

ORTEGA: An Efficient and Flexible Online Fault Tolerance Architecture for Real-Time Control Systems

Xue Liu, *Member, IEEE*, Qixin Wang, *Member, IEEE*, Sathish Gopalakrishnan, *Member, IEEE*, Wenbo He, *Member, IEEE*, Lui Sha, *Fellow, IEEE*, Hui Ding, and Kihwal Lee

Abstract—Fault tolerance is an important aspect in real-time computing. In real-time control systems, tasks could be faulty due to various reasons. Faulty tasks may compromise the performance and safety of the whole system and even cause disastrous consequences. In this paper, we describe On-demand Real-Time GuArD (ORTEGA), a new software fault tolerance architecture for real-time control systems. ORTEGA has high fault coverage and reliability. Compared with existing real-time fault tolerance architectures, such as Simplex, ORTEGA allows more efficient resource utilizations and enhances flexibility. These advantages are achieved through the on-demand detection and recovery of faulty tasks. ORTEGA is applicable to most industrial control applications where both efficient resource usage and high fault coverage are desired.

Index Terms—Industrial control, real-time and embedded systems, reliability and robustness.

I. INTRODUCTION

REAL-TIME and embedded systems now play an important role in our lives, with products covering a large variety of markets, such as aerospace, communication systems, automobiles, healthcare, and personal electronics. Real-time and embedded systems research is regarded as one of the next information technology (IT) frontier [1].

A real-time system has well-defined timing constraints. Different from general purpose computer systems, a real-time system is considered to function correctly only if it returns the correct result within the system-wide timing constraints [2], [3].

Reliable functioning of real-time systems is of paramount concern to the millions of users that depend on these systems everyday. However, fault and failures can occur in real-time systems. Failures can be caused by hardware malfunctions and/or

faults (e.g., electro-mechanical device), software faults (e.g., the processes/threads running on a computer), or communication medium faults.

Hardware and communication medium faults are typically tolerated by hardware redundancy [4] and techniques such as message buffering [5]. In this paper, we focus on how to tolerate software faults in real-time control systems.

One of the major deployment of real-time systems is in control applications. Real-time control system software faults can be categorized along three dimensions [6]:

- 1) resource sharing faults: The corruption of other module's code and data;
- 2) timing faults: failure to meet timing constraints;
- 3) semantic faults: producing wrong values.

Simplex [6], [7] is a software architecture which facilitates the building of dependable real-time control systems. It provides dynamic toleration of software faults. In Simplex, resource sharing faults are handled by address space protections. Timing faults are handled through real-time scheduling methods such as generalized rate-monotonic scheduling (GRMS) [8]. Semantic faults are handled through *analytical redundancy* by running redundant high-assurance controller to guard the system. Simplex has been successfully used in applications such as automated aircraft maneuvering [9] and semiconductor wafer-making [7].

However, Simplex fault tolerance architecture has two major drawbacks.

- 1) **Lack of efficiency:** In Simplex, the analytical redundant high-assurance controller runs in parallel with high-performance controller even when there are no faults. This wastes CPU cycles. In well-tested industrial applications, failures are infrequent. A parallel high-assurance controller nearly doubles the CPU execution time for a single controlled system. This makes Simplex a high-cost scheme, and hampers its application to many real-time embedded systems, which are resource-constrained.
- 2) **Lack of flexibility:** Simplex enforces the same execution period on the high-assurance controller and the high-performance controller. This simplifies the real-time scheduling analysis of systems running under Simplex. However, a control system's performance is affected by the sampling/control periods¹ used [10], [11]. As a result, in practice, different controllers may use different periods for different performance considerations. For example, when

¹In this paper, we refer "sampling/control period" as "period" for brevity.

Manuscript received October 13, 2008; revised November 25, 2008. Current version published January 21, 2009. Paper no. TII-08-10-0127.

X. Liu is with the School of Computer Science, McGill University, Montreal, QC H3A 2A7 Canada (e-mail: xueliu@cs.mcgill.ca).

Q. Wang is with the Department of Computing, Hong Kong Polytechnic University, Hong Kong, China (e-mail: wchshapp@yahoo.com).

S. Gopalakrishnan is with Department of Electrical and Computer Engineering, University of British Columbia, Vancouver, BC V6T 1Z4 Canada (e-mail: sathish@ece.ubc.ca).

W. He is with the Department of Computer Science, University of New Mexico, Albuquerque, NM 87131 USA (e-mail: wenbohe@cs.unm.edu).

L. Sha, H. Ding, and K. Lee are with Department of Computer Science, University of Illinois at Urbana-Champaign, Urbana, IL 61820 USA (e-mail: lrs@uiuc.edu; huiding@uiuc.edu; klee7@uiuc.edu).

Digital Object Identifier 10.1109/TII.2008.2010774

a fault occurs in the high-performance controller, ideally the system designer prefers to run the high-assurance controller at a faster rate to help recover from the fault and protect the system promptly. Unfortunately, this design is not possible under Simplex because it lacks the flexibility to allow the high-assurance controller and the high-performance controller to run at different periods.

In this paper, we present a new fault tolerance architecture called On-demand Real-TimE GuArD (ORTEGA). ORTEGA maintains high fault coverage and reliability as the original Simplex, meanwhile significantly improves the flexibility and resource utilization efficiency. Hence ORTEGA is applicable to a wider range of real-time control systems.

In ORTEGA, the high-assurance controller is running in an on-demand fashion. Only when a fault is detected, will the high-assurance controller be activated to replace the faulty high-performance controller. Since only one controller is active to control the system at any time instant, much resource is saved. Through careful design and schedulability analysis, ORTEGA also allows the high-assurance controller and high-performance controller to run at different rates, which greatly facilitates the flexibility in control and fault tolerance design. We implemented ORTEGA on a real-time inverted pendulum control system and carried out extensive evaluations. Results confirm the effectiveness and efficacy of ORTEGA.

The rest of this paper is organized as follows. Section II discusses related work. Section III presents the ORTEGA architecture. Section IV discusses in detail several design challenges related to ORTEGA and presents our solutions. Section V describes the implementation and evaluations of ORTEGA. Finally, Section VI summarizes the paper and points out future research directions.

II. RELATED WORK AND BACKGROUNDS

Fault tolerance is always an important issue in software systems [12]. There are well-developed fault tolerance techniques for general software systems. They can be classified into three general categories: fault masking (e.g., N-version programming [13]); backward fault recovery (e.g., recovery blocks [14] and/or checkpointing techniques in transaction-based systems [15]); and forward fault recovery (e.g., roll-forward checkpointing scheme in [16]).

A typical example of fault masking is N-Version programming [13], where multiple presumably independent program units are developed to accomplish the same task via (perhaps) different algorithms. The multiple units are executed in parallel and a majority determines the correct set of outputs.

A typical example of backward recovery techniques is recovery blocks [14], where a program unit is executed and then an acceptance test is applied. In the event that the acceptance test fails, the system is rolled back to a recovery point and an alternate program unit is executed. The sequence (execute \mapsto apply acceptance test \mapsto rollback \mapsto execute alternative) is repeated until either the acceptance test is passed or there are no additional alternates. The checkpointing technique commonly used in transaction-based database systems [15], whose primary concern is the atomicity, consistency, isolation,

and durability (ACID) properties, also belongs to backward recovery technique.

A typical example of forward fault recovery is roll-forward checkpointing scheme [16]. In this scheme, it is assumed that the organization consists of a pool of active processing modules and either a small number of spare modules or active modules with some spare processing capacity. The fault-tolerance scheme is based on checkpoints. Unlike traditional checkpointing schemes though, it requires no rollbacks for recovering from single faults. The objective of this design is to achieve performance of a triple modular redundant system using duplex system redundancy.

However, none of the above-mentioned general fault tolerance schemes are readily applicable to real-time systems. For real-time systems, as discussed in Section I, software faults can be categorized along three dimensions: 1) resource sharing faults; 2) timing faults; and 3) semantic faults. The general fault tolerance schemes above do not sufficiently take timing faults into consideration. What is more, how to *timely* recover the system from faults is an important research problem for real-time systems and has not been fully addressed in the general fault tolerance schemes.

Fault tolerance for real-time systems is an active research topic in the past decades. A comprehensive review of the recent progresses in this area can be found in [17] and [18]. Here we only list the research which is most relevant to our work.

The researchers of the FORTS project [19], [20] from University of Pittsburgh address the CPU scheduling of recovery jobs in real-time systems. They assume that all faults can be detected at the end of each periodic execution of the job, and the recovery is done by re-executing the job before its deadline. This is effective for faulty controllers in which faults are nonrecurrent. However, for control systems in which faults are recurrent or persistent, usually a recovery cannot be done by re-executing the same job within the same period. This is because most probably the re-execution will lead to the same fault again since it is still executed within the same faulty controller process. If we replace the faulty controller process with a higher-assurance controller process, the fault could be safely removed. A process replacement is different from a job re-execution and brings up many interesting research issues.

Lee *et al.* [21] develops a technique called *process resurrection* to recover a process from crash failure to meet the real-time requirements. Process resurrection is tightly coupled with the crash detection mechanism of the underlying operating system, which offers signals as event notification mechanism. Common error notification signals include SIGSEGV (segmentation faults), SIGBUS (bus error), SIGFPE (arithmetic operation error such as divide by zero), and SIGILL (execution of an illegal instruction). In process resurrection, a special signal handler is assigned for every crash related signals in order to override the default signal handler and trigger the recovery. However the fault coverage of this technique is limited to process crashes. Faults such as deadline missing, infinite loop, or deadlock cannot be handled.

Simplex [6], [7] is a software architecture which facilitates the building of dependable real-time systems. It provides dynamic toleration of system faults. The plant under control is divided into a high-assurance-control (HAC) subsystem and

a high-performance-control (HPC) subsystem. The HAC subsystem is a control software which was proven to be reliable. HAC's simple construction let the system designer leverage the power of formal methods and a rigorous development process. From the system level, high-assurance OS kernels such as certifiable runtimes are usually used in the HAC. From the application level, well-understood classical controllers designed to maximize the controlled plant's stability region is also used.

The HPC subsystem complements the conservative HAC core. From application level, an HPC can use more complex and advanced control technologies for higher control performance, including those difficult to verify, for example, neural network control [22]. These more complex and advanced control schemes may yield better control performance, however, they may have bugs or faults. From system level, COTS real-time OS and middleware designed to simplify the application development can be used in HPC. However, these software components may not be certifiable and could contain faults. In Simplex, the HAC and HPC subsystems run in parallel, but the software stays in separate processes. By using the redundant HAC to guard against possible faults in the HPC, Simplex achieves fault tolerance.

However, as we discussed in the Introduction section, Simplex is not resource-efficient and not flexible. In comparison, ORTEGA achieves the same fault coverage and functionalities as Simplex, but with significantly lower CPU resource usage and allows flexible controller implementations.

The conference version of this paper is published in [23].

III. ORTEGA ARCHITECTURE

In this section, we present the overall architecture and design considerations of ORTEGA.

A. Components Organization and Fault Recovery Procedure of ORTEGA

The architecture of ORTEGA is shown in Fig. 1. We call a plant for which ORTEGA provides fault tolerance protection an **FT-enabled plant**. In ORTEGA, there can be multiple FT-enabled plants. For each FT-enabled plant, there are three ORTEGA logical components: 1) a decision module; 2) an HPC module; and 3) an HAC module. Fig. 1 illustrates the case where there is one FT-enabled plant.

Similar to the Simplex architecture [7], the HAC subsystem is highly reliable but only provides basic performance. The HPC subsystem complements the conservative HAC core, but may contain faulty software components.

For both Simplex and ORTEGA, the decision module plays a key role in providing fault detection and recovery using its decision logic. In Simplex, at any time, both the HPC and the HAC are running; the decision module determines which control command among the two to be used for online control in each period. In contrast, ORTEGA runs the HAC only when it is necessary, i.e., only when the decision module detects an HPC fault/failure. As a result, *only one* instance of either the HAC or the HPC is running at any time. By eliminating the redundant execution of controllers, ORTEGA's run-time overhead is significantly less than that of Simplex. The saved CPU cycles can

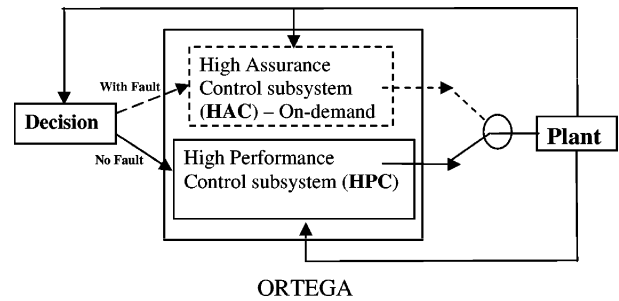


Fig. 1. Illustration of the ORTEGA fault tolerance architecture.

be used for other real-time or non-real-time tasks. It is worth noting that the on-demand execution can be implemented efficiently under ORTEGA to remove the potential overhead (in terms of delay) associated with controller thread/process start and stop.

In our implementation, the decision module uses a mutex semaphore to control which of the HAC or HPC is running. When the HPC is running well, the HAC thread simply blocks on the semaphore, i.e., the HAC is suspended. Only when a fault is detected in the HPC, the decision module will release the semaphore to let the HAC become active.

In ORTEGA, the HPC controls the plant during most of the time. In order to detect and recover HPC faults in a timely manner, the decision module should ensure that the HPC-controlled plant state stays within the HAC-established stability region. A method to determine stability regions for digital controllers will be presented in Section IV-A. When this condition is violated, the suspended HAC will be activated and takes over the plant to recover.

A second feature of ORTEGA is that the HAC and the HPC can run at different sampling rates. This allows better flexibility in designing the HAC and the HPC. For example, when the HPC is detected faulty, the carefully designed HAC can run at a faster rate to help recover the plant promptly.

In summary, the realization of ORTEGA is described as follows. On system start-up, all components are started but the HAC is blocked. As soon as a fault is detected in the HPC, the decision module deactivates the HPC and activates the HAC. Now the plant is under the control of HAC for recovery. If the type of fault is semantic (e.g., control command out of stability region), the HPC is allowed to be switched back later, after the HAC recovers the system from the previous fault and stabilizes the plant. If the failure is due to an execution error such as segmentation fault or entering infinite loop, the HPC will be restarted for later retry.

After the HAC has recovered and stabilized the plant, the active controller will switch back to the HPC for retry or performance considerations. Note unlike the recovery procedure, which must be done in a timely fashion in order for the plant state to stay within the HAC-established stability region,² the initiation time of the switch-back can be more flexible. In practice, the decision module can initiate the switch-back when the plant state is near the control set point and the CPU is in an idle

²Schedulability analysis of the controller switching will be discussed in Section IV-B.

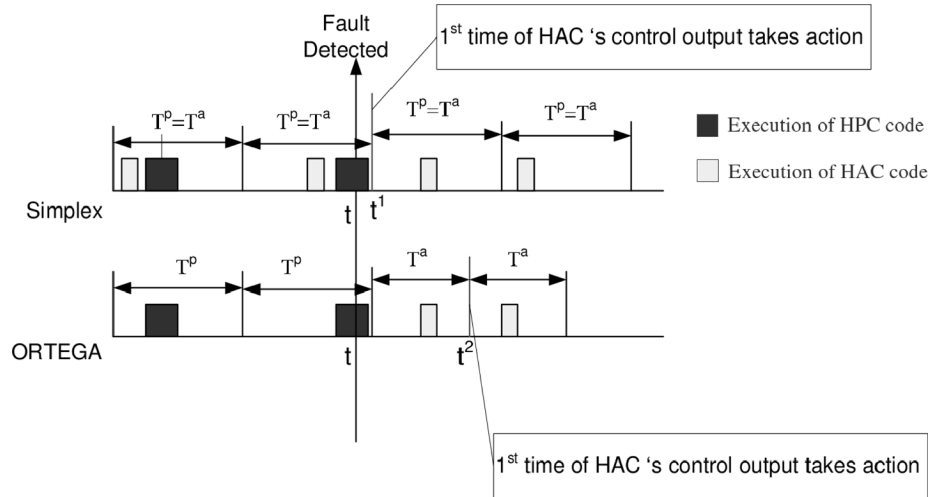


Fig. 2. Illustration of the extra delay in recovery using ORTEGA. T^p : period of HPC; T^a : period of HAC.

interval [24]. This mechanism avoids the possible mode-change problem and simplifies schedulability analysis. We will discuss the details of schedulability analysis in Section IV-B.

B. Analysis of CPU Usage Savings of ORTEGA

ORTEGA eliminates the redundant execution of controllers hence reduce the resource usage significantly. In this section, we quantify the resource savings of ORTEGA in terms of CPU usage.

We use the commonly adopted periodic task model [25] to model the control tasks. Each periodic task is denoted by τ_i . The *timing parameters* of a task τ_i is represented by the tuple (C_i, T_i) , where C_i is the worst-case execution time, and T_i is the task period. For control systems, a task's deadline is usually the same as its period, i.e., we have $D_i = T_i$.

Compared with Simplex, ORTEGA greatly reduces CPU resource usage due to the on-demand execution of the HAC. Suppose the timing parameters of an FT-enabled plant under the control of its HPC is (C^p, T^p) , while the timing parameters of the same plant under the control of its HAC is (C^a, T^a) . We use P_r to denote the percentage of time used for recovery (i.e., when HAC is active) during a total run time of T (in the unit of milliseconds).

In the original Simplex, the total CPU resource usage (in the unit of milliseconds) is

$$R_{Simplex} = (1 - P_r) \cdot \left(C^a \cdot \left\lceil \frac{T}{T^a} \right\rceil + C^p \cdot \left\lceil \frac{T}{T^p} \right\rceil \right) + P_r \cdot C^a \cdot \left\lceil \frac{T}{T^a} \right\rceil. \quad (1)$$

While in ORTEGA, the total CPU resource usage (in the unit of milliseconds) is

$$R_{ORTEGA} = (1 - P_r) \cdot C^p \cdot \left\lceil \frac{T}{T^p} \right\rceil + P_r \cdot C^a \cdot \left\lceil \frac{T}{T^a} \right\rceil. \quad (2)$$

As we can see, ORTEGA saves $(1 - P_r) \cdot C^a \cdot \lceil T/T^a \rceil$ (ms) CPU usage during a total run time of T (ms). In practical industrial applications where faults are rare, P_r is very small, thus ORTEGA saves much of the CPU resource.

C. Extra One Period Delay

The design of the on-demand execution of HAC in ORTEGA saves much of the CPU resource. However, we note due to the on-demand recovery, ORTEGA may introduce up to one period delay in the recovery procedure. In this section, we detail the cause of the delay and provide a solution.

Fig. 2 illustrates this extra delay. Suppose at time t , a fault is detected in the HPC. The upper half of Fig. 2 shows the recovery timeline under the original Simplex, while the lower half of Fig. 2 shows the recovery timeline under ORTEGA. For Simplex, since the HAC and the HPC are running in parallel, on detection of a fault, the HAC control command is immediately available and can take effect at the beginning of the next control period (i.e., at $t = t^1$). For ORTEGA, the HAC is running on-demand. On detection of a fault, the HAC needs to carry out the control computation and its control command only becomes available during the first period when the HAC begins running. So the HAC control command takes effect one period (T^a) later, i.e., at time $t = t^2$. As a result, compared with the recovery procedure using Simplex, the recovery procedure using ORTEGA will incur an extra delay up to $t^2 - t^1 = T^a$.

The extra delay incurred in ORTEGA can be *compensated* using state projection technique. The idea is illustrated in Fig. 3. At any decision time t , the decision module projects the plant state one HAC period ahead, i.e., projects $x(t + T^a)$. If the projected state is still within the stability region associated with the HAC, the HPC can still run; otherwise, the HAC will be activated to take over the plant.

It is worth noting that the flexibility of ORTEGA allows the period of the HAC (T^a) to be smaller than that of the HPC (T^p) for fast recovery. Hence the potential impact of the extra delay is small compared with the benefits gained in the increased stability region of the HAC in ORTEGA. The increased stability

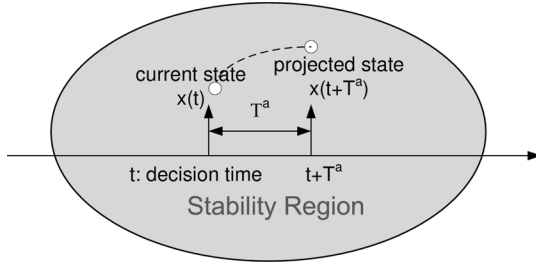


Fig. 3. Illustration of using state projection to handle the extra delay.

region allows for better fault-tolerance. This is discussed in Section IV-A. A simple and computationally-inexpensive state projection method is also presented in Section IV-A.

IV. DESIGN CHALLENGES AND SOLUTIONS

The design and implementation of ORTEGA raises several important research challenges. In this section, we discuss two most important challenges and present their solutions. The first challenge is how to determine the HAC stability region, which is used by the decision module for fault detection and recovery. The other is how to carry out schedulability analysis for ORTEGA.

A. Maximum Stability Region for Digital Controllers

ORTEGA's forward recovery scheme is based on the maximum stability region of the plant under the HAC controller. At any decision time, the decision module will check if the plant's (projected) state is still within the stability region associated with the HAC. If so, the HPC controls the plant; otherwise, the HAC will be activated to take over the control. In order to minimize the unduly restriction of the state space the HPC can use, we prefer a large stability region associated with the HAC. In this section, we propose a formal approach to determine the maximum stability region for a digitally-implemented control system.

Assume the plant to be controlled is governed by a continuous-time state space model as described in the following:

$$\frac{dx(t)}{dt} = Ax(t) + Bu(t) \quad (3)$$

where $x \in \mathcal{R}^n$ is the system state and $u \in \mathcal{R}^m$ is the control input. $A \in \mathcal{R}^{n \times n}$, $B \in \mathcal{R}^{n \times m}$ are the corresponding system matrices. Controllers are typically designed in a state feedback form, i.e., $u(t) = -Kx(t)$, where K is the corresponding controller gain.

Modern control systems are typically implemented on digital computers. Due to the digitization of continuous controllers, the sampling and control of the continuous-time system (3) is enforced at discrete time points. As a result, for the purpose of design and analysis, we need to convert the continuous-time system to its discrete-time form according to the digital implementation method used. For example, the continuous-time

system (3), when implemented using a zero-order hold with a sampling period h , can be represented as follows [26]:

$$x(k+1) = Fx(k) + Gu(k) \quad (4)$$

where $x(k) \triangleq x(kh)$ is the state of the plant at the k th sample time, and

$$F = e^{Ah} \\ G = \left(\int_0^h e^{As} ds \right) B. \quad (5)$$

Using (4) and (5), we can get a simple and computationally-inexpensive state projection method to compensate the extra delay when using ORTEGA as discussed in Section III-C. That is, at control interval k , since the decision module knows the current plant state $x(k)$ and the current control output value $u(k)$, it can calculate the predicted plant state at interval $k+1$ according to (4) and (5).

Corresponding to the continuous-time state feedback controller $u(t) = -Kx(t)$, the digitally-implemented state feedback controller is $u(k) = -Kx(k)$. By replacing the $u(k)$ term with $-Kx(k)$ in the discrete-time system (4), we get the closed-loop discretized control system as

$$x(k+1) = \bar{F}x(k) \quad (6)$$

where $\bar{F} = F - GK$.

To determine the stability of a closed-loop discretized control system as (6), we use the celebrated Lyapunov stability criteria which is summarized in the following theorem [26].

Theorem 1: A discrete-time linear time-invariant (LTI) system (6) is stable iff there exists a positive definite matrix $P > 0$ such that

$$\bar{F}^T P \bar{F} - P < 0. \quad (7)$$

For a real-life application, due to the limitation on physical plant and control actuators, there are constraints on the system states and/or control inputs. For system (4), these constraints can be represented as

$$a_i^T x \leq 1, \quad i = 1 \cdots l \quad (8)$$

$$b_j^T u \leq 1, \quad j = 1 \cdots r. \quad (9)$$

With digital controller $u(k) = -Kx(k)$, the constraints (8)–(9) can be combined and represented as a polytope (a multidimensional figure whose faces are hyperplanes) in the system's state space as follows:

$$\alpha_m^T x \leq 1, \quad m = 1, \dots, q \quad (10)$$

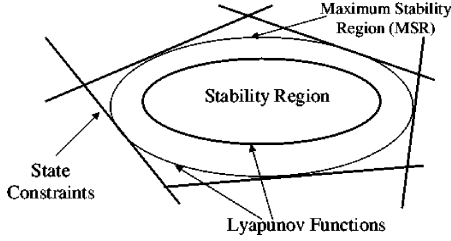


Fig. 4. Illustration of a stability region under state constraints.

where $\alpha_m^T = a_m^T$, for $m = 1, \dots, l$; $\alpha_m^T = b_j^T K$, for $m = l + 1, \dots, q$, $j = 1, \dots, r$, and $q = l + r$. The states inside the polytope are called *admissible states*, because they obey the operational constraints.

We formally define a stability region as a subset of the states within the polytope such that if the closed-loop discretized control system starts from a state within the stability region, the system states' future trajectory will always stay within the region and finally converge to the control set point.

From Lyapunov theory, we see a Lyapunov function $x^T P x$ inside the state constraint polytope represents a stability region [7], [27], [28]. Geometrically, it defines an n -dimensional ellipsoid in the n -dimensional system state constraint polytope, as illustrated in Fig. 4, where the state space is two-dimensional. An important property of a Lyapunov function is: if the system state is within the ellipsoid associated with a controller, it will always stay within the ellipsoid and finally converge to the equilibrium position (set point) under this controller.

Mathematically, for a closed-loop discretized control system (6), a *stability region* under a specific Lyapunov matrix P with state constraints is defined as the following ellipsoid:

$$S_P \triangleq \{x | x^T P x < 1\} \quad (11)$$

where P satisfies the Lyapunov stability criteria $\bar{F}^T P \bar{F} - P < 0$ [i.e., (7)] and x satisfies state constraints [i.e., (10)].

However, a Lyapunov matrix P is not unique for a given stable closed-loop discretized control system. As we discussed, in order not to unduly restrict the state space within the operational constraints, we should find the maximum stability region (MSR). To get the MSR, we first give Lemma 1.

Lemma 1: Given a discretized LTI system (6) with state constraints (10), the stability region S_P defined in (11) satisfies the constraints in (10) iff $\alpha_m^T P^{-1} \alpha_m \leq 1$, $m = 1, \dots, q$.

Proof: Please refer to [29, Lemma 4.1]. ■

Note that the area of the stability region defined in (11) is proportional to the determinant of matrix P^{-1} . Based on Lemma 1, the determination of the MSR of a closed-loop discretized control system (6) with state constraints (10) is then reduced to the following linear matrix inequality (LMI) problem [30].

1) *Problem 1:* Maximize $\log \det P^{-1}$

$$\begin{aligned} \text{s.t. :} \quad & P > 0 \\ & \bar{F}^T P \bar{F} - P < 0 \\ & \alpha_m^T P^{-1} \alpha_m \leq 1, \quad m = 1, \dots, q. \end{aligned}$$

Further, if \bar{F}^{-1} exists and let $Q \triangleq P^{-1}$, we convert the above LMI problem as the following new problem.

2) *Problem 2:* Maximize $\log \det Q$

$$\begin{aligned} \text{s.t. :} \quad & Q > 0 \\ & Q \bar{F}^T - \bar{F}^{-1} Q < 0 \\ & \alpha_m^T Q \alpha_m \leq 1, \quad m = 1, \dots, q. \end{aligned}$$

Problem 2 is a MAXDET problem [30]. It is readily-solvable using numerical software packages such as *sdp sol* [31] or *YALMIP* [32]. Note that the maximum stability region and its solution via LMI formulations (i.e., Problems 1 and 2) are different from those presented in [29]. Here we are dealing with discretized system under digital controllers, while in [29], the authors were dealing with continuous system under continuous-time controllers.

The resulting $P = Q^{-1}$ from the MAXDET problem above defines the maximum stability region $\{x | x^T P x < 1\}$ in the system state polytope (see Fig. 4).

Using the method presented above, when the continuous-time system model, the underlying continuous-time controller and its control loop period are given, a system designer can calculate the maximum stability region of the corresponding closed-loop discretized control system offline.

It is obvious that the maximum stability region (MSR) is a function of the control loop period T . Hence we denote the MSR for a plant under a digital controller with respect to the control loop period T as $MSR(T)$. We further use $A(T)$ to denote the area of $MSR(T)$. Since $MSR(T)$ encloses the admissible states of the system to keep it stable with respect to the control loop period T , we call $A(T)$ the **stability index** of the closed-loop discretized control system under control loop period T .

Now, let us look at a real-life control example. Consider a controlled inverted pendulum. The continuous-time system model of the inverted pendulum is shown in the following:

$$\frac{dx(t)}{dt} = Ax(t) + Bu(t) \quad (12)$$

where the system matrices are

$$A = \begin{pmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & -2.7528 & -10.9526 & 0.0043 \\ 0 & 28.5812 & 24.9179 & -0.0441 \end{pmatrix} \quad (13)$$

$$B = (0, 0, 9432, -4.4385)^T. \quad (14)$$

The designed high-assurance controller for the inverted pendulum uses state feedback control [26] as shown in the following:

$$u(t) = -(-5.7807, -42.2087, -14.0953, -8.6016)x(t). \quad (15)$$

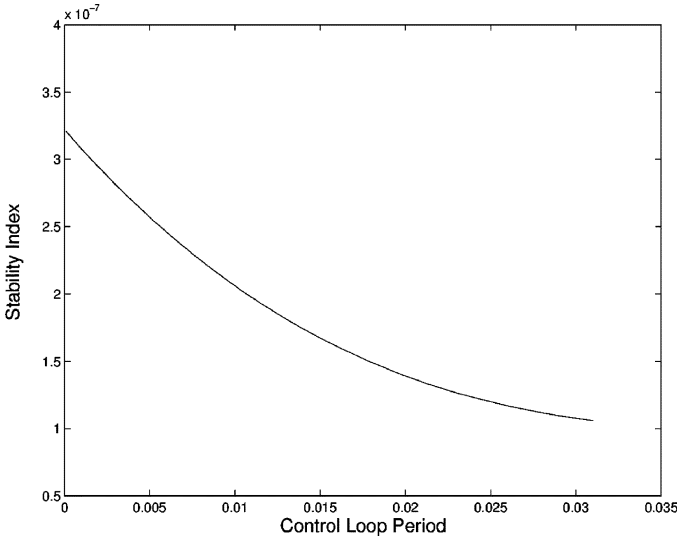


Fig. 5. Stability index versus control loop period for the inverted pendulum control system.

Here the plant state is represented as $x(t) = (x_1(t), x_2(t), x_3(t), x_4(t))^T$. $x_1(t)$ is the inverted pendulum's track position at time t ; $x_2(t)$ is the inverted pendulum's angle position at time t ; $x_3(t)$ is the inverted pendulum's track position velocity at time t ; and $x_4(t)$ is the inverted pendulum's angle velocity at time t .

We implemented the method presented above to determine the maximum stability region when varying the control loop period from 0.001 (seconds) to 0.032 (seconds). Fig. 5 illustrates the corresponding stability index versus control loop period for the inverted pendulum system.

The physical meaning of the decreasing shape of the stability index shown in Fig. 5 is clear: as the control loop period decreases (i.e., controller runs faster), the system becomes more stable, hence the stability index $A(T)$ increases. On the other hand, as the control loop period increases (i.e., controller runs slower), the plant becomes less stable, hence the stability index $A(T)$ decreases. So in order to have a larger maximum stability region, the HAC should run faster. This is supported in ORTEGA as the HAC can run at a rate faster than that of the HPC, as long as system schedulability is guaranteed. We provide the schedulability analysis of ORTEGA in next section.

B. Schedulability Analysis for ORTEGA Fault Tolerance Architecture

The new ORTEGA fault tolerance architecture saves CPU resource compared with Simplex. The saved CPU resource could be used for other real-time tasks. In this section, we discuss the schedulability analysis of ORTEGA together with these real-time tasks.

Consider an FT-enabled plant PL under the protection of ORTEGA. Let us denote the task when PL is being controlled by the HPC as τ^p and denote the task when PL is being controlled by the HAC as τ^a . Their timing parameters are (C^p, T^p) and (C^a, T^a) , respectively. When a fault is detected in the HPC,

the decision module will issue a recovery request (RR) to initiate the recovery procedure. As a result, τ^p will be aborted and the new task τ^a will be activated for recovery.

Using similar notations, the decision module is modeled as real-time task τ_d^p when PL is controlled under the HPC and modeled as task τ_d^a when PL is controlled under the HAC. In ORTEGA, when controller switches, the decision module's period also switches accordingly. So tasks τ^p (i.e., controller task) and τ_d^p (i.e., decision module task) have the same phase and period. From a schedulability analysis perspective, τ^p and τ_d^p can be modeled as one single task $\tilde{\tau}^p$. Its execution time is $\tilde{C}^p = C^p + C_d^p$, its period is $\tilde{T}^p = T^p = T_d^p$. Similarly tasks τ^a and τ_d^a can also be modeled as one single task $\tilde{\tau}^a$, where $\tilde{C}^a = C^a + C_d^a$, $\tilde{T}^a = T^a = T_d^a$.

As a result, when scheduled together with other real-time tasks, the decision modules and controllers for a single FT-enabled plant can be modeled as an abstract task $\tilde{\tau}$. It has two sub-tasks $\tilde{\tau}^p$ and $\tilde{\tau}^a$, where $\tilde{\tau}^p$ is running when the plant is under the control of the HPC and $\tilde{\tau}^a$ is running when the plant is under the control of the HAC. Task $\tilde{\tau}$ is called an **FT-enabled task**. This model abstraction simplifies the following schedulability analysis. In the discussions below, without confusion, we use τ_k to denote the combined decision and control task for an FT-enabled plant k together with other real-time tasks.

1) *Task Model and Definitions:* Assume there are a total of N real-time tasks, τ_1, \dots, τ_N , running on the CPU. They are ordered in the sequence of their priorities, with τ_1 having the highest priority and τ_N having the lowest priority. Among them, tasks τ_k , ($k \in \mathcal{FT}$) are the FT-enabled tasks, $\mathcal{FT} \subseteq \{1, \dots, N\}$ is the set of all FT-enabled tasks. Each τ_k , ($k \in \mathcal{FT}$) is composed of two subtasks: τ_k^p represents the combined decision and control task when the plant is being controlled by its HPC; τ_k^a represents the combined decision and control task when the plant is being controlled by its HAC.

Definition 1: Given the task set $\{\tau_i\}, i \in \{1, \dots, N\}$, among which tasks τ_k , ($k \in \mathcal{FT}$) are FT-enabled tasks, $\mathcal{FT} \subseteq \{1, \dots, N\}$. If the task set $\{\tau_i\}$ is schedulable under the ORTEGA task recovery and switch-back scheme with random failures, it is called **FT-schedulable**.

We will focus on the schedulability analysis when there is one FT-enabled task τ_k in the task set, i.e., $\mathcal{FT} = \{k\}, 1 \leq k \leq N$. In ORTEGA, when a fault in the HPC is identified, the decision module will issue an RR. At the same time, task τ_k^p will be aborted, and τ_k^a will be activated for recovery. This potentially raises a mode-change problem [33], since tasks τ_k^p and τ_k^a may have different timing parameters. We need to carry out schedulability analysis to guarantee that the tasks are schedulable during the recovery. Similarly, after the recovery, τ_k^a can be switched back to τ_k^p for retry or performance considerations. We also need to guarantee the tasks are still schedulable with the switch-back.

If a task set is schedulable under the ORTEGA task recovery, it is called *FT-schedulable for recovery*. If a task set is schedulable under the ORTEGA task switch-back, it is called *FT-schedulable for switch-back*.

Since the schedulability analysis methods are similar for both the recovery procedure and the switch-back procedure, we only present the analysis for recovery, i.e., under the mode-change from τ_k^p to τ_k^a in the following discussion.

2) *Schedulability Analysis for ORTEGA Recovery Scheme:* When the RR is initiated, the recovery procedure drives the system from the old operating mode (abbreviated as “old-mode”) to the new operating mode (abbreviated as “new-mode”). The plant is controlled by the HPC in the old-mode and is controlled by the HAC in the new-mode, respectively. Tasks in the old-mode are called old-mode tasks; while tasks in the new-mode are called new-mode tasks. In the following discussions, when we refer to task τ_k , we mean subtask τ_k^p when the context is in old-mode and we mean subtask τ_k^a when the context is in new-mode.

The switching from τ_k^p to τ_k^a imposes transitional scheduling overhead, which may make the whole task set unschedulable. In real-time analysis, this is called the mode-change problem [34], [33]. In order to determine whether the task set is FT-schedulable under potential recoveries, we develop a schedulability test. It is based on a variant of the proposal presented in [33] by Real *et al.* with simplifications. The basic idea here is to consider each real-time task τ_i which may be affected by the transitional scheduling overhead in either old-mode or new-mode. Then we perform an offline response time analysis to test if it is schedulable in both modes.

When there is only one FT-enabled task τ_k in the task set, it is the only task which may initiate the mode-change. For any other task τ_i ($i \neq k$), the task release pattern will not change before and after the mode-change. We call these tasks unchanged tasks.

Our schedulability test is divided into three parts.

Schedulability of Steady State Task Set: Let us define two task sets as follows:

$$\mathcal{S}_o \triangleq \{(C_1, T_1), \dots, (C_k^p, T_k^p), \dots, (C_N, T_N)\} \quad (16)$$

$$\mathcal{S}_n \triangleq \{(C_1, T_1), \dots, (C_k^a, T_k^a), \dots, (C_N, T_N)\}. \quad (17)$$

They are the old-mode and new-mode steady state task sets. We first test if both task sets are schedulable. This can be done using the standard response time analysis [25], [35], [36] for each task in the set. By solving the recurrence equations in the standard response time analysis, we get $R_i^{\mathcal{S}_o}$ (and $R_i^{\mathcal{S}_n}$), the maximal response time of task τ_i under the old-mode (and under the new-mode) in steady state. It should be smaller than or equal to the task deadline (period) T_i for schedulability. If the recurrence value exceeds the period, then task τ_i is unschedulable.

Schedulability of Old-Mode Tasks With Transitional Scheduling Overhead: In the old-mode, first, it is obvious to see that for each task τ_i ($i < k$), its schedulability is not affected by the mode-change. This is because they have higher priorities than the FT-enabled task τ_k .

Secondly, we consider task τ_k^p , which is the task aborted upon the RR. It cannot be affected by the new-mode task τ_k^a during

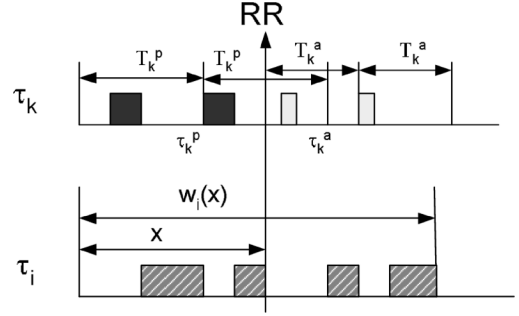


Fig. 6. Illustration of mode-change incurred by the recovery.

the old-mode, hence its schedulability has already been covered by the steady state schedulability analysis above [i.e., case (I)].

Lastly, we consider every task τ_i ($i > k$), which is an old-mode task who has lower priority than task τ_k . In order to account for the worst case phasing in the schedulability test, the RR is assumed to occur x time units after the task’s activation. We also define a temporal window $w_i(x)$, starting at the activation of old-mode task τ_i and finishing when τ_i completes. Fig. 6 illustrates x and $w_i(x)$.

Now we carry out the response time analysis by quantifying the possible interferences τ_i may receive from other tasks.

Firstly, in the old-mode, τ_i can be affected by the old-mode aborted task τ_k^p . The total worst-case interferences from τ_k^p is

$$\left\lceil \frac{x}{T_k^p} \right\rceil C_k^p + \min \left(x - \left\lfloor \frac{x}{T_k^p} \right\rfloor T_k^p, C_k^p \right). \quad (18)$$

Secondly, τ_i can also be affected by the new released task τ_k^a . The worst-case interferences from τ_k^a is

$$\left\lceil \frac{w_i(x) - x}{T_k^a} \right\rceil_0 C_k^a \quad (19)$$

where $\lceil y \rceil_0 \triangleq \max\{\lceil y \rceil, 0\}$.

Finally, τ_i can also be affected by the unchanged tasks who have higher priorities than it. These interferences are

$$\sum_{j < i, j \neq k} \left\lceil \frac{w_i(x)}{T_j} \right\rceil C_j. \quad (20)$$

By summing up all the sources of interferences, we got the total size of window

$$w_i(x) = C_i + \left\lceil \frac{x}{T_k^p} \right\rceil C_k^p + \min \left(x - \left\lfloor \frac{x}{T_k^p} \right\rfloor T_k^p, C_k^p \right) + \left\lceil \frac{w_i(x) - x}{T_k^a} \right\rceil_0 C_k^a + \sum_{j < i, j \neq k} \left\lceil \frac{w_i(x)}{T_j} \right\rceil C_j. \quad (21)$$

Solution to this equation is obtained by performing a recurrence calculation on $w_i(x)$ to find the smallest positive integer that satisfies it, just as the ordinary response time analysis. Then

the worst-case response time of the old-mode task τ_i is obtained as the duration of the largest time window of $w_i(x)$, i.e.,

$$R_i = \max(w_i(x)), \quad \forall x \in [0, R_i^{So}). \quad (22)$$

In practice, when calculating R_i using (22), we only care about *significant* values of x . Those x produce changes in the values of the *ceiling* and *floor* functions in (21).

After R_i is obtained, it will be compared with τ_i 's deadline to determine if τ_i is schedulable or not.

Schedulability of New-Mode Tasks With Transitional Scheduling Overhead: In the new-mode, first, it is obvious to see that for each task i ($i < k$), its schedulability is not affected by the mode-change. So this case has already been covered in the steady state schedulability test [i.e., case (I)].

Secondly, we consider task τ_k^a , which is the newly released task at RR. It can not be affected by the old-mode task τ_k^p in the new-mode, hence its schedulability has also been covered by the steady state schedulability analysis [i.e., case (I)].

Finally, we consider every task τ_i ($i > k$), which is a new-mode unchanged task who has lower priority than task τ_k . As with the analysis of old-mode tasks, we define a temporal window w_i to enclose the response time.

In the new-mode, τ_i can be affected by the new task τ_k^a . The interference is

$$\left\lceil \frac{w_i}{T_k^a} \right\rceil C_k^a. \quad (23)$$

Also, it can be affected by the unchanged tasks whose priorities are higher. These interferences are

$$\sum_{j < i, j \neq k} \left(\left\lceil \frac{w_i}{T_j} \right\rceil C_j \right). \quad (24)$$

Summing up all the sources of interferences, we get the window size

$$w_i = C_i + \left\lceil \frac{w_i}{T_k^a} \right\rceil C_k^a + \sum_{j < i, j \neq k} \left(\left\lceil \frac{w_i}{T_j} \right\rceil C_j \right). \quad (25)$$

Again, recurrence calculation of w_i should be performed until its convergence or when the recurrence value exceeds the task's deadline. Then the response time for task τ_i in the new-mode can be obtained as

$$R_i = \max(w_i, R_i^{Sn}). \quad (26)$$

The whole task set $\{\tau_1, \dots, \tau_N\}$ is schedulable even with random recoveries if it passes the schedulability tests (I-III), i.e., FT-schedulable for recovery. Similar tests can be done to ensure



Fig. 7. Inverted pendulum (IP). θ is the angular deviation from the vertical position.

the task set is still schedulable with the switch-back. Then we can determine if the task set is FT-schedulable.

We implemented the schedulability test presented above. Below we give a numerical example.

3) *Example 1:* Consider three tasks τ_1 , τ_2 , and τ_3 . Tasks τ_1 and τ_3 are ordinary real-time tasks with timing parameters (2, 4) and (3, 30), respectively. Task τ_2 is an FT-enabled task protected under ORTEGA. τ_2 's timing parameters for the HPC are $(C_2^p, T_2^p) = (2, 8)$. We vary the execution time of the HAC (C_2^a) for different values to simulate different versions of the HAC design. For each C_2^a , we find the smallest period of the HAC (T_2^p) such that the whole task set is still schedulable based on the schedulability analysis presented above. We denote such a period for the HAC as T_2^{*a} . The results are shown in the following:

- if $C_2^a = 2.0$, we have $T_2^{*a} = 6.5 < T_2^p$;
- if $C_2^a = 1.5$, we have $T_2^{*a} = 4.5 < T_2^p$;
- if $C_2^a = 1.0$, we have $T_2^{*a} = 3.0 < T_2^p$;
- if $C_2^a = 0.5$, we have $T_2^{*a} = 2.5 < T_2^p$.

We have two observations here. First, the HAC's period can be smaller than the HPC's period, in the mean time, all real-time tasks under ORTEGA are still schedulable. Second, as C_2^a decreases, T_2^{*a} has significantly dropped. This means the HAC can run at a much faster rate than that of the HPC during the recovery when fault occurs.

V. IMPLEMENTATION AND EVALUATION OF ORTEGA

We implemented and evaluated ORTEGA on an *inverted pendulum* (IP) testbed [37]. Fig. 7 explains the concept of IP. An IP is a metal rod with one end hinged onto a cart and the other end free. The cart moves back and forth along the x -axis to keep the rod (i.e., IP) stand up vertically, i.e., to maintain angular deviation θ around 0 degree (see Fig. 7).

IP is inherently unstable. A couple of missed control outputs is enough to make it fall, even when the angular deviation θ and angular velocity $\dot{\theta}$ are small. As a result, the fault detection and recovery must be carried out in a timely and predictable manner [21].

In our testbed, ORTEGA runs on a computer with a Pentium II 350-MHz processor, a 66-MHz memory bus, and 32 KB of level 1 cache memory on chip. The IP uses a Quadrature optical encoder interface for sensing input and a digital to analog converter for control output.

We implemented ORTEGA in C, and run ORTEGA on Linux kernel version 2.4.18-3 with RT scheduling and kernel preemption enhancements. Our ORTEGA uses rate monotonic scheduling [38], a fixed priority scheduling scheme widely supported

TABLE I
EXECUTION STATISTICS FOR THE NONFAULTY HPC AND THE HAC

Controller	Average Execution Time (μs)	Variance of Execution Time	Minimum Execution Time (μs)	Maximum Execution Time (μs)
HPC	2.6705	0.02181	2.3571	3.2857
HAC	1.1060	0.005812	0.9429	1.6371

by contemporary systems and standards, including the POSIX real-time extension [39].

In our tested, we use a field-tested state feedback controller as the HAC. It was designed to be simple so it can be easily checked that there is no bug. For the HPC, users can upload their arbitrary controllers dynamically to our testbed. These controllers may contain faults or bugs. In fact, we selected many faulty controllers as HPCs to evaluate the fault tolerance performance of ORTEGA. We also provide a default nonfaulty HPC. It serves two purposes. First, it is used as a benchmark in the evaluation of the CPU resource savings under ORTEGA (Section V-A). Second, it can be used as a controller template for users to insert various bugs for testing (Section V-B).

The following are our evaluation results.

A. CPU Resource Savings of ORTEGA

In order to measure the CPU savings of ORTEGA compared with the original Simplex, we collected the controllers' running temporal data from the testbed.

We accurately measure the nonfaulty HPC and the HAC controller's execution times by using the *rdscl()* call [40]. Table I shows the mean, variance, minimum, and maximum of execution times of the HPC and the HAC, respectively.

When the HPC and the HAC are running at the same frequency, as we can see from Table I, the percentage of CPU usage savings (in terms of controllers' execution times) using ORTEGA compared with that of using Simplex [cf. (1) and (2)] is

$$\frac{(1 - P_r) \cdot 1.1060}{(1 - P_r) \cdot (2.6705 + 1.1060) + P_r \cdot (1.1060)} \cong \frac{1.1060}{2.6705 + 1.1060} = 29.29\%$$

where P_r is the percentage of time used for recovery, which is assumed to be small.

ORTEGA can run HPC and HAC at different rates. For example, in our tests, the HPC is running at 33.3 Hz and the HAC is running at 50 Hz. Consider the difference in control frequencies of the HPC and the HAC, the percentage of CPU usage savings (in terms of controllers' execution times) using ORTEGA compared to Simplex is

$$\frac{(1 - P_r) \cdot 1.1060 \cdot 50}{(1 - P_r) \cdot (2.6705 \cdot 33.3 + 1.1060 \cdot 50) + P_r \cdot (1.1060 \cdot 50)} \cong \frac{1.1060 \cdot 50}{2.6705 \cdot 33.3 + 1.1060 \cdot 50} = 38.34\%.$$

B. Fault Tolerance Performance Under ORTEGA

For any fault tolerance scheme, two important evaluation criteria are false negative (i.e., type II errors) and false positive (i.e., type I errors) ratio [41]. For a good fault tolerance scheme, in order to minimize false negative ratio, the scheme needs to tolerate as many faults as possible; in order to minimize false positive ratio, a scheme should minimize the situation of classifying nonfaulty behaviors as faulty.

In order to evaluate the fault tolerance capabilities of ORTEGA, we conducted extensive tests on our IP testbed. The testing procedure uses various nonfaulty and faulty HPCs to control the inverted pendulum, and checks whether ORTEGA lets the nonfaulty HPCs to take control, and fixes the faulty HPCs by switching to the HAC.

The way to introduce faulty HPCs is by inserting different faults/bugs into the default nonfaulty HPC controller. Below we list a subset of the benchmark faults/bugs tested against ORTEGA. Not all faults/bugs of the benchmark are listed due to space limit of this paper. The faults/bugs benchmark is not intended to be complete, rather it is intended to show the broad range of faults/bugs that ORTEGA can tolerate. According to our results, ORTEGA tolerates all the faults/bugs in the benchmark. When the HPC controlled system fails the decision module's test, ORTEGA replaces the HPC with the HAC. Hence the inverted pendulum can keep running without falling down. It is worth noting that some of the tests (such as the "tricky design bug" discussed below) can also evaluate ORTEGA's false positive ratio. A video demo of all the tests listed here is available at <http://www.cs.mcgill.ca/~xueliu/Demos/>. The video demo shows the details of the dynamic movements of the cart and inverted pendulum before and after a bug was introduced. It captures the negative effect imposed by each bug on the pendulum, illustrates the transition when the HAC was activated, and shows how the HAC can keep the pendulum from falling down.

We also use other faulty HPCs to test the robustness of ORTEGA. The tested faults/bugs include (but are not limited to) bang-bang (BangBang) bug, divided by zero (DivZero) bug, infinite loop (InfLoop) bug, maximum control output (MaxCtrl) bug, nonperforming (NonPerf) bug, positive feedback control (PosFdbk) bug, and tricky designer (TrickyDsg) bug (see [23] for more details).

For each of the above bugs, we tested ten trials using ORTEGA and another ten trials using Simplex. In each trial, the corresponding bug is injected at time 0 s, and we track the IP angular deviation (θ , see Fig. 7) to evaluate the performance of ORTEGA and Simplex.

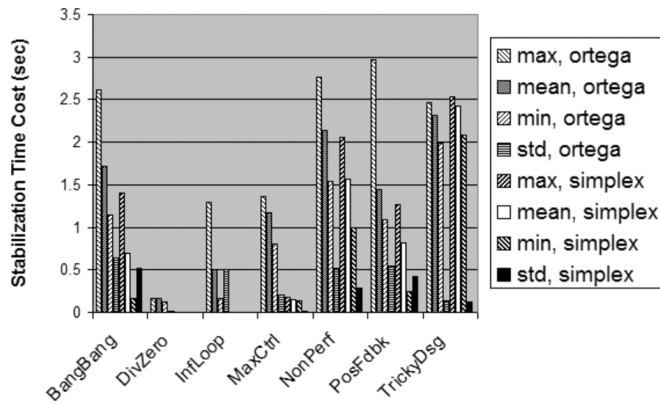
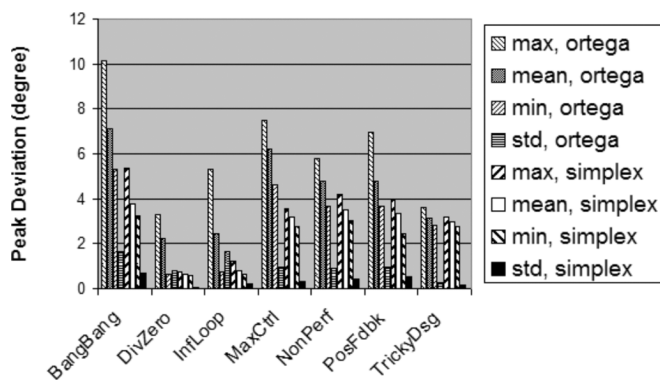


Fig. 8. Comparison on stabilization time cost.

Fig. 9. Comparison on peak deviation (i.e., $\max |\theta|$ after bug injection).

Figs. 8 and 9 compare the statistics between ORTEGA and Simplex using the results of our 140 trials. Fig. 8 compares ORTEGA and Simplex's performance on stabilization time cost. Empirically, we can regard the IP to be stabilized when $|\theta|$ never exceeds 1.5 degree. Stabilization time cost refers to the time cost since the bug takes place till the IP is stabilized. Fig. 9 compares ORTEGA and Simplex's performance on peak deviation, i.e., the maximum of $|\theta|$ after the bug takes place.

According to Figs. 8 and 9, ORTEGA can effectively tolerate all the faults/bugs in its 70 trials, and its performance is not unacceptably worse than Simplex. Given the 38.34% saving on CPU utilization compared to Simplex (see Section V-A), ORTEGA qualifies as a useful scheme.

VI. CONCLUSIONS

In this paper we presented a new fault tolerance architecture, ORTEGA, for real-time control systems. Similar to Simplex, ORTEGA is reliable and achieves high fault coverage. Compared with Simplex, ORTEGA has advantages including that it allows more efficient resource utilizations and enhances flexibility. This is achieved by running the high-assurance controller in an on-demand fashion instead of running in parallel. We implemented ORTEGA on an inverted pendulum control testbed and carried out extensive benchmark tests to evaluate the performance of ORTEGA. Results demonstrate the efficiency and effectiveness of ORTEGA. We believe ORTEGA is a promising real-time fault tolerance architecture and is applicable to a wider

range of industrial applications where both efficient resource usage and high fault coverage are desired.

In the future, we plan to make ORTEGA available to the industry and test its performance in more complex real-world deployments.

REFERENCES

- [1] National Research Council, *Embedded, Everywhere: A Research Agenda for Networked Systems of Embedded Computers*, National Academies Press, 2001.
- [2] J. A. Stankovic, "Real-time computing systems: The next generation," in *Tutorial: Hard Real-Time Systems*, J. A. Stankovic and K. Ramamritham, Eds. Washington, DC: IEEE Comput. Soc., 1998, pp. 14–37.
- [3] G. Buttazzo, *Hard Real-Time Computing Systems: Predictable Scheduling Algorithms and Applications*, 2nd ed. New York: Springer, 2005.
- [4] F. Cristian, "Understanding fault-tolerant distributed systems," *Commun. ACM*, vol. 34, no. 2, pp. 56–78, 1991.
- [5] S. Graham, G. Baliga, and P. R. Kumar, "Issues in the convergence of control with communication and computing: Proliferation, architecture, design, services, and middleware," in *Proc. 43rd IEEE Conf. Decision and Control*, Dec. 2004.
- [6] L. Sha, "Dependable system upgrade," in *Proc. IEEE Real-Time Systems Symp. (IEEE Computer Society; RTSS '98)*, 1998, p. 440.
- [7] D. Seto, B. H. Krogh, L. Sha, and A. Chutinan, "Dynamic control system upgrade using the simplex architecture," *IEEE Control System Mag.*, vol. 18, no. 4, pp. 72–80, Aug. 1998.
- [8] L. Sha, R. Rajkumar, and S. Sathaye, "Generalized rate monotonic scheduling theory: A framework of developing real-time systems," *Proc. IEEE*, vol. 82, no. 1, pp. 68–82, Jan. 1994.
- [9] D. Seto, E. Ferreira, and T. Marz, Case Study: Development of a Baseline Controller for Automatic Landing of an F-16 Aircraft Using Lmis, Carnegie Mellon Univ., Softw. Eng. Inst., 2000, Tech. Rep. CMU/SEI-99-TR-020.
- [10] D. Seto, J. P. Lehoczky, L. Sha, and K. G. Shin, "On task schedulability in real-time control system," in *Proc. 17th IEEE Real-Time Systems Symp.*, 1996, pp. 13–21.
- [11] L. Sha, X. Liu, M. Caccamo, and G. Buttazzo, "Online control optimization using load driven scheduling," in *Proc. Conf. Decision and Control*, Sydney, Australia, 2000.
- [12] *Software Fault Tolerance (Trends in Software, No. 3)*, M. Lyu, Ed. New York: Wiley, 1995.
- [13] A. Avizienis, "The methodology of n-version programming," in *Software Fault Tolerance*, M. R. Lyu, Ed. New York: Wiley, 1995.
- [14] B. Randell and J. Xu, "The evolution of the recovery block concept," in *Software Fault Tolerance*. New York: Wiley, 1995.
- [15] R. Ramakrishnan and J. Gehrke, *Database Management Systems*, 3rd ed. New York: McGraw-Hill, 2003.
- [16] D. Pradhan and N. Vaidya, "Roll-forward checkpointing scheme: A novel fault-tolerant architecture," *IEEE Trans. Comput.*, vol. 43, no. 10, pp. 1163–1174, Oct. 1994.
- [17] P. M. Melliar-Smith and L. E. Moser, "Progress in real-time fault tolerance," in *Proc. 23rd IEEE Int. Symp. Reliable Distributed Systems (SRDS'04)*, Florianopolis, Brazil, Oct. 2004, pp. 109–111.
- [18] K. H. Kim, "Slow advances in fault-tolerant real-time distributed computing," in *Proc. 23rd IEEE Int. Symp. Reliable Distributed Systems (SRDS'04)*, Florianopolis, Brazil, Oct. 2004, pp. 106–108.
- [19] H. Aydin, R. Melhem, and D. Mosse, "Optimal scheduling of imprecise computation tasks in the presence of multiple faults," in *Proc. 7th Int. Conf. Real-Time Computing Systems and Applications (RTCSA00)*, 2000.
- [20] H. Aydin et al., "Tolerating faults while maximizing reward," in *Proc. 12th Euromicro Conf. Real-Time Systems (Euromicro'00)*, Stockholm, Sweden, 2000.
- [21] K. Lee and L. Sha, "Process resurrection: A fast recovery mechanism for real-time embedded systems," in *Proc. 11th IEEE Real-Time and Embedded Technology and Applications Symp.*, San Francisco, CA, 2005.
- [22] L. Sha, "Using simplicity to control complexity," *IEEE Software*, vol. 18, no. 4, pp. 20–28, Jul.–Aug. 2001.
- [23] X. Liu, H. Ding, K. Lee, Q. Wang, and L. Sha, "ORTEGA: An efficient and flexible software fault tolerance architecture for real-time control systems," in *Proc. 20th Euromicro Conf. Real-Time Systems (ECRTS 2008)*, 2008.
- [24] K. Tindell and A. Alonso, "A Very Simple Protocol for Mode Changes in Priority Preemptive Systems," Tech. Rep., Univ. Politecnica de Madrid, Madrid, Spain, 1996.
- [25] J. Liu, *Real-Time Systems*. Englewood Cliffs, NJ: Prentice-Hall, 2000.
- [26] K. J. Astrom and B. Wittenmark, *Computer-Controlled Systems: Theory and Design*, 3rd ed. Reading, MA: Addison-Wesley, 1994.

- [27] M. Roozbehani, A. Megretski, and E. Feron, "Safety verification of iterative algorithms over polynomial vector fields," in *Proc. 45th IEEE Conf. Decision and Control*, 2006, pp. 6061–6067.
- [28] M. Roozbehani *et al.*, "Convex optimization proves software correctness," in *Proc. 2005 Amer. Control Conf.*, 2005, vol. 2, pp. 1395–1400.
- [29] D. Seto and L. Sha, An Engineering Method for Safety Region Development, CMU SEI, 1999, Tech. Rep. 18.
- [30] S. Boyd, L. E. Ghaoui, E. Feron, and V. Balakrishnan, *Linear Matrix Inequalities in System and Control Theory*. Philadelphia, PA: SIAM, 1994.
- [31] S.-P. Wu and S. Boyd, "Sdpsol: A parser/solver for sdp and maxdet problems with matrix structure," in *Recent Advances in LMI Methods for Control*, L. E. Ghaoui and S.-I. Niculescu, Eds. Philadelphia, PA: SIAM, 1999.
- [32] J. Lofberg, "YALMIP: A toolbox for modeling and optimization in matlab," in *Proc. 2004 IEEE Int. Symp. Computer Aided Control Systems Design*, Taipei, Taiwan, 2004.
- [33] J. Real and A. Crespo, "Mode change protocols for real-time systems: A survey and a new proposal," *Real-Time Syst.*, vol. 26, pp. 161–197, 2004.
- [34] P. Pedro and A. Burns, "Schedulability analysis for mode changes in real-time systems," in *Proc. 10th Euromicro Workshop Real-Time Systems*, Berlin, Germany, 1998.
- [35] M. Joseph and P. Pandya, "Finding response times in a real-time system," *Comput. J.*, vol. 29, no. 5, pp. 390–395, 1986.
- [36] N. Audsley, A. Burns, M. Richardson, K. Tindell, and A. Wellings, "Applying new scheduling theory to static priority preemptive scheduling," *Softw. Eng. J.*, vol. 8, no. 5, pp. 284–292, 1993.
- [37] Quanser, 2008. [Online]. Available: <http://www.quanser.com>.
- [38] C. L. Liu and J. W. Layland, "Scheduling algorithms for multiprogramming in hard real time environment," *J. ACM*, vol. 20, no. 1, pp. 40–61, 1973.
- [39] Real-Time System Services Working Group, IEEE STD 1003.1, 1998, 1996 ed. .
- [40] M. Rajagopalan, S. K. Debray, M. A. Hiltunen, and R. D. Schlichting, "Profile-directed optimization of event-based programs," in *Proc. SIGPLAN '02 Conf. Programming Language Design and Implementation (PLDI 02)*, 2002.
- [41] P. Pelliccione, N. Guelfi, H. Muccini, and A. Romanovsky, *Software Engineering and Fault Tolerance*. Singapore: World Scientific, 2007.



Xue Liu (M'06) received the B.S. degree in applied mathematics and the M.Eng. degree in control theory and applications from Tsinghua University, Beijing, China, in 1996 and 1999, respectively, and the Ph.D. degree in computer science from the University of Illinois at Urbana-Champaign in 2006.

He is currently an Assistant Professor in the School of Computer Science at McGill University, Montreal, QC, Canada. His research interests include real-time and embedded computing, performance and power management of server systems, cyber-physical systems,

real-time and embedded systems, fault tolerance, and control. He has authored/coauthored more than 40 refereed publications in leading conferences and journals in these fields.

Dr. Liu is a member of the ACM.



Qixin Wang (M'08) received the B.E. and M.E. degrees from the Department of Computer Science and Technology, Tsinghua University, Beijing, China, in 1999 and 2001, respectively, and the Ph.D. degree from the Department of Computer Science, University of Illinois at Urbana-Champaign in 2008.

He will join the Department of Computing in the Hong Kong Polytechnic University in 2009 as an Assistant Professor. His research interests include real-time/embedded systems and networking, wireless technology, and their applications in industrial control, medicine, and assisted living.

Dr. Wang is a member of the ACM.

Sathish Gopalakrishnan (M'06) received the Ph.D. degree in computer science from the University of Illinois at Urbana-Champaign.

He is an Assistant Professor of electrical and computer engineering at the University of British Columbia, Vancouver, BC, Canada. His research activities encompass several aspects concerning the design and implementation of highly reliable and predictable computer systems.

Dr. Gopalakrishnan is a member of the ACM.



Wenbo He (M'08) received the Ph.D. degree from the Department of Computer Science, University of Illinois at Urbana-Champaign in 2008.

She is currently an Assistant Professor in the Computer Science Department of the University of New Mexico, Albuquerque. Her research interests include pervasive and ubiquitous computing, cyber-physical systems, security, trust, and privacy.



Lui Sha (F'98) received the Ph.D. degree from Carnegie Mellon University, Pittsburgh, PA, in 1985.

He is currently Donald B. Gillies Chair Professor of Computer Science at the University of Illinois at Urbana-Champaign. His research area is dependable real-time and embedded systems.

Dr. Sha was elected a Fellow of ACM in 2005.

Hui Ding received the Ph.D. degree from the Department of Computer Science at the University of Illinois at Urbana-Champaign in 2006.

Kihwal Lee received the Ph.D. degree from the Department of Computer Science at the University of Illinois at Urbana-Champaign in 2006.