Industrial Automation as a Cloud Service

Tamir Hegazy, *Senior Member, IEEE* and Mohamed Hefeeda, *Senior Member, IEEE*

Abstract—New cloud services are being developed to support a wide variety of real-life applications. In this paper, we introduce a new cloud service: industrial automation, which includes different functionalities from feedback control and telemetry to plant optimization and enterprise management. We focus our study on the feedback control layer as the most time-critical and demanding functionality. Today's large-scale industrial automation projects are expensive and time-consuming. Hence, we propose a new cloud-based automation architecture, and we analyze cost and time savings under the proposed architecture. We show that significant cost and time savings can be achieved, mainly due to the virtualization of controllers and the reduction of hardware cost and associated labor. However, the major difficulties in providing cloud-based industrial automation systems are timeliness and reliability. Offering automation functionalities from the cloud over the Internet puts the controlled processes at risk due to varying communication delays and potential failure of virtual machines and/or links. Thus, we design an adaptive delay compensator and a distributed fault tolerance algorithm to mitigate delays and failures, respectively. We theoretically analyze the performance of the proposed architecture when compared to the traditional systems and prove zero or negligible change in performance. To experimentally evaluate our approach, we implement our controllers on commercial clouds and use them to control: (i) a physical model of a solar power plant, where we show that the fault-tolerance algorithm effectively makes the system unaware of faults, and (ii) industry-standard emulation with large injected delays and disturbances, where we show that the proposed cloud-based controllers perform indistinguishably from the best-known counterparts: local controllers.

Index Terms—Cloud computing, industrial automation, delay compensation, fault tolerance, feedback control

1 Introduction

1.1 Context and Motivation

Cloud computing is proving to be an important area that requires further research and development to accommodate more applications [24]. It has attracted significant interest from academia, industry, governments, and even individual users not only because of the promised cost savings, but also because it can improve existing computing services, e.g., video streaming [13]. In addition, cloud computing offers opportunities to create new services, e.g., robotics [8], [21] and manufacturing [35]. In this paper, we introduce a new cloud service: industrial automation. We show that the proposed service reduces the time and cost incurred in deploying automation systems, which are quite complex and require large human effort to build [26]. Further, we address how to migrate vital automation functionality to the cloud without compromising the system performance.

It is noteworthy that the work proposed in this paper fits an ongoing trend that evolved several decades ago when digital computers were first introduced to control systems around the year of 1960 in the form of Direct Digital Control (DDC) [34]. Ever since, evolution of control system has been associated with the advancement of computing devices [34]. Several functionalities, e.g., monitoring, logging, optimization, and asset management, have been added on top of the core direct digital control functionality, forming an *automation system*. As a result, a current industrial automation system is indeed a multi-tiered architecture entailing several hierarchical layers of computation, communication, and storage [33]. With such history, we see a great potential in studying the application of an evolving computing model, such as cloud computing to industrial automation systems, which could provide several benefits to end users, including cost saving and agility.

1.2 Background: Industrial Automation

The purpose of this section is to briefly introduce basic industrial automation concepts relevant to this work. This should help readers without enough industrial automation background to better understand the rest of the paper. We use a simplified, yet a real-life example to introduce the concepts. Such example will be used later in the evaluation of the proposed approach.

Consider the solar power plant, whose process diagram is shown in Fig. 1, which is a simplified version of the design in [17]. The operation of the solar power plant is divided into four main process cycles: synthetic oil cycle, salt cycle, steam cycle, and condensation cycle. The oil cycle collects the solar energy, while the salt cycle stores it to pump it in later. The steam cycle and the condensation cycle are responsible for operating a steam turbine. The oil cycle starts at the solar collector mirrors, which collect the sun heat along horizontal pipes passing oil. Solar collectors can be rotated to follow the sun direction. The oil absorbs the heat and passes it in two branches to interact with the salt cycle and the steam cycle. The salt cycle has two modes: heat storage and heat pumping. If the heat absorbed by the oil exceeds the required amount to run the plant, the salt is

- *T. Hegazy is with the School of Electrical and Computer Engineering, Georgia Institute of Technology, Atlanta, GA. This work was done while visiting Simon Fraser University. E-mail: tamir.hegazy@ece.gatech.edu.*
- *M. Hefeeda is with the School of Computing Science, Simon Fraser University, Burnaby, BC, Canada, and the Qatar Computing Research Institute, Doha, Qatar. E-mail: mhefeeda@cs.sfu.ca.*

Manuscript received 17 Mar. 2014; revised 11 Sept. 2014; accepted 18 Sept. 2014. Date of publication 22 Sept. 2014; date of current version 2 Sept. 2015. Recommended for acceptance by Y. Cui.
For information on obtaining reprints of this article, please send e-mail to: reprints@ieee.org, and reference the Digital Object Identifier below.
Digital Object Identifier no. 10.1109/TPDS.2014.2359894

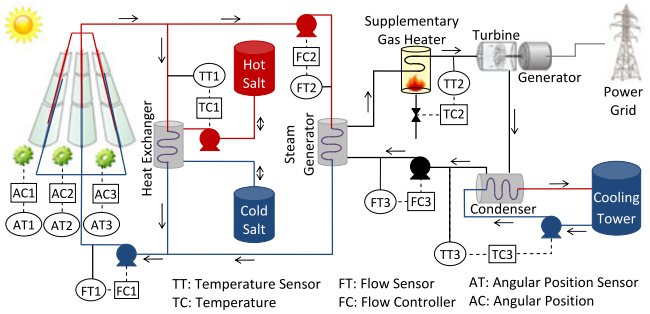


Fig. 1. Process diagram of a solar power plant.

pumped from the cold tank to the hot tank to store the excessive heat. At times where solar energy drops below required levels (e.g., cloudy weather), the salt flow direction is reversed to pump heat into the oil. The oil is pumped into a steam generator to heat up water to generate steam. A natural gas heater is used to maintain the evaporation temperature should the solar heat and the salt-pumped heat drop below required levels. The pressurized steam is fed through a steam turbine, which drives an electrical generator connected to the electrical power grid. The last cycle is the steam condensation required to create a vacuum at the downstream side of the turbine, which is necessary for an efficient steam cycle.

For the plant to operate as desired, several physical quantities need to be maintained at constant values (e.g., oil temperature) in spite of any disturbance, and other quantities need to follow a certain profile (e.g., solar collector angular position). To automatically maintain the desired performance, we need a system to keep monitoring the physical quantities through *sensors*, and make necessary adjustments through *actuators*. In our simplified example, we identified nine control loops. Note that mega industrial plants, however, could have tens of thousands of loops. The nine loops are shown in Fig. 1: (i) three angular position control loops for the the solar collector mirrors, where the sensors are potentiometers and the actuator are DC motors, (ii) three flow control loops; two for the oil cycle and one for the steam cycle, where sensors are flowmeters and actuators are pumps, and (iii) three temperature control loops, one for the oil cycle, one for the steam cycle, and one for the condensation cycle, where sensors are thermocouples and actuators are pumps and valves controlling the flow of heating/cooling material. Additional control loops (e.g., for the turbine) are not shown for simplicity.

We now show how a present-day industrial automation system such as the one shown in Fig. 2 is used to control the solar power plant. First of all, we need field devices, which are the sensors and actuators described in the previous paragraph. Field devices are shown at the very bottom of the figure. Then, we need controllers, which are special-purpose microprocessor-based computer systems running control algorithms on top of special-purpose operating systems. A single controller can usually interact with several control loops, depending on the processing speed as well as the sampling rates and complexity of the control algorithms. Controllers are hosted in or next to a control room. Such controllers belong to layer L1 in Fig. 2. For medium and large-scale systems, communication between

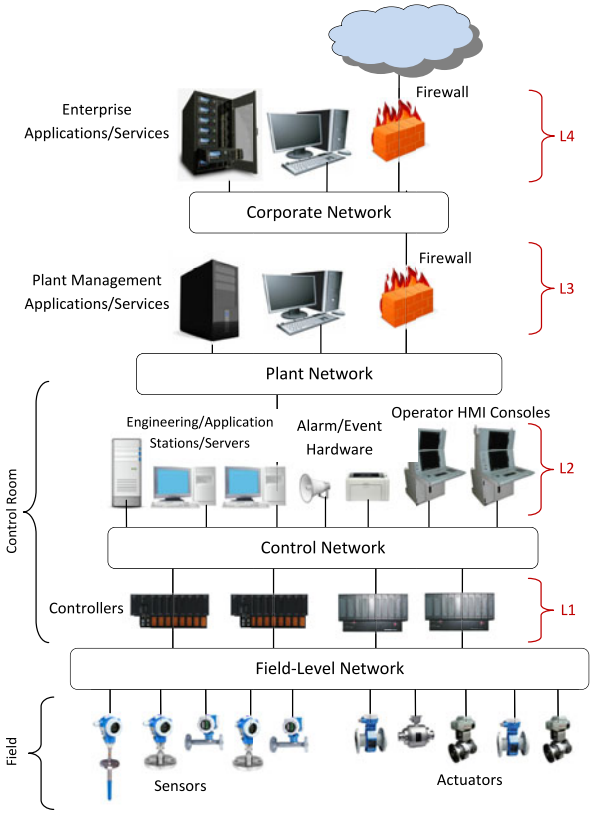


Fig. 2. Current automation system architecture.

field devices and controllers takes place over a special network called *field-level network*. Second, operators in the control room need to continuously monitor the *process variables* of the plant, such as oil flow rate, water temperature, and angular position of solar collectors. Therefore, they need Human-Machine Interface (HMI). Also, operators need to supervise the operation of individual control loops, e.g., change the desired oil temperature, through Supervisory Control and Data Acquisition (SCADA). Further, they need to log (or "historize") the process variables constantly to review their trends later and assess how the system health changes with time. L2 in Fig. 2 is for HMI/SCADA and historization. Other functionalities such as alarm management (e.g., when steam temperature dangerously goes beyond a certain threshold) and control software update (e.g., to improve a control algorithm) also belong to L2. Communication between controllers and HMI/SCADA machines take place over a special network called *control network*. Third, to optimize the overall operation of the solar power plant (e.g., across different seasons or weather conditions), higher-level optimization software (L3 in Fig. 2) is needed to coordinate the operating conditions of control loops so that the optimization objectives are met. For example, in our solar power plant, the objectives could be minimizing natural gas consumption, or maintaining a certain production rate/quality. Fourth, we need to link the plant to the outside world to tie the energy production to market demands as well as different assets, such as material and labor prices. L4 in Fig. 2 is there to perform such enterprise-level management. Plant optimization objectives in L3 are decided based on the analysis performed in L4.

1.3 Scope and Contributions

In this work, we show that if we adopt cloud computing to host the industrial automation layers described above, great cost and labor savings are achieved as we show in Section 6. However, industrial automation is an application area that requires tighter timeliness, reliability, and security than most other cloud applications. As a result, multiple research problems need to be addressed to enable cloud-based industrial automation.

We propose and make the case for an architecture for cloud-based industrial automation. Further, we solve the most challenging problems stemming from moving the automation functionalities to the cloud. We identify three main challenges facing the proposed architecture, namely timeliness, reliability, and security. We focus on the timeliness and reliability challenges. Addressing security is outside the scope of this work. Moreover, we reason that addressing the timeliness and reliability requirements for L1 poses the highest challenge, and hence, automatically addresses the other layers. The reason is that application timeliness and reliability requirements increase as we move down the hierarchy towards layer L1, making L1 the most demanding layer for the following reasons. First, closed-loop communication for L1 could occur as frequently as several times per second, whereas in L2 most of the communication is one way and it typically happens every 1-2 seconds. Second, failures at L1 could have severe consequences since L1 deals directly with the physical process, whereas the interaction between L2 and the physical process is indirect. L3 and L4 are even more relaxed in terms of timeliness and reliability since they do not deal with time-critical functionalities.

To maintain the operation of the controlled system under large variable Internet delays, we propose a delay compensator to mitigate such delays. Then, to handle controller/link failures, we propose and theoretically analyze a distributed fault tolerance algorithm that smoothly switches among redundant cloud controllers. Our theoretical analysis shows that the proposed fault-tolerant, cloud-based controllers guarantee zero overshoot and zero steady-state error if the original local counterparts guarantee the same. Further, the effect on the settling time is zero in most practical cases, and negligible in other rare, extreme cases.

To evaluate our approach, we designed and implemented a physical model of the solar power plant system presented in Fig. 1 and controlled its various processes through cloud controllers. In addition, we implemented the solar power plant model in an industry-standard emulation software system in order to show the viability and robustness of our proposed architecture under wide range of parameters. Specifically, to evaluate the distributed fault tolerance algorithm, we deployed redundant controllers on commercial clouds that are located thousands of miles away from each other and from the physical plant. Experimental results show that our algorithms can effectively make the physical plant unaware of link/controller failures. Further, to study the effect of very large variable delays and different disturbance models on the performance of the proposed system, we perform a series of experiments where distant cloud-based controllers control the emulated solar power plant hosted on a local machine in our lab. Through emulation, we inject communication delays and disturbances into the system. The results show that the cloud-based controllers effectively mitigate the effect of delays and perform indistinguishably from their local counterparts under disturbance.

The rest of the paper is organized as follows. We discuss related work in Section 2. Then, we introduce the proposed architecture as well as the proposed delay mitigation and fault tolerance components in Sections 3, 4, and 5, respectively. We analyze and evaluate the cost/time savings of our proposed approach in Section 6. Then, we experimentally evaluate our approach in Section 7, and we conclude the paper in Section 8.

2 Related Work

Kumar et al. [21] proposed an approach for assisting autonomous vehicles in path planning based on cloud-collected remote sensor data. Chen et al. [8] proposed "Robot as a Service" or RaaS where the service is available in both hardware and software. Also, Wu et al. [35] explored "Cloud Manufacturing", which is a cloud-based manufacturing model that is service-oriented, customer-centric, and demand-driven. RaaS and Cloud Manufacturing focus on planning and optimization, while we consider the whole automation hierarchy and focus on direct digital control, which is much more challenging in terms of timeliness and reliability. Several researchers/enterprises employed feedback controllers to manage their computing systems, e.g., [1], [23], [29]. The employed feedback controllers can be acquired as a service through our proposed cloud-based feedback control approach.

It has been recently proposed to offer certain industrial automation components through the cloud. First, enterprise-level (L4) asset management applications, such as SAP, are now offered through the cloud. Second, plant optimization (L3) can easily be offered through the cloud. For example, Honeywell Attune [31] offers cloud-based services for energy optimization. Although, it is mainly offered for building automation, the plant version is conceptually the same. Third, HMI/SCADA (L2) is now offered in a virtualized fashion as it is the case with Invensys Wonderware System Platform 2012 [19], which indicates that offering L2 as a cloud service is only a matter of moving the virtual machines (VMs) to the cloud. Finally, moving direct digital control (L1) to the cloud is challenging due to timeliness and reliability requirements. We are not aware of any commercially available system that offers direct digital control through the cloud, although researchers have partially addressed some of these challenges as we present in the following paragraphs.

Yang et al. [36] designed two predictive compensators for Internet delays in the feedforward and feedback directions, while Chen et al. [9] proposed a single compensator block for the feedforward path only. Yoo et al. [37] proposed the use of Smith Predictor with constant buffer, which can unnecessarily increase the settling time because it inserts delay inside the control loop to keep the delay at its maximum possible value. In contrast, our proposed delay mitigation approach requires only one remote component that adaptively compensates for the whole roundtrip delay without affecting the original controller design or requiring additional support from the controlled system.

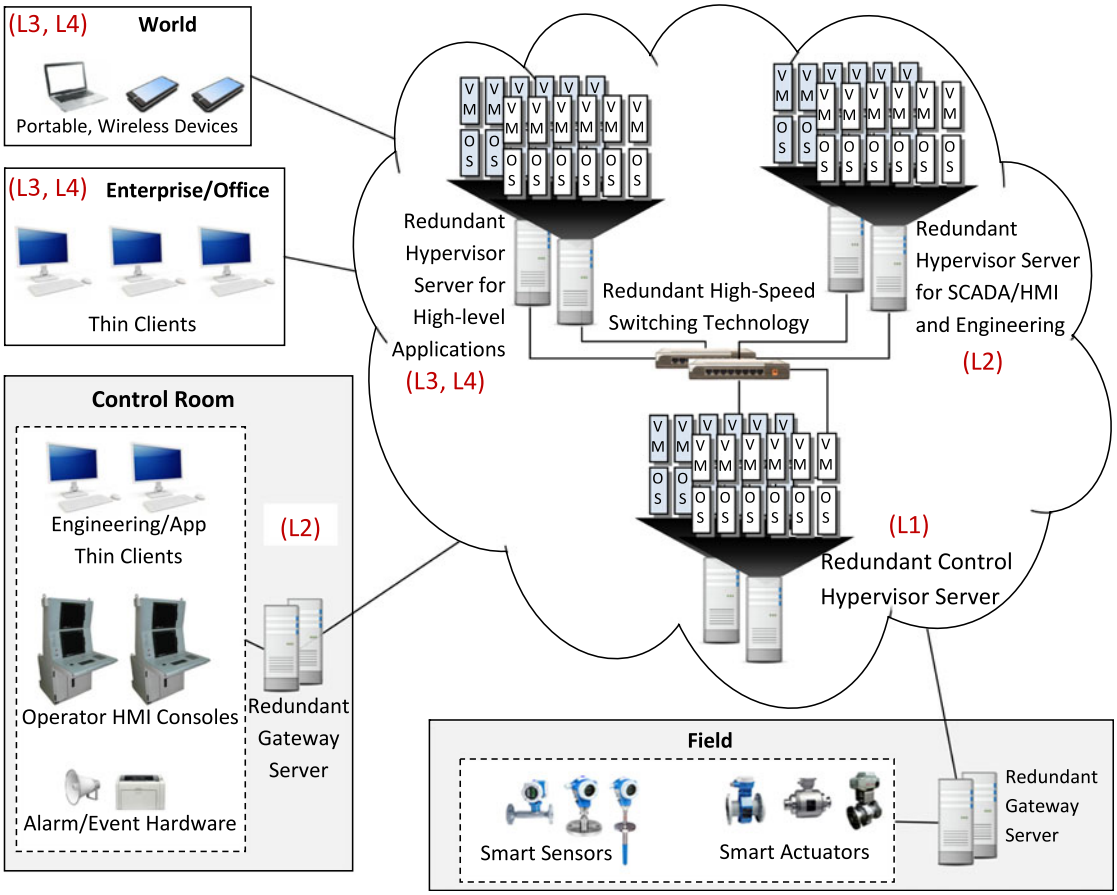


Fig. 3. Overview of the proposed architecture.

Most fail-stop failures in digital controllers are handled through redundancy [7]. For example, Yokogawa FFCS [18] employs double redundancy, and Invensys Triconex Tricon [20] employs triple redundancy. While the former is used for direct digital control, the latter is used for safety and emergency shutdown control of mission-critical processes, e.g., nuclear power plants. Our extensive literature review did not yield any work where redundant controllers are remote to both the process and the primary controller as we propose in this paper.

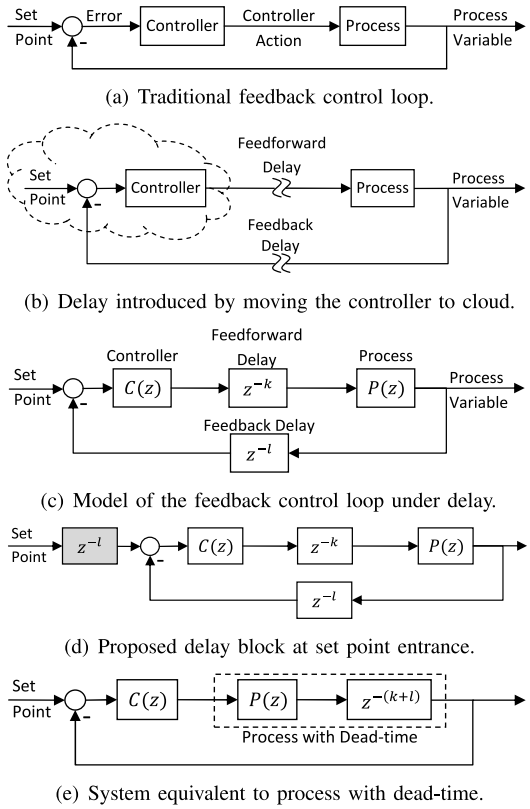
In summary, we are not aware of any work that proposed an architecture for the whole industrial automation hierarchy, including its most demanding layer of direct digital control. In addition, to the best of our knowledge, there is not any work that analyzes cost/time saving when using cloud computing in industrial automation, as we do in this paper. Furthermore, we have not come across any work that addressed fault tolerance through redundant software controllers when running on loosely coupled machines that are far away from one another. Thus, in the evaluation section, we compare the performance of our approach to the best known counterpart: local, physical, and tightly coupled controllers.

3 Proposed Architecture Overview

In our proposed approach, we move all computing functions of the automation system into the cloud in order to provide full automation as a service. This makes it easier, faster and less costly for users to deploy, maintain, and upgrade their automation systems. Moreover, our design supports switching to different cloud automation providers since all VMs can be group-migrated to a different provider. Some components are not movable to the cloud, such as sensors, actuators, and safety/emergency shutdown control functions. Fig. 3 illustrates our proposed automation architecture. Our proposed approach relaxes the existing systems layers. Fig. 3 reflects the relationship between each component and the layers shown in Fig. 2.

To connect sensors and actuators to the cloud, we use field-level protocols that run on top of TCP, such as Modbus/TCP and Profibus/TCP, which are either built in the devices or provided through separate I/O modules. For cases where advanced functions, such as security and message-level scheduling are required, we dedicate a gateway server, which could be replicated for more reliability.

In our approach, direct digital control algorithms (L1) run on cloud VMs instead of real hardware in the control room. Also, in existing automation systems (Fig. 2), controllers communicate with sensors/actuators over a network with mostly deterministic communication delays that are negligible. Whereas in our design, communication occurs over the Internet, which adds large and variable delays to the control loop. Therefore, straightforward migration of direct digital control algorithms to the cloud may affect the functionality of the control loop or even make the system unstable, and thus jeopardize the theoretical performance guarantees offered by traditional controllers. As a result,



(a) Traditional feedback control loop.

(b) Delay introduced by moving the controller to cloud.

(c) Model of the feedback control loop under delay.

(d) Proposed delay block at set point entrance.

(e) System equivalent to process with dead-time.

Fig. 4. Loop simplification for Internet delay mitigation.

more components are needed to mitigate the variable Internet delays and the lack of reliability of Internet links and VMs, which we design in this paper.

We propose providing the HMI/SCADA layer, L2, and all the way to L4, through Platform or Software as a Service (PaaS and SaaS) models. Thus, we provide engineers and operators with access to the control room applications through thin clients. In existing automation systems (Fig. 2), a control room is a complex environment loaded with servers, workstations, network switches, and cables. In our proposed design, a control room is comprised of a number of thin clients, which requires less hardware and wiring, making it a tidier environment [26]. In a manner similar to the field gateway server, we propose a control room redundant gateway server to reliably carry on advanced functions, such as security and message scheduling.

To show how to maintain the timeliness and reliability of migrated functionalities, we study L1, since it's the most challenging layer as we discussed in Section 1.3. We propose (i) an adaptive delay compensator (Section 4) which mitigates the effect of communication delays on the remotely-controlled physical plant, and (ii) a distributed fault tolerance algorithm (Section 5) which enables the controlled plant to maintain operation under controller/link failures. Security is outside the scope of this work.

4 Proposed Delay Mitigation

The roundtrip delay between the cloud controllers and controlled processes varies with time. In most cases, the Internet roundtrip delay ranges from tens to a few hundreds of milliseconds. Meanwhile, the sampling periods used in most industrial applications typically range from a few hundreds of milliseconds to several seconds. Therefore, most of the roundtrip delay will be absorbed within the sampling periods and will have no effect on the control loop [36], because the controlled process will still be receiving one action per sampling period. However, delay may occasionally change in a random fashion beyond the sampling period because of the dynamic nature of the Internet. To absorb such random variations, we need an adaptive delay mitigation approach.

In this section, we propose a method to handle varying communication delays. We start with the traditional feedback control loop shown in Fig. 4a. Moving the controller to a remote server as shown in Fig. 4b adds delay in both directions. We model the loop as shown in Fig. 4c, where $C(z)$ and $P(z)$ are the transfer functions of the controller and the controlled process, respectively, and z^{-k} and z^{-l} denote the feedforward and feedback delays, respectively.

We introduce an artificial delay block equal to z^{-l} at the entrance of the set point as shown in Fig. 4d. Introducing such delay is insignificant to the system performance as we discuss at the end of this section. We simplify the loop as shown in Fig. 4e. Thus, we have managed to reduce our cloud control problem to what is known as controlling a *process with dead-time* [32], where there is a time delay between the application of the process input and its effect on the output, e.g., when material traverses a long path within the process over a conveyor belt.

To control a process with dead-time, the controller is usually coupled with a *delay compensator*. We propose our adaptive version of Smith Predictor [30]. Thus, our compensator does not require precise knowledge of the delay component at design time. We first design the controller as if no delay is encountered. Then, we measure the delay and adjust the Smith Predictor. This is important because the communication delay over the Internet changes dynamically and cannot be known ahead of time.

The original design of Smith Predictor is as follows. Suppose the process consists of a non-delay component $P(z)$ followed or preceded by a pure time delay $z^{-(k+l)}$. If we first consider the process without delay and design a controller $C(z)$, the closed loop transfer function becomes $T(z) = C(z)P(z)/(1 + C(z)P(z))$. The objective is to find a controller $\bar{C}(z)$ for the process $P(z)z^{-(k+l)}$ such that the closed loop transfer function is $\bar{T}(z) = T(z)z^{-(k+l)}$, which involves solving the following equation for $\bar{C}(z)$

$$\frac{\bar{C}(z)P(z)z^{-(k+l)}}{1+\bar{C}(z)P(z)z^{-(k+l)}} = \frac{z^{-(k+l)}C(z)P(z)}{1+C(z)P(z)}. \quad (1)$$

The new controller is therefore given as:

$$\bar{C}(z) = \frac{C(z)}{1+(1-z^{-(k+l)})C(z)P(z)}. \quad (2)$$

Fig. 5 shows our proposed design of the virtualized controller, which has two main components: (i) controller with delay compensator, and (ii) communication delay estimator. The controller with delay compensator is shown in the dashed box which is a block diagram of the

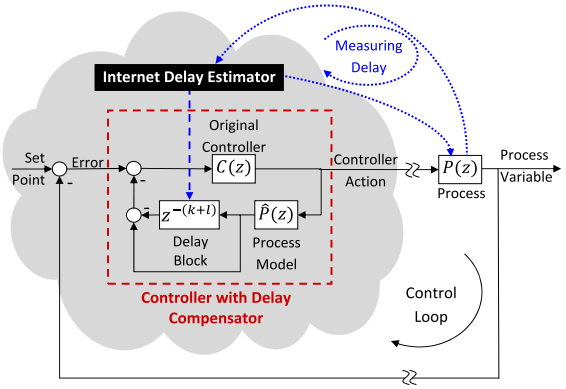


Fig. 5. Proposed virtualized controller that compensates for variable communication delays.

controller described by Eq. (2) with the combined feedforward and feedback delays $z^{-(k+l)}$, i.e., the roundtrip delay. It uses $C(z)$ which is the original controller designed for the process $P(z)$ with no delays. It also needs an approximation of the process transfer function which is denoted by $\hat{P}(z)$. In practice, a simple first or second-order approximation is sufficient [28]. The second component is shown in the black box in Fig. 5, and it estimates the roundtrip delay between the process and the remote controller. The roundtrip delay is used in the delay block $z^{-(k+l)}$. Our delay estimator employs an exponentially weighted moving average to estimate the communication delay mean as $D_i = \alpha d_i + (1-\alpha)D_{i-1}$, where D_i is the estimated mean delay and d_i is the measured delay at discrete time instant i. Similarly, our estimator employs an exponentially weighted moving variance to estimate the delay variance as $V_i = \alpha(d_i - D_i)^2 + (1-\alpha)V_{i-1}$, where V_i is the estimated variance at discrete time instant i. The delay value in the delay block is adjusted to $D_c = \lfloor (D_i + hV_i^{1/2})/T_s \rfloor$, where T_s is the sampling period, and h is a positive parameter to accommodate for delay values larger than the mean. Thus, the estimator adjusts to changes of delay while not overreacting to short delay spikes.

Now, we go back to the delay block introduced in Fig. 4d. Introducing such delay is insignificant to the operation of the system for two reasons. First, set points are kept constant for extremely long periods if not for the entire system lifetime. In control theory, a delayed version of such constant function is the same constant function. Second, even in the infrequent cases where set point has to be changed, it is often performed by a human operator. Adding a few tens or even hundreds of milliseconds of delay is insignificant to the operator response (several seconds, to reach a knob or software slider and to update the value).

In summary, the novelty of the proposed approach is that adding a single artificial delay block outside the control loop transformed the challenging cloud-based control problem to that of controlling a process with dead-time, which is solved using Smith Predictors. Using our adaptive version of Smith Predictor enables moving the controllers to a distant cloud provider without changing the design of the original controller or the process being controlled.

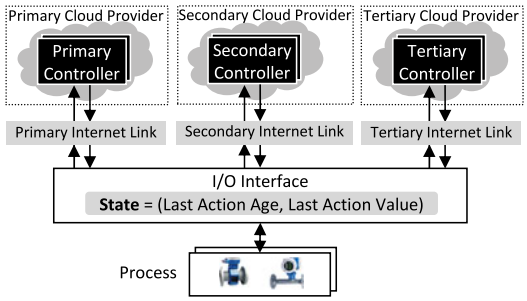


Fig. 6. Redundant cloud-based controllers for mitigating link/controller failures.

5 Proposed Fault Tolerance

5.1 Overview

In this section, we design a distributed fault tolerance algorithm which guarantees normal operation under failures, and we theoretically analyze its performance. We also theoretically prove that for most real-life scenarios, cloud feedback control using our algorithm has virtually no effect on the controlled process operation.

In most practical systems, controller failures are handled by double redundancy as in [11], or at most triple redundancy as in [20] for mission-critical processes. Upon failure, redundant controllers take over in a stateful manner. In current automation systems, redundant controllers are closely located and tightly synchronized. Thus, they easily share the control loop state periodically (typically once ever a few tens of milliseconds). Providing similar reliability for redundant cloud controllers is quite challenging because controllers would typically run on different machines and preferably in different data centers or even different cloud providers, through different Internet providers (multihoming) as shown in Fig. 6. Using different machines tolerates machine failures, whereas replicating across different data centers (or cloud providers) and using different Internet providers add more robustness to situations such as Internet link failures. Additionally, fine-grained clock synchronization and maintaining the state consistency on short time scales are complex and costly for geographically distant machines communicating over the *best-effort* Internet.

To achieve reliability in the proposed feedback control cloud service, we propose a distributed fault tolerance algorithm that is run *asynchronously* by all redundant controllers. We call this algorithm *Reliable Cloud Control* (RCC). RCC supports double and higher redundancy levels and provides the following guarantees:

- *G1* If the primary controller fails, the secondary controller is automatically hot-swapped in. This guarantee is generalizable to higher redundancy. For example, in triple redundancy, if the primary and secondary controllers fail, the tertiary controller is hot-swapped in.
- *G2* If the failed primary controller recovers, it takes over, forcing the secondary controller out of action. This guarantee is desirable when the secondary VM and/or link are chosen to be of less quality than the primary for cost savings. This guarantee is also generalizable to higher redundancy.

```
01 begin scan
02  begin step: Initialization
03    if firstCycle
04      initialize(myID);
05      initialize(myEngagementThreshold);
06    end if
07  end step
08  begin step: Polling
09    isTimedOut = pollIO(procVar, lastActn, lastActnAge(1:n));
10    if isTimedOut
11      firstCycle = TRUE; end scan;
12    end if
13  end step
14  begin step: Computing
15    iAmEngaged = TRUE;
16    for index = 1 to myID - 1
17      if lastActnAge(index) < myEngagementThreshold
18        iAmEngaged = FALSE;
19        exit for;
20      end if
21    end for
22    myNextActn = controller(procVar, iAmEngaged, lastActn);
23  end step
24  begin step: Conditional Acting
25    if iAmEngaged
26      writeToIO(lastActn:= myNextActn, lastActnAge(myID):= 0);
27    end if
28  end step
29 end scan
```

Fig. 7. Pseudocode of the proposed distributed fault tolerance algorithm: RCC.

G3 Handover of controllers is performed smoothly causing no undesirable transient response artifacts.

For RCC to provide such guarantees, we define the system state as the tuple $(a, u_1, u_2, u_3, \ldots)$, where a is the last controller action executed by the actuator, and u_i is the time elapsed since the last action performed by the redundant controller C_i. To be visible to all controllers, RCC stores the state tuple in the memory of the control I/O interface module as shown in Fig. 6. The state tuple is initialized when the I/O interface is first turned on. The last action a can be initialized arbitrarily according to the process design. The time since last action u_i is initialized to ∞ to indicate that the controller C_i has never acted.

We note that RCC does not require any clock synchronization. First, RCC does not use absolute timestamps to measure the time u_i elapsed since a controller C_i last acted. Instead, it performs relative delay measurements through time counters in the control I/O interface. Second, RCC is a periodic, soft real-time task whose relative deadline is equal to its sampling period. As a result, the core control algorithm is executed on every sampling period and is required to finish any time before the beginning of the next period. Delaying a control action within the same sampling period does not compromise the running control algorithm [36] because the process will still be receiving one action per sampling period. For these two reasons, RCC can run asynchronously on all VMs, and backup controller(s) could be started any time after the primary is started, without having to synchronize the clocks of the VMs hosting the controllers.

5.2 Detailed Operation

Fig. 7 shows the pseudocode of RCC which runs on top of every controller. We refer to the algorithm line numbers between parentheses. In addition to the *Initialization* step, RCC runs three steps in each sampling period: *Polling*, *Computing*, and *Conditional Acting*. The Initialization step (lines 2-7) runs on the very first cycle, where RCC initializes the ID i (line 4) and the engagement threshold D_i (line 5) for controller C_i to guarantee that only one controller is engaged at a time. IDs are set as 1 for the primary, 2 for the secondary, and so on. Also, for any controller pair (C_i, C_j) where $i > j$, the engagement thresholds must satisfy $D_i > D_j \geq T_s$, where T_s is the sampling period. Then, the main steps are executed with every sampling period. The Polling step (lines 8-13) fetches the following variables from the I/O interface (line 9):

(i) `procVar`: the current sensor measurement.
(ii) `lastActn`: representation of the state variable a, i.e., the last action executed by the actuator.
(iii) `lastActnAge`: a time counter array, where `lastActnAge(i)` represents the state variable u_i, i.e., the time elapsed since C_i was last engaged.

If the Polling step times out, e.g., due to link failure, the controller skips the current sampling period after resetting its `firtCycle` flag to `TRUE` (line 11). This is important for guarantee *G3* as will be shown in Section 5.3.

Then, the Computing step (lines 14-23) decides the controller mode. For a given controller C_i, if there is another controller C_j with a smaller ID ($j < i$) that is alive, then C_i will decide to run in the standby mode. On the other hand, for all C_j where $j < i$, if the age of the last action u_j is older than D_i, then C_i will decide to run in the engaged mode as it assumes that all controllers C_j have failed. Thus, RCC evaluates the flag `iAmEngaged` using the `for` loop that is scanning `lastActnAge` for controllers with lower IDs (lines 16-21). Then, RCC runs the control algorithm `controller()` (line 22), which normally requires the sensor measurement `procVar` only. For some control algorithms, guarantee *G3* dictates passing more parameters as discussed in Section 5.3.

Finally, the Conditional Acting step (lines 24-28) sends the computed action to the process (line 26) if the `iAmEngaged` flag is `TRUE` (line 25). It further sends zero to reset the action age counter. Otherwise, if the `iAmEngaged` flag is `FALSE`, the step performs no actions.

Without loss of generality, we now focus on the triple redundancy case to illustrate the interaction among 3 controllers under RCC. The `iAmEngaged` flag of the primary controller is always `TRUE` since it has the smallest ID. As the secondary controller polls `lastActnAge(1)`, it continuously checks whether the primary controller is alive. If the primary controller fails, the secondary controller will detect the failure when `lastActnAge(1)` exceeds the secondary controller's engagement threshold. In this case, `iAmEngaged` for the secondary controller will stay `TRUE` throughout the `for` loop. Thus, the secondary controller will run in the engaged mode and hence reset its counter `lastActnAge(2)` entry in the I/O interface to indicate it has just acted. Although the tertiary controller will also detect the failure of the primary, its engagement threshold is higher than that of the secondary controller. Before the value of `lastActnAge(1)` crosses the tertiary controller's engagement threshold, the secondary

controller will have already acted. Thus, when the tertiary polls the state on the following sampling period, `lastActnAge(2)` will have incremented to δ, such that $0 \leq \delta \leq T_s$ which is less than the tertiary's threshold, forcing the `iAmEngaged` flag for the tertiary controller to become `FALSE`. The tertiary controller will get engaged if and only if both the primary and secondary controllers become unavailable. This addresses guarantee *G1*.

If the primary controller recovers from failure, it will gain control over the process since it always operates in the engaged mode, forcing the secondary controller into the standby mode. Upon resetting `lastActnAge(1)` for the primary controller, the secondary will detect the recent primary action whose age is less than the secondary's engagement threshold. As a result, the `iAmEngaged` flag for the secondary controller will turn `FALSE`, causing it to operate in the standby mode. The same discussion applies to any two controllers when the lower-ID controller recovers from failure, achieving guarantee *G2*.

5.3 Smooth Controller Handover

Switching between controllers may result in a "bump" in the process output, which would violate guarantee *G3*. This occurs if the final value of the original controller action is not equal to the initial value of the new controller action. The main reason for this is that the redundant controllers do not necessarily start at the same time. With most controllers having an integrator component, the output of the controllers will not be the same since their integration intervals have different start times. To achieve smooth handover between cloud controllers, we propose to use the *bumpless transfer* concept from control theory [5], [32] in our cloud controllers. Bumpless transfer is originally designed to support switching from "manual" to "auto" control, and it is supported by most commercial PID controllers, which constitute more than 90 percent of the controllers employed in the industry [6], [12]. Bumpless transfer for PID controllers is achievable through adjusting the initial value of the integrator [32]. Other bumpless transfer methods have been proposed for advanced "auto" controllers, e.g., [5].

The smooth handover feature is implemented within the `controller()` function introduced in Fig. 7. Assume we have two standard PID controllers: a primary C_i in the engaged mode, and a backup C_j in the standby mode. After each of the controllers computes the proportional component, `P`, and the derivative component, `D`, the modified `controller()` function for each of the controllers evaluates the following logical condition: (**not** `iAmEngaged`) **or** `firstCycle`. If the condition is `TRUE`, it will adjust the integrator initial value as follows: `integInitVal = lastActn - P - D;`, which means that integrator initial value will be modified by subtracting the proportional and the derivative components `P`, `D`. This means that except for the first sampling period, the *engaged* controller C_i runs the PID control algorithm without applying the modification because the logical condition is `FALSE` (note that (**not** `iAmEngaged`) is `FALSE` for the engaged controller). On the other hand, for the standby controller C_j, the overall condition will be `TRUE`, so it overrides the regular value of the PID integrator by forcing it to be equal to the last control action (which was computed by the engaged controller, C_i) by subtracting `P`, `D`. In short, the smooth handover feature corrects any deviation of the integrator of C_j so it matches the integrator of C_i. Consequently, if C_i fails, and C_j takes over, then C_j starts with an action that is equal to the last action of C_i.

On the initial sampling periods (i.e., when `firstCycle` is `TRUE`), all controllers are required to correct the initial values of their integrators. This enables smooth handover between a recovered C_a with the currently engaged controller C_b if $a < b$. This is why RCC sets `firstCycle` to `TRUE` upon timeouts in Fig. 7. Consider a case where an engaged controller C_a suffers a link failure whereby the Polling step times out, and a backup controller C_b is swapped in. If the link recovers, then C_a takes over again after performing smooth handover with C_b because upon recovery, the `firstCycle` flag of C_a will be `TRUE`.

5.4 Analysis

We consider the fail-stop failure model [7] for the controller software, the hosting VM, the hosting server, the network switch, and the Internet link. In the following, we formally prove our guarantees. Proofs are given in Appendix B, which can be found on the Computer Society Digital Library at http://doi.ieeecomputersociety.org/10.1109/TPDS.2014.2359894

Theorem 1. *The proposed RCC algorithm guarantees normal operation of the controlled process as long as there is at least one operating controller that is accessible through at least one link.*

Theorem 2. *If the original control algorithm guarantees zero overshoot and zero steady-state error under no failure, then RCC algorithm guarantees the same overshoot and steady-state error performance under failure, provided that there is one operating reachable controller.*

Theorem 3. *The worst case increase in the settling time t_s under one failure is upper-bounded by the Internet roundtrip delay RTT_j and the engagement threshold D_j of the backup controller C_j, and is given by $\Delta t_s = \lceil (RTT_j + D_j)/T_s \rceil - 2$, where T_s is the sampling period.*

Theorem 4. *The RCC algorithm guarantees no change in process response upon controller recovery.*

6 Cost/Time Analysis and Evaluation

We present simple, parametrized models that can be used to estimate the savings in cost and time of different automation systems when they are offered from the cloud. We also present a realistic case study to show the range of achievable savings. Due to space limitations, we only present a summary of our analysis and results; more details can be found in Appendix A and in [15], which is available online.

6.1 Analytical Models

In [15], we showed that the total cost saving due to adopting the cloud-based model can be estimated as:

$$S_{Cost} = \frac{s_U C_U + N s_R C_R}{C_U + N C_R}, \tag{3}$$

TABLE 1
Real-Life System Used in Our Case Study

Oil/gas Plant		Analog I/O signal count	45,000	15,000 indication only
				15,000 control input
				15,000 control output
		Discrete I/O signal count	50,000	10,000 indication only
				20,000 control input
				20,000 control output
		Total I/O signal count	95,000	
		Simple analog control loop count	10,000	
		Complex analog control loop count	2,500	
		On-off control loop count	10,000	
Automation System	Software	Control/HMI/SCADA block count	145,000	
		Sampling periods	Ranging from 0.5 to 2 seconds	
		HMI graphic displays	4,000	
		HMI refresh rate	2 seconds	
	Hardware	Field-level network type	FOUNDATION Filedbus	
		Control network type	1Gbps redundant star network	
		Network switch count	10	
		Controller count	100 pairs	
		I/O module count	800	
		Engineering workstation/server count	10 (for configuration)	
		Application workstation/server count	23	
		HMI workstations	30	

where s_U, s_R respectively denote the saving in upfront and running costs due to adopting cloud-based automation model, C_U, C_R respectively denote the upfront and running costs of the current automation system, and N denotes the lifetime of the system. In turn, we showed that [15]:

$$s_U = \frac{s_H C_H + s_L C_L + s_C C_C}{C_h + C_L + C_C},\ s_R = s_M - \frac{C_V}{C_M}, \tag{4}$$

where s_H, s_L, s_C, and s_M respectively denote the saving in hardware, labor, commissioning/start-up, and maintenance costs due to the adoption of the cloud-based model. Labor cost corresponds to monetary compensation for engineers/technicians. And C_M and C_V denote the annual maintenance cost and the annual VM leasing cost.

Under cloud-based automation, the upfront cost can be reduced significantly. First, a big portion of the hardware is now replaced with cloud-based VMs, which counts towards running cost, not upfront cost. Second, the engineering labor is also reduced since hardware testing is greatly reduced. Third, the commissioning labor is reduced since hardware configuration and wiring is highly reduced.

Similarly, the running cost (mainly maintenance) will be reduced due to virtualization, which eliminates a lot of expensive hardware. Another source of reduction is by transferring most of the site labor into office labor. The hourly rate for an office engineer is around one-third of that of a site engineer [26]. The cloud-based approach will, however, incur an extra running cost component: VM leasing cost. Such additional cost is overcome by the saving in maintenance as we show in the case study (Section 6.2).

Next, we consider the time saving due to the adoption of cloud-based automation systems. We define *time to start up* (TTSU) as the duration between the point in time where all system components have been ordered and become available and the point in time the automation system is first turned on. The project goes through four main phases. In practice, negligible or no overlap exists between the project phases. Therefore, TTSU is composed of:

i) T_E: time for automation system engineering, where automation hardware is set up and configured, and automation software is implemented.
ii) T_F: factory acceptance testing, where automation system functions are tested against design specifications.
iii) T_S: system shipping from engineering location to site.
iv) T_C: site acceptance testing and commissioning, where the automation system is tested to make sure it, once installed in real conditions, is operating and interacting properly with the plant.

Thus, TTSU saving can be approximated as [15]:

$$S_{TTSU} = \frac{T - \hat{T}}{T} = \frac{s_E T_E + T_F + T_S + s_C T_C}{T_E + T_F + T_S + T_C}, \tag{5}$$

where s_X is the achieved saving in time component T_X. Thus the total saving increases linearly with both the saving in engineering and commissioning times, s_E and s_C.

6.2 Case Study

We focus on a case study inspired from the real industrial automation world. Particularly, we focus on the field and control room levels because we believe these are the most challenging components to be provided as a service. Consider an automation system for a large oil and gas plant. This system is characterized by the top part of Table 1, which is the standard method of characterization for designing automation systems. The case was carefully chosen to demonstrate a large plant and automation system. At the same time, the size is far from the extreme examples as some plants possess multiple the size of this case. A current automation system requires the software and hardware components listed in the lower part of Table 1.

Next, we discuss the impact on the required hardware and software components when adopting cloud-based automation systems. Software design of the cloud automation system follows that of the present-day system. Therefore, we assume the same block count estimated above in addition to extra blocks for delay compensation and distributed fault tolerance. We *conservatively* assume that the number of blocks increases by 50 percent, leading to 367,500 blocks. In the following, we show that if VMs of equivalent processing power are employed, then we need 300 VMs. The HMI/SCADA size remain unchanged in terms of number of displays, number of historian points, and other applications.

Bare-metal controllers will be replaced by VMs. Assuming that a dedicated VM with medium utilization can replace a physical controller, the number of VMs will be 50 percent more than the number of physical controllers since we have 50 percent increase in the number of software blocks for delay compensation and distributed fault tolerance. The I/O modules can be abandoned if all sensors and actuators are wireless TCP/Ethernet-enabled. However, to simplify our comparison, we assume traditional sensors and actuators but wireless TCP/Ethernet-enabled I/O modules with adequate intrinsic safety characteristics to be placed in or close to the field. We assume 50 percent increase in the cost per module due to the new wireless and intrinsic safety properties [26]. Cabinets are totally abandoned since the controllers are replaced by VMs, the I/O modules are scattered over the plant, and no need for marshaling since addressing is done by IP addresses, port numbers and channel numbers. VM-based controllers can

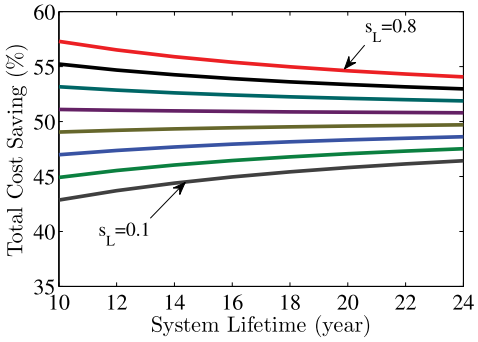


Fig. 8. Total cost saving under proposed cloud-based automation system for different values of s_L.

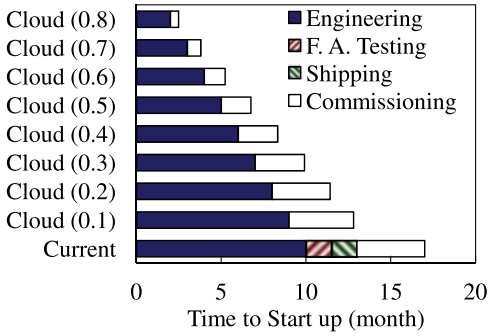


Fig. 9. TTSU saving under proposed system.

load the relevant addressing data of loop variables from a cloud-based database.

For the engineering and HMI/SCADA, all the servers and workstations are replaced by high-utilization dedicated VMs, in addition to thin clients. The workstation/server typically have multiple users logging through different accounts. While some thin clients can follow the same model, others are personal devices, e.g., cell phones. Therefore, we assume that the number of thin clients is more than the number of workstations/servers.

6.3 Cost/Time Saving Evaluation

Two samples of our analysis results are shown in Figs. 8 and 9. Fig. 8 shows the total cost saving under the proposed cloud system versus the system lifetime, computed according to Eq. (3). The figure contains eight lines, each line corresponds to a different engineering labor saving value, s_L. It is noteworthy that the total cost saving is at least 43 percent under 10 percent engineering labor reduction in a system with a lifetime of 10 years. As the system lifetime increases, the saving in running cost dominates as we spend more on running cost than upfront cost. Therefore, if the upfront cost saving is higher than the running cost saving, then the overall saving will decrease as in the case of $s_L = 0.8$. The opposite is also true as in the case of $s_L = 0.1$.

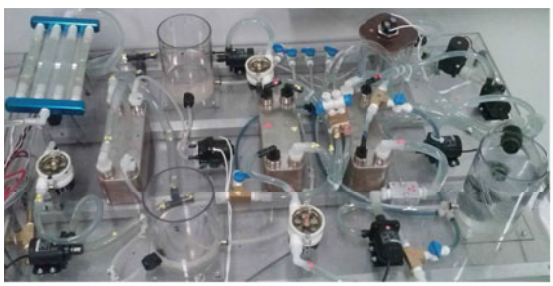
Next, we evaluate TTSU under the proposed system for different values of engineering time saving (s_E) and commissioning labor saving (s_C). In [15], we show that TTSU in the considered oil and gas plant example is 17 months using the current automation system. In contrast, Fig. 9 plots the TTSU values for a cloud-based automation system for a range of s_E (engineering time saving) values between 0.1 and 0.8. In each case, the commissioning time saving is conservatively set to $s_C = s_E/2$. As we can see from the figure, TTSU can drop to less than 13 months for $s_E = 0.1$ and to less than 2.5 months when $s_E = 0.8$, achieving time saving of 25 and 85 percent, respectively according to Eq. (5). We estimate that in practice s_S will fall in the range of 0.3 to 0.6, which, according to the figure, achieves a total saving in time in the range of about 40 to 70 percent. This means that an automation system development and setup can be accelerated by more than three times. More details are presented in Appendix A, available online.

7 Experimental Evaluation

In this section, we rigorously assess the performance of the proposed approach. We show how cloud-based controllers can effectively control an industrial plant that is more than 8,000 miles away. We also show that our approach can dynamically switch among redundant controllers upon failure to achieve smooth and reliable functioning of the controlled industrial plant and we compare our algorithms versus local controllers as they are the best known counterparts.

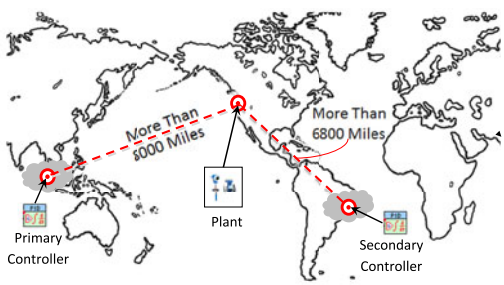
7.1 Experimental Setup

We designed and implemented a physical model (Fig. 10a) of the solar power plant presented in Fig. 1. We placed the physical model in our lab in Vancouver, Canada. For each plant process, we deployed two cloud controllers on the furthest (in terms of delay) Amazon cloud locations from our plant that we could find: Singapore and Sao Paulo, Brazil (Fig. 10c). In addition, to stress-test our approach, we



(a) Solar power plant physical model.

(b) TCP/Ethernet I/O module.



(c) Location of plant and cloud controllers.

Fig. 10. Experimental testbed: (a) Physical model of the solar power plant, (b) TCP/Ethernet-enabled I/O interface used to connect plant to cloud controllers, (c) Geographic locations of plant and its cloud controllers.

TABLE 2
Fault Tolerance Experimental Parameters

Parameter	Value	Parameter	Value
Sampling period (T_s)	200 ms	Flow set point	$0.69 + 0.35u(t-5)$ gal/min; $u(t)$ is unit step
Flow process time constant	2 s	Primary controller failure, recovery time instants	$t = 22$ s, $t = 78$ s, respectively
Average roundtrip delay	200 ms	Control algorithm, tuning method	PI, Ziegler-Nichols [38]

implemented an emulated version of the plant in LabVIEW, where we inject very high communication delays and disturbances. LabVIEW is a standard software for control and emulation in both automation industry and lab testing [22]. We evaluated our approach with the PID control method because it is, by far, the most commonly used in practice [6]. We deployed our cloud controllers according to the following stack:

- Packages: LabVIEW `PID and Fuzzy Logic`, `NI Modbus` packages.
- Control software: LabVIEW 2011 SP1 (32 bits).
- OS: Windows Server Datacenter.
- Cloud: Amazon Web Services EC2, regions: Asia Pacific (Singapore) and South America (Sao Paulo).

In additi on to the physical model of the solar power plant, Fig. 10b shows one of the four commercial I/O modules we used [2], [3]. The I/O modules convert the analog signals to sampled data wrapped in Modbus/TCP packets and vice versa, which is necessary for the sensors/actuators to communicate with the cloud controllers.

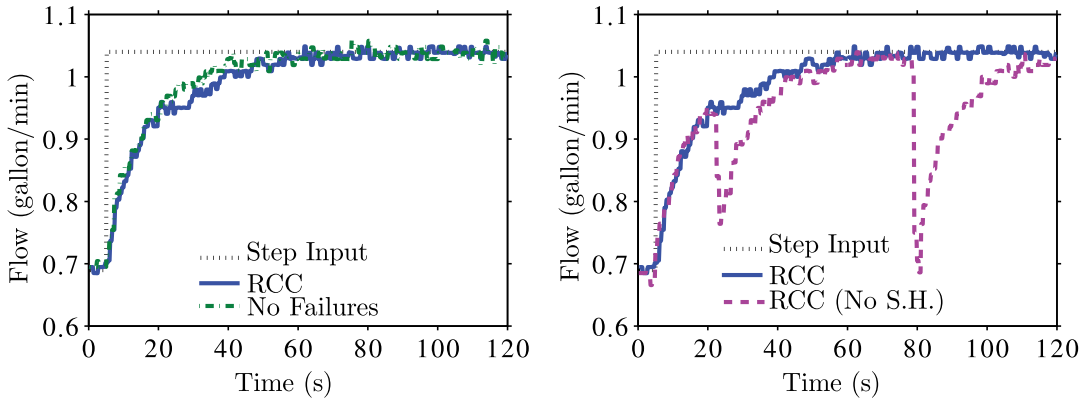
We will present performance results from representative control loops, which we introduced in Section 1.2; We derived the transfer functions of these loops and designed their PID cloud controllers using the Ziegler-Nichols method [38], and fine-tuned them by trial and error. The sampling period is typically set to 10 percent of the dominant time constant of the process [14]. Most continuous industrial processes have sampling periods in the range of 0.5 to 2.0 seconds [26]. We computed the dominant time constants for the control loops considered in the evaluation, and we *conservatively* set the sampling periods as 10 percent of these time constants with a maximum sampling period of 1 second. Smaller sampling periods stress our cloud-based control approach, as they require faster responses. Experimental parameters for each experiment are summarized in separate tables below.

7.2 Performance under Controller Failure

This section shows how cloud-based controllers can achieve (i) desired performance under real Internet delay and (ii) smooth handover in case of failures. We conduct the experiment with two redundant controllers; one placed in Singapore and the other in Sao Paulo, Brazil (Fig. 10c). We compare the time response of our RCC algorithm under failures to that of no failures. In our experiment, we introduce a step input to the flow process (FT1, FC1) shown in Fig. 1, while all plant processes (including other interacting flow processes) have been activated using step activation functions. The detailed experimental parameters are shown in Table 2.

We plot the response of the flow process in Fig. 11a against the set point step input (dotted line). Under only Internet delay and no failures, the dashed line shows that system reaches the steady state. This is the same result obtained when we used a local controller (not shown for clarity). The delay did not impact performance in this case, because the average measured delay was around 200 ms, i.e., much less than the loops time constant (around 2 s).

To examine our fault tolerance algorithm, we fail the primary controller by disconnecting it at $t = 22$ s, and return it back to operation at $t = 78$ s. These time instants are selected so that there is one handover event during the transient state and another during the steady state. The results show that the RCC algorithm successfully mitigated the failure and the performance appears as if there are no failures (solid line in Fig. 11a). Not employing RCC would leave the flow process at an intermediate value between $t = 22$ and $t = 78$, which is highly undesirable. Also, if the primary

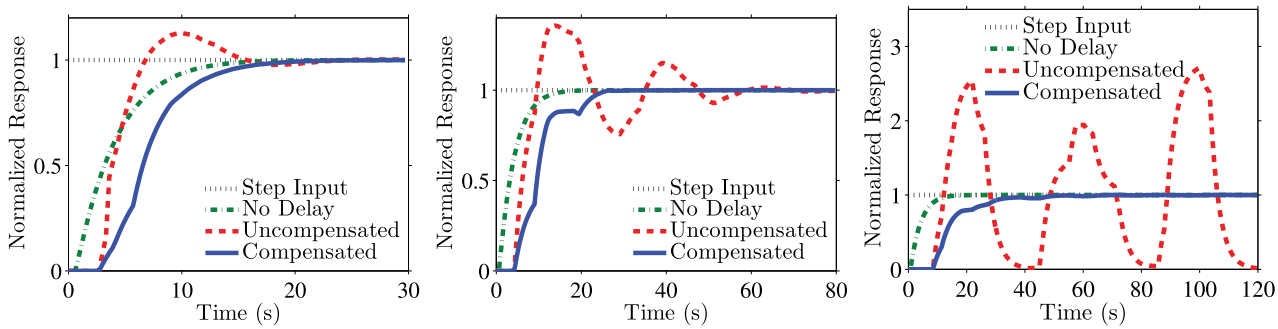


(a) Performance of RCC. (b) RCC with and without smooth handover.

Fig. 11. Robustness of cloud controllers under failures.

TABLE 3
Delay Compensation Experimental Parameters

Parameter	Value	Parameter	Value
Sampling period (T_s)	300 ms	Flow set point	unit step: $u(t)$ gal/min
Flow process time constant	3 s	Roundtrip delay (μ, σ, max)	(1, 0.7, 5), (2, 1.4, 10), (3, 2.1, 15), (4, 2.8, 20) seconds
Smith predictor delay formula	$\lfloor(\mu + 3\sigma)/T_s\rfloor$	Smith Predictor delay values	z^{-10}, z^{-20}, z^{-30} and z^{-40}
Delay generator	DS^2 [10]	Control algorithm, tuning method	PI, Ziegler-Nichols [38]



(a) μ =1 s, σ =0.7 s, *max*=5 s. (b) μ =2 s, σ =1.4 s, *max*=10 s. (c) μ =4 s, σ =2.8 s, *max*=20 s.

Fig. 12. Robustness of the proposed approach under large random delays.

controller took longer to return, this would lead to severe financial/safety risks.

To illustrate the importance of the smooth handover method, we conduct the same experiment while the smooth handover feature of RCC is disabled and plot the results in Fig. 11b. Controllers with no smooth handover capability ("RCC (No S.H.)") introduced two "bumps" in the process output during handover events. The negative impact of such phenomenon can range from compromising the efficiency of the controlled plant to damaging part or all of the plant. Thus, the smooth handover feature is important for providing the desired seamless performance under failure.

7.3 Performance under Variable Internet Delay

The observed real Internet delay in the previous section did not challenge our delay compensator because it was not significantly high. In this section, we show the performance under very large emulated delays. We consider the most common control-theoretic performance metrics [4]: (i) maximum overshoot percentage (M_p): normalized difference between the maximum overshoot and the final value; (ii) steady-state error (e_{ss}): difference between set point and final value of step response; and (iv) settling time (t_s): time taken by response to stay within 5% of final value.

We consider the flow process (FT3, FC3) with the experimental parameters shown in Table 3. Again, all plant processes are activated using step functions. To further stress-test the system, we artificially insert large random delay on top of the long Internet delay. We use delay distributions similar to the one in [10], with approximate values of (mean μ, standard deviation σ, and maximum max) of (100, 70, 500) ms, but we multiply the x-axis by a scaling factor to substantially increase the delay. We use scaling factors of 10, 20, 30 and 40 and appropriately scale the probability distribution so that the area under the curve remains equal to 1. This scaling yields the excessive delay values shown in Table 3. Thus, the inserted delay is up to two orders of magnitude larger than the sampling period. Further, highly variable delay causes packets to arrive out of order or even get lost under extreme delay values. TCP will ensure that lost packets are retransmitted and it will order the out-of-order packets. Such scenarios lead TCP to deliver several consecutive control actions to the Modbus protocol all at once. When Modbus receives such control actions, it will, in turn, write them to the actuator register in the I/O interface all at once. This is equivalent to executing only the last action and ignoring the rest of the "batch". Such scenario imposes even more challenge to our delay compensator.

For each delay distribution, we conduct an experiment with our delay compensator and another without it. We set the delay block in our delay compensator to $\lfloor(\mu + 3\sigma)/T_s\rfloor$, where T_s is the sampling period. Therefore, for the four delay distributions under consideration, the delay block is set to z^{-10}, z^{-20}, z^{-30} and z^{-40}, respectively. We compare the compensated and uncompensated performance to the zero-delay case as a baseline. To highlight the compensation benefit, we repeated each experiment several times and picked the worst performance for the compensated case, and the best performance in the uncompensated case.

We note that under large delays, the settling time has two components. The first component is due to the delay itself. The second component is due to the real distortion in the time response caused by the delay. To measure the real performance distortion, we define a metric called *pure settling time* as $t_{ps} = t_s - D_c$, where D_c is the value of the delay which the compensator is set to compensate for.

The results of this experiment are shown in Fig. 12 for the delay distributions plus the no delay case. The two distributions described by (μ, σ, max) = (3, 2.1, 15) and (4, 2.8, 20) yielded very similar results. Thus, to avoid redundancy, we only show the results of the latter distribution, and skip the

former. The figure shows that as we introduce larger delays, the "Uncompensated" cloud control loop overshoots (Figs. 12a and 12b) and eventually goes unstable (Figs. 12c). Whereas our method maintains smooth response with no apparent overshoots. Further discussion on the results are given in Appendix C, available online.

In summary, we tested our proposed system under extreme conditions: abrupt set point change under extremely large, variable delay, up to 20 s, i.e., 66 times the sampling period. Under such extreme conditions, our proposed cloud-based automation system kept the controlled process from overshooting or deviating from the final value.

7.4 Performance under Disturbance

In addition to the above results, we conducted several other experiments to show the robustness of the proposed cloud-based automation system under disturbances that may occur in real plants; details are given in Appendix D. We considered two control loops for two quite different processes (Fig. 10): (i) the solar collector positioning process marked by (AT1, AC1), and (ii) the temperature control process marked by (TT1, TC1) where the salt stores or pumps heat to regulate the oil temperature. We used LabVIEW to inject random and deterministic disturbances to these two processes. Our results confirmed that the performance of the proposed cloud controllers (deployed thousands of miles away) is indistinguishable from the performance of local controllers.

8 Conclusions

Cloud computing is proving beneficial to many real-life applications, including industrial automation. We proposed an architecture for offering industrial automation as a cloud service, which simplifies automation system design and saves time and cost. We focused on satisfying the timeliness and reliability requirements for feedback control as the most challenging industrial automation component. We presented and theoretically analyzed novel delay mitigation and distributed fault tolerance algorithms.

We evaluated our proposed approach using a physical model of a real-life plant, and deployed its controllers on the Amazon cloud. Our experimental results revealed that the our approach can effectively make the controlled plant unaware of controller/link failures, even when controllers are thousands of miles away from the plant. Further, we conducted emulation experiments with the purpose of injecting random and deterministic delays and disturbances. The results showed that our cloud-based controllers performed indistinguishably from local controllers, as the best known counterparts. By addressing the most challenging industrial automation layer, we have implicitly proved that the other layers are addressable. Hence, we can conclude that industrial automation can be offered as a cloud service.

Other than proving that cloud-based industrial automation is possible, another contribution of this paper is opening new horizons to employ cloud controllers in various scenarios, including:

- The soft real-time controllers used to manage many computing and communication systems, such as those designed for distributed caching [29], resource management in virtualized data centers [1], and video streaming [25], can now be offered as cloud services.
- Cloud controllers can act as backups for physical controllers of mission-critical systems. This is more economic than replicating all physical controllers.
- Cloud controllers can be used to temporarily manage systems while their physical controllers are being upgraded or replaced due to failures. This is inline with the on-demand nature of cloud services.
- Controllers can be deployed over private clouds to serve multiple facilities of the same company, consolidating automation functionalities in one data center.

Acknowledgments

This work was supported in part by the Natural Sciences and Engineering Research Council (NSERC) of Canada.

References

[1] T. Abdelzaher, Y. Diao, J. L. Hellerstein, C. Lu, and X. Zhu, "Introduction to control theory and its application to computing systems," in *Performance Modeling and Engineering*. Boston, MA, USA: Springer, 2008, ch. 7, pp. 185–215.

[2] Acromag, Inc. 972EN, 973EN Ethernet Analog Output Modules [Online]. Available: http://www.acromag.com/sites/default/files/972EN_973EN%20Ethernet%20Analog%20Output%20Modules.pdf, 2010.

[3] 968EN Ethernet Analog Input Modules [Online]. Available: http://www.acromag.com/sites/default/files/968EN%20Ethernet%20Analog%20Input%20Modules.pdf, 2010.

[4] M. Bandyopadhyay, *Control Engineering Theory and Practice*. New Delhi, India: Prentice-Hall, 2006.

[5] J. Bendtsen, J. Stoustrup, and K. Trangbaek, "Bumpless transfer between advanced controllers with applications to power plant control," in *Proc. IEEE Conf. Decision Control*, Dec. 2003, vol. 3, pp. 2059–2064.

[6] S. Bhattacharyya, A. Datta, and L. Keel, *Linear Control Theory: Structure, Robustness, and Optimization*. Boca Raton, FL, USA: CRC Press, 2009.

[7] J.-L. Boulanger, *Safety Management of Software-based Equipment*. Hoboken, NJ, USA: Wiley, 2013.

[8] Y. Chen, Z. Du, and M. Garcia-Acosta, "Robot as a service in cloud computing," in *Proc. IEEE Int. Symp. Serv. Oriented Syst. Eng.*, Jun. 2010, pp. 151–158.

[9] Z. Chen, L. Liu, and X. Yin, "Networked control system with network time-delay compensation," in *Proc. Ind. Appl. Conf.*, Oct. 2005, vol. 4, pp. 2435–2440.

[10] DS^2:Delay Space Synthesizer [Online]. Available: http://www.cs.rice.edu/~eugeneng/research/ds2/, 2006.

[11] Emerson. DeltaV Controller Redundancy [Online]. Available: http://www2.emersonprocess.com/siteadmincenter/PM%20DeltaV%20Documents/ProductDataSheets/PDS_DeltaV_ControllerRed.pdf, 2013.

[12] L. Desborough and R. Miller, "Increasing customer value of industrial control performance monitoring—Honeywell's experience," in *Proc. Preprint Chem. Process Control*, Jan. 2002, pp. 153–186.

[13] V. Gabale, P. Dutta, R. Kokku, and S. Kalyanaraman, "InSite: QoE-aware video delivery from cloud data centers," in *Proc. Int. Symp. QoS*, 2012, pp. 8:1–8:9.

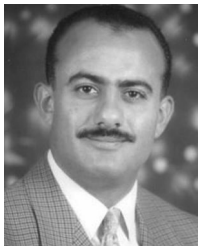
[14] M. Gopal, *Digital Control Engineering*. New Delhi, India: New Age International, 1998.

[15] T. Hegazy and M. Hefeeda. (2013, Jun.). The case for industrial automation as a cloud service, Tech. Rep. SFU-CMPT TR 2013-25-1, Simon Fraser University, Burnaby, BC, Canada [Online]. Available: http://nsl.cs.sfu.ca/techrep/SFU-CMPT_TR_2013-25-1.pdf

[16] T. Hegazy and M. Hefeeda. (2013, Sep.). Making industrial automation a cloud service, School Comput. Sci., Simon Fraser Univ., Burnaby, BC, Canada, Tech. Rep. SFU-CMPT TR 2013-25-3 [Online]. Available: http://nsl.cs.sfu.ca/techrep/SFU-CMPT_TR_2013-25-3.pdf

[17] U. Herrmann, B. Kelly, and H. Price, "Two-tank molten salt storage for parabolic trough solar power plants," *Energy*, vol. 29, nos. 5–6, pp. 883–93, Apr. 2004.
[18] K. Hiroyoshi, T. Hiroyuki, M. Hideo, and K. Kiyotaka. (2004). FFCS compact control station in CENTUM CS3000 R3, Yokogawa, Sugar Land, TX, USA, Tech. Rep. 38 [Online]. Available: http://www.yokogawa.com/rd/pdf/TR/rd-tr-r00038-002.pdf
[19] VMware, Inc. (2012). Wonderware system platform 2012 superior agility with VMware vSphere 5 [Online]. Available: http://www.vmware.com/resources/techresources/10314
[20] Invensys Operations Management. (2011). Tricon triple modular redundant (TMR) digital system for feedwater control and safety application in nuclear power plants [Online]. Available: http://iom.invensys.com/EN/pdfLibrary/ProductSpec_Triconex_Trident_03-10.pdf
[21] S. Kumar, S. Gollakota, and D. Katabi, "A cloud-assisted design for autonomous driving," in *Proc. SIGCOMM MCC Workshop Mobile Cloud Comput.*, 2012, pp. 41–46.
[22] National Instruments. (2009, Mar.). Why Use LabVIEW? [Online]. Available: http://www.ni.com/white-paper/8536/en/
[23] C. Lu, Y. Lu, T. Abdelzaher, J. A. Stankovic, and S. Son, "Feedback control architecture and design methodology for service delay guarantees in web servers," *IEEE Trans. Parallel Distrib. Syst.*, vol. 17, no. 7, pp. 1014–1027, Sep. 2006.
[24] NSF. (2012). NSF Report on Support for Cloud Computing [Online]. Available: http://www.nsf.gov/pubs/2012/nsf12040/nsf12040.pdf
[25] P. Patras, A. Banchs, and P. Serrano, "A control theoretic scheme for efficient video transmission over ieee 802.11e EDCA WLANs," *ACM Trans. Multimedia Comput. Commun. Appl.*, vol. 8, no. 3, pp. 29:1–29:23, Aug. 2012.
[26] "Private communication with senior automation engineers, project managers, site managers, and global technical managers serving the oil/gas industry," 2013.
[27] J. Rossiter, *Model-Based Predictive Control: A Practical Approach*, 2nd ed. Boca Raton, FL, USA: CRC Press, 2004.
[28] Siemens Industry Sector. (2009, Jul.). Smith Predictor for Control of Processes with Dead Times [Online]. Available: http://support.automation.siemens.com/WW/view/en/37361207
[29] G. Smaragdakis, N. Laoutaris, I. Matta, A. Bestavros, and I. Stavrakakis, "A feedback control approach to mitigating mistreatment in distributed caching groups," in *Proc. IFIP-TC6 Conf. Netw. Technol., Serv., Protocol*, 2006, pp. 331–343.
[30] O. Smith, "Closer control of loops with dead time," *Chem. Eng. Progress*, vol. 53, no. 5, pp. 217–219, May 1957.
[31] Honeywell Building Management. (2012). Attune™ Advisory Service [Online]. Available: https://buildingsolutions.honeywell.com/en-US/newsevents/resources/Publications/honeywell-hbs-service-attune%20advisory%20overview-brochure.pdf
[32] H. Wade, *Basic and Advanced Regulatory Control: System Design and Application*, 2nd ed. Research Triangle Park, NC, USA: ISA, 2004.
[33] L. Wang and K. Tan, *Modern Industrial Automation Software Design*. Hoboken, NJ, USA: Wiley, 2006.
[34] T. Williams, "Advances in industrial automation: Historical perspectives," in *Handbook of Automation*. New York, NY, USA: Springer, 2009, pp. 5–11.
[35] D. Wu, M. Greer, D. Rosen, and D. Schaefer, "Cloud manufacturing: Strategic vision and state-of-the-art," *J. Manuf. Syst.*, vol. 32, no. 4, pp. 564–579, 2013.
[36] S. Yang, X. Chen, L. Tan, and L. Yang, "Time delay and data loss compensation for internet-based process control systems," *Trans. Inst. Meas. Control*, vol. 27, no. 2, pp. 103–118, Jun. 2012.
[37] J. Yoo, Y. Zhou, S.-M. Lee, M. G. Joo, and J. H. Park, "An adaptive delay compensation technique for wireless sensor and actuator network," *Int. J. Smart Home*, vol. 6, no. 4, p. 187, Oct. 2012.
[38] J. Ziegler and N. Nichols, "Optimum settings for automatic controllers," *ASME Trans.*, vol. 64, pp. 759–68, 1942.

Tamir Hegazy received the MS and PhD degrees from Virginia Tech and Georgia Tech, in 2001 and 2004, respectively. He is currently with the ECE Research Faculty of Georgia Tech, where he also serves as a program manager for the Center of Energy and Geo Processing. He maintained a fine career balance between academia and industry in the period from 2005 to 2012. During this period, he held an adjunct assistant professor status at Virginia Tech. His position involved teaching, research, and administrative roles for Virginia Tech for Middle East and North Africa (VT-MENA), which serves as an extended campus of Virginia Tech in the region. He also held an assistant professor position at Mansoura University, Egypt. He simultaneously occupied several industry positions including research and development manager and global technical support manager at Invensys, a large, multinational automation firm. In 2012, he decided to focus on academic research. He was at Simon Fraser University, Canada between 2012 and 2013, where he researched real-time cloud computing services, and since 2013, he has been with Georgia Tech. He serves as a technical program committee member for IEEE conferences. His research interests include computer vision, human-computer interaction, real-time systems, industrial automation, and green energy. He is a senior member of the IEEE.



Mohamed Hefeeda received the BSc and MSc degrees from Mansoura University, Egypt, in 1997 and 1994, respectively. He received the PhD degree from Purdue University, in 2004. He is a professor in the School of Computing Science, Simon Fraser University, Canada, where he leads the Network Systems Lab. He is also a principal scientist in Qatar Computing Research Institute, Doha, Qatar. His research interests include multimedia networking over wired and wireless networks, peer-to-peer systems, mobile multimedia, and cloud computing. In 2011, he was awarded one of the prestigious NSERC Discovery Accelerator Supplements (DAS), which are granted to a selected group of researchers in all Science and Engineering disciplines in Canada. His research on efficient video streaming to mobile devices has been featured in multiple international news venues, including ACM Tech News, World Journal News, SFU NEWS, CTV British Columbia, and Omni-TV. He serves on the editorial boards of several premier journals such as the *ACM Transactions on Multimedia Computing, Communications and Applications (TOMM)*, and he has served on many technical program committees of major conferences in his research area, such as ACM Multimedia. He has coauthored more than 80 refereed journal and conference papers and has two granted patents. He is a senior member of the IEEE.