Reliability Optimization Models for Fault-Tolerant Distributed Systems

Naruemon Wattanapongsakorn • King Mongkut's University of Technology Thonburi • Bangkok
Steven Levitan • University of Pittsburgh • Pittsburgh

Key Words: Reliability optimization, System reliability, Reliability allocation, System design, Fault-tolerant, Redundancy

SUMMARY & CONCLUSIONS

This paper presents four models to demonstrate our techniques for optimizing software and hardware reliability for fault-tolerant distributed systems. The models help us find the optimal system structure while considering basic information on reliability and cost of the available software and hardware components. Each model is suitable for a distinct set of conditions or situations. All four models maximize reliability while meeting cost constraints.

The Simulated Annealing optimization algorithm is selected to demonstrate system reliability optimization techniques for distributed systems because of its flexibility in applying to various problem types with various constraints, as well as its efficiency in computation time. It provides satisfactory reliability results while meeting the constraints.

1. INTRODUCTION

For mission critical systems, redundancy techniques are commonly applied for high reliability. For distributed systems consisting of hardware, software and network components, redundancy can be achieved by applying extra copies of these components (in parallel) to handle the system work loads. These systems are called fault-tolerant systems; they are capable of tolerating software faults and/or hardware faults. Various system architectures that can tolerate these faults are discussed in Refs. 1–2. The architectures result from different ways of integrating software and hardware redundancy, together with some decision algorithms such as voting, acceptance tests and comparison.

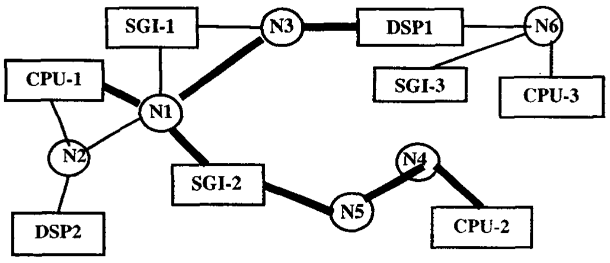

Figure 1: A Distributed System

A distributed system, shown in Figure 1, can be decomposed into a set of applications, each one represented by a series system. Each subsystem in the series system consists of both software and hardware components. The software components are software modules, and the hardware components, are processing units (with operating system, disk, etc.) or network elements. As an example, a series system extracted from Figure 1 is shown in Figure 2, consisting of 10 subsystems and having a simple speech recognition application. Redundancy allocation for this type of system is analyzed in this paper.

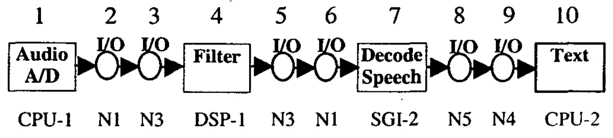

Figure 2: A Speech Recognition Application Running on the Distributed System

These systems are modeled as series-parallel fault-tolerant systems. The redundancy allocation problem for series-parallel systems is known to be difficult (NP-hard). Many researchers have proposed a variety of approaches to solve this problem using, for example, integer programming, branch-and-bound, dynamic programming, mixed integer and nonlinear programming (Refs. 3–6]. Recent optimization approaches (Ref. 2) are based on heuristics such as the Genetic Algorithm (GA), and the Tabu Search (TS). All of these approaches were developed for either optimizing reliability for software or hardware systems. Here we consider systems consisting of both software and hardware components. The software failure behavior, which is different from the hardware failure behavior, is considered toward the system failure.

Therefore, we have developed optimization models to select both software and hardware components and the degree of redundancy to optimize the overall system reliability, with a total cost constraint. In the system, there are a specified number of subsystems in series. For each subsystem, there are several hardware and software choices to be made. The system is designed using commercial off the shelf (COTS) components, each with known cost and reliability.

We use a Simulated Annealing (SA) algorithm as our optimization approach. The term Simulated Annealing derives from the roughly analogous metallurgical annealing process by heating and slowly cooling a substance. SA (Ref. 7) is a heuristic optimization model that has been applied effectively to solve many difficult problems in different fields such as scheduling, facility layout, and graph coloring/ graph partitioning problems. It is a stochastic algorithm with performance depending on the specification of the neighborhood structure and parameter settings for its cooling schedule. The algorithm is based on randomization techniques, having its iterative improvement based on neighborhood (or local) search. The term "temperature" is defined as an SA setting parameter. Decreasing temperature in the cooling schedule corresponds to narrowing of the random search process in the neighborhood of the optimal solution.

The assumption and notation used in these reliability optimization models as well as for the rest of this paper are presented next.

In section 2, the description of the fault-tolerant or redundant architectures that we model, shown in Figure 3, is discussed, followed by our four optimization models, in section 3. Next, in section 4, we describe our Simulated Annealing approach. Lastly, in section 5, we select two example systems to demonstrate the effectiveness of our optimization models, together with results and discussion.

For simplicity, we consider that all hardware redundancy is active, each component is statistically independent from each other, it is non-repairable, and it has 2 states: functional or fail, with known reliability (deterministic), that is either a constant or a time-dependent failure rate.

ASSUMPTIONS

1. Each software component, hardware component or the system has 2 states: functional or fail
2. Reliability of each software or hardware component is known
3. There is no failure repair for each component or system
4. Hardware redundancy is in active mode (i.e., hot spares)
5. Failure of individual hardware components are s-independent

NOTATION

X/i/j	System architecture *X*, with *i* hardware faults tolerated and *j* software faults tolerated
n	Number of subsystem within the distributed system
m_i	Number of hardware component choices available for subsystem *i*
p_i	Number of software versions available for subsystem *i*
R	Estimated reliability of the distributed system
R_i	Estimated reliability of the subsystem *i*
Rhw_{ij}	Reliability of hardware component *j* at subsystem *i*
Rsw_{ij}	Reliability of software component *j* at subsystem *i*
Chw_{ij}	Cost of using hardware component *j* at subsystem *i*
Csw_{ij}	Cost of developing software version *j* at subsystem *i*
Cost	Affordable Cost
Px	Probability that event *X* occurs
Qx	Probability that event *X* does not occur; Qx = 1 – Px
Pv_i	Probability of failure of software version *i*
Prv_{ij}	Probability of failure from related fault between two software versions, *i* and *j*
P_{all}	Probability of failure from related fault among all software versions, due to faults in specification
Pd	Probability of failure of decider or voter
Ph_i	Probability of failure of hardware component *i*. If only one hardware is applied, Ph_i = Ph

2. FAULT-TOLERANT SYSTEM ARCHITECTURES

2.1. *N-version Programming (NVP) Architecture* (Refs. 1–2)

This model consists of an adjudication module called a voter, and N independently developed software versions, which are functionally equivalent. N is usually an odd number. This NVP model is based on the same concepts as N-Modular Redundancy (NMR), which is a hardware fault-tolerant architecture (Ref. 8). In the NVP model, all N software versions are executed for the same task at the same time (i.e., in parallel), and their outputs are collected and evaluated by the voter. The majority of the outputs determine the voter decision. Two subclasses of NVP architecture are discussed in detail next.

2.1.1 *NVP/0/1 Architecture*

This model has zero hardware faults tolerated and a single software fault tolerated, as shown in Figure 3. The system architecture consists of three independent software versions (components) running in parallel on a single hardware component. This system fails if one of the following conditions are true: a single hardware fault, two out of three software version faults, a related fault between software versions, faults from software specification, and faults from the voting algorithm. The probability that an unacceptable result occurs during a single task iteration, 1-R(t), or P unreliability is given by:

$$\begin{gathered}(Prv12 + Qrv12\ Prv13 + Qrv12\ Qrv13\ Prv23) + (Qrv12\ Qrv13\ Qrv23\ Pd) + \\ (Qrv12\ Qrv13\ Qrv23\ Qd\ P_{all}) + (Qrv12\ Qrv13\ Qrv23\ Qd\ Q_{all}\ Ph) + \\ (Qrv12\ Qrv13\ Qrv23\ Qd\ Q_{all}\ Qh\ Pv1\ Pv2 + Qrv12\ Qrv13\ Qrv23\ Qd\ Q_{all}\ Qh \\ Qv1\ Pv2\ Pv3 + Qrv12\ Qrv13\ Qrv23\ Qd\ Q_{all}\ Qh\ Qv2\ Pv1\ Pv3)\end{gathered} \quad \text{Eq. 2-1}$$

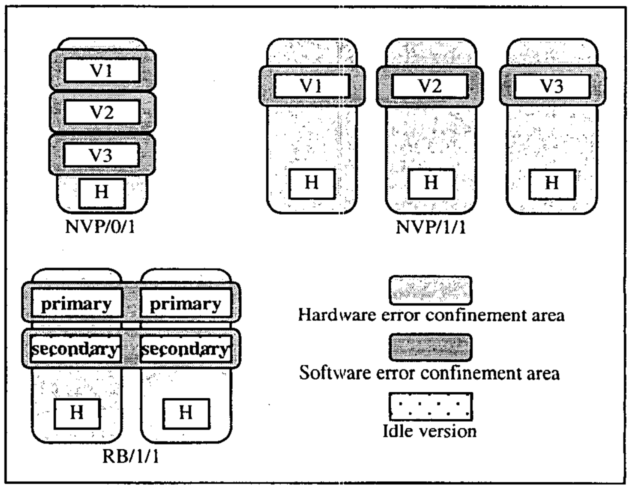

Figure 3: Fault-Tolerant Architectures: NVP/0/1, NVP/1/1 and RB/1/1 (Refs. 1– 2)

2.1.2 *NVP/1/1 Architecture*

This model consists of three independent software versions, each running on a separate hardware component.

Any hardware failure can cause the software running on it to produce unacceptable results. The system is functional if 2 out of 3 software versions (on working hardware) are functioning. Failures of a software version and an unrelated hardware component lead to system failures.

The probability that an unacceptable result occurs during a single task iteration, 1-R(t), or P unreliability is given by

$$\begin{gathered}(Prv + Qrv\,Prv + Qrv^2 Prv) + (Qrv^3\,Pd) + (Qrv^3\,Qd\,P_{all}) + \\ (Pv1\,Pv2\,Qrv^3\,Qd\,Q_{all} + Pv1\,Pv3\,Qv2\,Qrv^3\,Qd\,Q_{all} + Pv2\,Pv3\,Qv1\,Qrv^3\,Qd\,Q_{all}) \\ +Qv2\,Qv3\,Pv1\,Qrv^3\,Qd\,Q_{all}\,Ph^2\,Qh + \\ Qrv^3\,Qd\,Q_{all}\,Ph^2\,Qh\,(1\text{-}Pv3)\,(1\text{-}Pv1\,Pv2) + \\ Qv1\,Qv2\,Pv3\,Qrv^3\,Q_{all}\,Qd\,Ph^2\,Qh + \\ Qrv^3\,Qd\,Q_{all}\,Ph^2\,Qh\,(1\text{-}Pv2)\,(1\text{-}Pv1\,Pv3) + \\ Qv1\,Qv3\,Pv2\,Qrv^3\,Q_{all}\,Qd\,Ph^2\,Qh + \\ Qrv^3\,Qd\,Q_{all}\,Ph^2\,Qh\,(1\text{-}Pv1)\,(1\text{-}Pv2\,Pv3) + \\ Qv2\,Pv1\,Qv3\,Qrv^3\,Q_{all}\,Qd\,Ph\,Qh^2 + \\ Qv2\,Qv3\,Pv1\,Qrv^3\,Q_{all}\,Qd\,Ph\,Qh^2 + \\ Pv2\,Qv1\,Qv3\,Qrv^3\,Q_{all}\,Qd\,Qh^2\,Ph + \\ Qv1\,Qv3\,Pv2\,Qrv^3\,Q_{all}\,Qd\,Qh^2\,Ph + \\ Pv3\,Qv1\,Qv2\,Qrv^3\,Q_{all}\,Qd\,Qh^2\,Ph + \\ Qv1\,Pv3\,Qv2\,Qrv^3\,Q_{all}\,Qd\,Qh^2\,Ph\end{gathered}$$

Eq. 2-2

where $Prv_{12} = Prv_{13} = Prv_{23} = Prv$, and $Ph_1 = Ph_2 = Ph$

2.2 *Recovery Block (RB): RB/1/1 Architecture* (Refs. 1–2)

This model consists of an adjudication module called an *acceptance test*, and at least two software components, called alternates. At the beginning, the output of the first or primary alternate is tested for acceptance. If it fails, the process will *roll back* to the beginning of the process, and then let the second alternate execute and test its output for acceptance again. This process continues until the output from an alternate is accepted or all outputs of the alternates have been tested and fail.

The system consists of two hardware components, each running two independent software versions; primary and secondary. The primary version is active until it fails, and the secondary version is the backup spare. The system failures occur when both versions fail, or both hardware components fail. The probability that an unacceptable result occurs during a single task iteration, P unreliability is given by

$$Prv + Qrv\,Pd + Qrv\,Qd\,Prall + Qrv\,Qd\,Q_{all}\,Ph^2 + Qrv\,Qd\,Q_{all}\,(1\text{-}Ph^2)\,Pv1\,Pv2$$

Eq. 2-3

With these fault-tolerant architectures, we develop four optimization models for reliability of distributed systems, where each model covers different fault-tolerant structures suitable for different situations. The models are formulated as follows.

3. MODELS FORMULATION

Model 1: Find the optimal set of software and hardware allocations for all subsystems (without redundancy). The problem formulation is to maximize the system reliability, subjected to a specified cost constraint, *Cost*. The system has all subsystems connected in series. Each subsystem reliability is the product of a chosen software version and a hardware component available for selection.

Formulation: Maximize system reliability by choosing the optimal set of hardware and software components for each subsystem by:

$$Max\quad R = \prod_{i=1}^{n} R_i$$

Subject to

$$\sum_{j=1}^{m_i} X_{ij} = 1, \qquad i = 1, 2, \ldots, n$$

$$\sum_{k=1}^{p_i} Y_{ik} = 1, \qquad i = 1, 2, \ldots, n$$

$$\sum_{i=1}^{n}\sum_{j=1}^{m_i} X_{ij}C_{ij} + \sum_{i=1}^{n}\sum_{k=i}^{p_i} Y_{ik}C_{ik} \le \text{Cost}$$

$X_{ij} = 0,1 \quad Y_{ij} = 0,1 \quad i = 1, 2, \ldots, n \quad j = 1,2,\ldots,m_i \quad k = 1,2,\ldots,p_i$

where
$$R_i = \sum_{j=1}^{m_i}\sum_{k=1}^{p_i} X_{ij} Y_{ik}\, Rhw_{ij}\, Rsw_{ik}$$

Model 2: Find the optimal set of software and hardware allocations for all subsystems (with or without NVP/0/1 redundancy). This and the following models are suited for systems that handle more critical tasks. The problem formulation for this model is the same as in the previous model, except that each subsystem uses NVP/0/1 redundancy allocation as its reliability, calculated according to the NVP/0/1 redundancy configuration. The parameters considered for the reliability of the NVP architecture are assumed to be available as constant values. Each allocated software version is allowed to have a different reliability value, unlike several proposed models where all of the software versions have the same reliability value (Ref. 2).

Formulation: Maximize system reliability by choosing an optimal set of hardware and software components for each subsystem by:

$$Max\quad R = \prod_{i=1}^{n} R_i$$

Subject to

$$\sum_{j=1}^{m_i} X_{ij} = 1, \qquad i = 1, 2, \ldots, n$$

$$\sum_{k=1}^{p_i} Y_{ik} = 1 \ or\ 3, \qquad i = 1, 2, \ldots, n$$

$$\sum_{i=1}^{n}\sum_{j=1}^{m_i} X_{ij}C_{ij} + \sum_{i=1}^{n}\sum_{k=1}^{p_i} Y_{ik}C_{ik} \le Cost$$

$X_{ij} = 0,1 \quad Y_{ij} = 0,1 \quad i = 1, 2, \ldots, n \quad j = 1,2,\ldots,m_i \quad k = 1,2,\ldots,p_i$

where
$$R_i = \sum_{j=1}^{m_i}\sum_{k=1}^{p_i} X_{ij} Y_{ik}\, Rhw_{ij}\, Rsw_{ik}, \ if \quad \sum_{k=1}^{p_i} Y_{ik} = 1$$

$$R_i = Eq.(1), \qquad if \quad \sum_{k=1}^{p_i} Y_{ik} = 3$$

Model 3: Find the optimal set of software and hardware allocations for all subsystems (with or without NVP/1/1 redundancy). This model extends model 2, but instead of zero hardware faults tolerated, it has a single hardware fault tolerated.

$$Max\quad R = \prod_{i=1}^{n} R_i$$

Subject to

$$\left(\sum_{j=1}^{m_i} X_{ij} = 1 \ and \ \sum_{k=1}^{p_i} Y_{ik} = 1\right) or \left(\sum_{j=1}^{m_i} X_{ij} = 3 \ and \ \sum_{k=1}^{p_i} Y_{ik} = 3\right)$$

$$\sum_{i=1}^{n}\sum_{j=1}^{m_i} X_{ij}C_{ij} + \sum_{i=1}^{n}\sum_{k=1}^{p_i} Y_{ik}C_{ik} \le Cost$$

$X_{ij} = 0,1,3 \quad Y_{ij} = 0,1 \quad i = 1, 2, \ldots, n \quad j = 1,2,\ldots,m_i \quad k = 1,2,\ldots,p_i$

where

$$R_i = \sum_{j=1}^{m_i}\sum_{k=1}^{p_i} X_{ij}Y_{ik}Rhw_{ij}Rsw_{ik}, \quad if \ \sum_{j=1}^{m_i} X_{ij} = 1, \ \sum_{k=1}^{p_i} Y_{ik} = 1$$

$$R_i = Eq.(2), \quad if \ \sum_{j=1}^{m_i} X_{ij} = 1, \ \sum_{k=1}^{p_i} Y_{ik} = 3$$

Model 4: Find the optimal set of software and hardware allocations for all subsystems (with or without RB/1/1 redundancy). This model is also based on model 2, but captures optimization analysis for the Recovery Block architecture.

$$Max \ R = \prod_{i=1}^{n} R_i$$

Subject to

$$\left(\sum_{j=1}^{m_i} X_{ij} = 1 \ and \ \sum_{k=1}^{p_i} Y_{ik} = 1\right) or \left(\sum_{j=1}^{m_i} X_{ij} = 2 \ and \ \sum_{k=1}^{p_i} Y_{ik} = 2\right)$$

$$\sum_{i=1}^{n}\sum_{j=1}^{m_i} X_{ij}C_{ij} + \sum_{i=1}^{n}\sum_{k=1}^{p_i} Y_{ik}C_{ik} \le Cost$$

$$X_{ij} = 0,1,2 \quad Y_{ij} = 0,1 \quad i = 1,2,\dots,n \quad j = 1,2,\dots,m_i \quad k = 1,2,\dots,p_i$$

where

$$R_i = \sum_{j=1}^{m_i}\sum_{k=1}^{p_i} X_{ij}Y_{ik}Rhw_{ij}Rsw_{ik}, \quad if \ \sum_{j=1}^{m_i} X_{ij} = 1, \ \sum_{k=1}^{p_i} Y_{ik} = 1$$

$$R_i = Eq.(3), \quad if \ \sum_{j=1}^{m_i} X_{ij} = 2, \ \sum_{k=1}^{p_i} Y_{ik} = 2$$

4. SIMULATED ANNEALING ALGORITHM APPROACH

Our optimization is based on the Simulated Annealing optimization algorithm. For more information on this technique, see Ref. 7. The optimization behavior of the Simulated Annealing algorithm is determined by several parameters and a cost function. For these experiments, the cost is simply the sum of the costs of the components. The parameter settings for the optimization are described below.

Initial Solution: In our tests, the initial solution is selected randomly. In general, if we have some pre-defined knowledge of some particular subsystems (hardware and/or software units), we can select or fix the initial solution for the subsystems correspondingly.

Annealing Schedules (temperature reduction function): We set the temperature steps, *x*, in the range of 75-85, because the problem which we are trying to solve has a moderate problem size For this problem, the search space sizes for models 1-4 are 2.99 x10^6, 1.9x 10^8, 1.9x 10^8 and 7.3 x 10^8 respectively.

We implement a cooling schedule using the geometric function: $T = initT * Tfactor^x$, where T = temperature, x is the temperature step, *initT* is an initial temperature, and *Tfactor* is a factor to decrease temperature in each step. This geometric function causes the algorithm to start with a rapidly decrease in temperature, and slowly cool down at the later temperature steps.

From preliminary simulations, appropriate initial temperature, *initT*, ranges from 100 to 140. Lower temperatures from this range result in getting stuck in a local minimal, and higher temperatures would produce redundant iterations. The value of *Tfactor* used is in the range of 0.70-0.90. Smaller values of *Tfactor* cause the cooling temperature to be reduced too fast. This results in an insufficient random search process in each neighborhood, that is at each temperature step.

The number of iterations per temperature step is an increasing function:

$$F\ (iteration) = iteration * 1.18^x$$

where x = temperature step, the same as above. This iteration function causes more iterations in the later stages of the search, at lower temperature, so that only better solutions, with higher reliability and less cost are accepted; unlike the beginning of the search where most solutions are accepted. With a higher number of iterations, near-optimal or optimal solutions are expected to be found. The constant 1.18 is a predefined exponent, that works well for this particular problem. This number was obtained from our preliminary runs.

Summary of the values and ranges of the SA parameters are presented in Table 1.

SA Parameters	Values and Ranges
Temperature Steps (*x*)	75-85
Initial Temperature (*initT*)	100-140
Factor to Decrease Temperature (*T-factor*)	0.70-0.90
Factor to Increase the number of Iterations	1.18

Table 1: SA Parameters

Cost Constraint Consideration: We reject solutions that cannot meet the cost constraint. **Selection of New Neighborhood Solution:** Only one subsystem is randomly selected to be perturbed at each iteration. A new hardware component and a software component for the selected subsystem are randomly chosen. **Stopping Criterion:** We terminate the program when there is no improvement in the best solution within a specified number of temperature steps.

5. EXAMPLE SYSTEMS AND COMPUTATIONAL RESULTS

5.1 *General System*

Suppose we have a series system with 6 subsystems, having n = 6, m_i = 3, and p_i = 4. The reliability and cost of hardware and software components are given as inputs shown in Table 2. The results (based on 10 runs each), from our four models are shown as selected hardware and software components presented in the right half of the table for the given system cost constraints 180.0, 320.0, and 460.0 units. Since SA approach does not guarantee optimal solutions, it is necessary for us to have several simulation runs. The cost values here are chosen arbitrarily for illustration purposes. Each "h "and "s" indicates that the corresponding hardware or software component has been chosen in that solution to the optimization problem.

Notes: (i, j) = subsystem i, hardware component type j, and (i, k) = subsystem i, software component (version) type k.

We assume that the probability of failure of the voter/decider is 0.02, and probabilities of the related fault between software versions, and the related fault due to errors in software specification are 0.004, and 0.005, respectively.

Inputs						Results											
(i, j)	HW Cost	HW Reliability	(i, k)	SW Cost	SW Reliability	System Cost											
						180.0				320.0				460.0			
						1	2	3	4	1	2	3	4	1	2	3	4
11	30.0	0.995	11	30.0	0.950					h,s	h	h,s	s	h,s	h,s	s	s
12	10.0	0.980	12	10.0	0.908	h, s	h, s	h, s	h, s		s		h,h, s		s	s	2h
13	10.0	0.980	13	20.0	0.908						s					3h	
			14	30.0	0.950						s				s	s	s
21	30.0	0.995	21	30.0	0.965					h,s	s	s	s	h,s	s	s	s
22	20.0	0.995	22	20.0	0.908						h	h	s		h	h	2h
23	10.0	0.970	23	10.0	0.887	h, s	h, s	h, s	h, s				h,h				
			24	20.0	0.908												s
31	20.0	0.994	31	20.0	0.978	s	s	s	s	s	h,s	h,s	s	s	h,s	s	s
32	30.0	0.995	32	30.0	0.954	h	h	h	h	h			h	h		h	h
33	100.0	0.992	33	20.0	0.860												
			34	30.0	0.954												
41	30.0	0.990	41	20.0	0.950					h,s	s	s	s	h,s	h,s	s	s
42	10.0	0.980	42	10.0	0.908						s	s	s		s		2h
43	10.0	0.985	43	20.0	0.910	h	h	h	h		h	3h	h,h			3h, s	
			44	20.0	0.950	s	s	s	s		s	s			s	s	s
51	30.0	0.995	51	30.0	0.905					h	h	h		h	h	h	
52	20.0	0.980	52	20.0	0.967	h	h	h	h	s			h,s	s	s		s
53	30.0	0.995	53	10.0	0.967	s	s	s	s		s	s			s	s	2h, s
			54	30.0	0.905										s		
61	30.0	0.998	61	10.0	0.908					h		h		h	h		
62	20.0	0.995	62	30.0	0.968	h	h	h	h		h				s	3h, s	2h, s
63	20.0	0.994	63	20.0	0.968	s	s	s	s	s	s	s	h,s	s	s	s	s
			64	20.0	0.955										s	s	

Table 2: Inputs and Outputs (components selected from models 1-4)
where h = selected hardware component, and s = selected software component

To test if our models can find optimal reliability solutions, we first find the optimal component allocation for each model by selecting the best available components from inputs in Table 2.

For example, from the given reliability of the software and hardware components, the maximum system reliability, without redundancy, can be obtained easily, by manually selecting the software component, as well as the hardware component, with the highest reliability for each subsystem. Then, we calculate the total system reliability and cost based on the selected components. These results, shown in Table 3, are 0.772099 for system reliability, and 320.0 for system cost. Similarly, we find the optimal component allocation for models 2-4 assuming no cost constraints. This can be obtained just by selecting the maximum number of software and hardware components allowed by the models, for each subsystem. For example, with model 2 where NVP/0/1 redundancy is available for each subsystem, we select the best single hardware and three software components with the highest reliability for each subsystem. This results in finding the optimal results for models 2-4 without cost constraints, as shown in Table 3.

Model	Optimal System Reliability	At Cost
No.1: No Redundancy	0.772099	320.0
No.2: NVP/0/1	0.820978	590.0
No.3: NVP/1/1	0.838132	920.0
No.4: RB/1/1	0.829525	620.0

Table 3: Optimal System Reliability without Cost Constraint (by inspection)

Next, we evaluate all our optimization models by setting the cost constraint to 320.0. With the Simulated Annealing (SA) approach, our reliability result from model 1 is identical to the optimal result. We repeat this evaluation process for models 2-4, by setting the cost constraint for each model according to the cost in Table 3. As a result, all our models give the same reliability results as the optimal results with these cost constraints, or without cost constraints.

When we tighten the cost constraint to 180.0, as shown in Table 4, all our models give the same system reliability results. The results of component and redundancy allocations, shown as the outputs in Table 2, show that with this cost, no component redundancy can be obtained. The consistent results prove that our models are accurate in finding solutions, which could be the optimal result at this cost constraint.

Model	Cost 180.0	Cost 320.0	Cost 460.0
No.1: No Redundancy	0.636327	0.772099	0.772099
No.2: NVP/0/1	0.636327	0.793888	0.818629
No.3: NVP/1/1	0.636327	0.791116	0.815219
No.4: RB/1/1	0.636327	0.820243	0.831441

Table 4: Results of System Reliabilities from the Optimization Models

Furthermore, when we loosen the cost constraint to 460.0, without redundancy, model 1 can still find the optimal solution, which is 0.772099 as shown in Table 4. Similar to the cost constraint at 320.0, models 2 - 4 give better reliability results, because redundancy can be obtained. At any fixed system cost, the recovery block (RB) fault-tolerant architecture, that can be obtained from model 4, offers the same or better system reliability than the N-version programming (NVP) architecture obtained from models 2 - 3.

We note that the reliability values of components and the cost constraints of 180.0, 320.0 and 460.0 are chosen arbitrarily. However, we can see that, with a very limited system cost (shown here as 180.0), no fault-tolerant architecture can be obtained, resulting in the same system reliability of 0.636327 reported from all models. With higher system costs, the recovery block architecture gives the best reliability, given enough resources, which is consistent with other published results (Ref. 2).

5.2 *Embedded Distributed Speech Recognition System*

Extending our optimization models to a distributed system, shown in Figure 2, which consists of network-type subsystems (subsystems 2, 3, 5, 6, 8, and 9) and processor-type subsystems (subsystems 1, 4, 7, and 10). Each network subsystem has fixed component allocation and no redundancy. In addition, subsystems 2 and 6 use the same resource N1, and subsystems 3 and 5 use the same resource N3. Each resource is counted once towards the calculation for system reliability, therefore either subsystems 2 or 6 and either subsystems 3 or 5 can be considered for system reliability. Since the reliability for each network-type subsystem is known, to optimize the overall system reliability, only the component allocation and redundancy for processor-type subsystems is unknown and is the task to be solved. This results in a much smaller size of optimization problem. Instead of optimizing the problem with 10 subsystems, we can just optimize a partial subsystem with four processor-types, shown in Figure 4, and then combine it correspondingly with the network-type subsystems, shown in Figure 5.

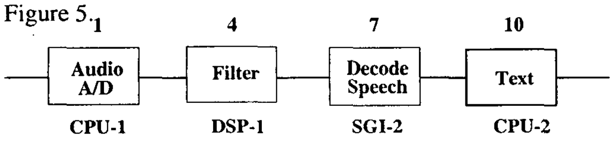

Figure 4: Processor-type Subsystems

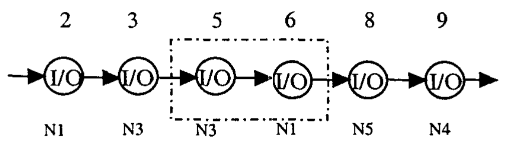

Figure 5: Network-type Subsystems

The cost and reliability for each software and hardware components available for each subsystem is shown in Table 5. There are three hardware component choices, and four software versions available for each component subsystem. Each of the hardware components consists of a hardware processing unit and its system software, i.e., the operating system. Each of the software versions represents application software designed specifically for a certain task such that software versions for the subsystem 1 perform audio A/D processing task, software versions for the subsystem 2 perform filter processing task, etc. Each of the network subsystems has the reliability and cost of hardware but not software, since it requires no application software for its operation.

(i,j)	HW Cost	HW Reliability	(i,j)	SW Cost	SW Reliability
1,1	40	0.990	1,1	177	0.995
1,2	35	0.980	1,2	150	0.950
1,3	30	0.970	1,3	110	0.950
			1,4	90	0.940
2,1	25	0.995	2,1	-	-
3,1	25	0.995	3,1	-	-
4,1	130	0.995	4,1	200	0.980
4,2	100	0.96	4,2	160	0.980
4,3	70	0.951	4,3	155	0.920
			4,4	100	0.950
5,1	25	0.995	5,1	-	-
6,1	25	0.995	6,1	-	-
7,1	135	0.998	7,1	250	0.980
7,2	120	0.980	7,2	195	0.895
7,3	95	0.970	7,3	192	0.970
			7,4	140	0.975
8,1	20	0.980	8,1	-	-
9,1	20	0.980	9,1	-	-
10,1	40	0.990	10,1	125	0.990
10,2	35	0.985	10,2	100	0.970
10,3	30	0.970	10,3	80	0.965
			10,4	47	0.965

Table 5: Inputs for Problem 2 (Components Available for the Four Models)

Case	Assumptions
1	Prv = 0.004, P_{all} = 0.005
2	Prv = 0.000, P_{all} = 0.005
3	Prv = 0.000, P_{all} = 0.000

Table 6: Probability of Related Faults in Multiple Software Versions

Applying this input data set, we perform system reliability optimization with cost constraints, using model 1 where no redundancy is allowed, and using model 4, where a choice of redundancy architecture RB/1/1 is available for each subsystem.

To verify the effects of related faults in the software versions to the system reliability, we assume three different cases for the probability of related faults between versions, Prv, and among software versions, P_{all}, shown in Table 6. We assume that, the probability of failure of the voter/decider is assumed to be 0.002 for each test case. In case 1, both related faults exist, while in case 2, related faults between versions do not exist, and in case 3, we assume no related faults or dependency failures in the software versions.

Results and Discussion

The corresponding system reliabilities for all cases at different costs are presented in Table 7. As expected, lower cost constraints (i.e., a higher cost budget) gives a higher system reliability provided by redundant components. At a cost of 1057, the maximum system reliability without component redundancy is obtained using model 1. At the same cost, the system reliability in model 4, however, can be increased via component redundancy.

In addition, with fewer or no related faults between or among software versions, higher system reliabilities are obtained, such that case 3 gives the highest system reliability results, and case 1 gives the lowest reliability results. These results are consistent with the models for the reliability of embedded systems with related faults discussed in Ref. 2.

6. FUTURE WORK

We are integrating these models into our dependability analysis framework (Refs. 9–10), which is integrated into the TimeWiz system design tool (Ref. 11). The result of this integrated work will be a fully integrated performance and dependability analysis and optimization framework for the design of fault-tolerant distributed embedded systems.

Case Number	Cost = 800		Cost = 1057		Cost = 1600		Cost = 1952	
	#1	#4	#1	#4	#1	#4	#1	#4
1	0.810574	0.810574	0.875453	0.880434	0.875453	0.905813	0.875453	0.908282
2	-	0.810574	-	0.885657	-	0.920452	-	0.922961
3	-	0.810574	-	0.894581	-	0.939093	-	0.941654

Table 7: Results of System Reliabilities from Problem 2

REFERENCES

1. J.-C. Laprie, J. Arlat, C. Beounes, and K. Kanoun, Definition and Analysis of Hardware- and Software-Fault-Tolerant Architectures, IEEE Computer, July 1990, pp 39-51.
2. M. R. Lyu, Editor in Chief, Handbook of Software Reliability Engineering, IEEE Computer Society Press, McGraw-Hill, 1996.
3. O. Berman, N Ashrafi, " Optimization Models for Reliability of Modular Software Systems", IEEE Trans. Software Engineering, vol 19, no. 11, 1993 Nov, pp 1119-1123.
4. D. W. Coit, A. E. Smith, "Reliability Optimization of Series-Parallel Systems Using a Genetic Algorithm", IEEE Trans. Reliability, vol 45, no. 2, 1996 June, pp 254-260.
5. D. W. Coit, A. E. Smith, "Adaptive Penalty Methods for Genetic Optimization of Constrained Combinatorial Problems", INFORMS Journal on Computing, vol 8, no. 2, 1996 Spring, pp173-182.
6. M. E. Helander, M. Zhao, N. Ohlsson, "Planning Models for Software Reliability and Cost", IEEE Trans. Software Engineering, vol 24, no. 6, 1998 June, pp 420-434.
7. Z. Michalewicz, D. Fogel, "How to Solve It: Modern Heuristics," Springer Verilog, 1999.
8. C. E. Ebeling, *An Introduction to Reliability and Maintainability Engineering*, McGraw-Hill Companies, Inc., 1997, pp 23-121.
9. N. Wattanapongsakorn, "Integrating Dependability Analysis and Optimization into the Distributed Embedded System Design Process", Ph.D. Thesis, University of Pittsburgh, 2000.
10. N. Wattanapongsakorn, S. P. Levitan, "Integrating Dependability Analysis into the Real-time System Design Process", Proc. Ann. Reliability & Maintainability Symp., 2000 Jan.
11. TimeWiz, "An Integrated Design Environment for Real-time Systems", TimeSys Corporation, http://www.timesys.com

BIOGRAPHIES

Naruemon Wattanapongsakorn, PhD, EE
Department of Computer Engineering
King Mongkut's University of Technology Thonburi,
91 Suksawasd 48, Ratburana, Bangkok 10140 Thailand

Internet (email): naruemon@cpe.eng.kmutt.ac.th

Naruemon Wattanapongsakorn is a faculty member in Computer Engineering at the King Mongkut's University of Technology Thonburi (KMUTT), Thailand. She received the B.S. degree in Computer Engineering (1994) and the M.S. degree in Electrical Engineering (1995), both from The George Washington University, and Ph.D. degree in Electrical Engineering (2000) from the University of Pittsburgh. Her research interests include distributed system dependability analysis, optimization algorithms, real-time system modeling, software fault-tolerance, and statistical analysis of system reliability. She is a member of the IEEE.

Steven P. Levitan, PhD, CS
348 Benedum Engineering Hall,
University of Pittsburgh, Pittsburgh, PA 15261 USA

Internet (email): steve@ee.pitt.edu

Steven P. Levitan is the John A. Jurenko Professor of Computer Engineering at the University of Pittsburgh. He received the B.S. degree from Case Western Reserve University (1972) and his M.S. (1979) and Ph.D. (1984), both in Computer Science, from the University of Massachusetts, Amherst. He worked for Xylogic Systems designing hardware for computerized text processing systems and for Digital Equipment Corporation on the Silicon Synthesis project. He was an Assistant Professor from 1984 to 1986 in the Electrical and Computer Engineering Department at the University of Massachusetts. In 1987 he joined the Electrical Engineering faculty at the University of Pittsburgh. His research interests include VLSI architectures, optoelectronic computing systems, parallel algorithm design, and HDL simulation and synthesis for VLSI. He is an Associate Editor of the ACM Transactions on Design Automation of Electronic Systems. He is Chair of the ACM Special Interest Group on Design Automation, a member of OSA, and a senior member of IEEE/CS.