



On the Scheduling of Flexible and Reliable Real-Time Control Systems

RAMESH CHANDRA
Department of Computer Science, Stanford University, USA

ramesh.chandra@cs.stanford.edu

XUE LIU
Department of Computer Science, University of Illinois at Urbana-Champaign

xueliu@cs.uiuc.edu

LUI SHA
Department of Computer Science, University of Illinois at Urbana-Champaign

lrs@cs.uiuc.edu

Abstract. Many real-time systems, such as manufacturing plants, have long life cycles. To enable the realization of technological innovations and to mitigate the risk and cost of bringing new control technologies into functioning systems, flexible and reliable real-time software architectures such as Simplex have been developed. There is also an emerging trend that integrates the design of controllers and schedulers. For example, algorithms that identify the optimal frequencies of control tasks subject to schedulability constraints have been developed, and the notion of feedback schedulers has been investigated.

However, the optimization of the performance of flexible and reliable architectures with analytically redundant software controllers has remained an open problem. In fact, a direct application of the existing optimization methods would not yield the optimal frequencies. In this paper, we present a method that correctly finds the optimal frequencies for systems using analytically redundant controllers. We also show that the proposed method is robust against inaccuracies in the estimation of failure rates of the controllers.

Keywords: real-time control systems, scheduling, dependable systems, analytical redundancy, co-design of controller and scheduler

1. Introduction

Computers and computer networks have revolutionized the production of goods and the delivery of services. However, the existing computing infrastructure also places formidable barriers to continuous process improvement, equipment upgrades, and agility in responding to changing markets and increased global competition. Consider the following anecdotal scenario from industry.

1.1. Process Improvement

A research department developed a process modification that significantly improved the product yield and quality. With a relatively minor modification of the processing sequence and new set-points for key process variables, the improvements were demonstrated in the laboratory. Nevertheless, these improvements were never implemented in the plant, because the line manager persuaded management that it could not be done cost-effectively. Although the required software modifications were

simple logic modifications, the process sequence was controlled by a set of networked PLCs coordinating many valves, sensors and PID loops. The last time a modification was attempted on the code, it took down the process completely, costing thousands of dollars in downtime. As a result, the line manager wanted no part in installing the so-called process improvements developed by the research department.

The Simplex architecture (Sha, 1998) was developed to solve such problems. More specifically, it was developed to (1) maintain a given level of system performance and functionality in spite of logical faults that could be introduced by changes in control software, and (2) permit changes in real-time control software without any need for shutting down the system's normal operations. The Simplex architecture uses analytically redundant controllers for software defect-tolerance. There is also an emerging trend of co-designing real-time schedulers and controllers. For example, Seto et al. (1996) developed an approach to select the frequencies that will optimize the control performance subject to schedulability constraints, and Stankovic et al. (1999) investigated the use of feedback schedulers.

However, the problem of optimization of the performance of control systems using analytically redundant controllers has remained open. In fact, a direct application of the approach in Seto et al. (1996) would not produce optimal frequencies, since Seto et al. (1996) assumes that each task controls a device, whereas in software fault-tolerant control only one of the analytically redundant controllers would be used to control the device at any time. In this paper, we generalize the results in Seto et al. (1996) and show how to identify the optimal frequencies correctly when redundant controllers are used. Furthermore, we also perform a sensitivity analysis of the proposed method and show that the method is robust against inaccuracies in the estimation of failure rates.

The rest of the paper is organized as follows. In Section 2, we provide a brief review of the technical background of this work. In Section 3, we present our algorithm for determining optimal task frequencies when analytically redundant controllers are used. In Section 4, we give some examples illustrating our algorithm. It is well-known that it is difficult to estimate the failure rate of software accurately. In Section 5, we show that our optimization technique is robust in the sense that it is insensitive to inaccuracies in the estimation of the failure rate. Finally, we present our conclusions in Section 6.

2. Background

In this section, we briefly review the concepts of analytical redundancy and the notion of performance loss index. These concepts form the base upon which we develop our solution techniques.

2.1. Forward Recovery Using Analytically Redundant Controllers

In fault-tolerant control, it is very difficult to determine if a controller's instantaneous output is correct, unless it is obviously wrong (such as an out-of-range control command). Some of the incorrect outputs from a faulty controller will inevitably be sent to the plant. Forward recovery techniques ensure that the results caused by incorrect actions are both

tolerable and recoverable. A good everyday example related to forward recovery in control is pilot training. The trainer will allow the student to make mistakes as long as the plane remains in a state that is stable and controllable by the trainer.

A noteworthy example of forward recovery using analytically redundant controllers in the real world is the flight control system of the Boeing 777 (Yeh, 1995). It uses triple-triple redundancy for system-level reliability. At the software application level, it uses two controllers. The sophisticated control software specifically developed for the Boeing 777 is the normal controller. The secondary controller is based on the control laws originally developed for the Boeing 747. The normal controller is very complex and is able to deliver optimized flight control over a wide range of conditions. On the other hand, control laws developed for the Boeing 747 have been used for over 20 years. It is a mature, old technology—simple, reliable and extremely well understood. To exploit the advantage of advanced control technologies and also ensure an unquestionable degree of reliability, a Boeing 777 under the control of the normal controller should fly within the stability envelope of its secondary controller. This is a fine example of using analytically redundant controllers to avoid potential faults in complex software systems.

The Simplex architecture (Sha, 1998) is a flexible and reliable software architecture that uses analytically redundant controllers. In addition, it supports the online insertion of new software controllers that implement advanced new control technologies without requiring that the system's normal operation be shut down. It also guarantees stability and a baseline control performance in the new software controllers, in spite of arbitrary application level errors. In the Simplex architecture, a device is controlled by two analytically redundant controllers, one that is a high-assurance (or high-reliability) controller, and another that is a high-performance controller. The high-assurance controller contains simple, mature, and highly reliable technology, typically with lower control performance. The high-performance controller contains the latest (but maybe not completely proven) optimizations, and hence could be less reliable. The system is normally under the control of the high-performance controller, with the high-assurance controller "supervising" it. We now provide a brief review of this approach. A detailed treatment of this subject can be found in Seto and Sha (1999).

In the operation of a plant (e.g., a car), there is a set of state constraints, called operation constraints, representing the safety requirements, physical device limitations, and environmental and other operation requirements. The operation constraints can be represented as a normalized polytope, $C^T X \leq 1$, in the N -dimensional state space of the system, as shown in Figure 1. Each line on the boundary represents a constraint. An example of a constraint is that the rotation of a car engine must not be greater than K rpm.

The states inside the polytope are called admissible states, because they obey the operation constraints. To limit the loss that can be caused by a faulty controller, we must ensure that the system states are always admissible. This means that (1) it must be possible to take the control away from a faulty control subsystem and switch it to the high-assurance control subsystem before the system state becomes inadmissible, (2) the system is controllable by the high-assurance control subsystem after the switch, and (3) the future trajectory of the system state after the switch will stay within the set of admissible states. Note that since physical systems have inertia, we cannot use the

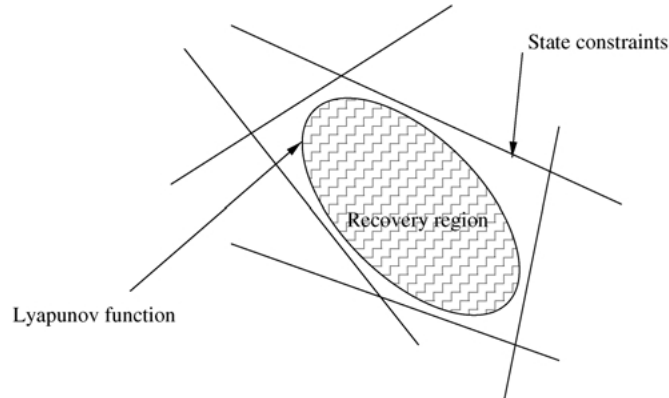


Figure 1. State constraints and the switching rule (Lyapunov function).

boundary of the polytope as the switching rule. For example, we cannot stop a car without collision when the car is about to touch a wall.

A subset of the admissible states that satisfies these three conditions is called a recovery region associated with the high-assurance controller. The recovery region is represented by a Lyapunov function inside the state constraint polytope. Geometrically, a Lyapunov function defines an N -dimensional ellipsoid in the N -dimensional system state space, as illustrated in Figure 1. A Lyapunov function satisfies the important property that as long as the system state is inside the ellipsoid associated with the controller, the system states under the given controller's control will stay within the ellipsoid and converge to the equilibrium position, which is the state that we want the system to reach. Thus, we can use the boundary of the ellipsoid as the switching rule.

The Lyapunov function is not unique for a given system and controller combination. In order not to unduly restrict the state space that can be used by high-performance controllers, we need to find the largest ellipsoid within the polytope that represents the operation constraints. Mathematically, finding the largest ellipsoid inside a polytope can be solved by the linear matrix inequality (LMI) method (Boyd et al., 1994). Thus, we can use Lyapunov theory and the LMI method to solve our recovery region problem.

For example, consider a dynamic system given by $\dot{X} = A^*X + BKX$, where X is the system state, A^* and B represent the system dynamics, and K represents a controller. We can first choose K by using well-understood controller designs with a robust stability, i.e., the system stability region is insensitive to model uncertainty. The system under the control of this reliable controller is given by $\dot{X} = AX$, where $A = A^* + BK$ is the stability condition satisfying $A^TQ + QA < 0$, with Q being the Lyapunov matrix. The operation constraints are represented by a normalized polytope, $C^T X \leq 1$. The largest ellipsoid inside the polytope can be found by minimizing $\log(\det(Q^{-1}))$ (Boyd et al., 1994), subject to stability conditions and operation constraints. The resulting Q defines the largest normalized ellipsoid $X^T Q X = 1$ inside the polytope, as shown in Figure 1. This is the recovery region associated with the high-assurance controller.

During normal operation, the plant is under the control of the high-performance control subsystem. At every sampling period, the plant state X is checked to see if it is within the N -dimensional ellipsoid, i.e., if $X^T Q X < k$, where k is a constant less than 1. Note that $(1 - k)$ is the margin against errors in system state measurement. If this check is not satisfied, the high-assurance control subsystem takes over. This ensures that the operation of the plant never violates the operation constraints. Once this is assured, statistical performance evaluations of the high-performance control subsystem can be conducted safely in the actual plant operations. The plant is also protected from latent faults that tests and evaluations fail to catch.

Due to the critical importance of the high-assurance controller, in the rest of this paper, we conservatively assume that the system would fail if the high-assurance controller failed.

2.2. Controller Performance Loss Index

The relationship between sampling frequency and control performance of a control system has been well-established. The dynamics we wish to control have a certain bandwidth. The recommendation of most control textbooks is that one should sample somewhere between 5 and 10 times faster than the highest frequency dynamics that one would like to control. Little additional control performance improvement is gained by sampling over 10 times the bandwidth of the system dynamics.

From a real-time computing perspective, the control textbooks' recommendation is less than precise, since the figure of 5 to 10 times represents a huge difference in system schedulability. Therefore, the choice of control frequencies should be carried out together with schedulability analysis. That is, we would like to choose the sampling frequencies such that all the tasks are schedulable and the control performance loss index is minimized. This problem was first solved in Seto et al. (1996), in which it was shown that for a digitized continuous controller, which is a commonly used controller design, the relationship between control performance loss and sampling frequencies can be modeled as an exponential function of sampling frequencies as shown below

$$\Delta J(f) = \alpha e^{-\beta f} \quad (1)$$

where $\Delta J(f)$ is the performance loss index (PLI) due to discrete sampling at frequency f , α is the magnitude coefficient, and β is the decay rate. Figure 2 is an example of the performance loss index of a diving control system.

Note that a higher value of $\Delta J(f)$ implies lower system performance. Also note that f_m is the lower bound on the sampling frequency, i.e., sampling below f_m either causes the system performance loss to become unacceptable, or even worse, causes the system to become unstable. It can be seen that as the sampling frequency becomes higher and higher, the digital controller's performance approaches that of the continuous time control. In the rest of this paper, we assume that each task has a given performance loss index function.

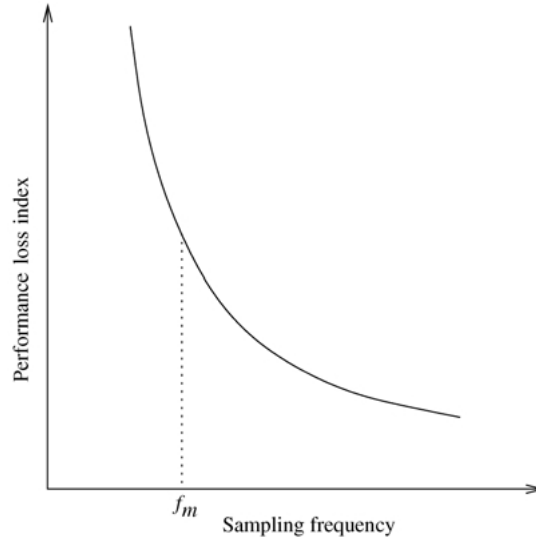


Figure 2. Control system performance loss index versus sampling frequency.

3. Task Scheduling

In this section, we study the problem of finding the controller task periods such that the expected system performance is optimized. For this purpose, we first introduce the notation used in this section and then formally define the problem. Then, we solve for the availabilities of the tasks, model the problem as an optimization problem, and propose a simple algorithm to solve it.

3.1. Notation

A set of n periodic real-time task pairs T_1, \dots, T_n is considered. Each task pair T_i has a high-performance specification and implementation T_{iP} , and a high-reliability (high-assurance) specification and implementation T_{iR} . C_{iP} and C_{iR} represent the task execution times of T_{iP} and T_{iR} , respectively. The performance loss indices of T_{iP} and T_{iR} are denoted by $\Delta J_{iP}(f)$ and $\Delta J_{iR}(f)$, respectively. f_{iP_m} and f_{iR_m} denote the minimum frequencies, while λ_{iP} and λ_{iR} denote the failure rates of T_{iP} and T_{iR} , respectively. μ_{iP} denotes the repair rate of T_{iP} . We assume that if T_{iR} fails, then the task T_i fails and that there is no repair possible. D denotes the mission duration for all the tasks. The performance loss on failure of task T_i is denoted by L_i . Also, the relative weight of task pair T_i in the system is represented by w_i .

3.2. Assumptions and Problem Statement

We assume that the high-assurance controllers are properly verified—that is, that their failure rate is negligible with respect to the application reliability requirement. We also assume that the controlled application would fail if the high-assurance controller failed. There will be no restart of the high-assurance controller, should it fail.

We assume that the high-performance controller has been reasonably tested. That is, although the high-performance controller may not meet the reliability requirement, it is not totally useless and will fail only occasionally. When the high-performance controller fails, the system switches to the control of the high-assurance controller, and the high-performance controller will be restarted. The system will pass the control back to the high-performance controller once it has been restarted. We assume that the high-performance controller’s restart time is much shorter than its mean time to failure.

Finally, for the calculation of the optimal task frequencies, we assume that the failure rates are known. However, it is often difficult to estimate the software failure rates accurately in practice. The impact of the uncertainty in failure rate will be examined in Section 5.

Problem Statement Given periodic task pairs $T_i = (T_{iP}, T_{iR})$, $i = 1, \dots, n$, scheduled using earliest deadline first algorithm and given C_{iP} , C_{iR} , $\Delta J_{iP}(f)$, $\Delta J_{iR}(f)$, f_{iP_m} , f_{iR_m} , λ_{iP} , λ_{iR} , μ_{iP} , D , L_i , and w_i for $i = 1, \dots, n$, find the frequencies f_{iP} and f_{iR} of T_{iP} and T_{iR} such that the expected system control performance loss is minimized.

3.3. Availability Analysis

To minimize the expected control performance loss index of the system, we first compute the availability of each of the control tasks for a mission duration D .

We assume that the failure rates and repair rates of tasks in the system are exponential. With this assumption, for each task T_i , we construct the Markov model representing the states of the task, as shown in Figure 3. State 2 of the Markov model represents the state

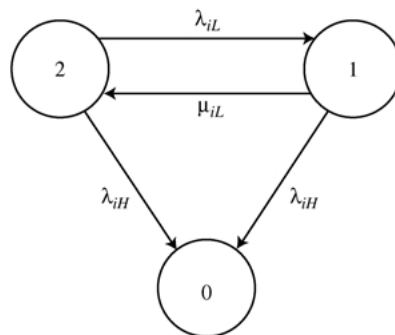


Figure 3. The Markov Model for Task T_i .

when both the implementations T_{iP} and T_{iR} of the task T_i are working. State 1 represents the state when the high-performance implementation T_{iP} has failed but the high-reliability implementation T_{iR} is still working. The system would still be working in this state, albeit with a loss in performance. Repair of T_{iP} in state 1 would move the state of the task back to state 2. State 0 represents the state when the high-reliability implementation has failed, in which case there is no provision for repair. The system initially starts in state 2.

The above Markov model is solved for the transition matrix $P(t) = [p_{ij}(t)]$ at time t . In particular, since state 2 is the initial state, we are interested in the probabilities $p_{22}(t)$, $p_{21}(t)$ and $p_{20}(t)$, which represent the probabilities that a system initially started in state 2 will, after time t , be in state 2, state 1, and state 0, respectively. Since the mission duration of the system, D , is known, the fraction of the mission duration spent in each of the states, represented by A_{i2} , A_{i1} , and A_{i0} , can be found using the following equations.

$$A_{i2} = \frac{\int_0^D p_{22}(t) dt}{D}, \quad A_{i1} = \frac{\int_0^D p_{21}(t) dt}{D}, \quad A_{i0} = \frac{\int_0^D p_{20}(t) dt}{D},$$

$$i = 1, \dots, n \quad (2)$$

Remark $P(t) = [p_{ij}(t)]$ can be solved using the matrix exponential function, `MatrixExp`, in *Mathematica*.

3.4. Optimization Problem

After the fraction of mission time spent in the different states is found for each of the tasks as explained above, the problem can be represented mathematically as the following optimization problem,

$$\text{Minimize } \Delta J(f_{1R}, f_{1P}, \dots, f_{nR}, f_{nP}) \quad (3)$$

where,

$$\begin{aligned} \Delta J(f_{1R}, f_{1P}, \dots, f_{nR}, f_{nP}) &= \sum_{i=1}^n w_i [A_{i1} \Delta J_{iR}(f_{iR}) + A_{i2} \Delta J_{iP}(f_{iP}) + A_{i0} L_i] \\ &= \sum_{i=1}^n w_i [A_{i1} \alpha_{iR} e^{-\beta_{iR} f_{iR}} + A_{i2} \alpha_{iP} e^{-\beta_{iP} f_{iP}} + A_{i0} L_i] \end{aligned}$$

$$\text{subject to } \sum_{i=1}^n (C_{iR} f_{iR} + C_{iP} f_{iP}) \leq U, \quad f_{iR} \geq f_{iR_m}, \quad f_{iP} \geq f_{iP_m},$$

$$i = 1, \dots, n \quad (4)$$

where $U \leq 1$ is the percentage of the CPU cycles available to run these tasks. Note that the term $\Delta J(f_{1R}, f_{1P}, \dots, f_{nR}, f_{nP})$ is the expected control performance of the system. The above problem can be transformed to an equivalent problem that contains $2n$ tasks,

T'_1, \dots, T'_{2n} , where $T'_1 = T_{1R}, T'_2 = T_{1P}, \dots, T'_{2n-1} = T_{nR}, T'_{2n} = T_{nP}$. The parameters are also correspondingly transformed, that is,

$$\alpha'_1 = \alpha_{1R}, \alpha'_2 = \alpha_{1P}, \dots, \alpha'_{2n-1} = \alpha_{nR}, \alpha'_{2n} = \alpha_{nP}$$

$$\beta'_1 = \beta_{1R}, \beta'_2 = \beta_{1P}, \dots, \beta'_{2n-1} = \beta_{nR}, \beta'_{2n} = \beta_{nP}$$

$$f'_1 = f_{1R}, f'_2 = f_{1P}, \dots, f'_{2n-1} = f_{nR}, f'_{2n} = f_{nP}$$

$$f'_{m1} = f_{1R_m}, f'_{m2} = f_{1P_m}, \dots, f'_{m2n-1} = f_{nR_m}, f'_{m2n} = f_{nP_m}$$

and

$$C'_1 = C_{1R}, C'_2 = C_{1P}, \dots, C'_{2n-1} = C_{nR}, C'_{2n} = C_{nP}$$

Also, the weights are transformed as

$$w'_1 = A_{i1}w_1, w'_2 = A_{i2}w_1, \dots, w'_{2n-1} = A_{n1}w_n, w'_{2n} = A_{n2}w_n$$

Note that the transformed problem is equivalent to the one solved in Proposition 3.1 of Seto et al. (1996). Hence, the solution to this transformed problem can be found using the following proposition, which is obtained from Proposition 3.1 of Seto et al. (1996) as shown in the proof below.

Proposition 3.4.1 Given a transformed set of $2n$ tasks, with the original tasks obeying the constraints given in Equation (4), there exists a unique optimal solution to the objective function given in Equation (3). This optimal solution is obtained at the sampling frequencies given by

$$f'_i = f'_{mi} \quad i = 1, \dots, p \quad (5)$$

$$f'_j = \frac{1}{\beta'_j} (\ln \Gamma_j - Q), \quad j = p+1, \dots, 2n \quad (6)$$

where,

$$\Gamma_j = \frac{w'_j \alpha'_j \beta'_j}{C'_j}, \quad Q = \frac{1}{\sum_{i=p+1}^{2n} C'_i / \beta'_i} \left(\sum_{i=1}^p C'_i f'_{mi} + \sum_{i=p+1}^{2n} \frac{C'_i}{\beta'_i} \ln \Gamma_i - U \right) \quad (7)$$

and f'_1, \dots, f'_{2n} are ordered according to f'_{m1}, \dots, f'_{m2n} , which are arranged as

$$\Gamma_1 e^{-\beta'_1 f'_{m1}} \leq \Gamma_2 e^{-\beta'_2 f'_{m2}} \leq \dots \leq \Gamma_{2n} e^{-\beta'_{2n} f'_{m2n}} \quad (8)$$

with $p \in \{1, \dots, 2n\}$ being the largest integer, such that

$$\sum_{i=1}^p C'_i f'_{mi} + \sum_{i=p+1}^{2n} \frac{C'_i}{\beta'_i} \left(\beta'_p f'_{mp} + \ln \frac{\Gamma_i}{\Gamma_p} \right) \geq U \quad (9)$$

Proof: Let $N=2n$. Now, the above transformed problem can be restated as the following optimization problem: Given a set of tasks T'_1, \dots, T'_N to be scheduled on a single CPU,

$$\begin{aligned} & \text{Minimize } \Delta J(f'_1, \dots, f'_N) \\ & \text{where } \Delta J(f'_1, \dots, f'_N) = \sum_{i=1}^N w'_i \Delta J'_i = \sum_{i=1}^N w'_i \alpha'_i e^{-\beta'_i f'_i} \\ & \text{subject to } \sum_{i=1}^N C'_i f'_i \leq U, \quad f'_i \geq f'_{mi}, \quad i = 1, \dots, N \end{aligned}$$

This is exactly the problem solved in Proposition 3.1 of Seto et al. (1996); hence the solution presented in Seto et al. (1996) can be used here. It can be easily seen that replacing N with $2n$ in that solution would yield the optimal solution given above.

Thus, Proposition 3.4.1 provides an algorithm for finding the optimal task frequencies in a reliable real-time control system such that the expected control performance given by $\Delta J(f_{1R}, f_{1P}, \dots, f_{nR}, f_{nP})$ is minimized, i.e., it provides a solution to the original optimization problem posed in Equations (3) and (4). ■

4. Applications

In this section, we apply our optimization algorithm to some sample applications. As this paper focuses on software fault tolerance, we assume that the software is running on some fault-tolerant hardware with CPU capacity normalized to 1. Also, in the following discussion, we denote the approach given in Seto et al. (1996) (which does not take the availability of the tasks into consideration), as Method 1, and the optimization approach proposed in this paper as Method 2.

We begin with the following simple example:

Example 4.1 An inverted pendulum (Sha, 1998) is governed by two controllers, a high-performance controller and a high-assurance controller. Suppose the inverted pendulum is running for $D=1$ day, and the control tasks' other parameters are as shown below:

Task	α	β	C (ms)	f_m (Hz)	Weight	$1/\lambda$ (days)	$1/\mu$ (min)
HR (T_{1R})	100	0.1	8	50	10	∞	—
HP (T_{1P})	200	0.2	15	20	10	0.1	10

Following the algorithm in Proposition 3.4.1, we determine that $p=1$ and that the optimal frequencies of the two tasks are $f_{1R} = 50$ Hz and $f_{1P} = 40$ Hz.

The fractions of time the high-assurance controller and the high-performance controller of task T_{1R} execute are calculated to be $A_{11} = 0.0645134$ and $A_{12} = 0.935487$; this yields an optimal performance loss index of $\Delta J = 1.0623$.

If we use the approach from (Seto et al., 1996), which deals with no reliability reasoning, the PLI is $\Delta J = 3.4702$. The results of the two methods are compared below:

Algorithm	p	(f_{IR}) (Hz)	f_{IP} (Hz)	PLI
Method 2	1	50	40	1.0623
Method 1	0	60.85	34.21	3.4702

Example 4.2 This example is a temperature control application similar to the one discussed in Seto et al. (1996). Suppose there are 3 units whose temperatures need to be automatically controlled by one processor. For each unit i , there are two analytically redundant controllers running at the same time; one is a high-performance controller, and the other is a high-assurance controller. The corresponding task pair is (T_{iP}, T_{iR}) .

The temperature for each of the units is governed by the dynamic equation:

$$\dot{\gamma}_i(t) = a_i\gamma_i(t) + b_iu_i(t) \tag{10}$$

where $\gamma_i(t)$ is the temperature difference between the i -th unit and the ambient temperature at time t , with $\gamma_i(0)$ being 0; a_i and b_i are constants depending on the insulation of the unit; and $u_i(t)$ is the rate of heat (cold air) supplied to the unit. For the high-performance controller design, we use the optimal control approach. Suppose we need to change the temperature in the i -th unit, and we require that the change be completed in no more than t_f time units and that it consumes a minimum amount of fuel. Let γ_{di} be the difference between the desired temperature and the ambient temperature. We also require that $|\gamma_i(t_f) - \gamma_{di}| \leq \delta_i$. Then, the optimization problem can be formulated as:

$$\min_{u_i} J_i = \frac{1}{2}p_i(\gamma_i(t_f) - \gamma_{di})^2 + \frac{1}{2} \int_0^{t_f} u_i^2(t)dt \tag{11}$$

where p_i is a weight coefficient. Then, the continuous-time optimal control and the final state are determined by

$$u_i^*(t) = \frac{\gamma_{di}p_i a_i b_i e^{a_i t}}{a_i e^{a_i t_f} + p_i b_i^2 \sinh(a_i t_f)}, \quad \gamma_i^*(t_f) = \frac{\gamma_{di} p_i b_i^2 \sinh(a_i t_f)}{a_i e^{a_i t_f} + p_i b_i^2 \sinh(a_i t_f)} \tag{12}$$

Using the digitizing method stated in Seto et al. (1996), we can obtain for each high-performance controller:

$$\Delta J_{iP} = \alpha_{iP} e^{-\beta_{iP} f_i} \tag{13}$$

which is exactly the exponential form we have assumed. The high-assurance controller may be either a common PID controller or a feedback controller using LMI (linear matrix inequality), as long as it has high-reliability. After digitizing the corresponding

continuous-time controller and using techniques such as least-square fitting approach, we get

$$\Delta J_{iR} = \alpha_{iR} e^{-\beta_{iR} f_i} \quad (14)$$

Suppose mission time is $D = 365$ (days), with α_{iR} , β_{iR} , α_{iP} , β_{iP} , the execution times C_{iR} , C_{iP} , and the minimum frequencies f_{iR_m} , f_{iP_m} of each task being as shown in the tables below:

Task	α	β	C (ms)	f_m (Hz)	Weight	$1/\lambda$ (days)	$1/\mu$ (min)
HR (T_{1R})	5	0.01	3	10	10	∞	—
HP (T_{1P})	6.67	0.03	2	20	10	7	10

Task	α	β	C (ms)	f_m (Hz)	Weight	$1/\lambda$ (days)	$1/\mu$ (min)
HR (T_{2R})	5	0.08	1	15	20	∞	—
HP (T_{2P})	6.67	0.15	2.4	12.5	20	4	8

Task	α	β	C (ms)	f_m (Hz)	Weight	$1/\lambda$ (days)	$1/\mu$ (min)
HR (T_{3R})	5	0.02	3	15	30	∞	—
HP (T_{3P})	6.67	0.05	2	10	30	20	15

Following the algorithm in Proposition 3.4.1, we determine that $p=2$, and that the optimal frequency for each task is $f_{1R} = 10$ Hz, $f_{1P} = 217.77$ Hz, $f_{2R} = 25.39$ Hz, $f_{2P} = 57.68$ Hz, $f_{3R} = 15$ Hz, and $f_{3P} = 162.84$ Hz.

The fractions of time the high-assurance controller and high-performance controller of task 1 execute are calculated to be $A_{11} = 0.000991061$ and $A_{12} = 0.999009$; the fractions of time the high-assurance controller and high-performance controller of task 2 execute are calculated to be $A_{21} = 0.00138694$ and $A_{22} = 0.998613$; the fractions of time the high-assurance controller and high-performance controller of task 3 execute are calculated to be $A_{31} = 0.000520547$ and $A_{32} = 0.999479$.

This set of frequencies yields an optimal PLI of $\Delta J = 0.299$. If we ignore the availability implications and apply the approach in Seto et al. (1996) to all the above tasks directly, we have $\Delta J = 10.987$. The detailed comparison of the methods is shown below:

Algorithm	p	f_{1R} (Hz)	f_{1P} (Hz)	f_{2R} (Hz)	f_{2P} (Hz)	f_{3R} (Hz)	f_{3P} (Hz)	PLI
Method 2	2	10	217.77	25.39	57.68	15	162.84	0.299
Method 1	0	62.33	80.50	56.18	30.23	120.75	80.49	10.987

Example 4.3 This example discusses a bubble control system similar to the one discussed in Seto et al. (1996). Suppose two such systems with different physical dimensions are installed on an underwater vehicle to control the depth and orientation of the vehicle, and they are controlled by one on-board processor. For each bubble control system i , there are two controllers running at the same time, of which one is a high-performance controller and the other is a high-assurance controller. The corresponding tasks are T_{iP}, T_{iR} . Suppose the mission time is $D = 365$ (days), and $U = 1$, with $\alpha_{iR}, \beta_{iR}, \alpha_{iP}, \beta_{iP}$, the execution times C_{iR}, C_{iP} , and the minimum frequencies f_{iR_m}, f_{iP_m} of each task being as shown in the tables below:

Task	α	β	C (ms)	f_m (Hz)	Weight	$1/\lambda$ (days)	$1/\mu$ (min)
HR (T_{1R})	10	0.1	10	15	5	100	20
HP (T_{1P})	10	0.12	15	10	5	5	10

Task	α	β	C (ms)	f_m (Hz)	Weight	$1/\lambda$ (days)	$1/\mu$ (min)
HR (T_{2R})	10	0.14	10	18	3	100	20
HP (T_{2P})	10	0.18	15	10	3	3	10

Following the algorithm in Proposition 3.4.1, we determine that $p = 2$, and that the optimal frequency for each task is $f_{1R} = 15$ Hz, $f_{1P} = 27.15$ Hz, $f_{2R} = 18$ Hz, and $f_{2P} = 17.51$ Hz.

The fractions of time the high-assurance controller and high-performance controller of task 1 execute are calculated to be $A_{11} = 0.000370$ and $A_{12} = 0.266482$; the fractions of time the high-assurance controller and high-performance controller of task 2 execute are calculated to be $A_{21} = 0.000616$ and $A_{22} = 0.26623$.

This yields an optimal PLI of $\Delta J = 0.8593$. If we ignore the availability implications and apply the approach in Seto et al. (1996) to all the above tasks directly, we have $\Delta J = 1.6704$. The detailed comparison is shown below:

Algorithm	p	f_{1R} (Hz)	f_{1P} (Hz)	f_{2R} (Hz)	f_{2P} (Hz)	PLI
Method 2	2	15	27.15	18	17.51	0.8593
Method 1	0	28.11	21.57	18.84	13.79	1.6704

The above examples demonstrate the superior performance of the task pairs with frequencies calculated using Method 2, as compared to their performance with frequencies calculated using Method 1. This is because in reliable real-time control systems, the availability of the different task pairs and the fact that only one task in each task pair would be controlling the system at any particular time are important factors to be considered during optimal task frequency selection. Method 1 does not consider these factors, leading to sub-optimal solutions.

5. Sensitivity Analysis

When a task's correctness cannot be verified, its reliability is questionable. In addition, it is difficult to obtain a very accurate failure rate by testing. This raises the question of how useful the optimization we have proposed is, given that the failure rate is a factor in the computation. Fortunately, the uncertainty in the estimated failure rate has relatively little effect as long as the mean time to failure is much longer than the mean time to repair¹. This is illustrated in the following example.

Suppose that we have two pairs of analytically redundant controllers for a total of four tasks on a processor, as described by the following tables.

Task Pair 1

Task	α	β	C (ms)	f_m (Hz)	Weight	$1/\lambda$ (days)	$1/\mu$ (min)
HR (T_{1R})	250	0.1	10	10	30	∞	—
HP (T_{1P})	250	1	12.5	8	30	7	10

Task Pair 2

Task	α	β	C (ms)	f_m (Hz)	Weight	$1/\lambda$ (days)	$1/\mu$ (min)
HR (T_{2R})	250	0.3	10	12	10	∞	—
HP (T_{2P})	250	1.5	12.5	5	10	4	8

In the above tables, T_{1R} and T_{2R} are the high-assurance controllers of tasks 1 and 2, and T_{1P} and T_{2P} are the high-performance controllers.

Suppose the nominal (or estimated) mean time to failure of task T_{1P} is 7 days, as given in the table above. Then its availability is 0.99901. However, if the actual mean time to failure is 3.5 days, the availability goes down to 0.9982. If the actual mean time to failure is 14 days, then the availability is 0.99950. Hence, it can be seen that the change in availability is quite small for even moderately large errors in estimation of mean time to failure.

Now, recall that when computing the system's performance loss index, the availability of each control task is a multiplier of the weight of the task's performance loss index. Since even a moderately large error in the estimation of failure rate results in very little change in availability, it follows that a large error in the estimation of failure rate has little effect on the optimal performance loss index computed using the approach proposed in this paper. Figure 4 shows the sensitivity of the proposed approach to errors in estimating failure rate. The estimated failure rate is $1/7$ times per day. The solid line is obtained by computing the optimal system frequencies using the proposed approach at the estimated failure rate of $1/7$ per day, and plotting the system performance at these frequencies, when the actual failure rate varies from $1/3.5$ per day to $1/14$ per day. The dotted line shows the optimal system performance for failure rates between $1/3.5$ per day and $1/14$

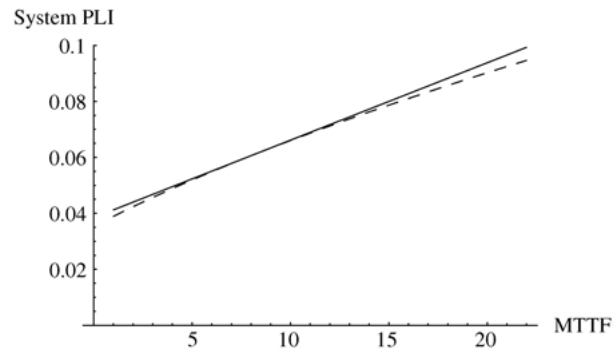


Figure 4. Sensitivity analysis.

per day. It can be seen from the figure that the loss in system performance is very small even if there is a considerable error in estimating failure rate.

In summary, the optimization technique we have presented is robust against uncertainty in the estimation of the failure rates, provided that the $MTTF \gg MTTR$.

6. Conclusions

The long life cycle of most real-time systems makes frequent upgrades necessary. This in turn necessitates the development of flexible and reliable software architectures aimed at reducing the costs and risks of software upgrades. Analytical redundancy is one such architecture and is a powerful tool to achieve software fault tolerance. The development of control systems using analytical redundancy has been well-studied. However, the problem of optimization of the performance of such systems has remained open.

In this paper, we proposed a unified approach to reliable control system design. More specifically, given a control system using analytical redundancy, we developed an algorithm to determine the control task frequencies that optimize the expected performance of the system such that all tasks are schedulable. We then presented a few examples to illustrate the proposed algorithm. Furthermore, we show that this algorithm is robust against inaccuracies in the estimation of failure rate.

Acknowledgments

The authors would like to thank the sponsorship from the Office of Naval Research and the discussions with Danbing Seto.

Notes

¹ Recall that in steady state, the availability is equal to $MTTF/(MTTF + MTTR)$. This number is close to 1 as long as $MTTF \gg MTTR$.

References

- Bertossi, A., Fusiello, A., and Mancini, L. 1997. Fault-tolerant deadline-monotonic algorithm for scheduling hard real-time tasks. In *Proceedings of the International Parallel Processing Symposium*, 133–138.
- Boyd, S., Ghaoul, L. E., Feron, E., and Balakrishnan, V. 1994. Linear matrix inequality in systems and control theory. *SIAM Studies in Applied Mathematics*.
- Burns, A., Davis, R., and Punnekkat, S. 1996. Feasibility analysis of fault-tolerant real-time task sets. In *Proceedings of the Euromicro Workshop on Real-Time Systems*, 29–33.
- Gerber, R., Hong, S., and Saksena, M. 1994. Guaranteeing end-to-end timing constraints by calibrating intermediate processes. In *Proceedings of the 15th IEEE Real-Time Systems Symposium*, December.
- Gray, J. 1990. A census of Tandem system availability between 1985 and 1990. *IEEE Transactions on Reliability*, October.
- Liberato, F., Lauzac, S., Melham, R., and Mossé, D. 1999. Fault-tolerant real-time global scheduling on multiprocessors. In *Proceedings of the Euromicro Workshop on Real-Time Systems*.
- Liu, C. L., and Layland, J. W. 1973. Scheduling algorithms for multiprogramming in a hard real-time environment. *Journal of the Association for Computing Machinery* 20(1): 46–61, January.
- Locke, C. D. 1986. Best-effort decision making for real-time scheduling. PhD thesis, Computer Science Department, Carnegie Mellon University.
- Seto, D., and Sha, L. 1999. Engineering method for safety region development. Technical Report CMU/SEI-99-TR-018, Software Engineering Institute, Carnegie Mellon University.
- Seto, D., Krogh, B., Sha, L., and Chutinan, A. 1998. Dynamic control systems upgrade using Simplex architecture. *IEEE Control*, August.
- Seto, D., Lehoczky, J. P., Sha, L., and Shin, K. G. 1996. On task schedulability in real-time control systems. In *Proceedings of the 17th Real-Time Systems Symposium*, December, 13–21.
- Sha, L. 1998. Dependable system upgrade. In *Proceedings of the 19th Real-Time Systems Symposium*, December, 440–448.
- Siewiorek, D. P., and Swarz, R. S. 1992. *Reliable Computer Systems—Design and Evaluation*. Digital Press (distributed by Butterworth), 2nd edn.
- Stankovic, J., Lu, C., and Son, S. 1992. The case for feedback control real-time scheduling. In *Proceedings of the 11th Euromicro Conference on Real-Time Systems*, 11–20.
- Yeh, Y. C. 1995. Dependability of the 777 primary flight control system. In *Proceedings of the DCCA Conference*.

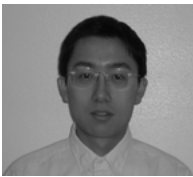


Dr. Lui Sha obtained his Ph.D. from Carnegie Mellon University in 1985. He was elected to be an IEEE Fellow in 1998 “for technical leadership and research contributions, which enabled the transformation of real-time computing practice from an ad hoc process to an engineering process based on analytic methods.” He is currently a professor of the Computer Science Department of University of Illinois at Urbana Champaign, the Chair of IEEE Real Time Systems Technical Committee from 1999–

2000, and an associated editor of the Journal of IEEE Transaction on Parallel and Distributed Systems and of the Journal of Real-Time Systems. His research interest includes flexible and reliable real time computing systems and the integration between control and real-time computing.



Ramesh Chandra is a doctoral student in the department of Computer Science at Stanford University, USA. He obtained his Bachelor in Technology from the Indian Institute of Technology, Madras, India, and his Master of Science from the University of Illinois at Urbana-Champaign, USA, both in Computer Science. His research interests are in the areas of distributed systems, communication networks, operating systems, and reliable systems.



Xue Liu is a MS/Ph.D. student in Computer Science Department of University of Illinois at Urbana-Champaign. He got his BS in Applied Mathematics in 1996 and Master of Engineering in Automation in 1999 both from Tsinghua University, P. R. China. He is a student member of IEEE. His main research topic is the integration between control and real-time computing and reliable real time computing systems.