Towards Precise Analysis for Software Rejuvenation (v 0.1)

Jiansen HE

May 5, 2014

1 Introduction

quick read

- §4.1.5
- §2 & §3

2 Background

2.1 Basic Failure Model in Reliability Study

In reliability study, the time when the first failure occurs in often assumed to follow a probability density function (pdf) if time is a continuous value, or a probability mass function (pmf) if time is a discrete value. (Musa, 2004)

The value of time is a continuous value if its unit is a *time unit* (e.g. second) and the value could be any real number. The value of time is a discrete value if its unit is a *natural unit* (e.g. number of operations) or its unit is a *time unit* but the user only cares the status of a system at discrete time (e.g. every second).

Although the precise definition of failure varies in different systems, measures of failures can be consistently defined in terms of a pdf or a pmf. Assuming that the pdf of a system is $f(t)$, measures of failures can be defined as in Figure 1 (Musa, 2004). Measures of failures can be defined in terms of pmf similarly.

In the above, the reliability during a specific time, $R(t)$, is the probability that a system can survive without failure throughout that time. The Mean Time to Failure (MTTF) is the average time when the first failure occurs. The hazard rate at time t is the failure rate at that time given the condition that the system has survived for t time. Equation 2.1.2 checks that the average failure rate (λ) is the reciprocal of MTTF.

$$R(t) = \int_t^\infty f(x)dx = 1 - \int_0^t f(x)dx \tag{2.1.1a}$$

$$MTTF = \int_0^\infty tf(t)dt = \int_0^\infty R(t)dt \tag{2.1.1b}$$

$$h(t) = \lim_{\Delta t \to 0} \frac{R(t) - R(t + \Delta t)}{\Delta t R(t)} = \frac{f(t)}{R(t)} \tag{2.1.1c}$$

Figure 1: Measures of Failures

$$\lambda = \int_0^\infty h(t)dt = \int_0^\infty \frac{f(t)}{R(t)}dt = \frac{\int_0^\infty f(t)dt}{\int_0^\infty R(t)dt} = \frac{1}{MTTF} \tag{2.1.2}$$

2.2 Traditional Software Rejuvenation Models

2.2.1 The Basic Model

The basic software rejuvenation model proposed by (Huang et al., 1995) is modelled by the state transition diagram in Figure 2(a). Huang et. al defines the following 4 states:

S The robust `start` state in which the system will not fail.

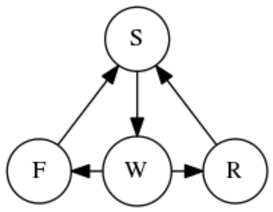
W The failure probable `working` state in which the system may fail at a certain probability.

F The `failure` state when the system reaches its boundary condition.

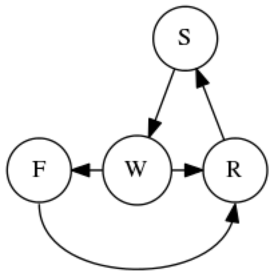
R The `rejuvenation` state when the system is plan to restart to a clean state.

(Huang et al., 1995) decides the rejuvenation thresholds using the following process.

1. compute the probabilistic *mean* rates of every state transitions somehow.
2. calculate the probability of the system being in each state, based on the state transition rates computed above.
3. calculate the *expected* total time of the system in state s during an interval of L time units as $T_s(L) = p_s \times L$, where p_s is the probability the system being in state s calculated in the second step.
4. calculate the cost of downtime during interval L as $Cost(L) = (p_f \times c_f + p_r \times c_r) \times L$, where p_f and p_r are probabilities the system in state F and R respectively, and c_f and c_r are unit cost of the system in state F and R respectively.



(a) Traditional Model 1



(b) Traditional Model 2

Figure 2: Traditional Software Rejuvenation Models

5. determine the optimal rejuvenation thresholds by finding the optimal transition rate between state W to state F so that the cost of downtime is minimum.

The original software rejuvenation model, proposed by Huang et al. (1995), has shown its significance in long running billing systems, scientific applications, and telecommunication systems (Huang et al., 1995). Readers may have noticed that the above estimation methodology gives an approximate statistical estimation for a long running system because the usage of *mean* state transition rates and the *expected* time of being in each state. By *long running system*, we refer to a system whose expected in-service time is a statistically long time regards to its MTTF, repair time, and rejuvenation time.

2.2.2 A Modified Model

Dohi et al. (2000) modifies the original model given by Huang et al. (1995) with the following modifications:

1. As shown in Figure 2(b), the completion of repair process is immediately followed by the software rejuvenation process.

2. Assuming that every state transition is a stochastic processes. For each state S, define a probability density function $F_s(t)$, which represents the probability that the system will stay in that state for time t.

The first modification is made to distinguish the process of system cleanup and process resuming from other repair tasks, the former of which is required in both the repair process and the rejuvenation process in practice. The second assumption is made so that the model is a Semi-Markov process, a well studied stochastic process with off-the-shelf analysis techniques. Although Huang et al. (1995) do not mention the relationship between their model and Markov process, Dohi et al. (2000) show that if the modified model gives the same analysis result as the one given in Huang et al. (1995) if (i) remove assumption 1; and (ii) assume that the sojourn times in all states are exponentially distributed.

Dohi et al. (2000) decides the optimal software rejuvenation thresholds by looking for the one that maximise the system availability, the ratio of uptime and total time. Same as the methodology in Huang et al. (1995), instead of precisely describe state transition probabilities, *mean* time of staying in the `start` state, the `failure` state, and the `rejuvenation` state is used. The expected time of staying the failure probable `working` state is calculated by equation 2.1.1b.

The methodology given in Dohi et al. (2000) does not give a general cost analysis as the one in Huang et al. (1995). It also suffers the problem of using *mean* times. Despite those two limitations, Dohi et al. (2000) derived a non-parametric statistical algorithms to estimate the optimal software rejuvenation thresholds that maximise the system availability, based on statistical complete sample data of failure times.

3 A New Model

3.1 Definitions

To overcome the limitations of classical models, which focus on the equilibrium status that will take a long time to reach, this paper propose a new model that enables effective simulation on early-stages of a system using rejuvenation. The new model works as the follows. It is initialised to a working state and its *software rejuvenate schedule* is set to time T. The failure rate during the working period is a function of the elapsed working time t. A failed system takes t_f time to repair. At the end of a repair process or after working for T time, the system enters to the rejuvenation process for t_r time before being set to the initial working state. Figure 3 gives the transition digram for the new Model, which consists of the following 3 *categories* of states:

- `W(`τ`),` $0 \leq \tau < T$ the state that the system has running without failure for τ time;
- `R(`τ`),` $0 \leq \tau < t_r$ the state that the system has entered the rejuvenation process for τ time.
- `F(`τ`),` $0 \leq \tau < t_f$ the state that the system has failed for τ time, during which period no rejuvenation process has been invoked, but other repair processes, such as data restore, maybe performed.

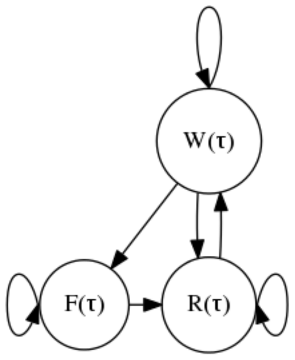


Figure 3: New Model

For the convince of later discussions, this paper index possible state x_i as follows:

Definition 1.

$$x_i = \begin{cases} W(i) & 0 \leq i < T \\ R(i-T) & T \leq i < T + t_r \\ F(i-T-t_r) & T + t_r \leq i < T + t_r + t_f \end{cases}$$

Let X_t be the state of the system at time $t \geq 0$, then the process $X = (X_t, t \geq 0)$ is a stochastic process. If time is a discrete value in the model, the process has discrete time and discrete state space. If time is a continuous value in the model, the process has continuous time and continuous state space.

3.2 Cast Continues State Space to Discrete State Space

As Section 3.3 will show, a standard Markov Chain can be defined if time is treated as a discrete value. Section 3.3.3 will give the simulation process corresponding to the Markov Chain. Although a general Markov process can be defined similarly for the case where time is a continuous value, unfortunately, the general Markov process does not have a precise simulation method. Therefore, this paper converts the continuous case to an approximate discrete case, where a simulation method can be applied.

To convert a model with continuous state space to one with discrete state space, a new time unit, $\Delta t > 0$, is picked such that $T/\Delta t$, $t_r/\Delta t$, and $t_f/\Delta t$ are integers. Let *pdf* be the failure density function in the continues model, then the failure massive function at $T + \Delta t$ in the discrete model is

$$pmf(T + \Delta t) = \int_{T}^{T+\Delta t} pdf(t)dt \tag{3.2.1}$$

Finally, times t in the continuous model can be converted to $t/\Delta t$ in the new discrete model.

3.3 Discrete Model as a Markov Chain

3.3.1 Probability Mass Function and Failure Rate

Let $f(t)$ be the probability mass function (pmf) which represents the probability that, without software rejuvenation, the system will fail when being running for exactly t. We can derive its reliability function ($R(t)$) and the failure rate function ($\lambda(t)$) as follows:

$$R(t) = 1 - \sum_{i=1}^{t-1} f(t) \qquad t > 0 \tag{3.3.1a}$$

$$\lambda(t) = f(t)/R(t) \qquad t > 0 \tag{3.3.1b}$$

Intuitively, $R(t)$ is the probability that the system can survive for as least t time and $\lambda(t)$ is the probability that the system, which has survived for time t, fails at the next moment.

3.3.2 The Transaction Matrix

We define the transaction matrix P whose (i, j)th element is the probability of moving from x_i to x_j in exactly *one* unit of time, we have:

$$p_{ij} = \begin{cases} 1 - \lambda(k) & if\ x_i = W(k), x_j = W(k+1), \\ & \quad 0 \le k < T-1; \\ \lambda(k) & if\ x_i = W(k), x_j = F(0), \\ & \quad 0 \le k < T-1, \\ 1 & if\ x_i = W(T-1), x_j = R(0), \\ & or\ x_i = R(k), x_j = R(k+1), \\ & \quad 0 \le k < t_r - 1, \\ & or\ x_i = R(t_r), x_j = W(0), \\ & or\ x_i = F(k), x_j = F(k+1), \\ & \quad 0 \le k < t_f - 1, \\ & or\ x_i = F(t_f), x_j = R(0); \\ 0 & otherwise \end{cases}$$

It is easy to verify that P satisfies the following two properties of *stochastic matrix*:

a) $p_{ij} \ge 0$, and

b) $\sum_i p_{ij} = 1$.

3.3.3 Simulation

Let μ_t be a row vector of probabilities so that $\mu_t(i)$ represents the probability that the system is at state x_i at time t, we can inductively simulate μ_t as the follows:

$$\mu_0 = \begin{cases} \mu_0(0) = 1 \\ \mu_0(i) = 0 \quad 0 < i < T + t_r + t_f - 1 \end{cases} \tag{3.3.2a}$$

$$\mu_t = \mu_0 P \tag{3.3.2b}$$

Particularly, a row vector π is called a stationary distribution if $\pi = \pi P$.

On the other hand, the Chapman-Kolmogorov Equation states the follows:

$$p_{ij}(m+n) = \sum_k p_{ik}(m) p_{kj}(n) \tag{3.3.3}$$

where $p_{ij}(n)$ is the probability that the system moves from state X_i to state X_j with in exactly n steps.

A nice corollary of Equation 3.3.3 is:

$$P_n = P^n \tag{3.3.4}$$

where P_n is the transaction matrix whose (i, j)th element is the probability of moving from x_i to x_j in exactly n unit of time.

From either Equation 3.3.2 or Equation 3.3.5, we have

$$\mu_t = \mu_0 P^t \tag{3.3.5}$$

a standard property of (time-homogeneous) Markov Chain.

3.3.4 Utility Analysis

Borrowed from economics, the notion of utility function is used to unifies downtime cost analysis, availability analysis, and other variants in literature.

Let $u(i)$ be *utility function* that returns a real value if the system stays at state x_i for one unit of time, the *expected* utility gained at time t, and the *total* utility gained until time t are Equation 3.3.6a and Equation 3.3.6b respectively.

$$u_t = \sum_{i=0}^{T+t_r+t_f-1} u(i) \mu_t(i) \tag{3.3.6a}$$

$$U_t = F_{i=0}^{t} u_i \tag{3.3.6b}$$

where F is an accumulating function.

CASE I: Availability Analysis

The utility function for availability analysis can be defined as:

$$u(i) = \begin{cases} 1 & 0 \leq i < T \\ 0 & T \leq i < T + t_r + t_f - 1 \end{cases} \tag{3.3.7}$$

The target of (Dohi et al., 2000) is finding a value T so that $a = \sum_{i=0}^{T+t_r+t_f-1} u(i)\pi(i)$ has a maximum value. Notice that, the changes of systems availability before the Markov model reaches its steady-state π is ignored in (Dohi et al., 2000).

For a safety crucial system expected to run t time, users may want to use the following objective function instead.

$$A_{min}(t) = \min_{i=0}^{t} u_i \tag{3.3.8}$$

The average availability of a system during its first t time is:

$$\bar{A}(t) = \frac{1}{t} \sum_{i=0}^{t} u_i \tag{3.3.9}$$

If the system can reaches its steady-state π, one may expect that

$$\lim_{t \to \infty} \bar{A}(t) = a \tag{3.3.10}$$

CASE II: Downtime Cost Analysis

As in (Huang et al., 1995), let c_f and c_r be the unit cost of the system in state F and R respectively, we have:

$$u(i) = \begin{cases} 0 & 0 \leq i < T \\ -c_r & T \leq i < T + t_r \\ -c_f & T + t_r \leq i < T + t_r + t_f \end{cases} \tag{3.3.11}$$

The target of (Huang et al., 1995) is finding a value T so that $c = \sum_{i=0}^{T+t_r+t_f-1} u(i)\pi(i)$ has a maximum value. Notice that, the changes of systems availability before the Markov model reaches its steady-state π is ignored in (Dohi et al., 2000). Actually, the expected total cost of running the system for L time is:

$$C(L) = \sum_{t=0}^{L} u_t \tag{3.3.12}$$

Only when the system can reaches its steady-state π, , one may expect that

$$\lim_{L \to \infty} C(L) = cL \tag{3.3.13}$$

CASE III: Benefit Analysis

The utility function in CASE I returns non-negative value while the utility function in CASE II returns non-positive value. In reality, a software application gains revenue when it runs and cost resources to restart it when it is down.

Now consider an international Voice over Internet Protocol (VoIP) service provider, who implements a few independent front-end applications, each of which has the following utility function, where the unit of time is hour:

$$u(i) = \begin{cases} r & 0 \le i < T \\ -c_r & T \le i < T + t_r \\ -c_f & T + t_r \le i < T + t_r + t_f \end{cases} \tag{3.3.14}$$

The goal is set a proper rejuvenation schedule T to maximize the total utility gained during the expect service time L defined as:

$$U(L) = \sum_{t=0}^{L} u_t = \sum_{t=0}^{L} \sum_{i=0}^{T+t_r+t_f-1} u(i)\mu_t(i) \tag{3.3.15}$$

3.4 Reflection on Markov Processes

The model in (Huang et al., 1995) has its root in Continuous Time Markov Chain. The model in (Dohi et al., 2000) is a Continuous Time Semi-Markov Chain. Our model is a devised Discrete Time Markov Chain (DTMC) or a General Markov Process.

The General Markov Process is a precise model for software rejuvenation, but it may not have an analysitcal form for the simulation purpose. Our devised DTMC adds a time parameter to standard DTMC, result in a special form of CTMC and SMC.

The problem of our devised DTMC is the expanded state space. Recall that the simulation process, $\mu_t = \mu_{t-1}P$, computes the product of a $1 \times n$ matrix and a $n \times n$ matrix, where n is the number of states in the model. It appears that the computational cost of the simulation process of the devised DTMC is impractical in a general case where n is a large number.

Fortunately, the devised DTMC is an acceptable model for the software rejuvenation problem discussed in this paper for two reasons. Firstly, non-zero values in the state transition matrix are sparsely distributed, and therefore matrix multiplication does not necessary require a long time to compute. Secondly, similar to the process of reducing a continuous model to a discrete model, the state space of a descrete model can be further reduced for achieving better performance, with the cost of sacrifying accuracy.

4 Numerical Illustrations

4.1 Examples from (Huang et al., 1995)

4.1.1 Example A

- Mean Time Between Failures (MTBF): $12 \times 30 \times 24$ hours
- failure repair time: 30 minutes
- base longevity interval: 7×24 hours
- rejuvenation time: 20 minutes
- average cost of unscheduled downtime: \$1700 / hour
- average cost of scheduled downtime: \$40 / hour

For $12 \times 30 \times 24$ hours			
	no rejuvenation	once every 3 weeks	once every two weeks
Hours of Down Time	0.49	5.965	8.727
\$Cost of Down Time	490	554	586

Table 1: Huang etc's estimation for Example A

4.1.2 Example B

- Mean Time Between Failures (MTBF): $3 \times 30 \times 24$ hours
- failure repair time: 30 minutes
- base longevity interval: 3×24 hours
- rejuvenation time: 10 minutes
- average cost of unscheduled downtime: \$5000 / hour
- average cost of scheduled downtime: \$5 / hour

For $12 \times 30 \times 24$ hours			
	no rejuvenation	once every 2 weeks	once a week
Hours of Down Time	1.94	5.70	9.52
\$Cost of Down Time	9675.25	7672.43	5643.31

Table 2: Huang etc's estimation for Example B

4.1.3 Example C

- Mean Time Between Failures (MTBF): $3 \times 30 \times 24$ hours
- failure repair time: 2 hours
- base longevity interval (BLI): 10×24 hours
- rejuvenation time: 10 minutes
- average cost of unscheduled downtime: \$5000 / hour
- average cost of scheduled downtime: \$5 / hour

For $12 \times 30 \times 24$ hours			
	no rejuvenation	once every 2 weeks	once a week
Hours of Down Time	7019	6.83	6.36
\$Cost of Down Time	3.6K	2.48K	1.11K

Table 3: Huang etc's estimation for Example C

4.1.4 Simulation Results

- 4 experiments for each example: two different rejuvenation schedules, two simulated longevity (1 year and 20 years)
- basic time unit: 10 minutes
- failure distribution: uniform distributio, $U(BLI, 2 \times MTBF - BLI)$

4.1.5 Observations

- In each of the above examples, it takes more than 1.5 years to reach steady-state (see availability graph). the property of a reliable system 1.5 years from now is less valuable as the its property now.
- The cost for the first year is lower than the expected annual cost
- For a reliable system, the main cost is the rejuvenation cost.

4.2 Examples from Dohi

Use the same parameter used in the paper, assume weibull distribution as in the paper

5 Efficiency

In Example HuangB & HuangC. The time is set to 10 minutes. It takes 1 hour to simulate the first year. Not bad!

The matrix is sparse, space efficiency can be improved.

TODO: measure the complexity

6 TODO List

- Examples from Dohi
- Space Efficiency improvement
- ! compose sub-systems

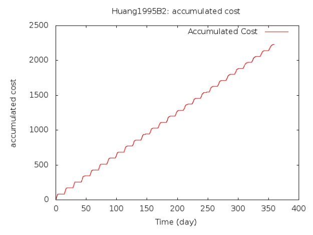
7 Conclusion

References

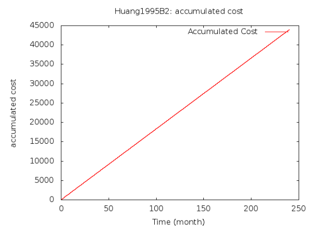
J. D. Musa, *Software Reliability Engineering: More Reliable Software Faster and Cheaper 2nd Edition*, 2nd ed., Sep. 2004.

Y. Huang, C. Kintala, N. Kolettis, and N. D. Fulton, "Software rejuvenation: Analysis, module and applications," in *Fault-Tolerant Computing, 1995. FTCS-25. Digest of Papers., Twenty-Fifth International Symposium on.* IEEE, 1995, pp. 381–390.

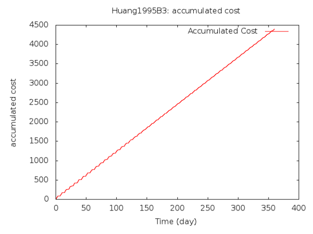
T. Dohi, K. Goseva-Popstojanova, and K. S. Trivedi, "Statistical non-parametric algorithms to estimate the optimal software rejuvenation schedule," in *Dependable Computing, 2000. Proceedings. 2000 Pacific Rim International Symposium on.* IEEE, 2000, pp. 77–84.



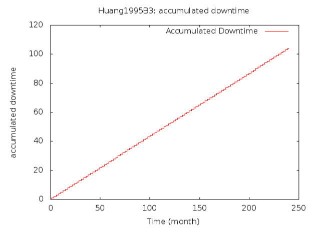
(a)



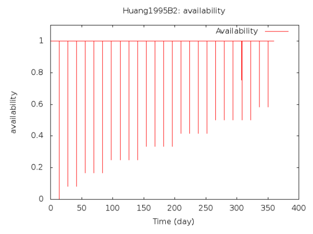
(b)



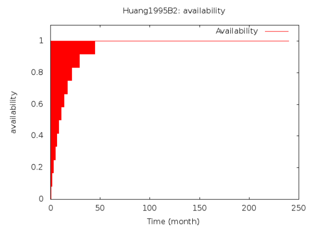
(c)



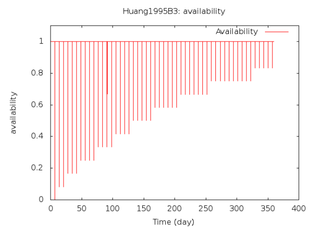
(d)



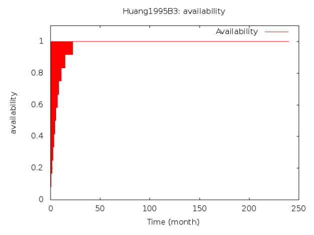
(e)



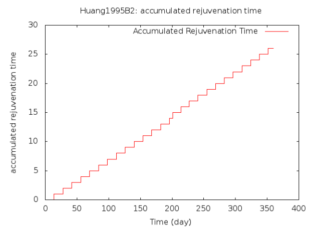
(f)



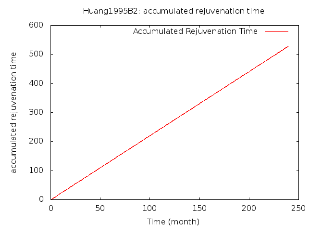
(g)



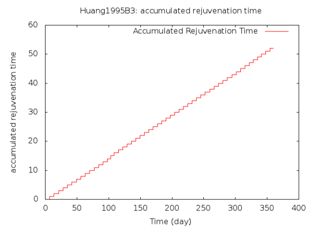
(h)



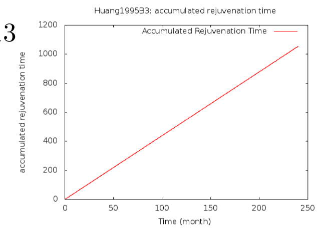
(i)



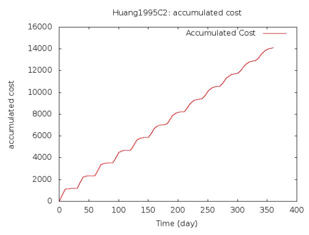
(j)



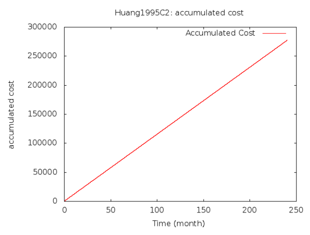
(k)



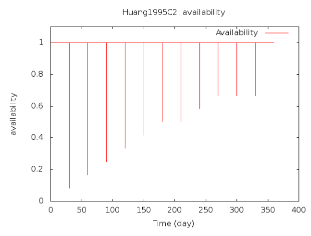
(l)



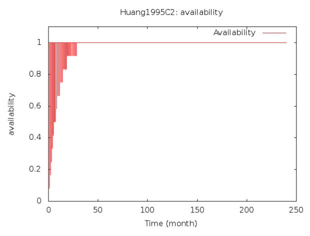
(a)



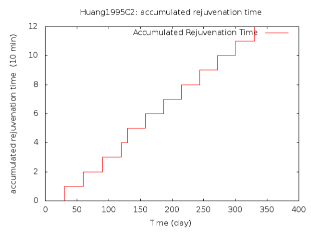
(b)



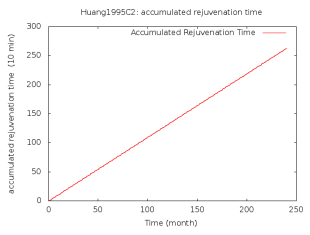
(c)



(d)



(e)



(f)

Figure 5: Example C