Adaptive Fault-Tolerance for Cyber-Physical Systems

C.M. Krishna and I. Koren
Department of Electrical and Computer Engineering
University of Massachusetts

***Abstract*—Cyber-physical systems are increasingly used in life-critical applications, where the probability of catastrophic failure has to be kept below very low levels. Massive fault-tolerance has been used to mask failure to achieve such low levels. However, fault-tolerance is expensive. We argue here that the fault-tolerance needs of an application change depending on its current position in its state space and the range of control inputs that can be applied. We illustrate the applicability of such an approach using the inverted pendulum as a case-study, and discuss its potential benefits.**

I. Introduction

Life-critical systems such as aircraft, nuclear and chemical reactors, power generation systems, and automobiles are increasingly being controlled by computer. The field of cyber-physical systems has expanded its application set tremendously over the past two decades. Controlling by computer allows for far more complex and efficient control algorithms to be used in comparison to the traditional hard-wired analog controller.

With a more complex control system comes the need for effective fault-tolerance. Fault-tolerance has long been a major concern for cyber-physical systems [2], [7]. In cyber-physical systems, it often takes the form of massive redundancy, with voting to mask component failures.

Our main claim in this paper is that while massive hardware redundancy is required, its actual runtime deployment can be done adaptively, based on the current state of the controlled plant. In many, if not most, instances, the plant is in a state which allows for a lowered level of fault-tolerance. Such a lowered level can be useful in several ways:

- A reduced computational load allows power management techniques to be used to reduce the amount of energy consumed by the computing system. Given that many cyber-physical applications are energy-constrained, this is often very useful.
- Reducing the energy consumed leads to a reduction in the amount of heat generated; processors can be run cooler under a lighter load. Since temperature is strongly correlated to the rate at which processors age, running cooler itself contributes to an increase in reliability.
- Reducing the computational load associated with critical tasks can often allow the system to devote more resources to non-critical tasks which improve the quality of control.

In this paper, we use the case study of an inverted pendulum to illustrate some of the key issues associated with implementing adaptive fault tolerance. An inverted pendulum has often been used in the cyber-physical community to act as a proxy for higher-order systems; for a detailed justification, see [14]. We use this case study as a means to identify the operational choices that are available when adapting fault-tolerance levels to current system state.

The rest of this paper is organized as follows. In Section II, we subdivide the state-space of the controlled plant according to how much fault-tolerance is required in each of them. Section III discusses the implications on scheduling of an adaptive fault-tolerance approach; in particular, the fact that the computational burden is no longer linearly related to the task dispatch frequency as is conventionally the case. Section IV covers the impact on processor reliability. In Section V, we use the inverted pendulum as a case study to illustrate our ideas. The paper concludes with a brief discussion in Section VI.

II. Adaptive Fault-Tolerance

We will consider a single control task. Later in this paper, we briefly mention how this scheme might be extended to multiple, interacting, control tasks. We assume that control tasks are executed repeatedly. They could be periodic or quasi-periodic (i.e., where the periods vary a bit from one iteration to the next).

We start with the well-known notion of the *allowed state-space*. The allowed state-space of a plant is where it is functioning within acceptable limits. For example, in [12], we have an example of the allowed state-space of an aircraft in the final phase of landing for the altitude, descent rate, pitch angle and pitch angle rate. So long as the plant remains within the allowed state-space, it is defined as not having failed. (Note that this does not mean the plant performs optimally: that is usually related to being in a much smaller subset of the plant state-space.)

We distinguish between the state at sampling instants and the rest. A sampling instant is when the control task is initiated. We subdivide the allowed state-space *at the sampling instant* (see Remark 2 below) of the controlled plant according to the level of fault-tolerance required in each. The most basic classifications are as follows:

- S_N: No fault-tolerance is required. The controlled plant is in a region of the state-space where even if the actuators are held at their worst-case incorrect setting until the next iteration of the control task, it will not leave the allowed state-space. Of course, if this should happen, the plant is likely to move closer to the edge of its allowed state-space and may have to expend more energy or other resources at the next iteration of the control task.
- S_{FS}: It is sufficient for the computer to be fail-stop. By *fail-stop*, we mean a system in which only two types

of computer output are possible: correct or no (or zero) output. A duplex is the most obvious fail-stop system: two processors replicate the computation and compare their results. If they agree, the output can be declared correct; if they disagree by more than some margin, we know that at least one of the outputs is wrong (but not which is wrong).

- S_F: Full fault-masking is required. That is, if the controller produces an incorrect or no output, the plant cannot be guaranteed to remain in the allowed state-space. A triplex (also known as Triple Modular Redundancy) is an example of such fault-masking: three processors replicate the computation and vote on their results [6]. So long as no more than one processor is faulty, a correct output is produced.

Note that $S_N \subseteq S_{FS}$.

Note further that there is no need to concern ourselves about what to do when the controlled plant is outside the allowed state-space, since it is defined as having already failed. We do not handle in this paper the issue of post-failure recovery of the controlled plant.

What factors control the size of the subspaces? The most obvious are the dynamics of the controlled process, as represented for instance by its state equations. This is not under our control, so we assume these dynamics to be the given context under which we have to operate. The most important factors that we can control are the rate at which the control task is executed and the range of outputs that the actuator is capable of delivering. The greater the rate at which the control output is updated, the greater the resilience of the system to an incorrect output or no output at all. The lower the range of outputs that the actuator can produce, the greater its resilience to producing an incorrect output; however, lowering the actuator force can increase the minimum possible rate at which the control task needs to be executed.

Remark 1: We are assuming that the state of the controlled plant is known correctly; in other words, we are assuming that the sensors have not failed. Sufficient redundancy is always required of sensor information so that any sensor faults are fully masked. Since the entire response of the system depends on the correctness (up to some designated level of precision) of the values received from the sensors, this must be regarded as an absolute requirement. (As an example, there have been recent high-profile disasters caused mainly or partly by failures of sensor input, most notably the AF 447 and Aeroperu 603 air-crashes.) Similarly, it is assumed that the computational task dispatcher is sufficiently fault-tolerant. We concentrate on the fault-tolerance level required by application tasks only, not the system software.

Remark 2: The allowed state-space is the region in which the controlled plant is deemed to be functioning satisfactorily. This is purely based on the application requirements, not on the capabilities of the controller.

In order to always remain within the allowed state-space, the plant must be in some given subset of the allowed state-space at sampling instants. That is, we can define a subset of the allowed state-space, S_σ, such that if the state $s \in S_\sigma$ at a sampling epoch, it is guaranteed to remain within the allowed state-space throughout the inter-sampling interval.

III. Implications for Scheduling

Let us start with the case where the control task is periodic. To minimize the computational load on the controller, the conventional approach would be to pick the maximum period under which quality-of-control requirements are met, subject to a given safety margin. Alternatively, one may trade off computational load against quality of control. One can analyze the dynamics of the controlled plant in order to obtain task deadlines (and thus periods, for periodic control tasks): see, for example, [10], [11], [12]. In every instance, when a traditional fault-tolerance scheme is used, the computational load increases linearly with the control task dispatch frequency, i.e., with the inverse of the period.

When adaptive fault-tolerance is used, however, there is no longer a monotonic relationship between the computational load and the period. In other words, reducing the period by a factor of k does not increase the load by the same factor; the reason is that the required level of fault-tolerance may be different for each of the cases.

To make this more precise, let us denote by $\psi_{FS}(P)$ and $\psi_N(P)$ the fraction of time that the system spends in subspaces S_{FS} and S_N when the control task period is P. (In this discussion, to begin with, we focus on the case of a single control task with the period of the other tasks fixed.) We can simulate the controlled plant to determine $\psi_{FS}(P)$ and $\psi_N(P)$. Then, if the control task execution time is e, the average task utilization is given by

$$u(P) = \frac{e}{P}\left\{\psi_N(P) + 2\psi_{FS}(P) + 3(1 - \psi_N(P) - \psi_{FS}(P))\right\}.$$

Compare this with the $3e/P$ utilization associated with using non-adaptive fault-tolerance.

If, as is generally the case, we have $n > 1$ interacting control tasks, an iterative approach can be used to pick the appropriate periods, $P_1, P_2, \cdots, P_n$. Pick an initial value for each P_i. Optimize P_1, keeping $P_2, \cdots, P_n$ fixed. Then, with this new value of P_1, optimize P_2 using the same approach, and so on.

When the task is not dispatched periodically, but is sporadic, the system must decide when to trigger a control task; this can be done anytime the controlled plant enters some designated subset, S_π, of S_σ. Determining S_π is an interesting problem that is outside the scope of this paper: the extent of fault-tolerance required depends on the plant state, as before. One strategy may, for example, be to dispatch the task when it is on the edge of the S_N subset; this ensures that (barring emergencies) no redundant copies of the control task are required; in such a case, S_π will be a region covering the outer boundary of the S_N subspace.

IV. Implications for Reliability

It has long been held that task loading is positively correlated with the failure rate [5], [9], [13]. This is increasingly because of thermally-induced accelerated failure rates. For

example, one analysis of electromigration-related failure of VLSI interconnect suggests that the mean time to failure is inversely related to the operating temperature [8]:

$$MTTF \propto \frac{1}{E\left[j(t)\frac{\exp\left(\frac{-Q}{kT(t)}\right)}{kT(t)}\right]}$$

where $E[]$ is the expectation operator, $j(t)$ is the current at time t, Q is the material activation energy, k is Boltzmann's constant, and $T(t)$ is the absolute temperature at time t.

The temperature profile of a chip is obviously a function of the task loading and can be greatly improved by reducing redundancy levels. For example, if we only need to run one copy of a task rather than the default of three, we can assign them one by one to the processors in a round-robin fashion, so that each of the three processors in a triplex is idle for two of every three periods. This can allow for a significant reduction in the chip temperature, with a consequent benefit to reliability. This, in turn, can reduce the number of backup (cold-start) processors that are needed to maintain a certain level of reliability. Such a reduction can be especially beneficial in applications where repair is either not possible or is very expensive (e.g., space missions).

V. Case Study: Inverted Pendulum Control

We use an inverted pendulum to illustrate our ideas. An inverted pendulum, as the term implies, is a rigid rod with the bob at the top and attached by a hinge at the bottom. Movement is restricted to one dimension by the hinge. The hinge is affixed to a bogie which moves along just one dimension: it is the acceleration of this bogie that is the single control input. The objective is to keep the pendulum as close to vertical as possible. Obviously, the pendulum is in either a state of unstable equilibrium or of inequlibrium.

The system admits of a simple state-space model: the state variables are $\theta(t)$ and $\dot{\theta}(t)$, where $\theta(t)$ is the angle the pendulum makes with the vertical (positive or negative). Define $x(t) = [\theta(t) \;\; \dot{\theta}(t)]^T$. The state equations are as follows [14]:

$$\dot{x}(t) = Ax(t) + Bu(t)$$

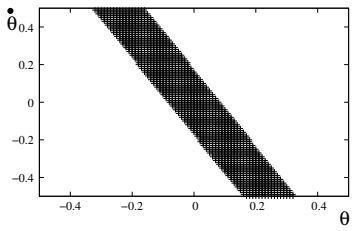
where we have the following notation:

$$\begin{aligned} g &\quad \text{Gravitation constant, } 9.81ms^{-2} \\ \ell &\quad \text{Length of pendulum} \\ J &\quad \text{Moment of inertia, } g\ell^2 \\ \gamma &\quad \text{Friction constant (at the hinge)} \\ A &= \begin{bmatrix} 0 & 1 \\ g/\ell & -\gamma/J \end{bmatrix} \\ B &= \begin{bmatrix} 0 \\ \ell/J \end{bmatrix} \end{aligned}$$

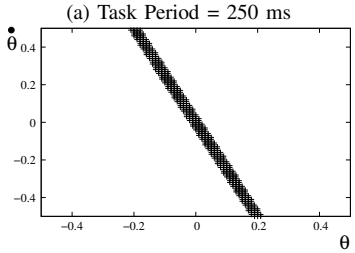
The control input is calculated every time the control task is executed; a zero-order hold (ZOH) approach is taken whereby the input is held constant until the next input becomes available [1]. Define $\tau(t)$ to be the latest time prior to t when a calculation of the control input is carried out and ξ the time prior to this when the value of the system state variables is last sampled. Then, the control input is given by $u(t) = -K\theta(\tau(t) - \xi)$ where K is a given constant of proportionality, subject to a maximum magnitude of u_{lim}[1].

Variable	Value
m	1 kg
ℓ	1 m
K	20
u_{lim}	1.0 N
θ_{lim}	0.5 rad
$\dot{\theta}_{lim}$	0.5 rad/sec
γ	1E-4

TABLE I: Default Parameter Values



(a) Task Period = 250 ms



(b) Task Period = 350 ms

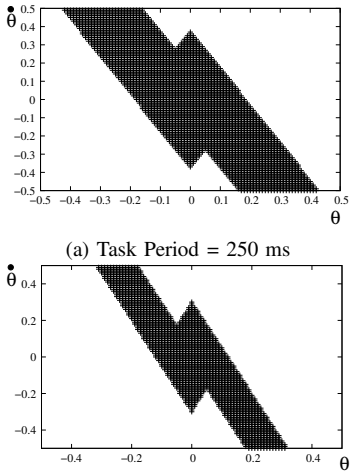
Fig. 1: S_N Subspace

We start by plotting the S_N, S_{FS}, and S_σ subspaces for the parameter values selected in Table I. These are determined as follows. For S_N, we must allow for the actuators to be placed in the maximally wrong position and still not leave the allowed state-space anytime until the next sampling instant. For S_{FS}, we must allow for the actuators to be set to 0 (neutral setting upon discovery of an error in the computation) without leaving the allowed state space. For S_σ, we assume that the actuators are always correctly set. Obviously, $S_N \subseteq S_{FS} \subseteq S_\sigma$.

The parameters of the inverted pendulum are shown in Table I. The shaded portions of Figure 1 indicates the subspace where no fault-tolerance is required: even if the software produces a wrong output of maximum possible magnitude from the actuator, the system will stay within the allowed state-space ($|\theta| \leq 0.5$ rad and $|\dot{\theta}| \leq 0.5$ rad/sec). The shaded region in Figure 2 shows the subspace where it is sufficient to have just fault-detection, rather than full-blown fault-masking. Finally, the shaded region in Figure 3 shows the subspace S_σ. These subspaces are not convex. The reason is the control algorithm that we use in this paper is based only on the currently observed angle, $\theta(t)$ and not by its rate, $\dot{\theta}(t)$.

Obviously, the controlled plant parameters and the fre-

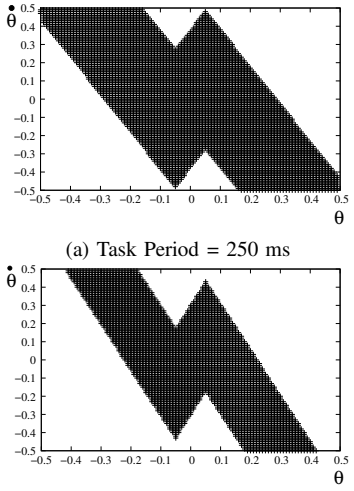
[1]Note that we are not considering noise effects in this expository example. If noise were to be included, since its amplitude is stochastic, the edges of the various subspaces would be fuzzy. Except for this, everything would be along the same lines.



(a) Task Period = 250 ms

(b) Task Period = 350 ms

Fig. 2: S_{FS} Subspace



(a) Task Period = 250 ms

(b) Task Period = 350 ms

Fig. 3: S_σ Subspace

quency with which the pendulum-control task is dispatched, together determine the subspaces. As an illustration, note how much smaller the S_N and S_{FS} subspaces are when the period is lengthened to 350 ms, from 250 ms. Related to this matter, Figure 4 indicates the fraction of the allowed subspace that is covered by the S_{FS} and the S_N subspaces. As the task period

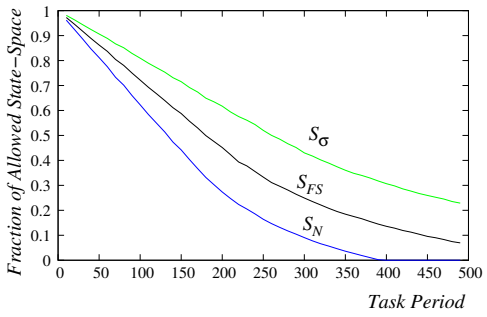


Fig. 4: Fraction of Allowed State-Space covered by Subspaces

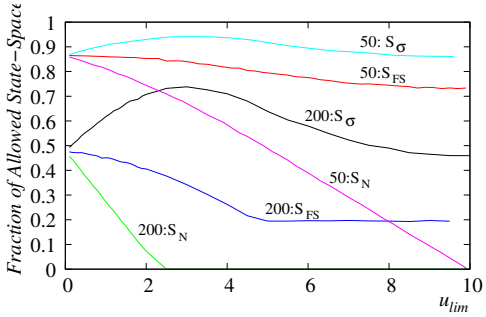


Fig. 5: Impact of u_{lim} on Subspaces

drops, the fraction of the allowed subspace that requires fault-tolerance drops as well. If the period is below about 150 ms, for example, no fault-tolerance is required for fully 50% of the allowed state-space; for the same percentage, if the period is below about 200 ms, fail-stop is a sufficient level of fault-tolerance.

Let us now consider the impact of u_{lim} on the subspaces. Figure 5 shows this for two values of control task period: 50 ms and 200 ms. An increase in u_{lim} always causes S_N to decline: the more powerful the actuator, the greater the worst-case impact of the actuator being applied incorrectly. As u_{lim} increases, S_σ initially increases, since an increase in u_{lim} widens the capability of the control (when applied correctly) to keep the plant in the allowed state-space. Beyond a certain point, however, the fact that the actuator is updated only once every period and is held fixed between updates starts to have a negative impact. This accounts for the reduction in both S_σ and S_{FS} beyond a certain point. (Note also how S_{FS} starts to decline earlier for a period of 200 ms rather than of 50 ms.)

All these simulations assumed that the task was lightweight enough that computer response time was not significant. A significant response time means that the system is essentially increasing the effects of working with outdated sensor information, and tends to further reduce the size of the S_N and S_{FS} subspaces.

It is important to point out that these fractions do NOT necessarily represent the fraction of time that the plant spends in these subspaces, since not all points in the state-space are equally likely to be visited (after all, the purpose of the control is to keep the state at some optimal value, which in our present example is $\theta = \dot{\theta} = 0$).

We now turn to the implications for task scheduling. Starting the pendulum in an initial random state $\theta \in [-0.5\theta_{lim}, 0.5\theta_{lim}]$ and $\dot{\theta} \in [-0.5\dot{\theta}_{lim}, 0.5\dot{\theta}_{lim}]$, we run a simulation of the pendulum control for up to 100 seconds or until it leaves the allowed state-space, whichever comes first. Data are collected for $\psi_N(P)$ and $\psi_{FS}(P)$; these are presented in Figure 6. As the control task period increases, an ever smaller fraction of the allowed state-space can be handled without any fault-tolerance. Finally, this fraction goes to 0. Correspondingly, the fraction of state-space requiring fault-detection (but not correction) increases at first. Beyond a certain point, however, this fraction starts to decline as an increasing fraction of instances require full fault-tolerance.

Based on this, we then calculate the utilization, $u(P)$. In Figure 7, we present the ratio of the task utilization of

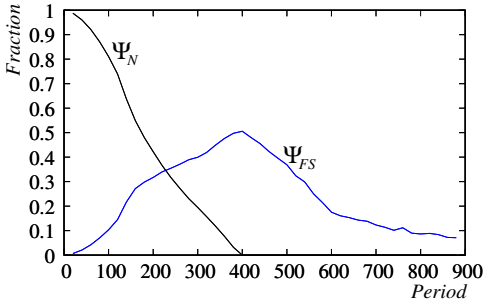


Fig. 6: Dependence of Subspace Size on Control Task Period

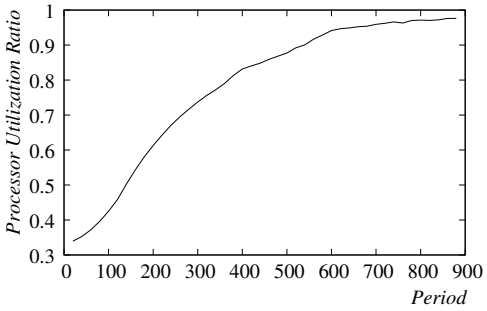


Fig. 7: Ratio of Adaptive to Non-Adaptive Utilization

the adaptive scheme to the non-adaptive one (which always uses a triplex). Note that for small periods, the utilization is substantially less for the adaptive case. As periods drop (or, equivalently, the control task dispatch frequency increases), an increasing fraction of control task instances can be handled by lower amounts of redundancy. This counteracts the increasing task loading caused by an increased dispatch frequency. This is better shown in Figure 8, which shows how the adaptive and non-adaptive task utilizations change with the period: note how much smaller is the increase in task utilization as the period decreases. (These numbers were calculated assuming $e = 1$ ms).

VI. Discussion

The traditional approach to dealing with the life-critical nature of many cyber-physical systems is the use of massive redundancy for fault-tolerance. Our aim in this paper has been to point out that the cyber-side reliability requirements must not be regarded as unvarying, but rather as a function of the current state of the controlled plant. Indeed, since one would normally expect the controlled plant to mostly operate deep within its allowed state-space, one can expect that the actual amount of redundancy required will be much lower than the full fault-masking capability required in some cases. While this does not reduce the need to use massive hardware redundancy so that it can be put into operation if the plant approaches the edge of its allowed state-space, it does allow us to reduce the computational burden for much of its period of operation. As explained above, this can significantly reduce the thermal stress on the controller, which in turn can greatly improve controller reliability. The benefits to energy consumption and the thermal stress can be quantified using standard and well-known processor energy and heatflow models.

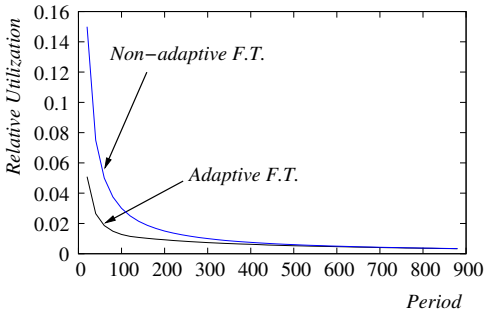


Fig. 8: Relative Utilization of Adaptive and Non-Adaptive Schemes

Altering the level of fault-tolerance during runtime is not difficult; it is just a matter of deciding how many copies to dispatch.

The computational burden is no longer now directly proportional to the task dispatch frequency. As this frequency increases (or conversely the task period decreases), the plant tends to operate deeper within its allowed state-space, thereby reducing the need for redundancy and hence the computational burden. We have also pointed out that the range of actuator force also affects the subspaces. These tradeoffs must be resolved by the designer.

Acknowledgements

This work was supported in part by the National Science Foundation under grant CNS-0931035.

References

[1] K.A. Astrom and B. Wittenmark, *Computer-Controlled Systems: Theory and Desigh*, Dover, 2011.
[2] A. Burns and A. Wellings, *Real-Time Systems and Their Programming Languages*, Academic Press, 2007.
[3] G.C. Buttazzo, G. Lipari, and L. Abeni, "Elastic Task Model for Adaptive Rate Control," *IEEE Real-Time Systems Symposium*, 1998.
[4] E. Elnozahy, R. Melhem, and D. Mosse, "Energy-Efficient Duplex and TMR Real-Time Systems," *IEEE Real-Time Systems Symposium*, 2002.
[5] R.K. Iyer, D.J. Rossetti, and M.C. Hsueh, "Measurement and Modeling of Computer Reliability as Affected by System Activity," *ACM Transactions on Computing Systems*, Vol. 4, No. 3, 1986.
[6] I. Koren and C.M. Krishna, *Fault-Tolerant Systems,* Morgan-Kaufman, 2007.
[7] C.M. Krishna and K.G. Shin, *Real-Time Systems*, McGraw-Hill, 1997.
[8] Z. Lu, W. Huang, M.R. Stan, K. Skadron, and J. Lach, "Interconnect Lifetime Prediction for Reliability-Aware Systems," *IEEE Transactions on Very Large Scale Integration Systems*, Vol. 15, No. 2, 2007.
[9] J. Meyer and L. Wei, "Analysis of Workload Influence on Dependability," *IEEE Fault-Tolerant Computing Symposium*, 1988.
[10] D. Seto, J.P. Lehoczky, L. Sha, and K.G. Shin, "On Task Schedulability in Real-Time Control Systems," *IEEE Real-Time Systems Symposium*, 1996.
[11] K.G. Shin and H. Kim, "Derivation and Application of Hard Deadlines for Real-Time Control Systems," *IEEE Transactions on Systems, Man and Cybernetics,*, Vol. 22, No. 6, Nov/Dec 1992.
[12] K.G. Shin, C.M. Krishna and Y.-H. Lee, "A Unified Method for Characterizing Real-Time Computer Controllers and Its Application," *IEEE Transactions on Automatic Control*, Vol. AC-30, No. 4, April 1985, pp. 357–366.
[13] D. Tang, R.K. Iyer, and S.S. Subramani, "Failure Analysis and Modelling of a VAX Cluster System," *Fault-Tolerant Computing Symposium*, 1990.
[14] F. Zhang, K. Szwaykowska, W. Wolf, and V. Mooney, "Task Scheduling for Control Oriented Requirements for Cyber-Physical Systems," *IEEE Real-Time Systems Symposium*, 2008.