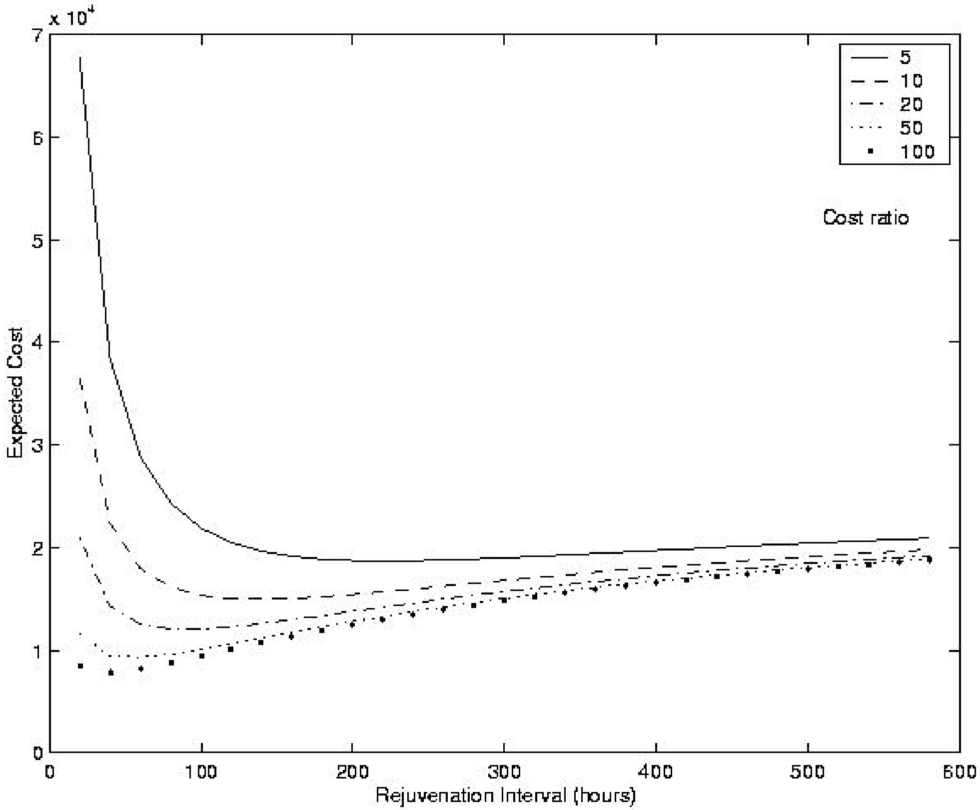


- As rejuv. int. increases, rejuvenation is performed less frequently
- When rejuv int is close to zero, the system is rejuvenating very frequently resulting in high cost/downtime
- When rejuv. int. goes beyond optimal value, system failures become frequent resulting in high cost/downtime

Results

Simple Time-Based Rejuvenation

Effect of $cost_{nodefail}/cost_{rejuv}$ for the 8/1 configuration

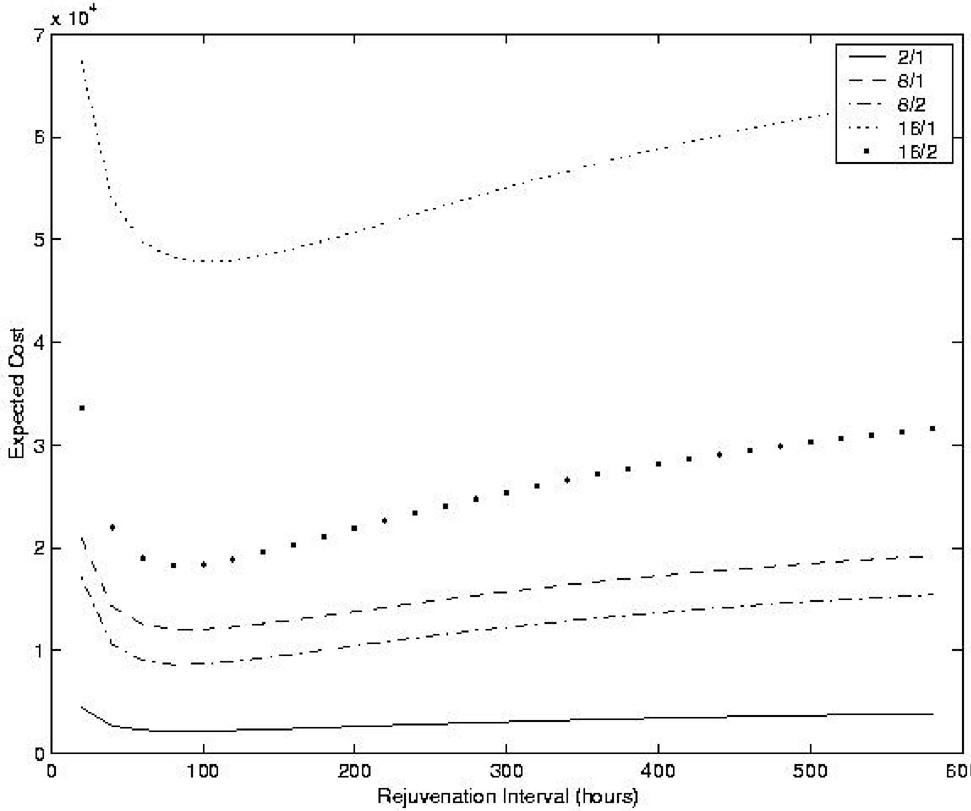


- •Cost of node failure is fixed
- •Decrease in cost ratio implies increase in cost of rejuvenation
- •Hence, decrease in cost ratio increases total expected cost
- •As rejuvenation interval increases, rejuvenation is performed less frequently
- •As rejuvenation tends to infinity, almost no rejuvenation is performed and all the plots tend to the same value

Results

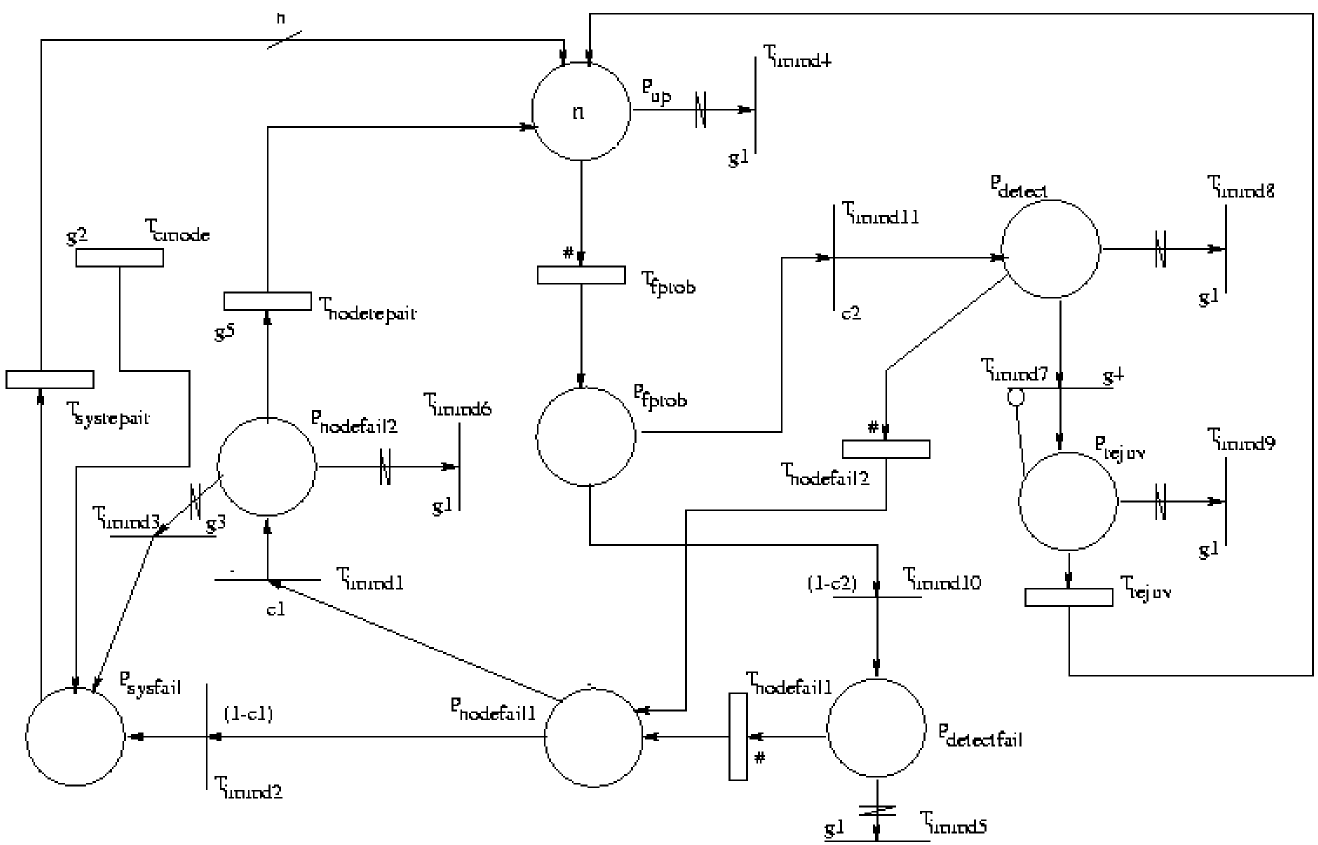
Simple Time-Based Rejuvenation

Expected cost for various configurations



SRN Model

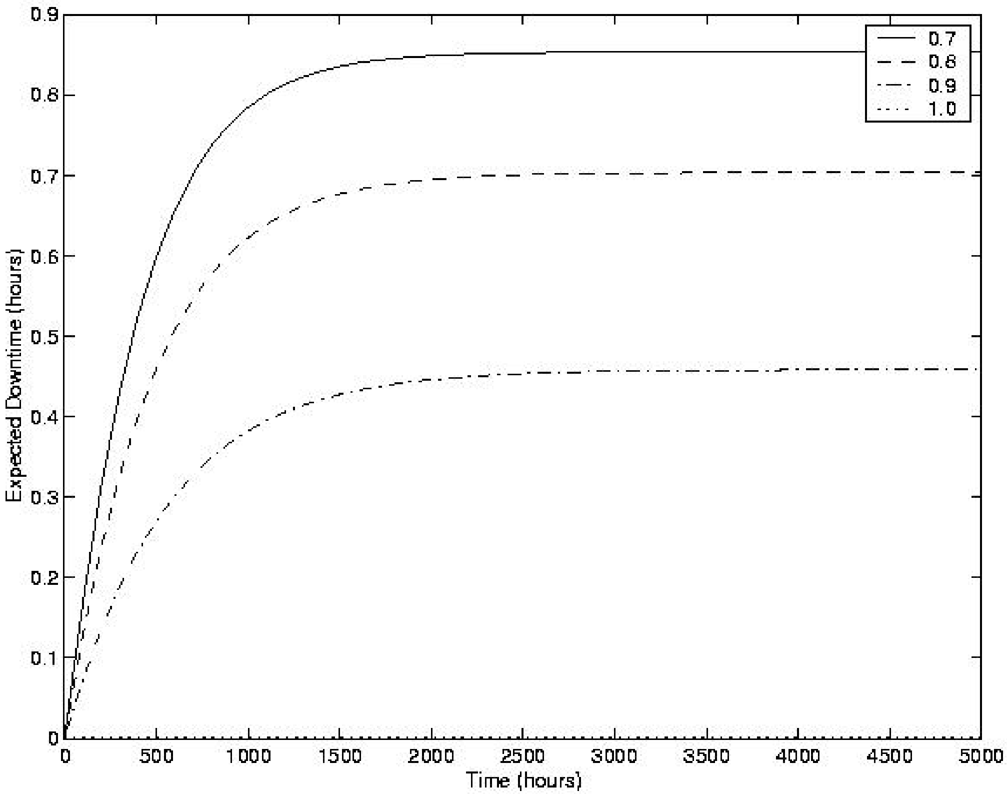
Condition-Based Rejuvenation



Results

Condition-Based Rejuvenation

Effect of detection coverage on expected downtime



Results

UA with rejuvenation/UA without rejuvenation

Rejuvenation Policy	Configuration				
	2/1	*8/1*	*16/1*	*8/2*	*16/2*
Time-Based	0.74	0.74	0.76	0.02	0.05
Condition-Based	0.38	0.38	0.39	0.15	0.15

Time-Based: Optimal rejuvenation interval

Condition-Based: 90% coverage

Lower the ratio, greater the benefit

100(1-ratio) is the % of UA reduced due to rejuvenation

Recap

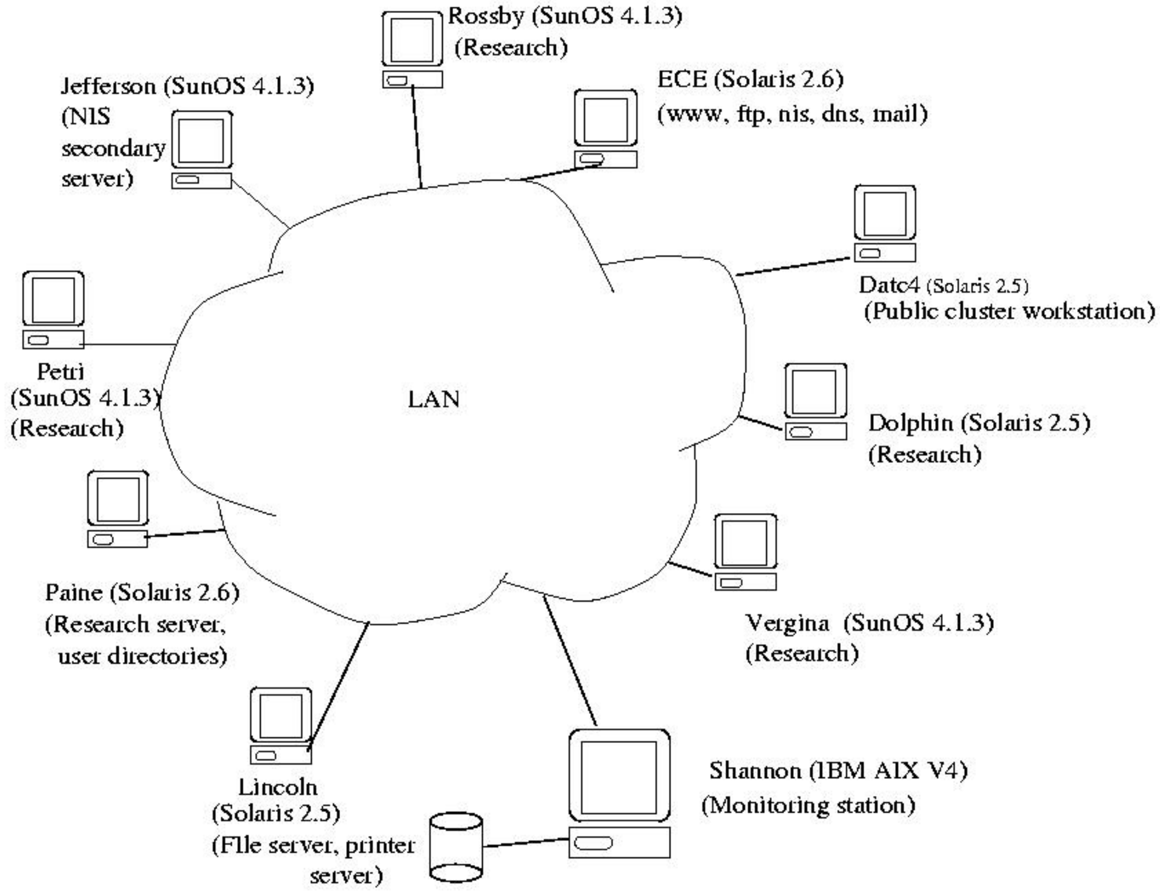
Analysis of Rejuvenation for Cluster Systems

- Huge benefit in terms of UA and cost improvement for systems with more than one spare
  - Simple time-based policy better than prediction-based for some cases
- Condition policy much better for large node repair times and low node-failure coverage
- Future work
  - Consider other performability measures
  - Explore non-ideal effects of common-mode failure and node-failure coverage

Measurement-Based Approach

- Objective
  - Detection and validation of aging
- Periodically monitor and collect data on the attributes responsible for the “health” of the system
- Quantify the effect of aging on system resources
  - Proposed metric – Estimated time to exhaustion
- Three approaches
  - Time-based (workload-independent) estimation [Garg98]
  - Workload-based estimation [Vaidyanathan99]
  - ARMA/ARX models [Li02]

Data Collection
Experimental Setup

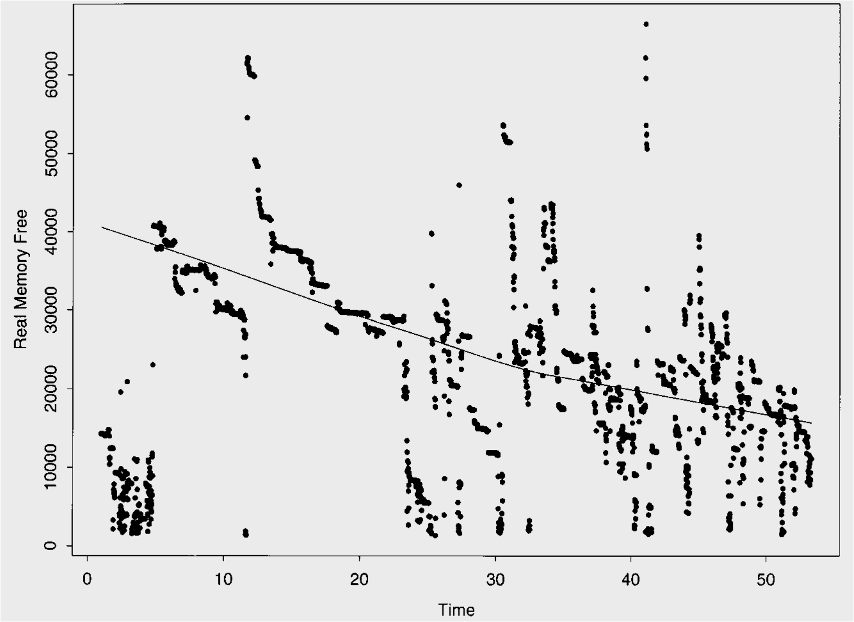


SNMP-based resource monitoring tool:

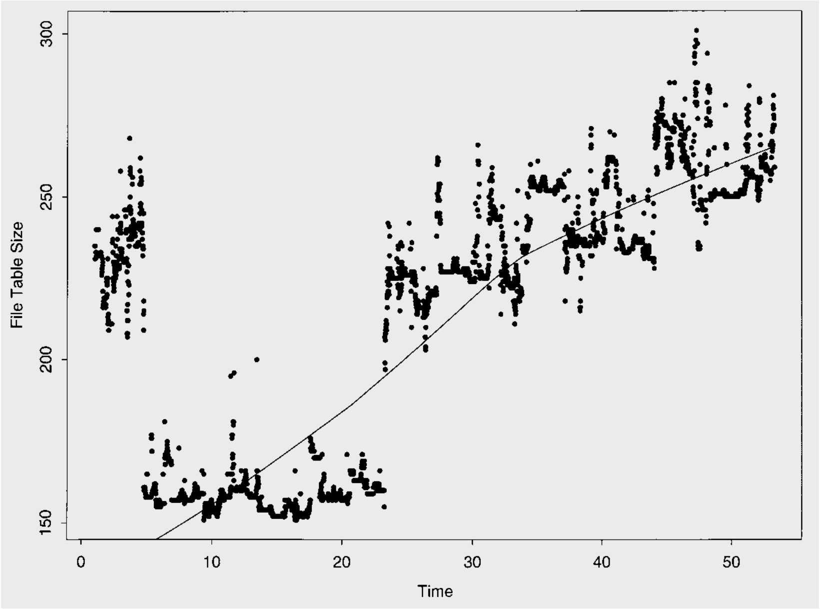
Data related to OS resource usage (memory, process table, file table etc.) and system activity (CPU usage, paging, swapping, NFS, interrupts etc.) collected for over 3 months at 10 min intervals

Time Plots
Non-parametric Regression Smoothing

Real Memory Free



File Table Size



Trend detection: Seasonal Kendall test for trend

Time-Based Approach (Workload-independent)

Estimated Time to Resource Exhaustion

Resource Name	*Initial Value*	*Max Value*	*Sen's Slope Estimation*	*95% Confidence Interval*	*Estimated Time to Exhaustion (days)*
Rossby					
Real Memory Free	40814.17	84980	-252.00	-287.75 : -219.34	161.96
File Table Size	220	7110	1.33	1.30 : 1.39	5167.50
Process Table Size	57	2058	0.43	0.41 : 0.45	4602.30
Used Swap Space	39372	312724	267.08	220.09 : 295.50	1023.50
Velum					
Real Memory Free	63276.03	116924	-188.00	-253.91 : -132.31	336.57
File Table Size	251	3628	0.67	0.58 : 0.70	5065.50
Process Table Size	60	1034	0.16	0.13 : 0.17	6168.67
Used Swap Space	17516.01	262076	418.00	394.22 : 446.00	585.07
Jefferson					
Real Memory Free	67638.54	114608	-972.00	-1006.81 : -939.08	69.59
File Table Size	268.83	7110	1.33	1.30 : 1.38	5144.36
Process Table Size	67.18	2058	0.30	0.29 : 0.31	6696.41
Used Swap Space	47148.02	524156	577.44	545.69 : 603.14	826.07

Workload-based Approach
Motivation

- Time-based approach for estimation of resource exhaustion
  - Non-parametric regression smoothing
  - Seasonal Kendall test for trend
  - Simple linear equation using Sen’s slope estimate
  - Doesn’t incorporate workload
- Intuitive that rate of resource exhaustion depends on current system load
  - Strong correlation between workload and system reliability/availability

Workload Characterization
Cluster Analysis

- Workload parameters
  - cpuContextSwitch, sysCall pageIn, pageOut
- Statistical cluster analysis
  - Hartigan's *k-means* algorithm

Cluster no.	Cluster Center				% of data points
	cpuContextSwitch	sysCall	pageOut	pageIn	
1	48405.16	94194.66	5.16	677.83	0.98
2	54184.56	122229.68	5.39	81.41	0.76
3	34059.61	193927.00	0.02	136.73	0.93
4	20479.21	45811.71	0.53	243.40	1.89
5	21361.38	37027.41	0.26	12.64	7.17
6	15734.65	54056.27	0.27	14.45	6.55
7	37825.76	40912.18	0.91	12.21	11.77
8	11013.22	38682.46	0.03	10.43	42.87
9	67290.83	37246.76	7.58	19.88	4.93
10	10003.94	32067.20	0.01	9.61	21.23
11	197934.42	67822.48	415.71	184.38	0.93

Clusters {1,2,3} and {4,5} merged to get 8 clusters

Workload Characterization

Transition Probability Matrix

$$p_{ij} = \frac{observed\ no.\ of\ transitions\ from\ state\ i\ to\ state\ j}{total\ observed\ no.\ of\ transitions\ from\ state\ i}$$

$$\mathbf{P} = \begin{bmatrix} 0.000 & 0.155 & 0.224 & 0.129 & 0.259 & 0.034 & 0.165 & 0.034 \\ 0.071 & 0.000 & 0.136 & 0.140 & 0.316 & 0.026 & 0.307 & 0.004 \\ 0.122 & 0.226 & 0.000 & 0.096 & 0.426 & 0.000 & 0.113 & 0.017 \\ 0.147 & 0.363 & 0.059 & 0.000 & 0.098 & 0.216 & 0.088 & 0.029 \\ 0.033 & 0.068 & 0.037 & 0.011 & 0.000 & 0.004 & 0.847 & 0.000 \\ 0.070 & 0.163 & 0.023 & 0.535 & 0.116 & 0.000 & 0.023 & 0.070 \\ 0.022 & 0.049 & 0.003 & 0.003 & 0.920 & 0.003 & 0.000 & 0.000 \\ 0.307 & 0.077 & 0.154 & 0.231 & 0.077 & 0.154 & 0.000 & 0.000 \end{bmatrix}$$

Workload Characterization

Sojourn Time Distribution

Workload State	Sojourn Time Distribution, $F(t)$	Distribution type
W_1	$1 - 1.602919e^{-0.9t} + 0.6029185e^{-2.392739t}$	Hypo-exponential
W_2	$1 - 0.9995e^{-0.4459902t} - 0.0005e^{-0.007110071t}$	Hyper-exponential
W_3	$1 - 0.9952e^{-0.3274977t} - 0.0048e^{-0.0175027t}$	Hyper-exponential
W_4	$1 - 0.841362e^{-0.3275372t} - 0.158638e^{-0.03825429t}$	Hyper-exponential
W_5	$1 - 1.425856e^{-0.56t} + 0.4258555e^{-1.875t}$	Hypo-exponential
W_6	$1 - 0.80694e^{-0.5509307t} - 0.19306e^{-0.03705756t}$	Hyper-exponential
W_7	$1 - 2.86533e^{-1.302t} + 1.86533e^{-2t}$	Hypo-exponential
W_8	$1 - 0.9883e^{-0.2655196t} - 0.0117e^{-0.02710147t}$	Hyper-exponential

Model Validation

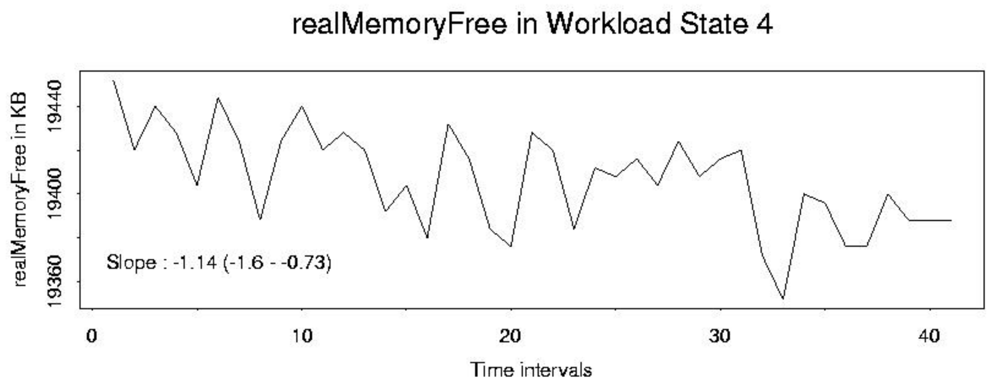
Steady-state probabilities computed from the model match very closely with actual probabilities obtained from the observed data

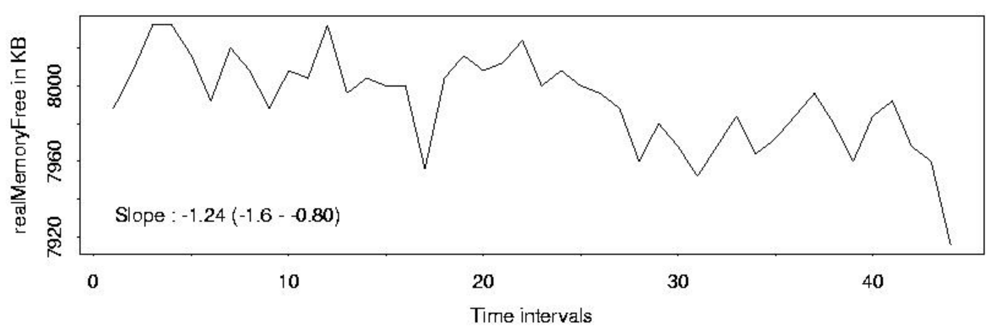
State	Observed value	Value from model	% Difference
W_1	2.664146	2.8110	5.512235
W_2	9.058096	8.3464	7.857015
W_3	6.548642	6.0576	7.498379
W_4	11.77381	10.8480	7.863300
W_5	42.86696	44.4310	3.648591
W_6	4.932967	4.5767	7.222165
W_7	21.22723	22.1030	4.125691
W_8	0.928154	0.82577	11.030928

Modeling Resource Usage

Sen's Slope Estimate

Slope estimate (in KB/10min) for a each resource in a given workload state – nearly same during different visits to that state





Modeling Resource Usage
Reward Function

Reward function for each resource - Sen's slope estimate (in KB/10 min) for each resource at every workload state

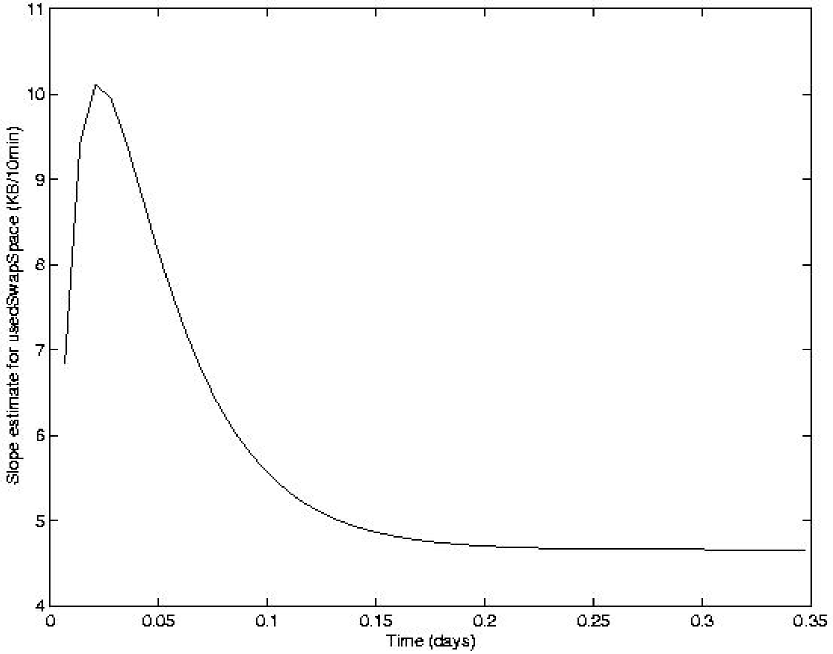
Workload State	usedSwapSpace		realMemoryFree	
	Slope Estimate	95 % Conf. Interval	Slope Estimate	95 % Conf. Interval
W_1	119.33	5.54 - 222.39	-133.67	-137.67 - -133.33
W_2	0.57	0.40 - 0.71	-1.47	-1.78 - -1.09
W_3	0.76	0.73 - 0.80	-1.43	-2.50 - -0.62
W_4	0.57	0.00 - 0.69	-1.23	-1.67 - -0.80
W_5	0.78	0.75 - 0.80	0.00	-5.65 - 6.00
W_6	0.81	0.64 - 1.00	-1.14	-1.40 - -0.88
W_7	0.00	0.00 - 0.00	0.00	0.00 - 0.00
W_8	91.78	72.40 - 110.95	0.00	-369.88 - 475.17

Solution Method

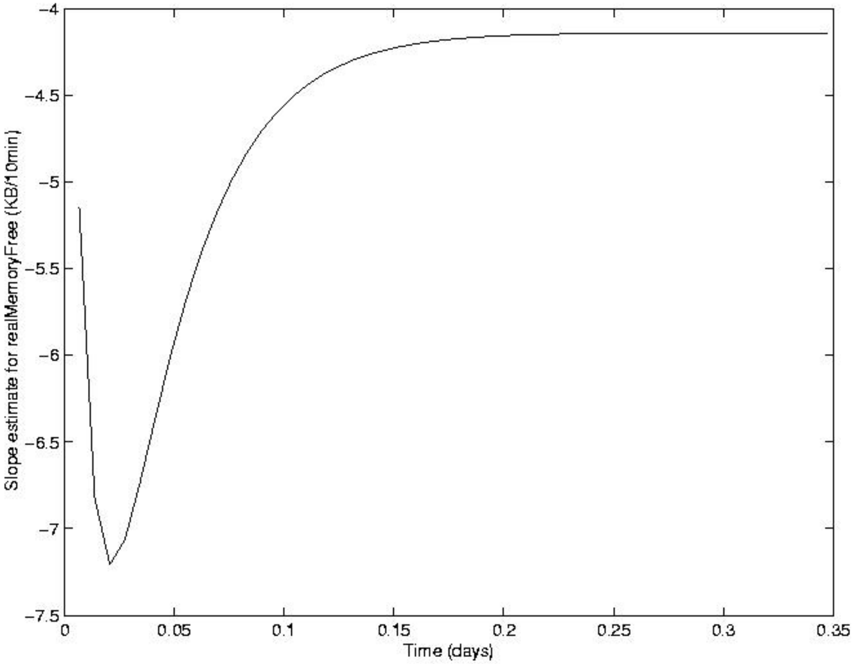
- Semi-Markov reward model Markovized into Markov reward model
- Solved using SHARPE (Symbolic Hierarchical Automated Reliability and Performance Estimator)
- Measures obtained (reward rate ⇔ rate of exhaustion)
  - Expected instantaneous reward
  - Expected reward rate at steady state
  - Expected accumulated reward at time *t*
  - Mean job completion time

Results
Transient slope estimates

Slope for *usedSwapSpace*



Slope for *realMemoryFree*



Results

Estimates for slope and time to exhaustion

Estimations for slope (KB/10min) and time to exhaustion (days) for *usedSwapSpace* and *realMemoryFree*

Method of Estimation	*usedSwapSpace*			*realMemoryFree*		
	Slope Estimate	95 % Conf. Interval	Est. Time to Exh.	Slope Estimate	95 % Conf. Interval	Est. Time to Exh.
Time based	0.787	0.786 - 0.788	2276.46	-2.806	-3.026 - -2.630	60.81
Workload based	4.647	1.191 - 7.746	490.50	-4.1435	-9.968 - 2.592	41.38

Workload-based approach: lower time to exhaustion than the time-based approach

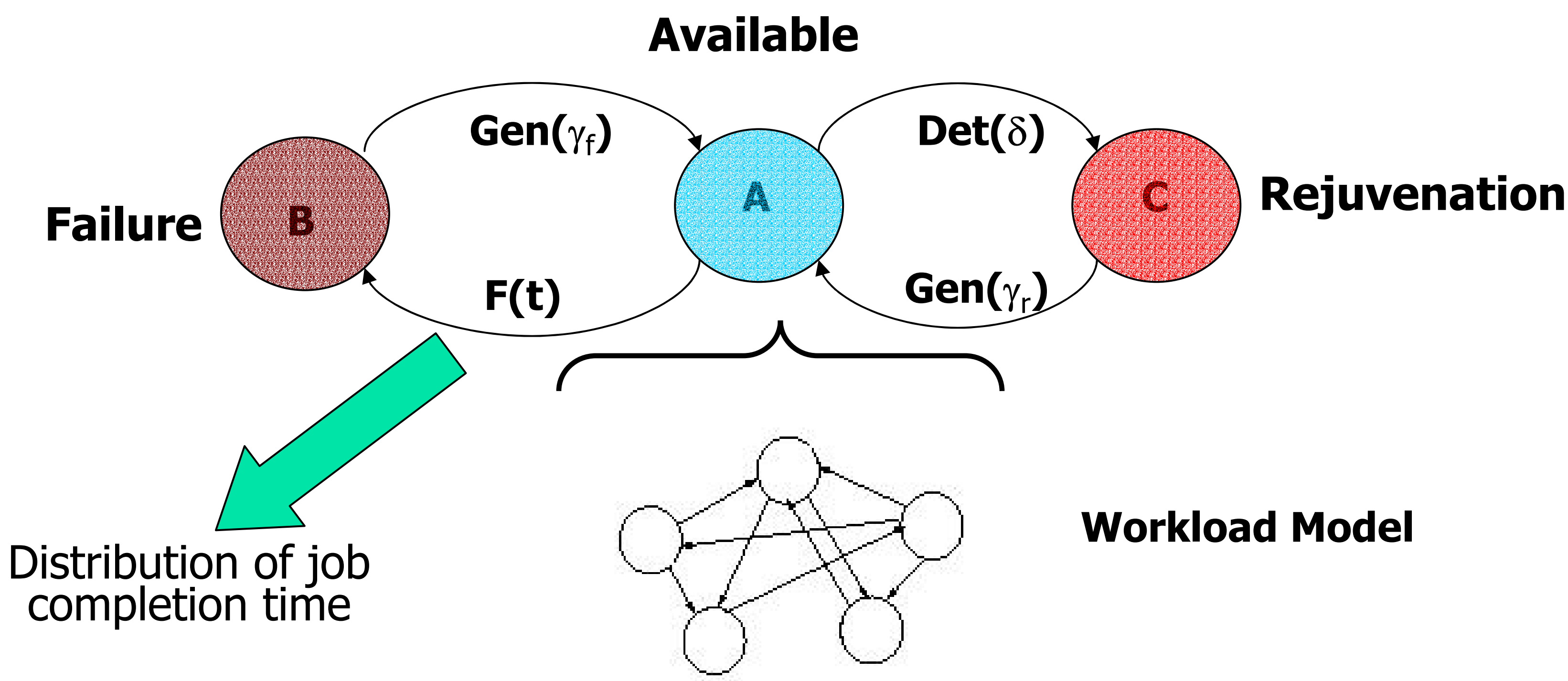
DUKE

Recap

Workload-based Approach

- Developed measurement-based model which incorporates workload
- Demonstrated relation between workload and resource usage
- Estimates for slope and time to exhaustion
- Not actual machine failure times
  - Need more accurate models
  - Dependencies between various resources
- A step further towards predicting failures resulting from resource exhaustion
  - New/better preventive maintenance policies

Comprehensive Model
Description



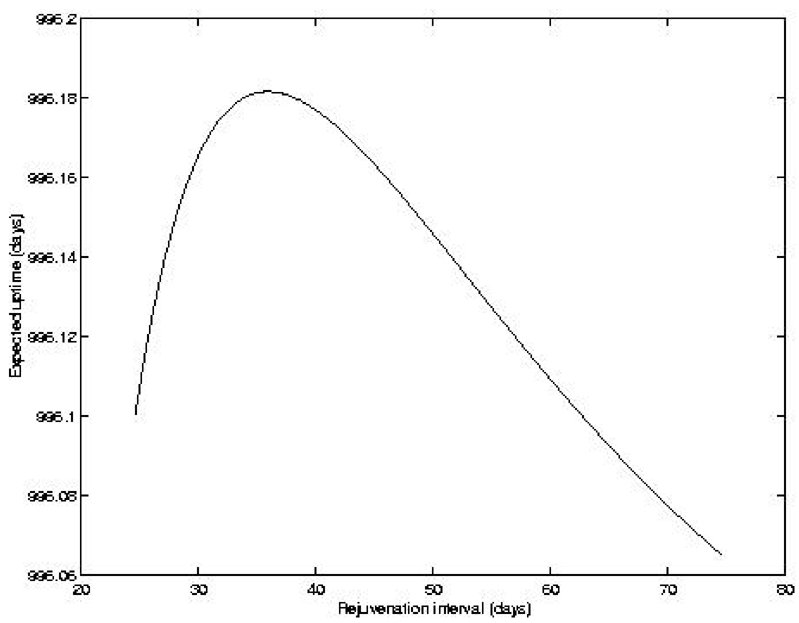
Comprehensive Model

Model Parameters

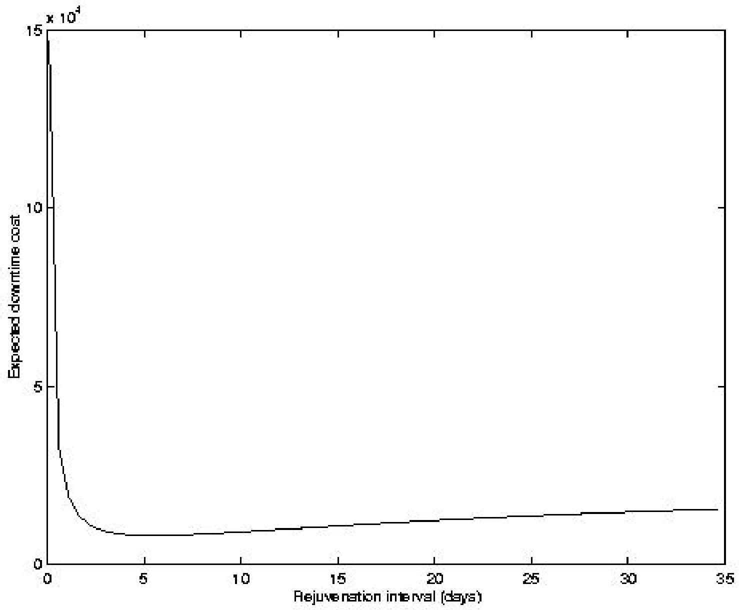
- Distribution of job completion time difficult to compute
  - Assume 2-stage Erlang (IFR) with mean = mean JCT = 41.38 days
- Mean recovery time = 4 hours
- Mean rejuvenation time = 1 hour
- Cost of failure = $5000/hr
- Cost of rejuvenation = $500/hr
- Compute expected uptime and expected cost over a given interval (1000 days)

Comprehensive Model
Results

Expected uptime



Expected downtime cost



Optimal δ = 36.10 days

Optimal δ = 5.60 days

Comprehensive Model
Summary

- Integrated analytic and measurement-based model
- Future work
  - Compute the distribution JCT more accurately
  - Better approximations for F(t)
  - Take into account multiple resources in the model

Time Series and ARMA models
Study of Rejuvenation for Apache

- Application of software rejuvenation
  - Web server needs to run forever
  - Current method: Periodically restart server
  - Our objective: Predict the appropriate/optimal time to restart server
- Experimental setup
  - Linux Monitoring Tool
    - Procmon: extracts information from files in /proc file system
  - Web Server Benchmark
    - Httperf
      - Connection rate
      - Response time, reply rate, timeout error

Experiments

- Capacity of the web server

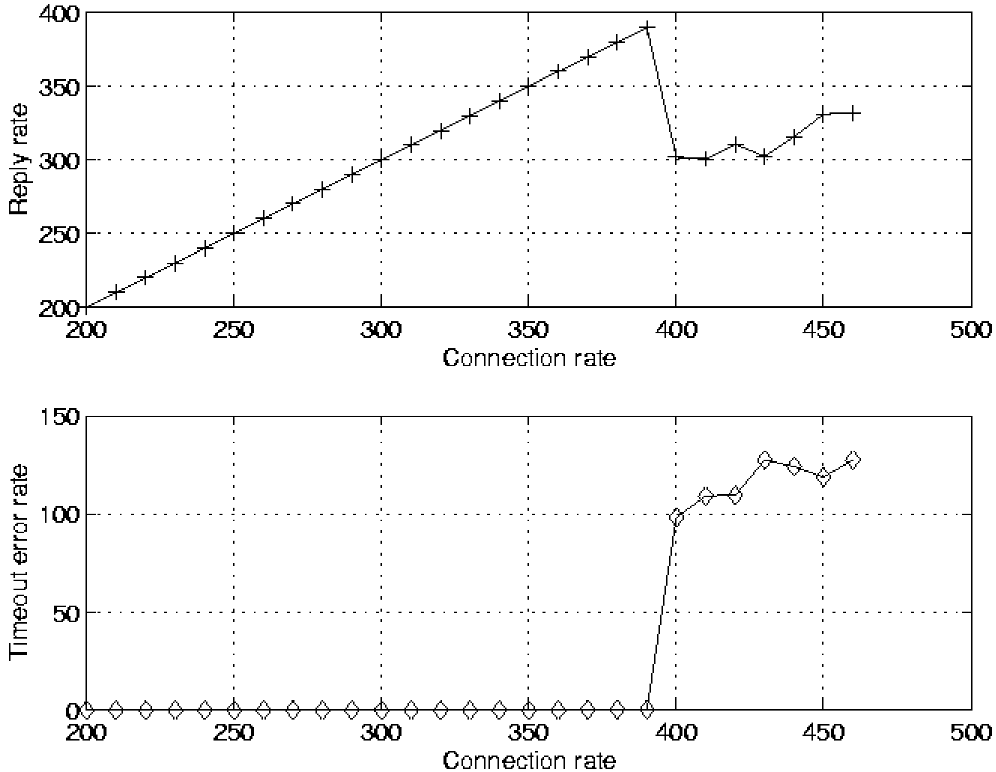


Figure 3.2: Capacity of the web server

Data Analysis and Results

- Collected Data
  - 7-day period, connection rate of 350 per second
  - 25-day period, connection rate of 400 per second
  - 14-day period, connection rate varies from 350 to 390 per second
  - More than 100 parameters recorded
- Selection of parameters
  - Physical meaning
  - Relationship to the system resources

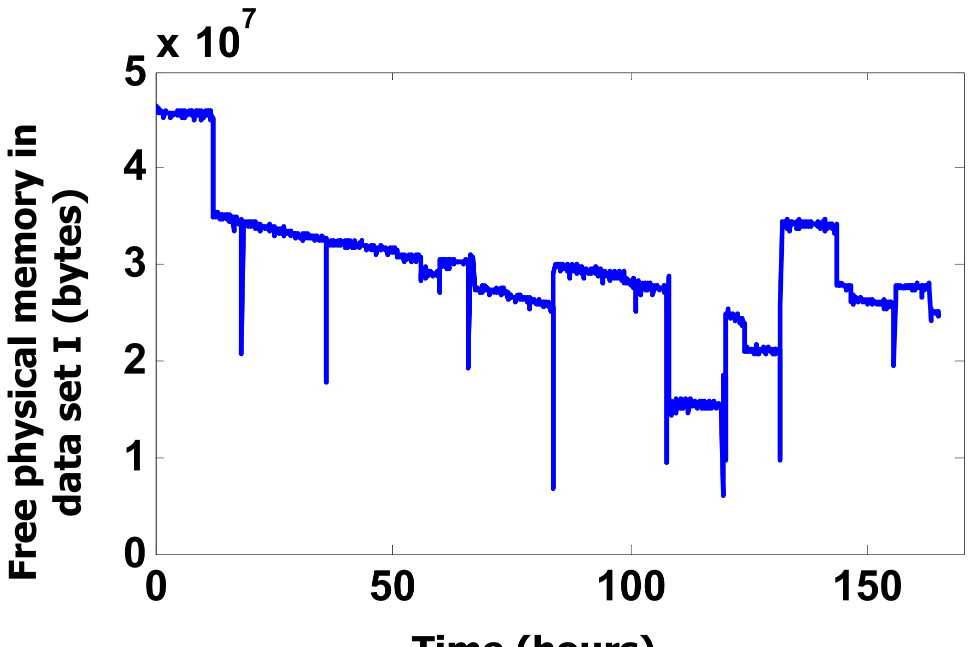
Data Analysis and Results

Selected parameters and their physical meaning

Parameter	Physical meaning
PhysicalMemoryFree	Free physical memory
SwapSpaceUsed	Used swap space
LoadAvg5Min	Average CPU load in the last five minutes
NumberDiskRequests	Number of disk requests in the last five minutes
PageOutCounter	Number of pages paged out in the last five minutes
NewProcesses	Number of newly spawned processes in the last five minutes
ResponseTime	The interval from the time a Httperf sends out the first byte of request until it receives the first byte of reply

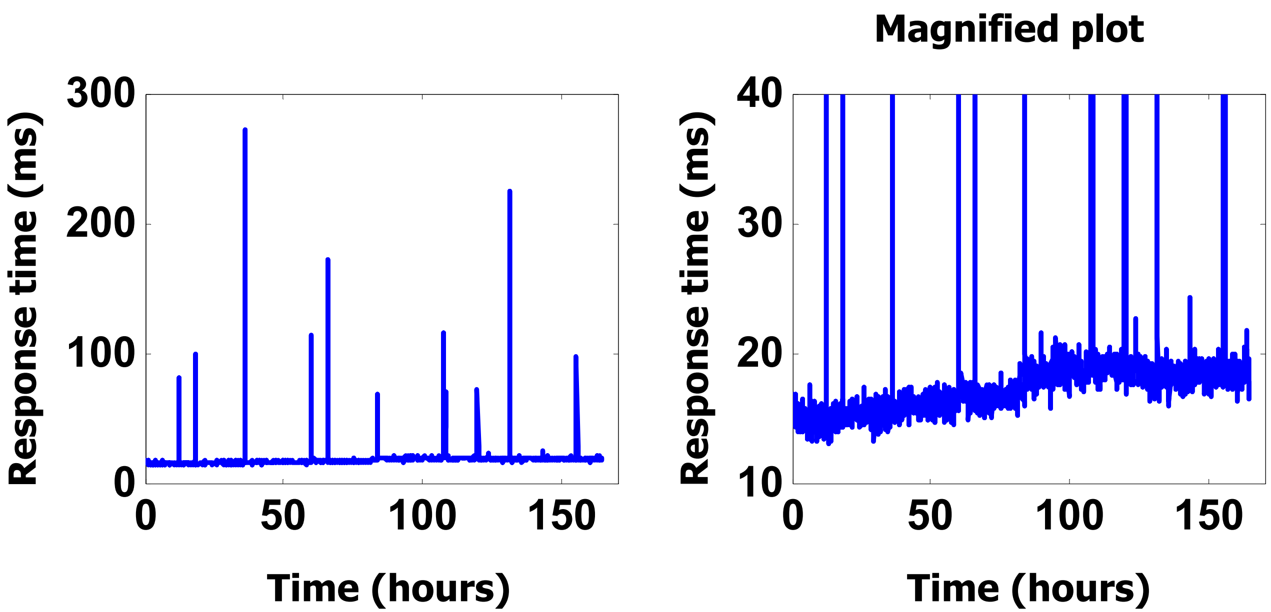
Data Analysis and Results

- Trend Detection
  - Visual observation
    - Exhaustion of system resources



Data Analysis and Results

- Trend Detection
  - Visual observation
    - Degradation of performance



Response time of the web server in data set I

Data Analysis and Results

- Trend Detection
  - Linear regression

Estimated slopes of parameters

Data Set	Parameter	Slope	95% confidence interval
I	Response time	0.027 ms/hour	(0.019, 0.036) ms/hour
	Free physical memory	-88.472 kB/hour	(-93.337, -83.607) kB/hour
	Used swap space	29.976 kB/hour	(29.290, 30.662) kB/hour
II	Response time	0.063 ms/hour	(0.057, 0.068) ms/hour
	Free physical memory	15.183 kB/hour	(14.094, 16.271) kB/hour
	Used swap space	7.841 kB/hour	(7.658, 8.025) kB/hour

Data Analysis and Results

- Modeling Software Aging
  - ARMA model
    - AR(p): $X_t = \phi_1 X_{t-1} + \phi_2 X_{t-2} + \ldots + \phi_p X_{t-p} + W_t$
    - MA(q): $X_t = W_t + \theta_1 W_{t-1} + \theta_2 W_{t-2} + \ldots + \theta_q W_{t-q}$
    - ARMA(p,q):

$$X_t - \phi_1 X_{t-1} - \phi_2 X_{t-2} - \ldots - \phi_p X_{t-p} = W_t + \theta_1 W_{t-1} + \theta_2 W_{t-2} + \ldots + \theta_q W_{t-q}$$

Data Analysis and Results

- Modeling software aging
  - Determine the order of the model
    - Autocorrelation function (ACF)

$$\rho(h) = \mathrm{corr}(X_{t+h}, X_t) = \frac{\mathrm{Cov}(X_{t+h}, X_t)}{\mathrm{Var}(X_t)}$$

    - Partial autocorrelation function (PACF)
      - Best linear prediction

$$X_{t+h}^{(h-1)} = \beta_1 X_{t+h-1} + \beta_2 X_{t+h-2} + \ldots + \beta_{h-1} X_{t+1}$$

$$\bar{X}_t^{(h-1)} = \beta_1 X_{t+1} + \beta_2 X_{t+2} + \ldots + \beta_{h-1} X_{t+h-1}$$

      - PACF

$$\psi_{11} = \mathrm{corr}(X_{t+1}, X_t) = \rho(1)$$

$$\psi_{hh} = \mathrm{corr}(X_{t+h} - X_{t+h}^{(h-1)}, X_t - \bar{X}_t^{(h-1)}), \quad h \geq 2$$

Data Analysis and Results

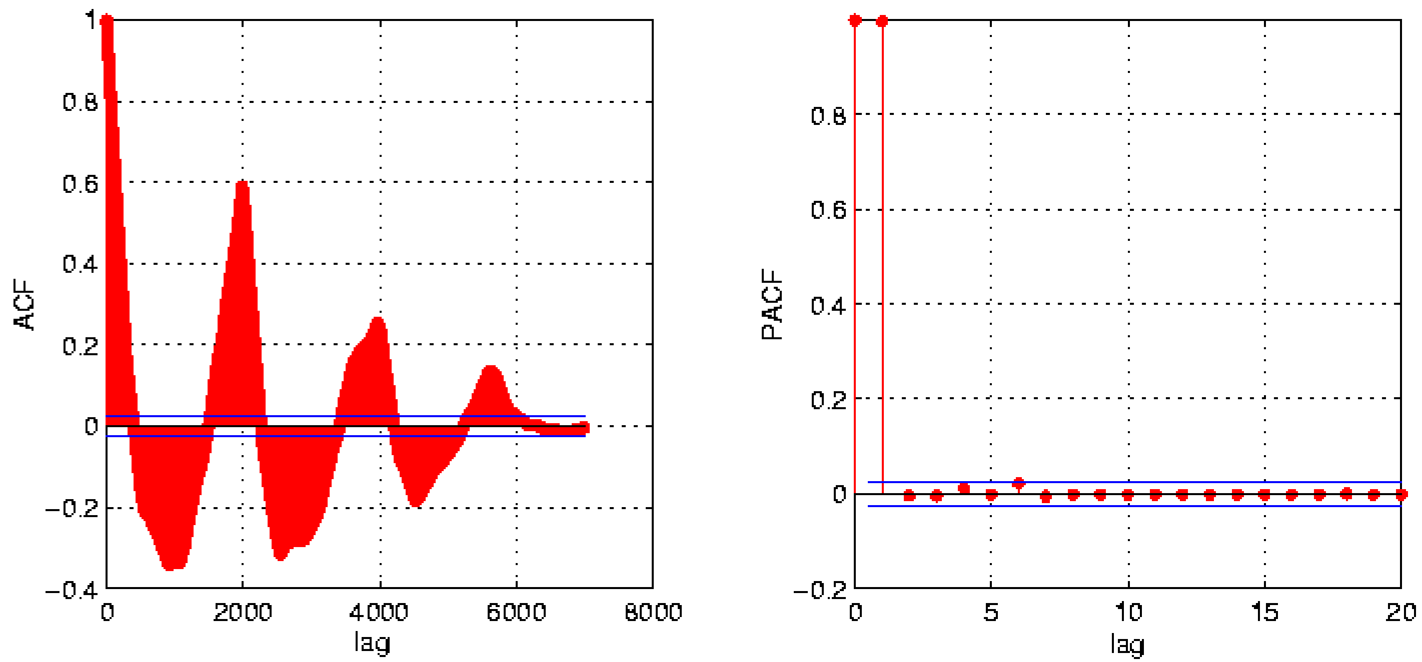
- Modeling software aging
  - Determine the order of the model
    - Behavior of the ACF and PACF for ARMA Models

Table 3: Behavior of the ACF and PACF for ARMA Models

	AR(p)	MA(q)	ARMA(p,q)
ACF	Trails off	Cuts off after lag q	Trails off
PACF	Cuts off after lag p	Trails off	Trails off

Data Analysis and Results

- Modeling software aging
  - Determine the order of the model



ACF and PACF of used swap space in data set II

Data Analysis and Results

- Modeling software aging
  - Determine the order of the model
    - Used swap space
      - AR(1) is suitable for the swap space
      - Add inputs to make it ARX model
        - Connection rate: X
        - Linear trend: L
        - Weekly periodicity: W
        - Daily periodicity: D

$$Y_t = aY_{t-1} + b_1 X_t + b_2 L_t + b_3 W_t + b_4 D_t$$

    - All the parameters can be modeled by ARX model

Data Analysis and Results

- Modeling software aging
  - Combine all the MISO ARX models into a MIMO ARX model
    - Leading relationship of the parameters
      - Compute crosscorrelation functions for each pair of parameters

Table 3.6: Leading relationship between parameters

	Phy	Swa	Loa	Num	Pag	New	Res
PhysicalMemoryFree		-2	0	-5	-5	1	-3
SwapSpaceUsed	2		0	0	0	8	0
LoadAvg5Min	0	0		-1	-1	1	-3
NumberDiskRequests	5	0	1		0	0	-2
PageOutCounter	5	0	1	0		0	-2
NewProcesses	-1	-8	-1	0	0		-3
ResponseTime	3	0	3	2	2	3	

Data Analysis and Results

- Modeling software aging

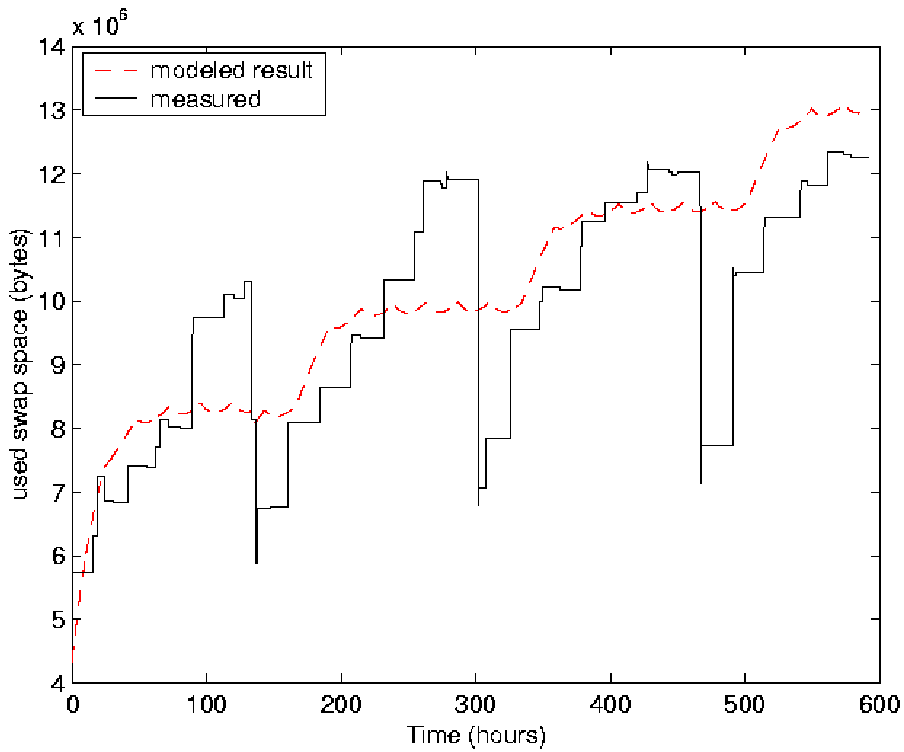


Figure 3.16: Measured and modeled used swap space

Center for Advanced Computing and Communication, Duke University

Data Analysis and Results

- Modeling software aging

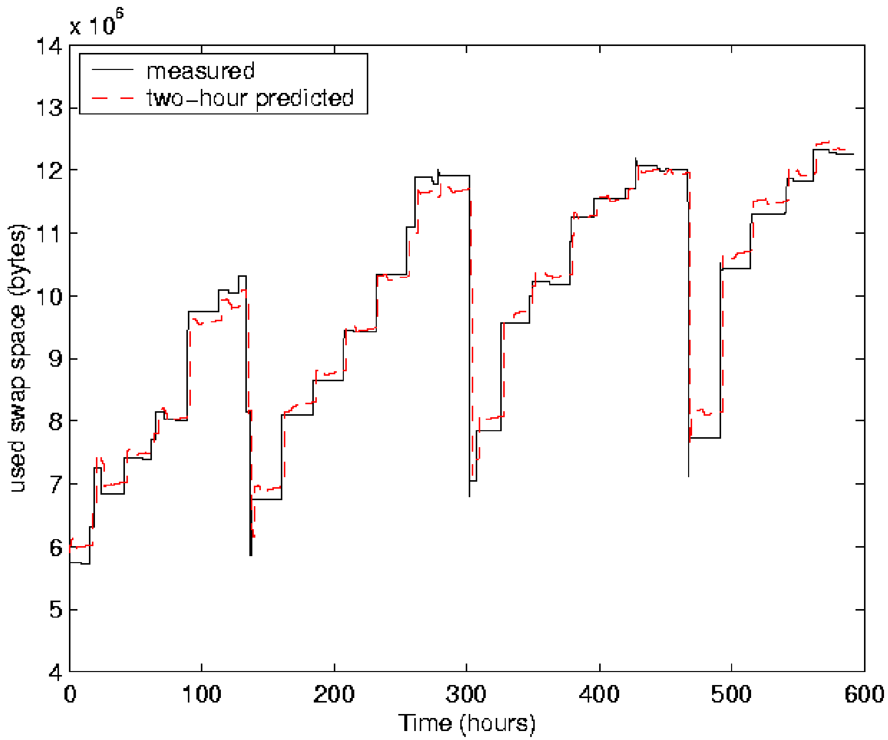


Figure 3.17: Measured and two-hour ahead predicted used swap space

Data Analysis and Results

- Modeling software aging

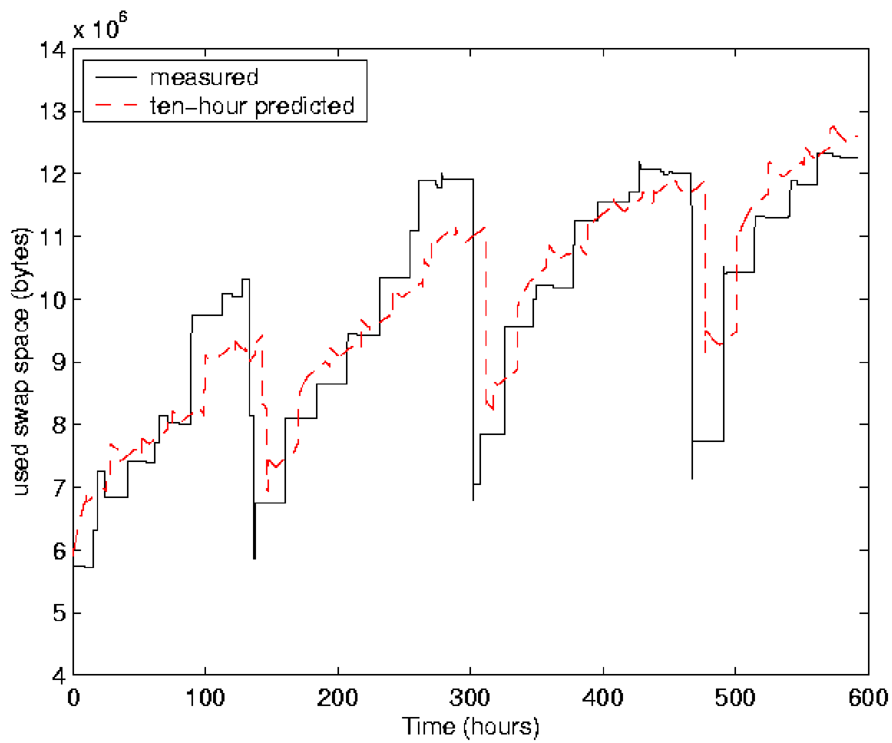


Figure 3.18: Measured and ten-hour ahead predicted used swap space

Related research and comparison

- Comparison of various methods

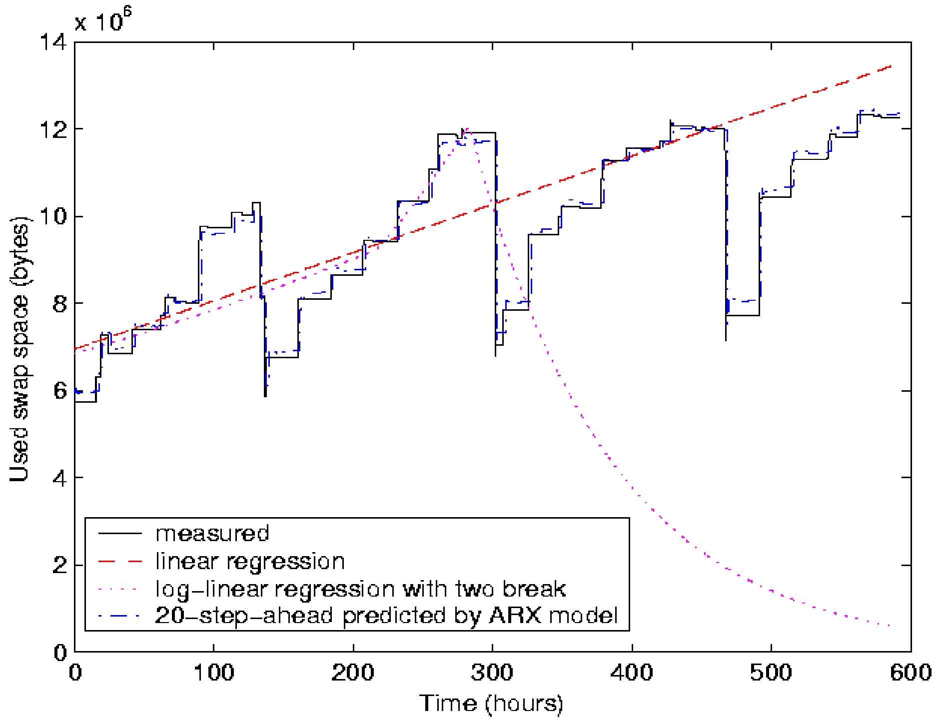


Figure 4.2: Model results of used swap space in data set II

IBM xSeries Software Rejuvenation Agent (SRA)

- Implemented in a high-availability clustered environment
- Monitors consumable resources, estimate time to exhaustion and generates alerts if within user notification horizon
- IBM Director system management tool
  - Provides GUI to configure SRA
  - Acts upon alerts
- Two versions
  - Periodic rejuvenation
  - Prediction-based rejuvenation

IBM xSeries SRA

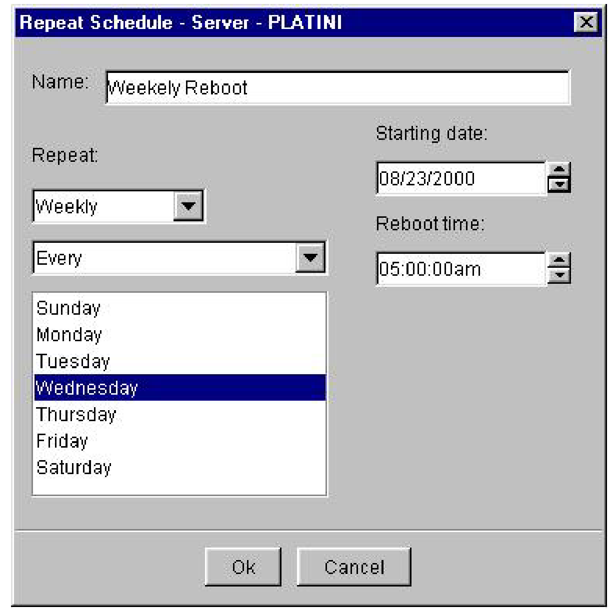
Design Goals

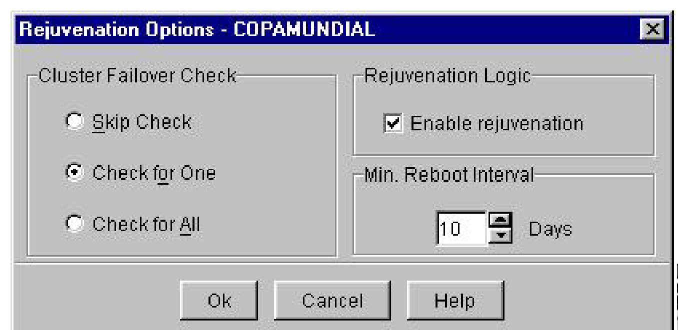
- No modification of the application allowed
- Designed for portability
- Uses published and architected interfaces for data acquisition, alerting and rejuvenation
- Simple GUI
  - Minimum tunable parameters
- Must adapt to monitor new parameters and incorporate new prediction algorithms

Rejuvenation in IBM Director

Periodic Rejuvenation

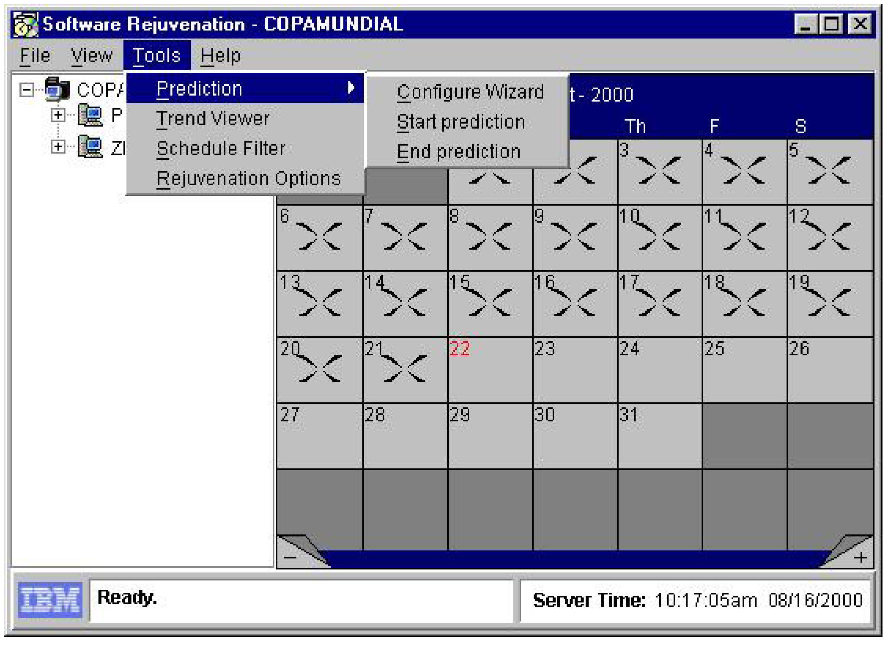
Rejuvenation Menu

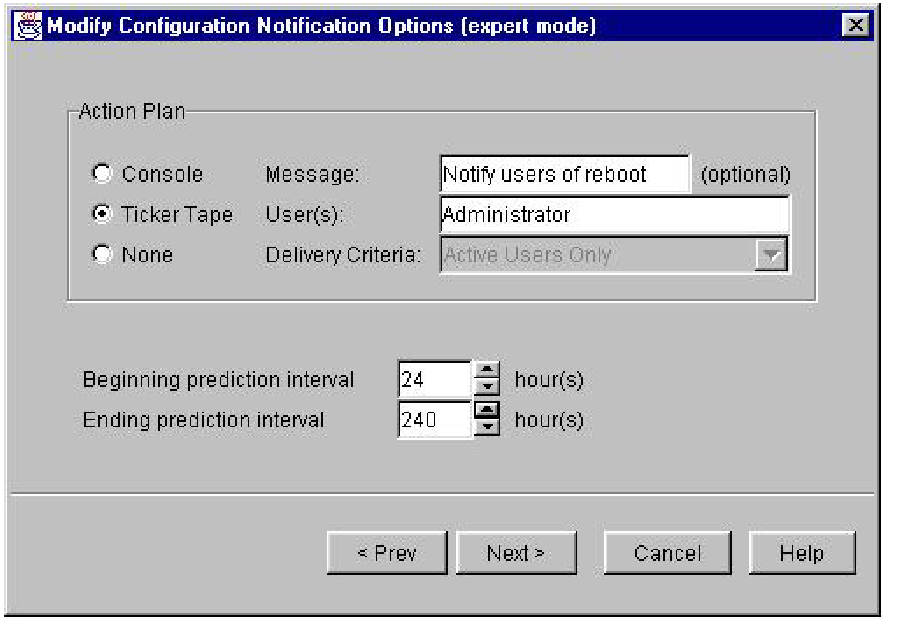




Rejuvenation in IBM Director

Prediction-Based Rejuvenation





SRA Design

- Monitor resources, estimate exhaustion times and send alerts
- Data acquisition specific to OS
  - **Windows NT:** registry performance counters
    - available bytes, committed bytes, non-paged pool, paged pool, handles etc.
  - **Linux:** /proc directory
    - memory, file descriptors, inodes, swap space etc.
- Data logged on disk and used by prediction algorithms

Prediction Algorithms

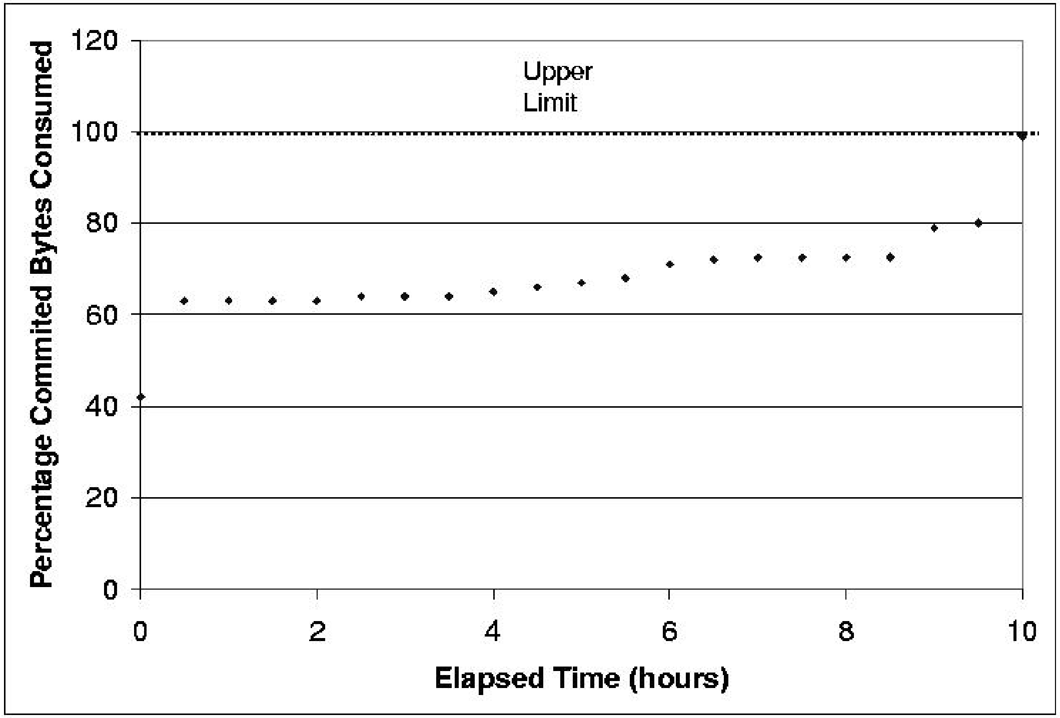
- Curve-fitting analysis and projection
  - Sliding window based on a fraction of the notification horizon
  - Sampling interval automatically selected
  - Fits several types of models and selects best the using a model-selection criterion
- Projected data compared to upper/lower exhaustion thresholds within notification time horizon
  - Identify the offending process/sub-system if possible

Rejuvenation Granularity

- Level 1 rejuvenation
  - Restart service
  - Only when stoppage of service saves necessary states
- Level 2 rejuvenation
  - OS reboot
  - Application failover and recovery by cluster management software

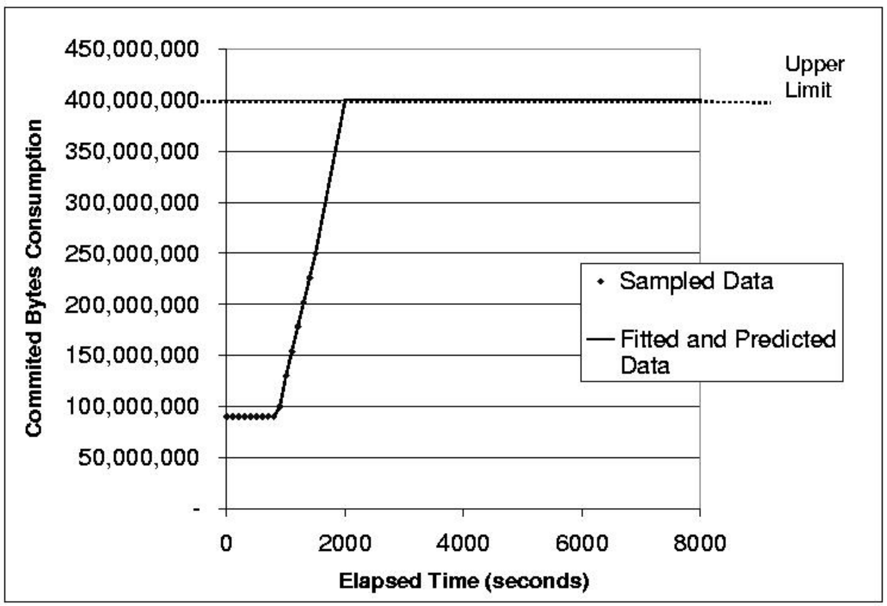
Empirical Measurements
Database Application

Application hang after committed bytes exhausted in 10 hours (Windows NT)

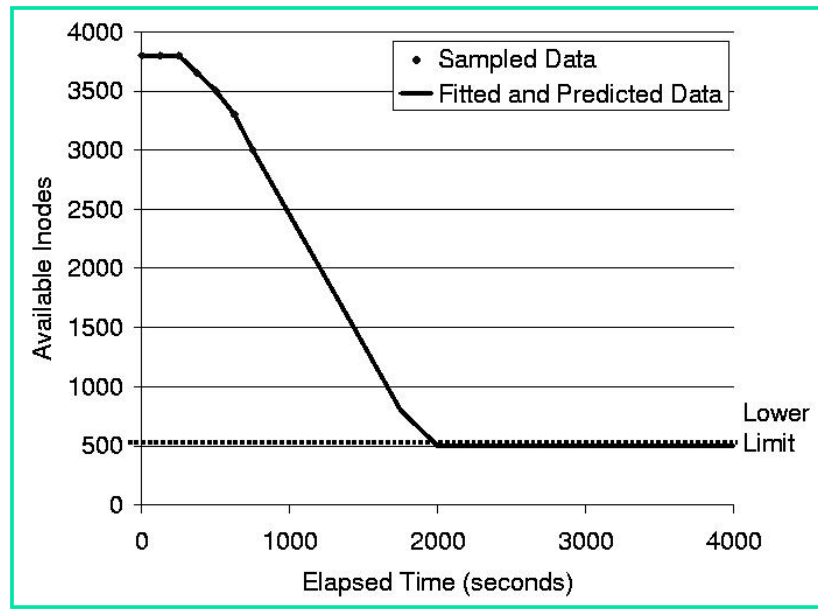


Empirical Measurements
"Bad Boy" Programs

Committed bytes consumption for a leaky application (Windows NT)



Inodes consumption for a leaky application (Linux)



Summary

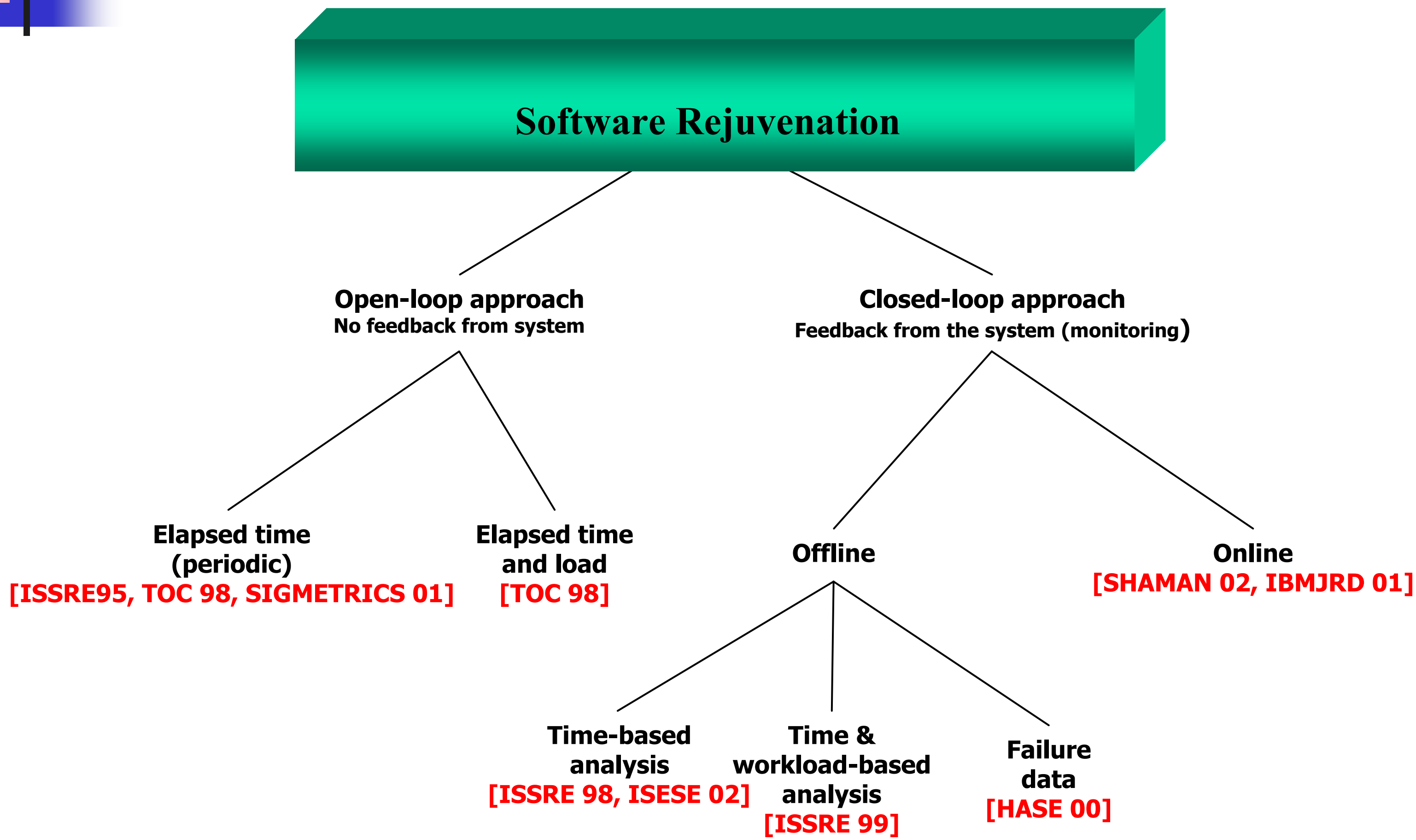
- Should consider five dimensions of software reliability: testing/operational phase, different types of bugs, redundancy type, measurement or modeling and different layers of software

  www.software-reliability.com
- Software aging not anecdotal – real life scientific phenomenon
- Interesting problems for modeling community

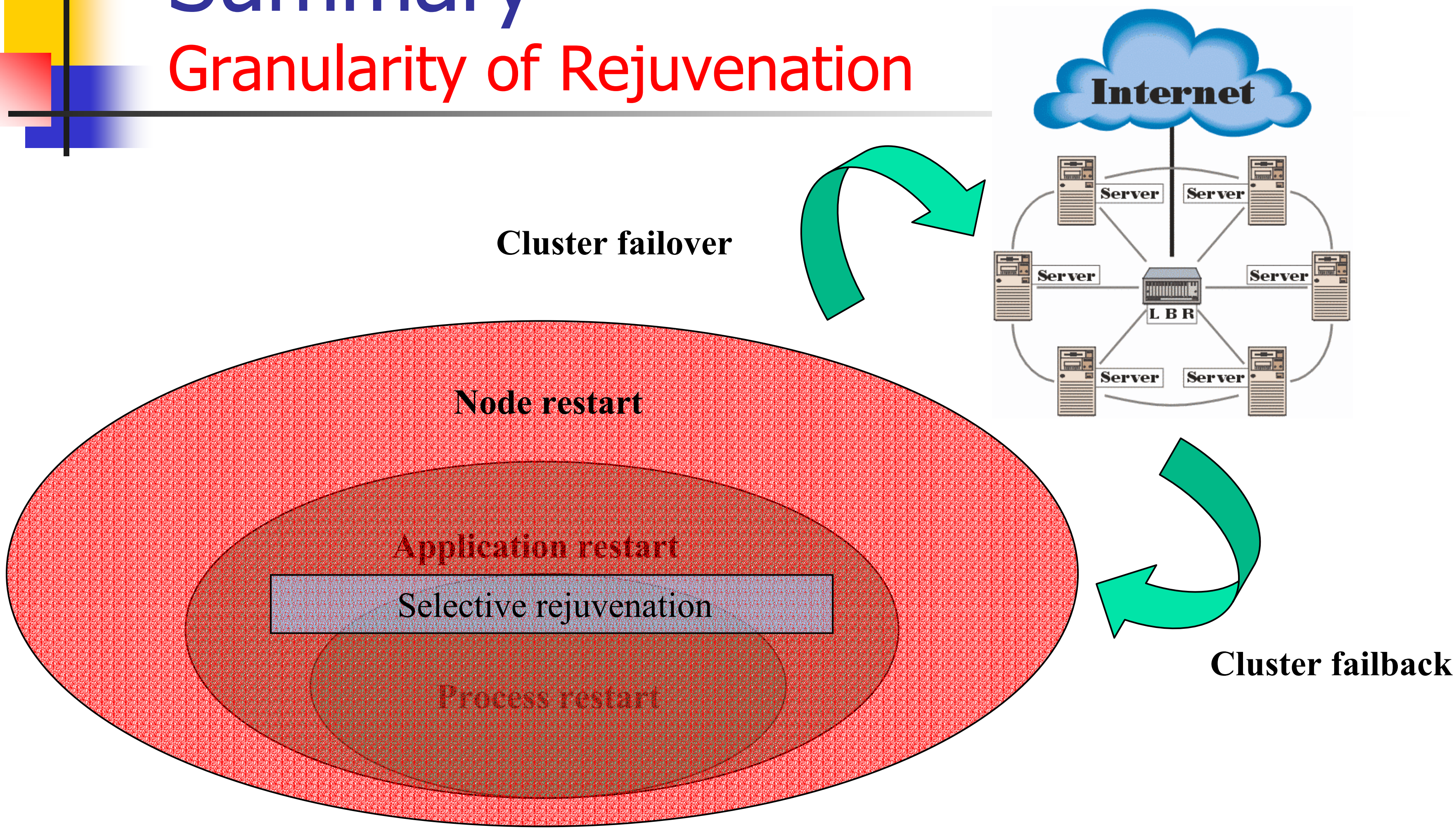
  www.software-rejuvenation.com

Summary

Approaches to Rejuvenation



Summary
Granularity of Rejuvenation



References

- .**Software Rejuvenation: Analysis, Module and Applications**, Y. Huang, C. Kintala, N. Kolettis and N. Fulton, In *Proc. of the 25th IEEE Intl. Symp. on Fault Tolerant Computing (FTCS-25)*, Pasadena, CA, June 1995.
- **Analysis of Software Rejuvenation using Markov Regenerative Stochastic Petri Net**, S. Garg, A. Puliafito M. Telek and K. S. Trivedi, In *Proc. of the Sixth IEEE Intl. Symp. on Software Reliability Engineering*, Toulouse, France, October 1995.
- **Analysis of Preventive Maintenance in Transaction Based Software Systems**, S. Garg, A. Puliafito, M. Telek and K. S. Trivedi, *IEEE Trans. on Computers, 47(1). January 1998*.
- **Analysis of Software Cost Models with Rejuvenation**, T. Dohi, K. Goseva-Popstojanova and K. S. Trivedi, *Proc. of the IEEE Intl. Symp. on High Assurance Systems Engineering, HASE-2000*, November 2000.
- **Statistical Non-Parametric Algorithms to Estimate the Optimal Software Rejuvenation Schedule**, T. Dohi, K. Goseva-Popstojanova and K. S. Trivedi, *Proc. of the 2000 Pacific Rim Intl. Symp. on Dependable Computing, PRDC 2000*, Los Angeles, December 2000.
- **A Methodology for Detection and Estimation of Software Aging**, S. Garg, A. van Moorsel, K. Vaidyanathan and K. S. Trivedi. In *Proc. of the Ninth IEEE Intl. Symp. on Software Reliability Engineering*, Paderborn, Germany, November 1998.
- **A Measurement-Based Model for Estimation of Resource Exhaustion in Operational Software Systems**, K. Vaidyanathan and K. S. Trivedi. In *Proc. of the Tenth IEEE Intl. Symp. on Software Reliability Engineering*, Boca Raton, Florida, November 1999.

References (contd.)

- **Modeling and Analysis of Software Aging and Rejuvenation**, K. S. Trivedi, K. Vaidyanathan and K. Goseva-Popstojanova. In *Proc. of the 33rd Annual Simulation Symp.*, Washington D.C., April 2000.
- **Analysis and Implementation of Software Rejuvenation in Cluster Systems**, K. Vaidyanathan, R. E. Harper, S. W. Hunter and K. S. Trivedi. In *Proc. of the Joint Intl. Conf. on Measurement and Modeling of Computer Systems*, *ACM SIGMETRICS 2001/Performance 2001*, Cambridge, MA, June 2001.
- **Proactive Management of Software Aging**, V. Castelli, R. E. Harper, P. Heidelberger, S. W. Hunter, K. S. Trivedi, K. Vaidyanathan and W. Zeggert, *IBM Journal of Research & Development*, Vol. 45, No. 2, March 2001.
- **An Approach to Estimation of Software Aging in a Web Server**, L. Li, K. Vaidyanathan and K. S. Trivedi. In *Proc. of the Intl. Symp. on Empirical Software Engineering, ISESE 2002*, Nara, Japan, October 2002.
- **Analysis of Inspection-Based Preventive Maintenance in Operational Software Systems**, K. Vaidyanathan, D. Selvamuthu and K. S. Trivedi. In *Proc. of the Intl. Symp. on Reliable Distributed Systems, SRDS 2002*, Osaka, Japan, October 2002.
- **Probability & Statistics with Reliability, Queuing and Computer Science Applications, (2nd ed.),** K. S. Trivedi, John Wiley, 2001.