Sustainable Computing: Informatics and Systems 28 (2020) 100411

Contents lists available at ScienceDirect

Sustainable Computing: Informatics and Systems

journal homepage: www.elsevier.com/locate/suscom

Enhancing dependability and energy efficiency of cyber-physical systems by dynamic actuator derating☆

Shikang Xu*, Israel Koren, C. Mani Krishna

Department of Electrical and Computer Engineering University of Massachusetts at Amherst, Amherst, USA

ARTICLE INFO

Article history:
Received 15 March 2019
Received in revised form 18 February 2020
Accepted 16 March 2020
Available online 27 June 2020

Keywords:
Cyber-physical systems
Energy consumption
Adaptive fault-tolerance
Lifetime and dependability
Sustainable computing

ABSTRACT

Thermal issues in the cyber part of cyber-physical systems (CPSs) has attracted considerable attention in recent years. Heat generation due to energy consumption results in accelerated thermal damage to the processing units, reducing their lifetime. Fault tolerance contributes to a large fraction of this thermal behavior in CPS since it is implemented using redundant computations. This paper studies the use of dynamic actuator derating (i.e., artificially limiting the maximum actuator output) for reducing the need to apply maximum redundancy. By targeting the use of fault-tolerance, we are able to obtain significant reductions in computer energy expenditure and thermal stress without lowering the reliability. This has beneficial effects on processor lifetime as well as the required energy storage.

1. Introduction

Thermal management of the processors in a cyber-physical system (CPS) has emerged as an important problem in recent years. As control computers get tasked with controlling ever more complex plants under constrained circumstances, their energy consumption has increased and is dissipated as heat. Increased energy demands require larger battery or supercapacitor arrays and lead to higher operating temperatures. Such thermal stress results in increased hardware failure rates and therefore an increased required hardware replacement rate. When these systems are used in applications where heating can damage the operating environment there is an additional motivation to keep energy consumption and heat dissipation low.

The critical nature of many applications contributes greatly to the worsen thermal issue. Increasingly, CPSs are used in life-critical applications requiring very high levels of reliability. For example, the traditional reliability requirement for an aircraft control computer is $1{-}10^{-9}$ for a 10-hour flight [19]. Since systems used in such applications must be far more reliable than their individual components, fault-tolerance using massive redundancy is used [15] which consumes large amounts of energy and thus generates large amount of heat.

This paper considers the energy and thermal dependability impact of applying redundancy adaptively and dynamically. The insight that drives this work is the observation that incorrect computer/control output to the actuators of the controlled plant does not necessarily lead to plant failure all the time. In other words, the controlled plant operates in a state-space, only a part (often a very small part) of which is critically vulnerable to incorrect computer output. Furthermore, this vulnerable fraction of the state-space is linked to the actuator limits, e.g., the maximum output of a motor or the maximum deflection of an aircraft elevator. In many instances, dynamically derating an actuator (i.e., limiting it to a lower level of output) helps expanding the state-space region of plant invulnerability. In such subspaces, no (or greatly reduced) fault-tolerance is required and energy demands and heat generation can be greatly reduced. In this paper, we explore the advantages of adaptive actuator derating based on current state of the controlled plant and its cost in terms of the quality of control provided.

Derating consists of limiting the output of an actuator to below its physical limit. No physical changes to the actuator are required of the actuator: derating can be done by simply replacing an actuator command of magnitude M with one of magnitude $M_{derated} < M$, where $M_{derated}$ is the derated limit. The concept of derating is not new; however, the purpose has always hitherto been to reduce the stress on the plant actuators. For example, in aviation, it is used to reduce engine wear: derating engine output is common practice during takeoff under light loading and favorable weather and

☆ This work has been partially supported by the National Science Foundation under grants CNS-1329831 and CNS-1717262.

* Corresponding author.
E-mail address: shikang@umass.edu (S. Xu).

https://doi.org/10.1016/j.suscom.2020.100411
2210-5379/© 2020 Elsevier Inc. All rights reserved.

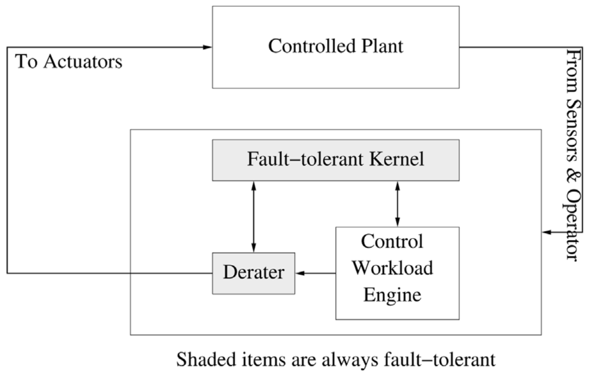


Fig. 1. High-level system block diagram.

runway conditions [14]. By contrast, our purpose in using dynamic actuator derating is to aid the cyber side of a CPS.

This paper is organized as follows. Section 2 presents our system model. More details concerning adaptive actuator derating and its context are provided in Sections 3 and 4. Models are then provided in Sections 5, 6 and 7 for evaluating thermally accelerated aging and energy consumption under our approach. We then present two case studies to illustrate our approach. These have been selected from areas of practical importance: robotics and unmanned aerial vehicles (UAVs). Robots and UAVs can be considered either individually or as members of a networked cluster working cooperatively.

2. System model

A typical CPS consists of a controlled plant (the physical system) with a computer (the cyber part) in its feedback loop [19,21]. The computer is provided with sensors and other inputs which allow it to understand the current state of the plant. It executes a real-time workload, consisting of tasks which provide commands to the actuators. These tasks can be triggered either periodically or sporadically; routine control tasks are usually triggered periodically, while emergency tasks are triggered sporadically.

The computer typically consists of multiple processors. When no fault-tolerance is needed with respect to a task output, a single copy is all that needs to execute. When full fault-tolerance is needed, fault-masking is required. In such cases, a common approach is to use a triplex of processors, with three copies being executed and voted on to mask up to one failure [15]. (For greater fault-tolerance the triplex can be expanded to encompass even more processors, but we will focus on triplexes for fault-tolerance in this paper.) Note that the kernel of the cyber side must always be fault-tolerant, i.e., the portion of the processing that determines the appropriate level of fault-tolerance to be applied. However, this portion is usually but a small part of the whole cyber workload, which mostly consists of running control algorithms. Fig. 1 illustrates this; the role of the actuator derater is explained in detail below.

The performance of the computer in a CPS can only truly be judged within the context of the controlled plant [17]. The computer calculates actuator inputs designed to optimize a given performance functional; a typical quadratic performance functional for a plant over a given interval, $[0, T]$, of operation, is given by

$$\mathcal{F}(T) = \int_0^T \left(x^T(t)Qx(t) + u^T(t)Ru(t) \right) dt \tag{1}$$

[2,6]. Here, $x(t)$ is a vector depicting plant state (or in a tracking problem, the deviation of plant state from the desired state-space trajectory) and $u(t)$ a vector of actuator inputs, both at time t, and Q, R are appropriate matrices, selected by the user to weigh the various parts of the functional in line with the priorities of the application.

CPSs are commonly controlled using zero-order control. That is, control inputs are updated at specific epochs and held constant between update epochs [4,23]. The performance functional is then usually calculated by replacing the integral in the above expression by a sum, with state and input vectors being represented at specific multiples of some constant time interval, i.e., $x(i)$, $u(i)$.

To avoid failure, the job of the computer is to keep the controlled plant within a given portion of its state-space. In discrete-time terms, we can define a *Safe State Space, SSS*, which must be maintained, i.e., we define failure as occurring if there exists any i for which $x(i) \notin SSS$; for further details, see [28].

We further divide SSS into two portions. One portion, called S_1, is a subset of *SSS* in which the controlled plant will stay in SSS prior to the next update of the actuator settings no matter what the actuator settings (i.e., no matter how incorrect the settings are) [18]. This volume subspace is a function of: (a) the dynamics of the controlled plant, (b) the impact of the operating environment on the plant, (c) the rate at which the actuator settings are updated, and (d) the range of actuator settings that are possible (this is where derating comes in). When the state of the plant under control in is S_1, no fault tolerance is needed. The rest of the SSS is denoted by S_3 in which a triplex is needed to mask up to one faulty output [15] as the subscript signifies. (Note that S_i is only defined for $i = 1, 3$.) Our work proceeds from the observation that by using no fault-tolerance when the plant is in S_1, significant improvements in energy consumption and failure rate can be achieved.

3. Adaptive actuator derating: motivation and approach

Subspace S_1 requires no fault-tolerance on the part of the computer. That is, even if a faulty processor commands the worst-case (wrong) control output, the plant will remain in its safe state space. Of course, if there exists a actuator with infinite capability, the above statement will not hold, i.e., there is no S_1. Thus, a trade-off exists: as the maximum actuator capability increases, the quality of control tends to increase upon delivery of a correct actuator output; however, the potential cost of a faulty output also increases. In other words, for actuators with higher capability, the volume of *SSS* will increase. But the actuator will also have larger capability to move the state out of *SSS* on a faulty output. As a result, beyond a point, S_1 tends to shrink with increased actuator capability and thus, redundancy will be needed more often. Therefore, derating the actuator (i.e., artificially reducing its capability) is worth considering in certain plant states. Actuator derating is very simple to implement: the commanded output to the actuator passes through a fault-tolerant limiter which saturates at the specified derated value. This limiter has to be fault-tolerant; however, its function is so simple that it imposes no meaningful cost on the cyber subsystem. In fact, we go further and adapt the derated value to the current plant state according to Algorithm 1.

Algorithm 1. Adaptive actuator derating

```
1:  Input: Actuator derated ranges, $A_1, A_2, \ldots, A_k$, assuming an order
    relation between them, i.e., $A_1 \subset A_2 \subset \cdots \subset A_k$. ($A_k$ is the
    maximum, physically limited, range of the actuator.)
2:  Obtain $S_1$ associated with the specified operating environment for
    each of the derated ranges: $S_1(A_1), S_1(A_2), \ldots, S_1(A_k)$.
3:  Define $S_3 = SSS - \cup_{i=1}^{k} S_1(A_i)$.
4:  If the controlled plant state is estimated as $\sigma$, and
    $j = \max\{i \mid \sigma \in S_1(A_i)\}$ exists, then select $A_j$ as the derated value to
    be used, and proceed with the control output calculation without
    fault-tolerance. If no such $j$ exists, then the associated control
    output calculation must be fully fault-tolerant.
```

If the real-time operating system allows, we can adjust the control task dispatch rate adaptively as well. The way to do this is to pick from the available menu of iterative task periods the biggest period for which the current plant state will be in S_1 for the selected actuator value.

There are three questions which we need to address to make the above approach practical:

(1) *Given the current state of the plant, how do we determine which subspace it is in?* Given a state, the subspace can be calculated in real time by using formulas, decision trees, or neural classifiers that have previously been derived offline. In other words, the state-space has to be analyzed *offline* and the subspaces demarcated by using control-theoretical reachability analysis [3,20]. (Alternatively, simple heuristic sampling techniques can be used to obtain the impact of maximal control input in a large number of randomly chosen directions.) Reachability analysis can be used to determine from which states the plant is guaranteed *not* to leave the SSS no matter what the actuator output may be (subject only to its output constraint, including derating if any).

Once the subspaces have been generated offline, lightweight (in time and storage) online mechanisms are used to determine, for any state vector, which subspace the plant is in. Classification (using neural nets or other machine intelligence approaches) mechanisms are used for this purpose. In the examples we have worked with, the online computation associated with this takes no more than a microsecond. Given that the minimum period in cyber-physical systems is typically of the order of milliseconds, the online overhead of subspace detection is very small. Note that this is part of the core fault-tolerant computation; it needs to be done once every control update period, not for every control task that is executed.

(2) *Does SSS itself depend on the derated range?* No. The outermost part of SSS is S_3, which requires full fault-tolerance. We do not derate actuators in S_3 since failure is masked when in S_3; their only limits are the original physical limits (e.g., maximum engine thrust, rudder deflection, braking intensity).

(3) *How do we select the actuator derated ranges?* This is a matter for engineering judgement. It requires modeling how S_1 varies with the actuator deratings. Whenever the actuators saturate due to derating, the control becomes suboptimal. We therefore would like to select the derated values so as to effectively trade off the loss in quality of control against the improvement in energy consumption and thermal stress.

In Algorithm 2, the minimum actuator derating range will shrink if there is an actuator saturation in the current control period. Likewise, the derating range will be widen when there is no saturation anytime in the past W control period(s). Here, W is an input parameter to the algorithm.

4. Dynamic derating constraint

Algorithm 2. Dynamic derating constraint update

```
1:  Input: Actuator derated output magnitude ranges, $A_1, A_2, \cdots, A_k$,
    $A_1 \subset A_2 \subset \cdots \subset A_k$. Window sizes $W_r$ and $W_l$.
2:  Current (derated) output range: $A_c$
3:  if actuator saturation occurred every time over the past $W_r$ periods
    then
4:    $\ell = \min(k, c+1)$
5:  else if no actuator saturation occurs anytime over the past $W_\ell$
    periods then
6:    $\ell = \max(1, c-1)$
7:  end if
8:  Change Algorithm 1 allowed derating ranges to $A_\ell, A_{\ell+1}, \cdots, A_k$
```

Control saturation of the actuator can result from one of two causes. First, and most likely, the optimal actuator setting as commanded by the control algorithm exceeds the artificial restrictions imposed by the derating process. Second, and far less likely, it could be the result of a computational error. In the former, the quality of control can potentially be improved by relaxing the derating. In the latter, being within S_1, this incorrect command does not harm safety in any way; if this causes the plant to move into S_3, then full fault-tolerance is switched on and the fault is detected and masked. (Note that simple acceptance testing by way of range checks can always detect faults which command truly excessive actuator values.)

We now introduce an approach to update the actuator derating ranges. Recall that Algorithm 1 takes as input a set of allowed actuator derating ranges and selects the appropriate one to be applied. Here, the focus is on reducing the computational burden, by trying to keep the controlled plant in S_1 to the extent possible. However, there is a price to be paid for this: by restricting the range of actuator output, the quality of control can degrade. In particular, if the derating makes it impossible to deliver the actuator output commanded by the optimal control algorithm, control quality degrades.

Algorithm 2, which is a supplement to Algorithm 1, is meant to address this potential control quality degradation. It works by restricting the amount of derating that Algorithm 1 is allowed. In other words, it dynamically adjusts the set of derating ranges that Algorithm 1 is allowed to work with.

Algorithm 2 is lightweight and is an adaptive heuristic. If the output commanded by the optimal control algorithm consistently causes saturation of the derated actuator over a specified window (W_r control periods), we relax the derating (if that is possible). We do so even if this pushes the current plant state into S_3, thereby (only temporarily, we hope) increasing the computational workload; this is considered an acceptable price to pay for not degrading control quality. Equally, if the currently applied derating range is such that actuator saturation does not happen anytime over a window (W_l control periods), we allow Algorithm 1 to select a more restrictive actuator derating.

5. Modeling accelerated processor aging

The system cycles between having to use full fault-tolerance and none. The latter happens whenever the plant is in S_1 and the former otherwise. Denote the power consumption per processor when the plant is in subspace S_i ($i = 1, 3$) by ω_i. Clearly, $\omega_3 > \omega_1$ (otherwise, a non-fault-tolerant scheme would be more power-hungry than a fault-tolerant scheme). We aim to calculate the average thermally-accelerated aging suffered per unit time by a processor under these conditions.

To do this, we embed a Markov chain at the epochs at which the plant enters subspaces S_1. Denote by $\pi_{\theta_i}(\theta_{i+1})$ the density function of a processor's temperature at the $i+1$st entry into S_1 given that its temperature at the ith entry was θ_i. Denote by Θ_ω^{ss} the steady-state temperature of a processor when run at a power of ω; let its thermal time constant (see [7]) be τ. Denote by $\theta_\omega(\Delta t, \theta_0)$ the temperature at time $t_0 + \Delta t$ with initial temperature θ_0 (at time t_0) under a constant power consumption ω. Then, the heat-flow differential equation (see [11]) can be solved to yield $\theta_\omega(\Delta t, \theta_0)$:

$$\theta_\omega(\Delta t, \theta_0) = \theta_0 e^{-\Delta t/\tau} + \Theta_\omega^{ss}(1 - e^{-\Delta t/\tau}) \tag{2}$$

Consider the following sequence of events: start with the plant entering S_1, staying there for some time Δt_1, transiting at that time to S_3, and staying there for time Δt_3 before transiting back to S_1. Denote the temperatures of the processor at these instants by $\theta_a, \theta_b, \theta_c$, respectively. Then, the processor temperature when the plant enters S_3 is given by $\theta_b = \theta_{\omega_1}(\Delta t_1, \theta_a)$; when it returns to S_1, it is given by $\theta_c = \theta_{\omega_3}(\Delta t_3, \theta_b)$. Based on these equations, we can solve for Δt_3 as a function $f(\Delta t_1, \theta_b, \theta_c)$ as follows:

$$\Delta t_3 = f(\Delta t_1, \theta_b, \theta_c) = -\tau \ln \left[\frac{\Theta^{ss}_{\omega_3} - \theta_c}{\Theta^{ss}_{\omega_3} - \theta_b} \right] \tag{3}$$

when $\theta_c > \theta_b$. If $\theta_c = \theta_b$, then $\Delta t_3 = 0$. Note that since $\omega_3 > \omega_1$, we will always have a temperature upper-bounded by $\Theta^{ss}_{\omega_3}$; furthermore, $\theta_b > \theta_c$ is not possible under the assumptions made.

Now, denote the probability density function (pdf) of Δt_1, Δt_3 by $\psi_1(\cdot)$, $\psi_3(\cdot)$, respectively. Based on the above derivation, we can now write the following expression:

$$\pi_{\theta_i}(\theta_{i+1}) = \int_0^\infty \psi_1(t)\psi_3(f(t, \theta_i, \theta_{i+1}))dt \tag{4}$$

(When the function $f(\cdot, \cdot, \cdot)$ does not exist, the density function is 0).

It is easy to show that the embedded Markov chain is recurrent non-null under all realistic conditions (Δt_1, Δt_3 finite with probability 1; power consumption finite). Hence a steady-state temperature density function ($p(\theta)$) exists for this embedded Markov chain; the equation is

$$p(\theta) = \int_0^\infty p(\sigma)\pi_\sigma(\theta)d\sigma \tag{5}$$

We can solve the above numerically through iterations to obtain the probability density function (pdf) of the temperature when the plant enters S_1.

From the above, we can study the impact of thermally accelerated processor aging. High temperatures cause premature aging of processors; indeed, this rate is a sharply and non-linearly increasing function of operating temperature [11]. If the processor temperature is θ, denote its instantaneous aging rate by $\alpha(\theta)$. This is the rate of change of the "effective" processor age due to a temperature of θ. A common expression for this quantity is $\alpha(\theta) = \exp(-E_a/(k\theta))$ where E_a is the *activation energy* (typically 0.5–1.0 eV) [18].

The thermally-accelerated aging, $A(\theta_1, \Delta t_1, \Delta t_3, \omega_1, \omega_3)$, suffered by a processor which enters S_1 at temperature θ_1, stays in S_1 for time Δt_1, then transitions to S_3 where it stays for a further Δt_3 before returning to S_1 is given by

$$\begin{aligned} A(\theta_1, \Delta t_1, \Delta t_3, \omega_1, \omega_3) &= \int_0^{\Delta t_1} \alpha(\theta_{\omega_1}(t, \theta_1))dt \\ &+ \int_0^{\Delta t_3} \alpha(\theta_{\omega_3}(t, \theta_{\omega_1}(\Delta t_1, \theta_1))dt \end{aligned} \tag{6}$$

The average aging over one such cycle is obtained by integration:

$$A_{av}(\omega_1, \omega_3) = \int_0^\infty \int_0^\infty \int_0^\infty A(x, y, z, \omega_1, \omega_3)p(x)\psi_1(y)\psi_3(z)dzdydx \tag{7}$$

Dividing by the average duration between embedded epochs (i.e., successive entries by the plant into subspace S_1) gives us the average aging rate suffered by the system. The thermally accelerated aging factor [18], TAAF, is the ratio of the total thermally-accelerated age to the chronological age. Reducing the rate at which individual units need to be replaced is a contribution to sustainability: the TAAF is an indication of the factor by which the replacement rate can be reduced.

6. Impact on system provisioning

From the preceding sections, it is clear that the failure rate is affected by the processor workload. As permanent failures occur, the workload per processor tends to increase, which raises the temperature, and thereby the failure rate.

Failure in such cases is usually modeled as a Poisson process whose mean is the reciprocal of the mean processor lifetime [18]. Thus, when there are k processors left alive, and the load is fairly evenly spread among them, then we can define the permanent failure rate per processor as $\lambda(k)$. (If the load is not evenly spread, some processors will fail more readily than others: a similar analysis can be used, except that each processor will need its own customized failure rate. This makes the notation more tedious but not the reasoning behind the analysis.) Therefore, if we denote by $T(k)$ the time interval in which the system has k processors functional, and $n_{\min}$ is the minimum number of processors required to run the critical workload, the time to run out of processors is a random variable $L = T(N) + T(N-1) + \cdots + T(n_{\min})$, where $T(k)$ is exponentially distributed with parameter $\mu(k) = k\lambda(k)$ and N is the number of processors available initially. The density function of L is well-studied: see, for example [1]. It is given by:

$$f_L(t) = \sum_{i=n_{\min}}^{N} \frac{\prod_{j=n_{\min}}^{N} \mu(j)}{\prod_{\substack{j = n_{\min} \\ j \neq i}}^{N} (\mu(j) - \mu(i))} e^{-\mu(i)t} \tag{8}$$

The probability that the system runs out of processors due to permanent failures over some period of operation, T_{op}, can then be calculated as

$$p_{sysFail}(T_{op}) = 1 - \int_{t=0}^{T_{op}} f_L(t)dt \tag{9}$$

Critical systems often specify a maximum probability of failure over a given period. In order to meet this requirement, the system must be provisioned with sufficient processors to begin with, i.e., N must be selected so that $p_{sysFail}(T_{op})$ is sufficiently small. Note that this calculation is carried out in the context of the computational workload and the thermal characteristics of the system. The designer can control the impact of these factors by adjusting the task dispatch rate, selecting the control tasks appropriately (e.g., Model Predictive Control vs Linear Quadratic), designing the appropriate heat sink capability, and so on.

7. Energy modeling

Much (if not most) of the time, the controlled plant is expected to be in subspace S_1, meaning that fault-tolerance is not required. As a result, the processors will consume significantly reduced amounts of energy. This results in substantially reduced requirements for energy storage.

The total amount of energy $E(t)$ consumed over a given interval, t, is a random variable whose cumulative (probability) distribution function (CDF) can be obtained as follows. Suppose the sojourn time distribution in subspaces S_1, S_3 of the system is denoted by $F_{s1}(\cdot)$, $F_{s3}(\cdot)$, respectively. Let us make the reasonable approximation that these sojourn times are conditionally independent of one another, conditioned on the state of the operating environment. In such a case, the controlled plant's traversal of these subspaces can be modeled as an alternating renewal process [22]. In particular, the CDF of the time, $\Delta t_3(\xi)$, spent in S_3 over a period ξ is given by

$$F_{\Delta t_3(\xi)}(x) = \sum_{n=0}^{\infty} F^{(n)}_{s3}(x)[F^{(n)}_{s1}(\xi - x) - F^{(n+1)}_{s1}(\xi - x)] \tag{10}$$

where the superscript $^{(n)}$ denotes an n-fold convolution. When the sojourn times can be modeled as Gaussian random variables, their convolution is also Gaussian; for large n, a good Gaussian approximation can be obtained; finally, in arbitrary cases, they can be calculated numerically or estimated by simulation. (In our case study, we obtain empirical distributions for the plant sojourn times and then use standard bootstrap techniques [8].) In practice, the infinite sum can be truncated after a relatively small number of summands.

The time, $\Delta t_1(\xi)$, spent by the system in subspace S_1 over an interval of operation ξ can be obtained by $\Delta t_1(\xi) = \xi - \Delta t_3(\xi)$. We can now write the total energy consumed during a period ξ of operation:

$$E(\xi) = \omega_1 \Delta t_1(\xi) + \omega_3 \Delta t_3(\xi) \tag{11}$$

Since we have the CDFs of $\Delta t_1(\xi)$ and $\Delta t_3(\xi)$, we can compute the CDF of the energy consumed by the cyber subsystem.

This information can be used to size the energy storage subsystem for the cyber subsystem appropriately. In many applications, there are significant form-factor restrictions, which make overprovisioning of energy storage resources very expensive. For example, consider a UAV with a bank of supercapacitors for energy storage. We wish the storage to be sufficient so that the UAV can be controlled effectively for $\mathcal{T}$ hours with a probability of at least 0.9 (as its energy store dwindles, the UAV returns to base and lands). The energy storage for the cyber part of the system should therefore be at least $\mathcal{E}$ such that the Prob$\{E(\mathcal{T}) \le \mathcal{E}\} = 0.9$.

8. Case study 1: robot

Our first case study is that of a robot [26]. It is has joints at its ankle, knee and hip; there are three controls, one for each joint. There are six state variables, comprising the angles and the angular rate measured at the ankle, knee, and hip joints. It can be modeled as a linear system, with its continuous-time state equation given by $\dot{x}(t) = Ax(t) + Bu(t)$ where [26]

$$A = \begin{bmatrix} 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 18.337 & -75.864 & 6.395 & 0 & 0 & 0 \\ -22.175 & 230.549 & -49.01 & 0 & 0 & 0 \\ 4.353 & -175.393 & 95.29 & 0 & 0 & 0 \end{bmatrix}$$

$$B = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0.292 & -0.785 & 0.558 \\ -0.785 & 2.457 & -2.178 \\ 0.558 & -2.178 & 2.601 \end{bmatrix}$$

Transformation to a discrete-time system with zero-order control updates happening at intervals of length ΔT results in the difference equation representing the state at sampling epochs: $x(k+1) = \Phi x(k) + \Gamma u(k)$ where $\Phi = \exp(AT)$, $\Gamma = \int_0^{\Delta T} \exp(At) B dt$ [10]. To this, we add the impact of noise from the environment.

The robot can obviously perform a variety of functions; for the purpose of this case study, let us consider when it is at rest, maintaining its position despite impulses that it receives from the operating environment. The environment disturbance is modeled as force applied to the mass center of each body part of the robot. The arrival of the disturbance follows a Poisson distribution with arrival rate of 1, 2 or 5 Hz (denoted by $r1$, $r2$ and $r5$ in figures below). The force associated with each disturbance, follows a truncated Gaussian distribution with maximum amplitude of 50 or 100 N (denoted as Env50 and Env100 in figures below). The mean values equal half of the maximum and the standard deviation is 1/6 of the maximum. Control is provided by means of a linear quadratic regulator [23].

There are two parameters associated with control: the actuator derating and the actuator update rate. The latter translates to the rate at which the control tasks are dispatched. The principal factors not under our control are (a) the physical limits on the actuators' capabilities and (b) the operating environment.

Table 1
Plots notation key.

Notation	Meaning
mN	Maximum actuator force of *N*
envK	Environmental disturbance amplitude of *K*
rM	Disturbance arrival rate of *M*
pJ	Control task period of *J*
AP	Adaptively varying period
TAAF	Thermal Accelerated Aging Factor

We start by considering how S_1 and S_3 change in volume as a fraction of an absolute reference value. For an environment imposing a maximum disturbance of 50 N per "shock", Fig. 2a shows results for four typical task periods: 1, 2, 3, and 5 ms (corresponding to dispatch rates of 1000, 500, 333 and 200 Hz, respectively) and four levels of fixed actuator derating: 150, 250, 400 and 500 N (see Table 1 for a key to the notation used in the plots).

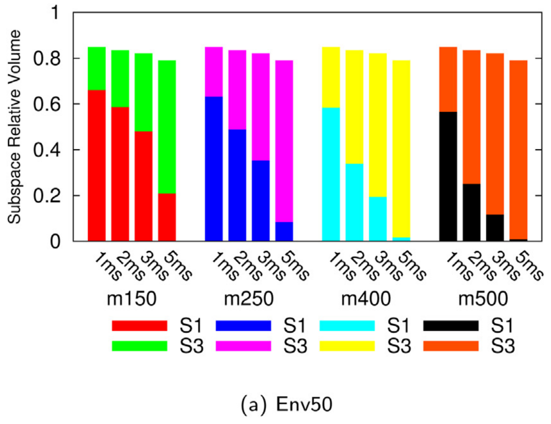
The total volume of the SSS (union of S_1 and S_3) drops gently as the task period is increased, for each of the four derating values. However, the volume of S_1 is far more sensitive to task period; it drops noticeably as the period increases. The greater the actuator maximum value allowed, the faster is the rate at which the S_1 volume goes down. Clearly, for such an environment, a task dispatch period of 5 ms is largely useless for adaptive fault-tolerance, since S_3 occupies almost all the SSS.

Fig. 2b shows equivalent results when the environmental disturbance amplitude is twice as great at 100 N. Here, one can see a sharper fall in the S_1 volume as the task period increases. Furthermore, notice that even at a period of 1 ms, well under half the SSS volume is occupied by S_1. This indicates the burden that a more hostile environment places on the system.

The raw volumes of S_1 and S_3 do not tell us the whole story: of primary interest is the fraction of time the system spends in S_1, where the benefits of adaptive fault-tolerance can be gained. This is because the plant spends more time in some regions of the plant state-space than in others. In Fig. 3a, we show the fraction of time the system spends in S_1 for each of the four periods, under the two environmental amplitudes of 50 and 100 N mentioned above, with environmental disturbances arriving as a Poisson process with rates 1, 2, and 5 per second. Adaptive actuator derating is used. We can see that (from the same group of bars in different colors), for a disturbance amplitude of 50, doubling the task period from 1 to 2 results in only a marginal reduction in the time spent in S_1. Increasing it further, however, results in a dramatic drop in this fraction. When the environment disturbance amplitude is 100 N, very little of the time is spent in S_1. This illustrates the cost to the system of having to perform in a hostile environment.

Fig. 3a also illustrates the energy savings possible from this approach. The energy consumed by the control tasks when the plant is in S_1 will be no greater than 1/3 of the baseline case; savings will be even greater if dynamic voltage and frequency scaling is used [24]. Fig. 4 shows a lower bound to the estimated energy savings for this application. The columns denoted by "w AP" show the improvement when the task dispatch rate is also adaptively changed. The lowest dispatch rate is chosen as long as the current state is in S_1. Note that the energy savings are obtained mostly for the case where control tasks are dispatched at a high rate: this is precisely the dispatch rate at which we have improved quality of control. Indeed, this approach saves energy best where it is most useful: for being able to sustain high control update rates.

Fig. 3b shows the time the system commands an output leading to the actuator(s) being saturated (i.e., reaching their derated output) as a fraction of the total time spent by the system in S_1. This fraction is very low for high dispatch rates, meaning that the con-



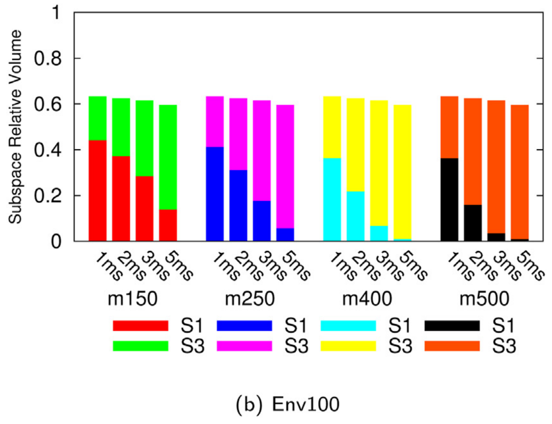
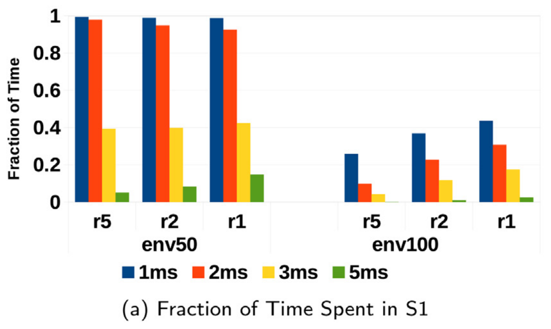


Fig. 2. Subspace volume relative to the SSS size for Env0.



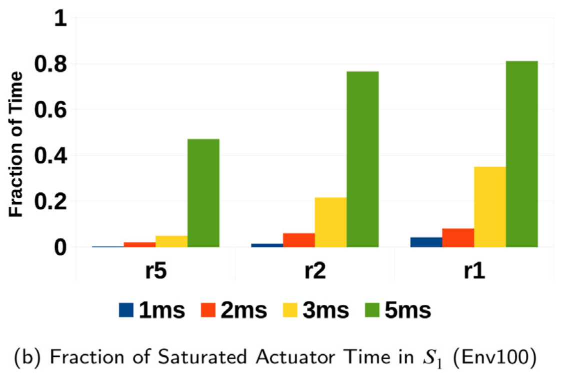


Fig. 3. Timing of the controlled plant in S_1.

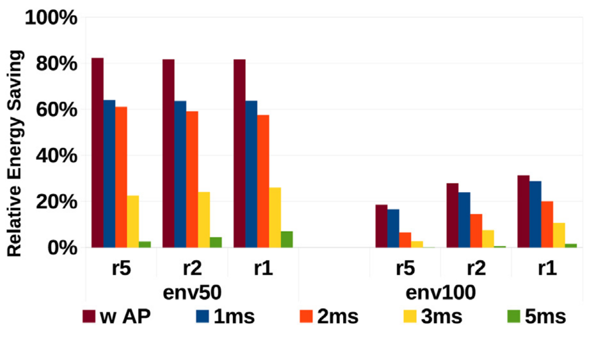


Fig. 4. Energy savings.

trol applied is optimal for most of the time; once the actuators are saturated, the control becomes suboptimal since the optimal value is outside the derated range allowed. Note that Fig. 3b assumes a 100 N environmental disturbance amplitude: under a more benign, 50 N disturbance amplitude (not shown here), the actuators were observed to remain unsaturated for almost 100% of the time.

For calculating thermal stress, it is not just the average fraction of the time spent in S_1 and S_3 that matters: it is the cumulative distribution function. The CDF of time spent in S_1 and S_3 is illustrated for a selected number of conditions in Fig. 5; note that the x-axis is logarithmic indicating the significant impact that the operating environment and task period play. As is shown in Fig. 5, with benign environment (r1, Env50) and high control dispatch rate (p1), the S1 duration per visit is longer, i.e., the system stays in safer states for longer time before the environment drive it to relative dangerous states. Besides thermal stress, with the distribution of time spent in S_1 and S_3, the energy consumption can be estimated for a given time interval using the approach described in Section 7.

The saturation of actuators is likely to result in a degradation in the quality of control (QoC). A standard linear quadratic measure is used to express QoC (see Eq. (1) in Section 2). (Note that under this measure, a bigger numerical quantity indicates *worse* QoC.) We used identity matrices for both Q and R. The degradation of QoC was observed to be negligible when the environmental disturbance amplitude is 50N, and is therefore not shown here. Results for an amplitude of 100 N are shown in Fig. 7. Only under the most hostile environmental conditions studied does the degradation (barely) exceed 10%. Other variations are minimal and statistically insignificant. Note the QoC degradation shown in Fig. 7 is the relative value w.r.t. when there is no degradation. In hostile environment (r5) and with smaller dispatch rate (i.e., larger period), the QoC is worse and thus the degradation may be smaller.

We now turn to thermally accelerated aging (see [16] for background). A standard thermal equivalent circuit is used to model heat flow [11]. The equivalent thermal resistance and capacitance are $R_{th} = 2\Omega$ and $C_{th} = 0.5F$. An activation energy of $E_a = 0.7$ eV is assumed; as mentioned previously, the instantaneous acceleration factor at temperature θ is assumed to be $\alpha(\theta) = \exp(-E_a/(k\theta))$ where k is Boltzmann's constant.

The improvement in the thermally accelerated aging factor, TAAF, is calculated as $1 - \text{TAAF}_{algorithm}/\text{TAAF}_{baseline}$ and is shown in Fig. 6. The baseline case is the standard one where fault-tolerance is on all the time. The rule when running a single task (when in S_1) is to run it on the coolest core available. Results are shown both for our dynamic derating (Fig. 6a) and for the case where the actuator limit is set constant at 500 N (Fig. 6b). Both cases show an improvement over the baseline; however, the adaptive approach has clear advantages at lower task dispatch rates and under a more

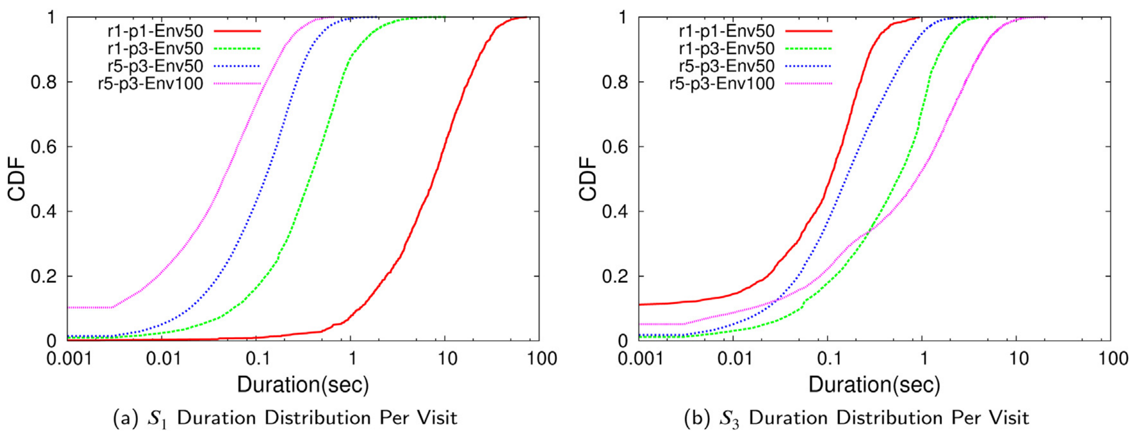


Fig. 5. Plant sojourn time CDFs.

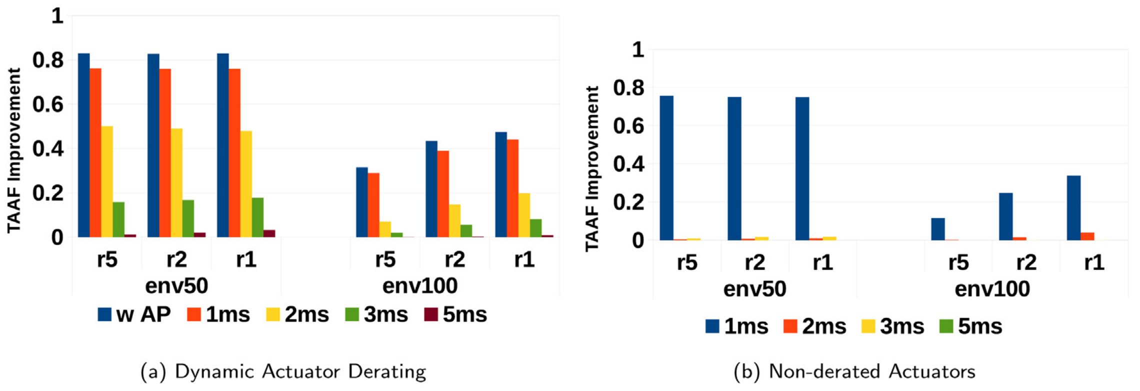


Fig. 6. Improvement in thermally accelerated aging factor.

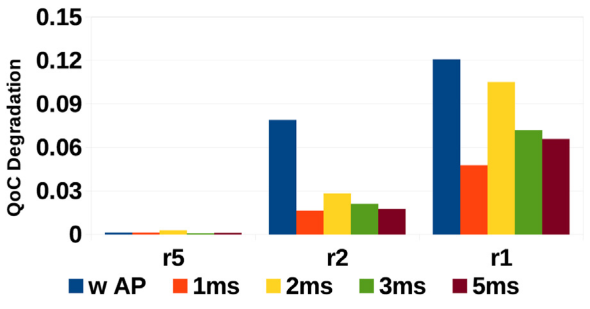


Fig. 7. Degradation in quality of control.

Table 2
Additional notation for Figs. 6–10.

	Adaptive period adjustment	Actuator derating w/o Algorithm 2	Actuator derating w Algorithm 2
Baseline	No	No	No
AP	Yes	No	No
DMAP	Yes	Yes	No
DMAP-D	Yes	Yes	Yes
pN		N number of functional processors	
W_r, W_ℓ		Window size used in Algorithm 2	

hostile operating environment. We can therefore see clear advantages in both computational energy requirements and processor aging when using this approach.

In Section 6, we presented a model for system lifetime. Based on this model, Fig. 8a shows the relative failure rate as a function of the number of functional processors. (See Table 2 for the notation used.) The unit of failure rate is that of a processor at room temperature, 300 K. The disturbance arrival rate is 5 per second. As is shown, with a harsher environment and fewer functional processors, the failure rate is higher and the improvement in failure rate using the proposed adaptive fault-tolerance techniques is greater. Fig. 8b shows the CDF of the time needed for the system to have fewer than 3 processors functional, when we start with 5 processors. The CDF for the baseline will be unaffected by environmental disturbances since it is a static triplex configuration. These results indicate the significant improvements made possible in system lifetime as a result of our adaptive approach.

In Section 4, we introduced Algorithm 2 to dynamically change the actuator derating constraints in order to improve the control quality of when such derating causes frequent actuator saturation. In Fig. 9, we show the impact on QoC of adaptive control period (AP), dynamic actuator derating with adaptive period (DMAP) and dynamic actuator derating with adaptive period and dynamic derating constraints (DMAP-D) (see Table 2). Note that the QoC expression is based on deviation from the optimal state trajectory, so that larger numbers indicate poorer QoC. The AP approach, without any derating, provides the best QoC since it places no limits on the actuators. The DMAP-D QoC degrades (involving the use of Algorithms 1 and 2) by a small amount; not using Algorithm 2 causes a small further degradation. These losses in QoC are offset by gains in the TAAF (leading to the reliability gains illustrated previously). Fig. 10, which seeks to balance the gains of one against losses in the other indicates the tradeoff: now, AP performs the worst since it pays heavily in terms of workload-related stress.

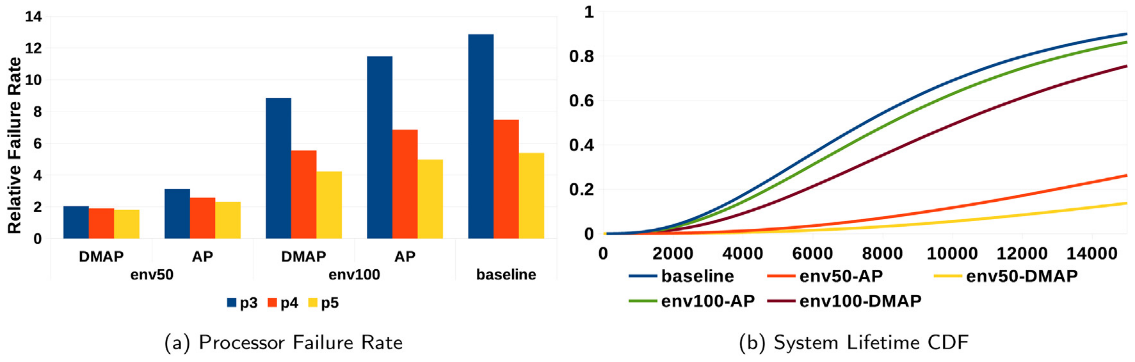


Fig. 8. Processor failure rate and CDF of time to system failure.

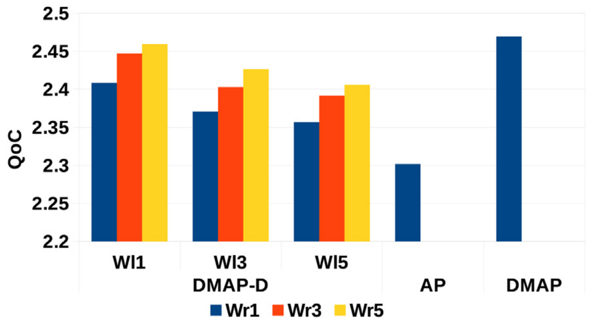


Fig. 9. QoC using different adaptive fault-tolerance.

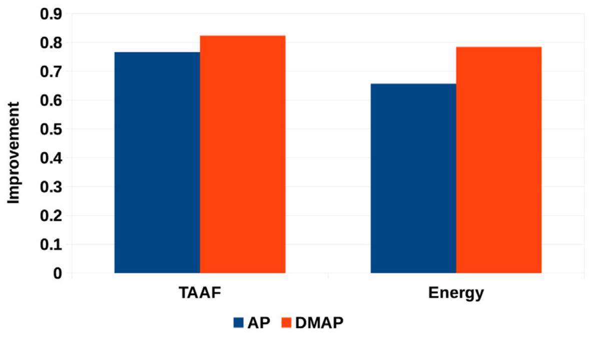


Fig. 11. Improvement in TAAF and energy.

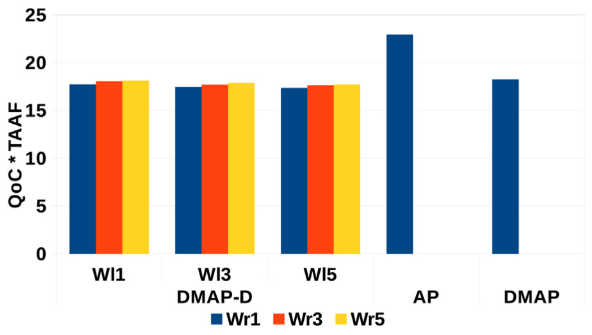


Fig. 10. Product of QoC and TAAF.

9. Case study 2: quadcopter

The application here is a quadcopter hovering in position, trying – in the face of atmospheric disturbances – to retain its location (vertical and horizontal) and body position (rotation, pitch and yaw angle). The quadcopter has 6 degrees of freedom. Thus there are 12 state variables (one for each degree of freedom and another for its first derivative). There are four controls, one for each propeller.

Quadcopter dynamics can be fairly accurately linearized at the desired state with its continuous-time state equation being given by $\dot{x}(t) = Ax(t) + Bu(t)$ where A and B are sparse 12×12 and 12×4 matrices, respectively. The non-zero elements of these matrices have the following values: $A(1, 4) = A(2, 5) = A(3, 6) = A(7, 10) = A(8, 11) = A(9, 12) = 1$; $A(4, 4) = A(5, 5) = A(6, 6) = -0.5$; $A(4, 8) = 9.81$, $A(5, 7) = -9.81$; $B(6, 1) = B(6, 2) = B(6, 3) = B(6, 4) = 0.06$; $B(10, 1) = 1.5$; $B(10, 3) = -1.5$; $B(11, 2) = 1.5$; $B(11, 4) = -1.5$; $B(12, 1) = B12, 3) = 0.1$; $B(12, 2) = B(12, 4) = -0.1$. For a fuller discussion of the system, see [27].

Model predictive control (MPC) is used for control. The quadcopter has four propellers; each propeller is separately controlled. Its motion is managed by applying differential controls to these; i.e., the actuators are the motors driving each propellers. The busy power of each processor used here is 15 W; idle power is 0.5 W. The execution of the control task per propeller takes 10 ms.

The disturbance from the operating environment is modeled as having a Gaussian distribution truncated at a maximum value of 1 N. The mean values equal half the maximum; the standard deviation is 1/6 of the maximum.

As in Case study 1, the baseline is the traditional approach where full fault-tolerance is used all the time. Fig. 11 shows the improvement in thermally accelerated aging over baseline, where the actuator derating levels are 10 and 15 N; the physical (non-derated) maximum is 20 N. The results show the improvement due to adaptive control period (AP) and dynamic actuator derating with adaptive period (DMAP) in Fig. 11.

The quality of control (QoC) degradation is minimal. The QoC is defined as the sum of the square of the distance from the desired position. With full fault-tolerance all the time, the QoC is around 0.0001 while the QoC is 0.0002 for AP and 0.01 for DMAP.

The failure rate with different number of functional processors is shown in Fig. 12. As is in the previous case study, using actuator derating and adaptive period together achieves the most failure rate improvement.

10. Related prior work

Many researchers have focused on thermal management in recent years. Some of this work tries to keep the energy consumption or the temperature below a given threshold while continuing to meet the performance requirement or optimize the trade-offs between thermal behaviour and the performance [31–33]. There is also work on reducing thermal damage while taking the timing constraint in real-time systems into consideration [34–36]. However, few of these works take into consideration that the workload itself

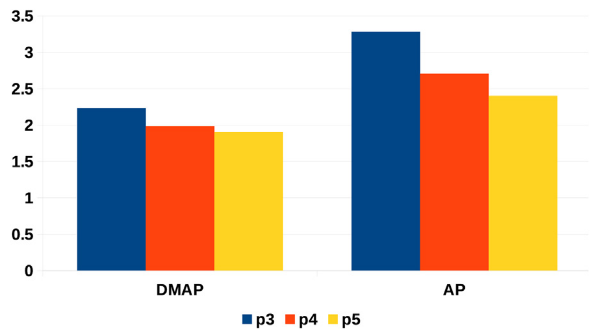


Fig. 12. Failure rate with different number of functional processors.

can be reduced by reducing redundancy in CPSs under conditions where this can be done safely.

Perhaps the earliest comprehensive look at the use of adaptive fault-tolerance in highly reliable systems was the work of Kim and Lawrence [13] in which the basic concept and research issue of adaptive fault-tolerance is discussed and a case study was presented to examine changing fault-tolerance levels with respect to the environment. A software infrastructure for adaptive fault-tolerance, *Chameleon*, is discussed in [12]. Object-oriented approaches to implementing adaptive fault-tolerance are described in [9,25]. The Simplex approach uses two versions of control tasks: a simple, suboptimal, but highly reliable one, and a complex, high-performance, but less reliable version [5]. The version to be run is selected based on the vulnerability of the controlled plant. The focus here is on protecting against software faults in the high-performance version. In [29,30], a component-based adaptive fault-tolerance approach was proposed. During the lifetime of a system, different fault-tolerance strategies would be selected adaptively based on changes in application characteristics and resources.

In the previous works mentioned above, the term *adaptive* means the adaptive selection of fault tolerance strategies, i.e., using different control tasks or different amounts of available resources. None of them allows turning on/off fault tolerance adaptively.

In [18], the plant state space was divided into three subspaces based on how much fault-tolerance was required of the cyber subsystem. It was shown there how the plant, operating environment and the controller interacted. However actuator limits and task periods were assumed fixed. No such restrictions apply in the present paper; we study here the benefits of adaptively and dynamically derating actuators and control task dispatch rates based on plant state.

11. Discussion

Cyber-physical systems are increasingly used in applications demanding high dependability. Dependability is defined in terms of the application: it is defined as the ability of the system to keep the controlled plant always within some defined Safe State-Space. The processors which form the cyber side of CPSs often operate under highly constrained conditions of power, temperature and cost. Fault-tolerance is expensive with respect to all these dimensions; however, it is essential in high-dependability CPS applications. The conventional approach has been to use fault-tolerance all the time. By contrast, an adaptive application of fault-tolerance allows for significant power, temperature and cost benefits. In this paper, we have outlined an approach which uses dynamically adaptive actuator derating to allow more economical use of fault-tolerance.

Derating reduces the capability of an incorrectly set actuator to drive the plant to failure, i.e., to outside its Safe State Space. It comes at the price of lowering the range of controls that can be applied, and therefore potentially lowers the quality of control. How this benefit/cost tradeoff works out depends on how the controlled plant interacts with the disturbances caused by its operating environment.

Our approach leads to significant computational energy reduction. In many applications, this can result in substantially reduced stress on the energy storage system (a lower current draw leading to greater efficiency; fewer charge/discharge cycles leading to a longer battery/supercapacitor lifetime) and reduced thermal stress on the processors (leading to longer processor lifetimes).

Authors' contribution

Shikang Xu: conceptualization, methodology, software, validation, writing (original draft and editing). Israel Koren: conceptualization, resources, writing (editing), supervision, project administration, funding acquisition. C.M. Krishna: conceptualization, resources, writing (editing), supervision, project administration, funding acquisition.

Conflict of interest

None declared.

References

[1] M. Akkouchi, On the convolution of exponential distribution, J. Chungcheong Math. Soc. 21 (4) (2008).
[2] B.D.O. Anderson, J.B. Moore, Optimal Control: Linear Quadratic Methods, Courier Corporation, 2007.
[3] E. Asarin, T. Dang, A. Girard, Reachability analysis of nonlinear systems using conservative approximation, International Workshop on Hybrid Systems: Computation and Control (2003) 20–35.
[4] K.j. Åström, B. Wittenmark, Computer-Controlled Systems: Theory and Design, Courier Corporation, 2013.
[5] S. Bak, D.K. Chivukula, O. Adekunle, M. Sun, M. Caccamo, Lui Sha, The system-level simplex architecture for improved real-time embedded system safety, 15th IEEE Real-Time and Embedded Technology and Applications Symposium, 2009. RTAS 2009 (2009) 99–107.
[6] A. Bemporad, M. Morari, V. Dua, E.N. Pistikopoulos, The explicit linear quadratic regulator for constrained systems, Automatica 38 (1) (2002) 3–20.
[7] J. Choi, C. Cher, H. Franke, H. Hamann, A. Weger, P. Bose, Thermal-aware task scheduling at the system software level, Proceedings of the 2007 International Symposium on Low Power Electronics and Design (2007) 213–218.
[8] B. Efron, R.J. Tibshirani, An Introduction to the Bootstrap, CRC Press, 1994.
[9] J. Fraga, F. Siqueira, et al., An adaptive fault-tolerant component model, Ninth International Conference on Object-Oriented Real-Time Systems (2003) 179.
[10] G.F. Franklin, J.D. Powell, M.L. Workman, Digital Control of Dynamic Systems, Addison-wesley, Menlo Park, CA, 1998.
[11] W. Huang, S. Ghosh, S. Velusamy, K. Sankaranarayanan, K. Skadron, M. Stan, Hotspot: A compact thermal modeling methodology for early-stage VLSI design, IEEE Trans. Very Large Scale Integr. Syst. 14 (5) (2006) 501–513.
[12] Z.T. Kalbarczyk, R.K. Iyer, S. Bagchi, K. Whisnant, Chameleon: A software infrastructure for adaptive fault tolerance, IEEE Trans. Parallel Distrib. Syst. 10 (6) (1999) 560–579.
[13] K.H. Kane Kim, T.F. Lawrence, Adaptive fault-tolerance in complex real-time distributed computer system applications, Comput. Commun. 15 (4) (1992) 243–251.
[14] D. King, I.A. Waitz, Assessment of the Effects of Operational Procedures and Derated Thrust on American Airlines B777 Emissions from London's Heathrow and Gatwick Airports, 2005.
[15] I. Koren, C.M. Krishna, Fault-Tolerant Systems, Morgan Kaufmann, 2010.
[16] I. Koren, C.M. Krishna, Temperature-aware computing, Sustain. Comput.: Informatics Syst. 1 (1) (2011) 46–56.
[17] C. Krishna, K. Shin, Performance measures for control computers, IEEE Trans. Autom. Control 32 (6) (1987) 467–473.
[18] C.M. Krishna, Ameliorating thermally accelerated aging with state-based application of fault-tolerance in cyber-physical computers, IEEE Trans. Reliab. 64 (1) (2015) 4–14.
[19] C.M. Krishna, K.G. Shin, Real-Time Systems, McGraw-Hill, 1996.
[20] A.A. Kurzhanskiy, P. Varaiya, Ellipsoidal techniques for reachability analysis of discrete-time linear systems, IEEE Trans. Autom. Control 52 (1) (2007) 26–38.
[21] J.W.S. Liu, Real-Time Systems, Wiley, 2000.
[22] T. Nakagawa, Maintenance Theory of Reliability, Springer Science & Business Media, 2006.
[23] K. Ogata, Discrete-Time Control Systems, vol. 2, Prentice Hall, Englewood Cliffs, NJ, 1995.

[24] P. Pillai, K.G. Shin, Real-time dynamic voltage scaling for low-power embedded operating systems, ACM SIGOPS Operating Systems Review, vol. 35 (2001) 89–102.

[25] E. Shokri, H. Hecht, P. Crane, J. Dussault, K.H. Kim, An approach for adaptive fault tolerance in object-oriented open distributed systems, Int. J. Software Eng. Knowl. Eng. 8 (03) (1998) 333–346.

[26] https://github.com/pydy/pydy tutorial-human standing.

[27] https://github.com/ketakiB/quadcopter.

[28] Y. Xu, I. Koren, C.M. Krishna, Adaft: a framework for adaptive fault tolerance for cyber-physical systems, ACM Trans. Embedded Comput. Syst. 16 (3) (2017) 79.

[29] M. Lauer, M. Amy, J. Fabre, M. Roy, W. Excoffon, M. Stoicescu, Engineering adaptive fault-tolerance mechanisms for resilient computing on ROS, A2016 IEEE 17th International Symposium on High Assurance Systems Engineering (HASE) (2016) 94–101.

[30] M. Stoicescu, J.-C. Fabre, M. Roy, Architecting resilient computing systems: a component-based approach for adaptive fault tolerance, Special Issue on Reliable Software Technologies for Dependable Distributed Systems, vol. 73 (2017) 6–16.

[31] A.K. Coskun, T.S. Rosing, K.A. Whisnant, K.C. Gross, Static and dynamic temperature-aware scheduling for multiprocessor SOCS, IEEE Trans. Very Large Scale Integr. Syst. 16 (9 (September)) (2008) 1127–1140.

[32] H.F. Sheikh, I. Ahmad, Sixteen heuristics for joint optimization of performance, energy, and temperature in allocating tasks to multi-cores, ACM Trans. Parallel Comput. 3 (2) (2016).

[33] H. Sheikh, I. Ahmad, S. Arshad, performance, energy, and temperature enabled task scheduling using evolutionary techniques, Sustain. Comput.: Syst. Informatics (2019) 272–286.

[34] S. Xu, I. Koren, C. Krishna, Thermal-aware task allocation and scheduling for heterogeneous multi-core cyber-physical systems, International Conference on Embedded Systems, Cyber-Physical Systems and Applications (2016).

[35] S. Xu, I. Koren, C. Krishna, Improving processor lifespan and energy consumption using DVFS based on ILP monitoring, Workshop on Low-Power Dependable Computing (2015).

[36] S. Xu, I. Koren, C. Krishna, Thermal aware task scheduling for enhanced cyber-physical systems sustainability, IEEE Trans. Sustain. Comput. (2019), http://dx.doi.org/10.1109/TSUSC.2019.2958298.