Improving System Reliability for Cyber-Physical Systems

Leon Li Wu

Submitted in partial fulfillment of the
Requirements for the degree
of Doctor of Philosophy
in the Graduate School of Arts and Sciences

COLUMBIA UNIVERSITY

2015

Abstract

Improving System Reliability for Cyber-Physical Systems

Leon Li Wu

Cyber-physical systems (CPS) are systems featuring a tight combination of, and coordination between, the system's computational and physical elements. Cyber-physical systems include systems ranging from critical infrastructure such as a power grid and transportation system to health and biomedical devices. System reliability, *i.e.*, the ability of a system to perform its intended function under a given set of environmental and operational conditions for a given period of time, is a fundamental requirement of cyber-physical systems. An unreliable system often leads to disruption of service, financial cost and even loss of human life. An important and prevalent type of cyber-physical system meets the following criteria: processing large amounts of data; employing software as a system component; running online continuously; having operator-in-the-loop because of human judgment and an accountability requirement for safety critical systems. This thesis aims to improve system reliability for this type of cyber-physical system.

To improve system reliability for this type of cyber-physical system, I present a system evaluation approach entitled *automated online evaluation (AOE)*, which is a data-centric runtime monitoring and reliability evaluation approach that works in parallel with the cyber-physical system to conduct automated evaluation along the workflow of the system continuously using computational intelligence and self-tuning techniques and provide operator-in-the-loop feedback on reliability improvement. For example, abnormal input and output data at or between the multiple stages of the system can be detected and flagged

through data quality analysis. As a result, alerts can be sent to the operator-in-the-loop. The operator can then take actions and make changes to the system based on the alerts in order to achieve minimal system downtime and increased system reliability. One technique used by the approach is *data quality analysis using computational intelligence*, which applies computational intelligence in evaluating data quality in an automated and efficient way in order to make sure the running system perform reliably as expected. Another technique used by the approach is *self-tuning* which automatically self-manages and self-configures the evaluation system to ensure that it adapts itself based on the changes in the system and feedback from the operator. To implement the proposed approach, I further present a system architecture called *autonomic reliability improvement system (ARIS)*.

This thesis investigates three hypotheses. First, I claim that the automated online evaluation empowered by data quality analysis using computational intelligence can effectively improve system reliability for cyber-physical systems in the domain of interest as indicated above. In order to prove this hypothesis, a prototype system needs to be developed and deployed in various cyber-physical systems while certain reliability metrics are required to measure the system reliability improvement quantitatively. Second, I claim that the self-tuning can effectively self-manage and self-configure the evaluation system based on the changes in the system and feedback from the operator-in-the-loop to improve system reliability. Third, I claim that the approach is efficient. It should not have a large impact on the overall system performance and introduce only minimal extra overhead to the cyber-physical system. Some performance metrics should be used to measure the efficiency and added overhead quantitatively.

Additionally, in order to conduct efficient and cost-effective automated online evaluation for data-intensive CPS, which requires large volumes of data and devotes much of its processing time to I/O and data manipulation, this thesis presents COBRA, a cloud-based reliability assurance framework. COBRA provides automated multi-stage runtime reliability evaluation along the CPS workflow using data relocation services, a cloud data store, data

quality analysis and process scheduling with self-tuning to achieve scalability, elasticity and efficiency.

Finally, in order to provide a generic way to compare and benchmark system reliability for CPS and to extend the approach described above, this thesis presents FARE, a reliability benchmark framework that employs a CPS reliability model, a set of methods and metrics on evaluation environment selection, failure analysis, and reliability estimation.

The main contributions of this thesis include validation of the above hypotheses and empirical studies of ARIS automated online evaluation system, COBRA cloud-based reliability assurance framework for data-intensive CPS, and FARE framework for benchmarking reliability of cyber-physical systems. This work has advanced the state of the art in the CPS reliability research, expanded the body of knowledge in this field, and provided some useful studies for further research.

Contents

List of Figures

List of Tables

Acknowledgments

I am grateful to my advisor Prof. Gail E. Kaiser for her guidance and support along my graduate study at Columbia University. I also would like to thank my PhD thesis committee members (Prof. Gail E. Kaiser, Prof. Christian Murphy, Prof. Roger N. Anderson, Prof. Luca Carloni, and Prof. Stephen A. Edwards) for their valuable suggestions and help. Furthermore, I would like to thank the members of the Programming Systems Laboratory, the Department of Computer Science, the Center for Computational Learning Systems at Columbia University, the PENSA Laboratory at the Department of Operations Research and Financial Engineering at Princeton University, and the Prediction Analysis Laboratory at MIT Sloan School of Management for their assistance in my research.

The Programming Systems Laboratory is funded in part by NSF CCF-1302269, CCF-1161079, NSF CNS-0905246, and NIH U54 CA121852. The Center for Computational Learning Systems is supported in part by Finmeccanica, Rudin Management Company, Consolidated Edison, U.S. Department of Energy, General Electric, FedEx, and New York State Energy Research and Development Authority.

To my parents

Chengfa Wu & Genxiang Wu

who gave me unconditional love.

Chapter 1

Introduction

Cyber-physical systems (CPS) are engineered systems that are built from, and depend upon, the seamless integration of computational algorithms and distributed physical components [163]. Unlike small, single-sourced embedded systems, modern cyber-physical systems incorporate components from different providers using explicit interface standards that specify communication protocols, physical operation characteristics, real-time sensing and human operators informed by real-time data from the cyber-physical sensors.

As Prof. Edward A. Lee at UC Berkeley explains, "Applications of CPS arguably have the potential to dwarf the 20-th century IT revolution. They include high confidence medical devices and systems, assisted living, traffic control and safety, advanced automotive systems, process control, energy conservation, environmental control, avionics, instrumentation, critical infrastructure control (electric power, water resources, and communications systems for example), distributed robotics (telepresence, telemedicine), defense systems, manufacturing, and smart structures. It is easy to envision new capabilities, such as distributed micro power generation coupled into the power grid, where timing precision and security issues loom large. Transportation systems could benefit considerably from better embedded intelligence in automobiles, which could improve safety and efficiency. Networked autonomous vehicles could dramatically enhance the effectiveness of our military and could offer substantially

more effective disaster recovery techniques. Networked building control systems (such as HVAC and lighting) could significantly improve energy efficiency and demand variability, reducing our dependence on fossil fuels and our greenhouse gas emissions [134]."

System reliability, *i.e.*, the ability of a system to perform its intended function under a given set of conditions for a period of time, is widely recognized as a critical requirement for cyber-physical systems. An unreliable system often leads to disruption of service, financial cost and even loss of human life. Ideally, cyber-physical systems should not be deployed into certain mission critical applications such as traffic control, automotive safety and health care without improved reliability and predictability [134]. Barnum *et al.* put together a roundtable discussion on the reliability of embedded and cyber-physical systems [24]. It lists "a precursor generation of cyber-physical systems in areas as diverse as aerospace, automotive, chemical processes, civil infrastructure, energy, healthcare, manufacturing, transportation, entertainment, and consumer appliances" and "a complicated set of quality-based attributes (dependable, trustworthy, available, maintainable, fault-tolerant, robust, failure immune, secure, confidential, data integrity, safe, resilient, reliant, and several others) that are layered on top of a highly complicated base system."

It is a daunting challenge to make sure a system's computational and physical elements perform and maintain their functions in both routine circumstances and in hostile and unexpected circumstances. Many cyber-physical systems deployed in the field are evidently and even increasingly unreliable. For example, the electric power grid cyber-physical system has become less reliable and more outage-prone in the past years. According to two data sets, one from the U.S. Department of Energy and the other one from the North American Electric Reliability Corp., the number of power outages greater than 100 Megawatts or affecting more than 50,000 customers in the U.S. has nearly doubled every five years over the past fifteen years, resulting in about $49 billion in outage costs per year [8].

1.1 Definitions

Before I further discuss the problem statement, requirements, and approach, this section first formalizes some of the terms used throughout this thesis. Some of these definitions are similar to what is defined in the ANSI/ISO/ASQ standards [105, 14].

- A *cyber-physical system (CPS)* is a system featuring a tight combination of, and coordination between, the system's computational and physical elements. The applicable domains of cyber-physical systems include critical infrastructure such as power grids and highway transportation systems, health and biomedical systems, energy and industrial automation systems, automated defense and combat systems, and agricultural automation systems [54].

- *Failure* is the inability of a system or component to perform its required function within the specified performance requirements [13]. It is the manifestation of a fault in the system or human error.

- *System reliability* is defined as the ability of a system to perform its intended function under a given set of environmental and operational conditions for a given period of time. It includes all parts of the system, including hardware, software, supporting infrastructure, operators and procedures. For quantitative comparison, it is often reported as a probability.

- *Component reliability* is the ability of a component to perform a required function under stated conditions for a period of time [105]. The term 'reliability' is also used as a reliability characteristic denoting a probability of success or a success ratio.

- *Software reliability* is the probability of failure-free software operation for a specified period of time in a specified environment [13]. For cyber-physical systems, software reliability is an integral part of system reliability.

- *Robustness* is the ability of a computer system to cope with errors during execution. Robustness can also be defined as the ability of an algorithm to continue operating despite abnormalities in input, calculations, etc. In this thesis, robustness is considered one of the aspects of the overall system reliability.

- *Fault* or *defect* is an incorrect step, process, or data definition in a system [13]. In a software system, this can be a programming or design error that leads to an erroneous result during execution.

- *Fault density* is the number of faults, usually expressed as faults per thousand lines of code. This is a common software reliability metric.

- *Software bug* is the common term used to describe a fault in a software program that produces an incorrect or unexpected result, or causes it to behave in unintended ways.

- *Error* is the difference between a computed, observed, or measured value or condition and the true, specified, or theoretically correct value or condition.

- *Data quality* is an assessment of data's fitness to serve its purpose in a given context. Some aspects of data quality include: accuracy, completeness, update status, relevance, consistency across data sources, appropriate presentation, and accessibility [231, 229, 181, 116, 170, 218]. Cyber-physical systems' data quality is an important factor for its system reliability.

- *Autonomic* is a system characteristic that means being able to monitor its operational context as well as its internal state and being able to control and change its internal operations (*i.e.*, its configuration, state and functions) [122]. The purpose is to assess whether or not the system's current operation serves its purpose and possibly make self-adjustments accordingly.

- *Operator-in-the-loop* or *human-in-the-loop* is defined as a model that requires human interaction [121]. This allows the user to change the outcome of an event or process.

- An *actuator* is a component or separate device that is responsible for taking physical actions on cyber-physical systems. It can be a type of motor that is responsible for moving or controlling a mechanism or equipment.

1.2 Problem Statement

It is difficult to make cyber-physical systems reliable. Modern cyber-physical systems are becoming more and more complex and distributed, incorporating components from different providers using explicit interface standards that specify communication protocols, physical operation characteristics, real-time sensing and human operators informed by real-time data from the cyber-physical sensors [163]. The increasing complexity and the trend towards distribution make it difficult to ensure all parts of the system, including hardware, software, supporting infrastructure, operators and procedures, work together reliably, especially for those mission-critical online systems, *e.g.*, electric power grid, that require continuous system uptime.

Also, cyber-physical systems deployed in the field have to run in live environments, which are not controlled. The running environments are often unpredictable with abnormal conditions, thus posting challenges for cyber-physical systems to run smoothly.

Furthermore, for those deployed cyber-physical systems that process large amounts of real-time data on the fly, erroneous data is a significant problem when it comes to system reliability. For example, a data sensor may record some extreme measurements due to a power surge or an electronic component failure. The erroneous input data and incorrect output data from software components may lead to failure of subsequent system components, causing system malfunction and execution disruption. It is a challenge to ensure the data quality such as consistency and accuracy for better system reliability.

1.3 Requirements

A solution to these problems must meet the following requirements:

1. The approach should be able to improve system reliability for cyber-physical systems to ensure their reliable execution.

2. The system reliability improvement brought by the approach should be able to be measured and verified quantitatively and compared to the baseline derived from the system without implementation of the approach.

3. The approach should be able to process large amounts of real-time system data, deal with erroneous data input and abnormal data output from software and other components, and derive useful information from them intelligently for reliability analysis and improvement.

4. The approach should be efficient and not add significant overhead to the existing systems. The overhead caused by the approach should be measured quantitatively.

1.4 Scope

One example of a cyber-physical system are the energy control systems, in which the sensors and actuators physically monitor and control the energy processes; the computer-based systems analyze and store data; and the communication networks interconnect the process and computer systems [166]. Another example are the defense systems that are more attuned to their environments, receiving and processing massive amounts of data, to determine courses of action [166]. These systems will not be operating in a controlled environment, and must work reliably and continuously under the unexpected conditions and the subsystem failures [54]. Also, these systems commonly employ operator-in-the-loop because of human judgment and accountability requirement for safety critical systems. It is often not possible

to perform robust tests on these cyber-physical systems prior to actual deployment because the physical devices are so expensive that they cannot be replicated in a testing lab and the external environments are impossible to replicate. It is imperative to have an online reliability evaluation and improvement process to ensure the live system in the field is running as expected.

This thesis is limited to cyber-physical systems in these domains and aims to improve system reliability for cyber-physical systems that meet following criteria:

- Processing large amounts of system data including input data and output data from different components such as data collected from sensors and output from software
- Employing software as a system component, *e.g.*, software components used to support or control other system components
- Running online continuously, which demands as few failures as possible and the minimum system downtime during each failure
- Having operator-in-the-loop, *i.e.*, operator can take actions on the system to make changes to the system execution

These types of cyber-physical systems are important and becoming more prevalent [54]. Systems that meet these criteria include smart power grids, smart building systems, highway transportation systems, defense systems, and factory automation systems.

1.5 State of the Art

CPS Summit 2008 at Carnegie Mellon University stated in its executive summary that architectures and tools are needed in order to build reliable and resilient cyber-physical systems [54]. The report also described software reliability affects the overall system reliability. For architectural design of reliable cyber-physical systems, Sha *et al.* proposed

a hybrid approach that combines fault-tolerant architectures with formal verification to support the design of safe and robust cyber-physical systems [203]. La *et al.* proposed a service-based cyber-physical system based on service-oriented architecture (SOA) to achieve dynamic composition, dynamic adaptation, and high confidence [127]. Göhringer *et al.* described an approach of a reliable and adaptive network-on-chip architectures (NoC) for CPS such as FPGA-based system [88]. Sanislav and Miclea presented the analysis-to-design procedure for the event-driven multi-agent model development for CPS with dependability features [199]. One real-world constraint of these approaches is that many cyber-physical systems such as power grids have expansive infrastructure already built and these legacy systems are often too hard and expensive to replace. Additionally, the running environments of cyber-physical systems are often unpredictable and simulating all kinds of abnormal conditions in a test environment is almost impossible. Improving the reliability of these systems entails working with the software, hardware and physical devices that have already been deployed.

Another research direction is to achieve CPS reliability through workflow design. Leonardi *et al.* proposed a methodology, and its embodiment into a design flow, to realize execution platforms for high-performance building applications, in order to solve the design-space exploration problems by progressing through a sequence of refinement steps from specification to detailed implementation [135]. Wang *et al.* proposed a reliable workflow for CPS service substitution according to service compatibility and time/space operation in order to ensure CPS components' reliable replacement [230]. These workflow-based approaches also do not deal with deployed cyber-physical systems running in the field.

Some prior work has been done on monitoring wireless sensor network (WSN), a type of CPS, because of the sub-optimal end-to-end reliability that is intrinsic to wireless technologies. Bapat *et al.* presented a stabilizing protocol called Chowkidar that provides accurate and efficient network health monitoring in wireless sensor networks using message-passing

rooted spanning tree construction and its use in propagation of information with feedback (PIF) [23]. Doherty and Teasdale described a centralized monitoring time division multiple access (TDMA) network with policies chosen to maximize the number of received packets while maintaining low power characteristics using a method for detecting and diagnosing packet loss [63]. These monitoring techniques are targeting certain communication protocols specific to wireless network and do not provide a mechanism based on data-centric evaluation.

For cyber-physical transportation systems, Clarke *et al.* proposed some of the demanding challenges in applying formal analysis technique on autonomous transportation control for cars, trains, and aircraft. Their paper listed scalable analysis with respect to complexity and dimensionality, large-scale verification architectures, dynamic networks, probabilistic effects in cyber-physical transportation as some of the main challenges [46]. Their work explained the difficulty of applying formal analysis on the complex cyber-physical systems.

For electric power grid cyber-physical systems, Singh *et al.* concluded that the current techniques for power system reliability are insufficient because they focus mainly on the current carrying part of the power grid with some work done in the inclusion of protection systems. The paper also points out that the literature on the reliability of the cyber part is practically non-existent and the analysis of the power system as a cyber-physical system appears to be a challenging task because of the dimensionality and complexity issues [207]. Their work suggested that a holistic evaluation approach encompassing all parts of the cyber-physical systems is needed.

Security for cyber-physical systems has also been an important research topic in recent years. Vaseashta *et al.* described some vulnerabilities and countermeasures for sensor network, a type of cyber-physical system [226]. Walters *et al.* gave a general overview on wireless sensor network security: obstacles, requirements, attacks, and defenses [228]. An unreliable system may pose more security vulnerabilities that can be exploited by malicious attackers. “A system can’t be reliable if it’s not secure, and to some degree, if it’s not reliable,

at least in a security context, it can't be secure, either [24]." While the focus of this thesis is not to target potential security issues or defend the possible malicious attacks, instead I try to improve system reliability for the cyber-physical system so that the system can run reliably with or without a security attack.

1.6 Proposed Approach

To solve the problems mentioned in section 1.2, I propose a system evaluation approach entitled *automated online evaluation (AOE)*, as illustrated in Figure 1.1, that is able to improve reliability for cyber-physical systems effectively and efficiently in the domain of interest as indicated above. AOE is a data-centric runtime monitoring and reliability evaluation approach that works in parallel with the cyber-physical system to conduct automated evaluation at the multiple stages along the workflow of the system, continuously using computational intelligence and self-tuning techniques and provide operator-in-the-loop feedback for taking actions to achieve reliability improvement. For example, abnormal input and output data can be detected and flagged through data quality analysis. As a result, alerts can be sent to the operator-in-the-loop. The operator can then take actions and make changes to the system based on the alerts in order to achieve minimal system downtime and higher system reliability.

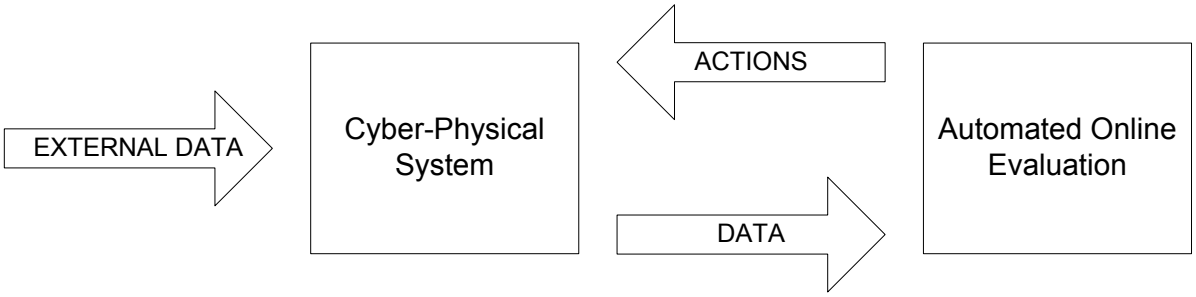


Figure 1.1: Automated online evaluation, a "big data" approach.

The evaluation is online, which differs from many statically analyzed systems that often

employ a pre-deployment or postmortem evaluation and analysis, although they can be used to compare results achieved by my approach. The evaluation is also autonomic because it works in parallel with the cyber-physical system to automatically alert the operator when abnormal events happen and it is able to self-tune the evaluation system adaptively. To implement the proposed approach, I further propose a system architecture called *Autonomic Reliability Improvement System (ARIS)*.

One technique used by the approach is *data quality analysis using computational intelligence* that applies computational intelligence in evaluating data quality in an automated and efficient way to ensure data quality and make sure the running system perform as expected. It employs machine learning, data mining, statistical and probabilistic analysis, and other intelligent techniques. In a cyber-physical system, the data collected from the system, *e.g.*, input and output data, software error reports, system status logs and performance reports, are stored in some databases. This data is analyzed via data mining and other intelligent techniques so that useful information on system reliability including erroneous data and abnormal system states can be analyzed for recommended actions. This reliability related information is directed to operators so that proper actions can be taken, sometimes proactively based on the predictive results, in order to ensure the proper and reliable execution of the systems.

Another technique used by the approach is *self-tuning* which automatically self-manages and self-configures the evaluation system itself to ensure its proper functioning, which leads to a more robust evaluation system and hence improved system reliability. For example, the self-tuning adapts the evaluation system based on the changes in the system such as software models or thresholds and feedback from the operator to ensure that the evaluation system runs reliably.

Additionally, in order to conduct efficient and cost-effective automated online evaluation for data-intensive CPS, which requires large volumes of data and devotes much of its processing time to I/O and data manipulation, I propose COBRA, a cloud-based reliability

assurance framework that provides automated multi-stage runtime reliability evaluation along the system workflow using data relocation service, cloud data store, parallel data quality analysis and process scheduling with self-tuning to achieve scalability and efficiency.

Finally, in order to provide a generic way to compare and benchmark system reliability for CPS and to extend the approach described above, I propose FARE, a generic framework for benchmarking the reliability of cyber-physical systems with or without implementation of the automated online evaluation. The framework provides a generic CPS reliability model, a set of methods and metrics on evaluation environment selection, failure analysis, and reliability estimation for benchmarking CPS system reliability. It not only provides a retrospective evaluation and estimation of the CPS system reliability using the past data, but also provides a mechanism for continuous monitoring and evaluation of CPS system reliability for runtime enhancement.

1.7 Hypotheses

In this thesis, I will present the research for following hypotheses.

The foremost hypothesis is that **the approach is effective**, *i.e.*, the automated online evaluation empowered by data quality analysis using computational intelligence can work effectively to improve system reliability for cyber-physical systems in the domain of interest as indicated above. In order to prove this hypothesis, a prototype system needs to be developed and deployed in complex cyber-physical systems to measure the system reliability improvement that the approach brings to the system.

The second hypothesis is that **the self-tuning can effectively self-manage and self-configure the evaluation system based on the changes in the system and feedback from the operator-in-the-loop to improve system reliability**.

The third hypothesis is that **the approach is efficient and does not add too much overhead**. The evaluation system should not have a large impact on the overall system

performance and only introduce minimal extra overhead to the cyber-physical system.

The main objective of this thesis research is to validate the hypotheses outlined above by exploring the automated online evaluation approach for improving system reliability for CPS in the domain of interest. The research will advance the state-of-the-art research in system reliability for cyber-physical systems not only in its novel architectural design and capability in improving system reliability, but also in the new techniques developed and employed.

1.8 Outline

The rest of this thesis is organized as follows:

- Chapter 2 describes motivation, previous work and the systems in the domain of interest.
- Chapter 3 describes automated online evaluation (AOE) and an example of AOE architecture called ARIS (autonomic reliability improvement system), along with empirical studies of the approach on the smart power grids and the smart building systems.
- Chapter 4 describes a cloud-based reliability assurance framework called COBRA for data-intensive CPS and the experiments on the smart building systems.
- Chapter 5 describes a reliability benchmark framework called FARE and the experiments on the smart building systems.
- Chapter 6 describes further related work and compares them with my approach.
- Chapter 7 summarizes the main contributions of this work, discusses future research directions, and concludes the thesis, followed by a bibliography and appendix.

Chapter 2

Background

In this chapter, I will describe motivation and the systems in the domains of interest.

2.1 Motivation

This line of research began with work in which I addressed the data quality assurance and performance measurement of machine learning and data mining for *preventive maintenance*, *i.e.,* maintenance of equipment or systems before fault occurs, of power grid [242]. A power grid is the electricity distribution and transmission system that connects electricity generators and consumers. It is a power and information network consisting of power plants, transformers, high-voltage long-distance power transmission lines, substations, feeders, low-voltage local power lines, meters, and consumer appliances.

One of the main causes of power grid failure is electrical component failure. The electrical component failures may even lead to catastrophic cascading system failures. In 2004, the U.S.-Canada Power System Outage Task Force released their final report on the 2003 U.S. Northeast blackout, naming some strained high-voltage power lines in Ohio as the main cause of the blackout. These power lines later went out of service, which led to the cascading effect that ultimately forced the shutdown of more than 100 power plants [80].

For cyber-physical systems with high reliability requirements such as critical infrastruc-

tures including a power grid, preventive maintenance is often employed to improve system reliability. In New York City, underground primary feeders are one of the most failure-prone types of electrical components. To ensure the power grid is running smoothly, the electrical components including feeders, which are a type of transmission lines, are proactively taken offline for maintenance or replacement according to their susceptibility to failure. In order for power companies to benefit from the use of knowledge discovery methods and statistical machine learning for preventive maintenance, Rudin *et. al* introduced a general process for transforming historical electrical grid data into models that aim to predict the risk of failures for components and systems [197]. These models can be used directly by power companies to assist with prioritization of maintenance and repair work.

2.2 Systems in the Domain of Interest

Typical applications of CPS with operator-in-the-loop include sensor-based systems and intelligent control systems. Sensor-based systems such as smart building management systems and wireless sensor networks utilize many distributed sensors to measure and collect system or environmental data and transmit this information to a centralized system for processing. Intelligent control systems utilize some mechanisms to make intelligent decisions and use them to manage or operate the systems. Some examples of intelligent control systems include smart power grid operation control systems, railway transportation control systems, air traffic control systems, industrial process control systems, and distributed robotics.

These systems are different from CPS without operator-in-the-loop, such as *avionics*, which are the electronic systems used on aircraft, artificial satellites, and spacecraft. Avionic systems include communications, navigation, the display and management of multiple systems, and the varied systems that are fitted to aircraft to perform individual functions. Because these systems perform their functions without operator-in-the-loop, they do not fall

into the scope of this study.

In this section, I will provide background information on several systems in the domain of interest including smart power grid CPS and smart building CPS. These are the cyber-physical systems used in my experiments.

2.2.1 Smart Power Grid CPS

A power grid is a typical continuously running cyber-physical system that processes large amount of data, uses software as a system component, and has operator-in-the-loop. In the last few years, the power grid has been transitioning to *smart grid*, which is an automated electric power system that monitors and controls grid activities, ensuring the two-way flow of electricity and information between power plants and consumers—and all points in between [79]. As one of the critical infrastructures, a smart grid puts information and communication technology into electricity generation, delivery, and consumption, making systems cleaner, safer, and more reliable and efficient [47]. Without the smart grid, many emerging clean energy technologies such as electric vehicles and solar, wind or cogeneration power could not be adopted on a large scale [10].

A typical smart grid is illustrated in Figure 2.1. It consists of power generation, green energy sources, transmission networks, transformers, substations, electricity consumers, plug-in electric vehicle charging stations, grid control centers, and other electrical and intelligent systems. Smart meters are installed at various joints and customer locations to provide accurate reporting of the electricity consumption and power network status.

It is a critical challenge to ensure power grid reliability. In fact, the power grid has become less reliable and more outage-prone in past years. According to two data sets, one from the U.S. Department of Energy and the other one from the North American Electric Reliability Corp., the number of power outages greater than 100 Megawatts or affecting more than 50,000 customers in the U.S. nearly doubled every five years during the past fifteen years, resulting in about $49 billion in outage costs per year [8].

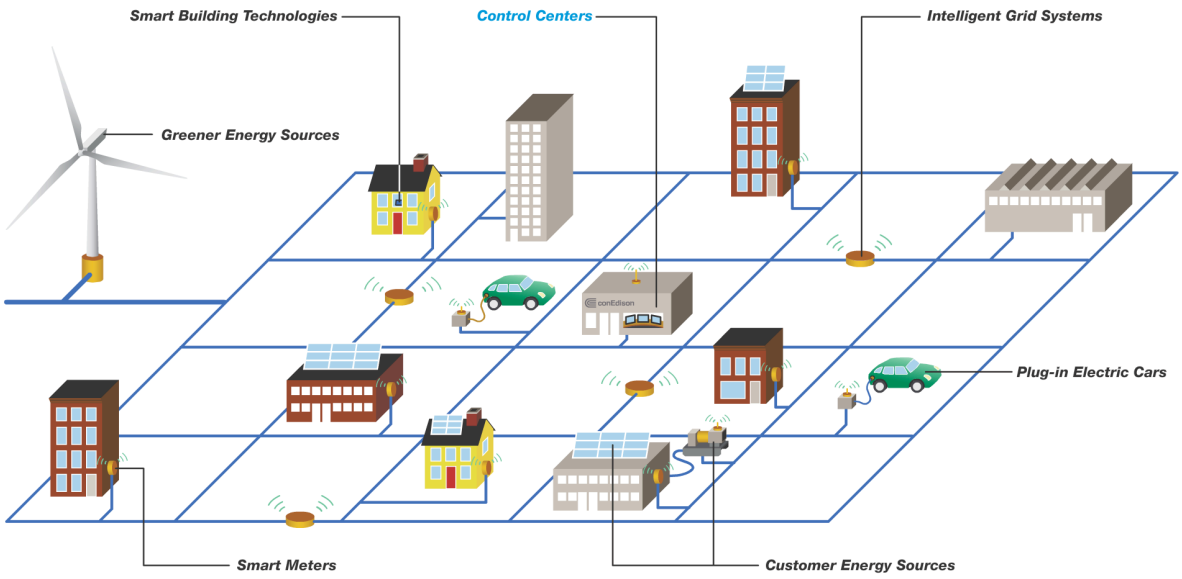


Figure 2.1: A smart grid [47].

One of the causes of the power grid failure is electrical component failure. The smart grid of the future will have to operate efficiently in order to satisfy the increasing capacity demand, and should use the current legacy grid as much as possible to keep costs lower. The legacy grid often contains old and unreliable electrical components.

To tackle this electrical component failure problem, researchers at Columbia University have collaborated with the Consolidated Edison of New York, the main power utility provider of New York City, and developed several machine learning and data mining systems to rank some types of electrical components such as feeders, *i.e.*, transmission lines with radial circuit of intermediate voltage, by their susceptibility to impending failure. MartaRank [91, 25] and ODDS [197, 90] are two of these feeder-ranking systems. The rankings are then used for planning fieldwork aimed at preventive maintenance, where the components are proactively inspected and/or repaired in order of their estimated susceptibility to failure [197, 205, 72, 177, 247, 176, 195, 196, 91, 175, 67].

MartaRank employs an ensemble-based approach using three different algorithms for

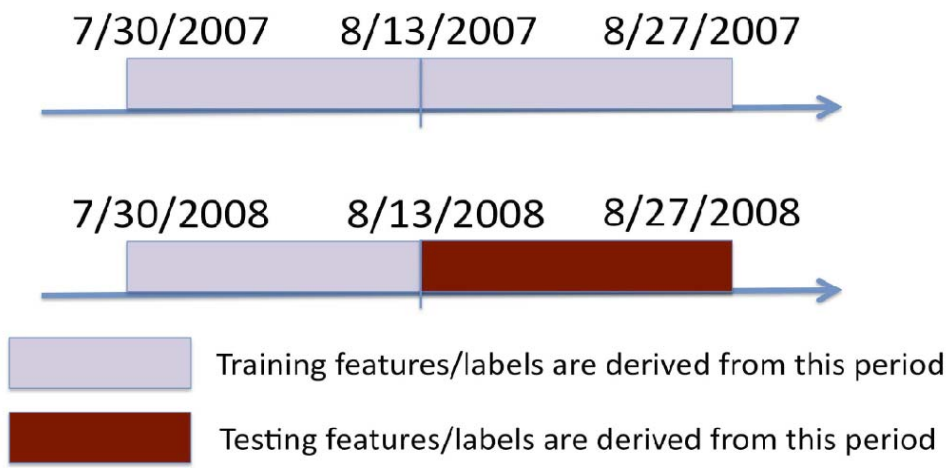


Figure 2.2: Example illustrating the training and test time windows in ODDS. The current time is 8/13/2008, and failure data for training was derived from the prediction period of 7/30/2007–8/27/2007 and 7/30/2008–8/13/2008 [197].

ranking: (1) SVM Score Ranker, which ranks objects by sorting the decision values of a linear Support Vector Machines (*SVM*) [225]. Typically, the SVM produces a classifier that labels examples x with $y = sign(w^t x + b)$, but MartaRank does not threshold the outputs so it can sort and rank the examples by how strongly the linear classifier predicts the class of each example. (2) RankBoost, which is a set of weak rankings that can be "boosted" to obtain a final ranking [81]. (3) MartiRank, which is a boosting-style ranking algorithm [143] that greedily selects the attribute that is most correlated with the positive examples (in this case power outages).

The ODDS, which stands for Outage Derived Data Sets, ranking system uses ranked lists obtained from a linear SVM. ODDS captures the dynamic data from right before the failure using "time-shifted" positive examples, as illustrated in Figure 2.2, where the positive examples are still created from the past outages within the prediction period, but the dynamic features are derived only from the time period shortly before the failure happened. It helps to reduce the imbalance between positive and negative examples. ODDS also creates a new model every 4 hours on the current data set to address the pervasive "concept drift,"

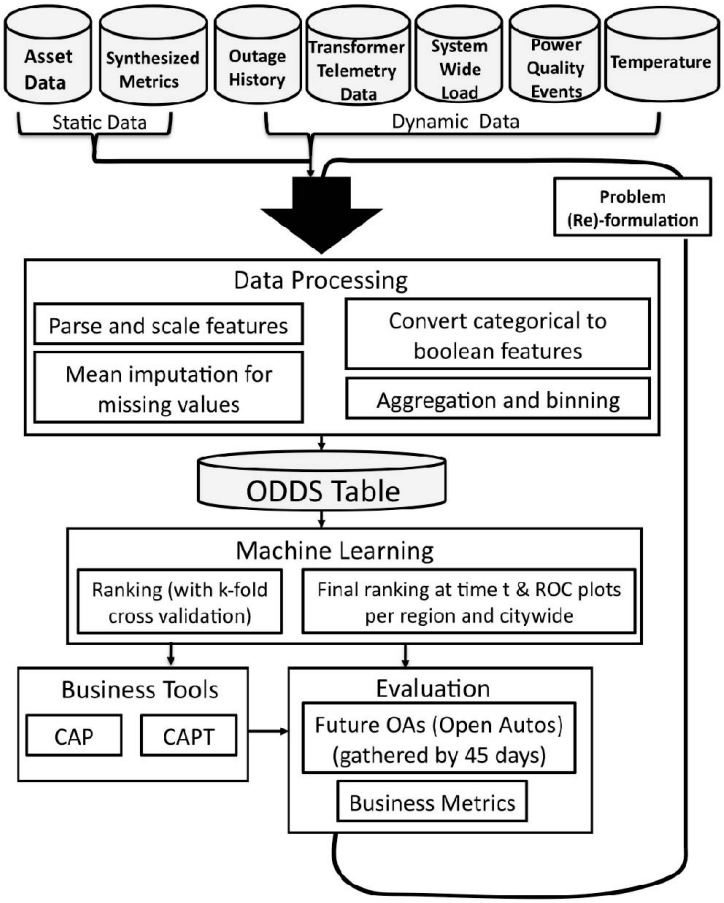


Figure 2.3: Process diagram of ODDS feeder ranking ML system [197].

meaning that patterns of failure change fairly rapidly over time, so that a machine learning model generated on data from the past may not be completely representative of future failure patterns. Features can become inactive or change in quality. Causes of this include: repairs being made on components, causing the nature of future failures to change; new equipment having different failure properties than current equipment; and seasonal variation in failure modes (*e.g.*, a greater likelihood of feeder failure in the summer) [197, 90]. Figure 2.3 outlines of the overall process of ODDS.

Previous Work in Smart Power Grid Reliability

As a critical infrastructure, smart power grids demand high reliability. NERC, an international regulatory authority established to evaluate reliability of the bulk power system in North America, reported the reliability considerations from the integration of smart grid [165]. Their report stated that current reliability engineering tools, although effective for today's electric power systems, cannot thoroughly capture the effect of integrating smart technologies.

"As the reliance on communication and control increases, systems can be operated closer to their physical limits to increase asset efficiency, although this potentially makes them vulnerable to system and device defects/attacks/failures. The challenges ahead for developing systematic analytical tools that can properly model the impact of new technology include: coupling between cyber and physical components; coupling between system dynamics and component stress; and uncontrolled and unpredictable changes to demand and supply [165]." Some of these challenges are what my approach is trying to meet.

Bose discussed the models and methods of reliability analysis for the smart grid [28]. He pointed out that the failure modes of the relays and controls in the smart grid are no longer independent but are causally connected to each other through software and communications. These make it difficult to develop models required to conduct reliability analysis because these models are often complex and the techniques for analysis can be very cumbersome. Faza *et al.* described the use of software fault injection combined with physical failures in identifying integrated cyber-physical failure scenarios for the Smart Grid [76]

Dominguez-Garcia proposed a framework for developing systematic reliability analysis tools to address planning and operation challenges of future electric power systems [64]. The paper recognized that current reliability analysis tools are inadequate for capturing the impacts of the increased system complexity due to new technologies and the introduction of new sources of uncertainty in systems already inherently complex. This work further supports the importance of developing new technology for improving reliability of smart

power grids.

Another factor related to power grid reliability is *power quality (PQ)*, which determines the fitness of electrical power to consumer devices. It includes proper synchronization of the voltage frequency and phase that allows electrical systems to function in their intended manner without significant loss of performance or life. Without the proper power, an electrical device (or load) may malfunction, fail prematurely or not operate at all. Modern systems use sensors called phasor measurement units (PMU) distributed throughout their network to monitor power quality and in some cases respond automatically to them. Using smart grids features such as rapid sensing and automated self healing of anomalies in the network promises to bring higher quality power and less downtime while simultaneously supporting power from intermittent power sources and distributed generation, which would degrade power quality if left unchecked. The root mean square (RMS) mathematical operation is widely used in power engineering, especially in representing PQ system measurement [71]. The PQ RMS process has a frequency response characteristic and an associated time constant which is important especially for short term signals [4]. The approach described in this thesis is also applicable to smart power grid PQ RMS systems and is complementary to the existing PMU related electrical engineering techniques.

2.2.2 Smart Building CPS

As defined by Smart Buildings LLC, a US-based engineering and design firm, "A *smart building* is the integration of building, technology, and energy systems. These systems may include building automation, life safety, telecommunications, user systems and facility management systems. Smart buildings recognize and reflect the technological advancements and convergence of building systems, the common elements of the systems and the additional functionality that integrated systems provide. Smart buildings provide actionable information about a building or space within a building to allow the building owner or occupant to manage the building or space [140]." IBM also offers its definition of smart building: "Smarter

buildings are well managed, integrated physical and digital infrastructures that provide optimal occupancy services in a reliable, cost effective, and sustainable manner. Smarter buildings help their owners, operators and facility managers improve asset reliability and performance that in turn reduces energy use, optimizes how space is used and minimizes the environmental impact of their buildings [101]."

A typical smart building is illustrated in Figure 2.4. Within the physical building, there are various sub-systems including lighting, ventilation, entry and exit control, air-conditioning, elevators, security surveillance, energy storage and alternate power generation, electric vehicle charging stations, and building management system (BMS). Installed throughout the building are many smart sensors such as temperature, humidity and CO2 sensors, which provide building operators with information regarding the current state of the building.

According to the U.S. Department of Energy (DOE), commercial office buildings lead the industrial and transportation sectors in total energy consumption [222]. Although new buildings are often designed with energy efficiency and system reliability in mind, the use of energy-efficient materials and advanced building management systems does not always guarantee efficient or reliable building operation. A high percentage of new buildings consume energy at levels that exceed specifications and experience system failures after being put into use [224]. This problem is even worse for older buildings.

One promising approach applies machine learning (ML) to historical and real-time building Supervisory Control and Data Acquisition (SCADA) data and other building information such as weather information and data from utilities to improve the efficiency of building systems without requiring large amounts of additional capital investment. Researchers at Columbia University have developed a prototype of this application and implemented it in a large multi-tenant office building in New York City [209, 244].

The ML approach, termed *predictive building energy optimization*, uses a model to produce accurate building energy demand forecasts as well as automated analyses that can

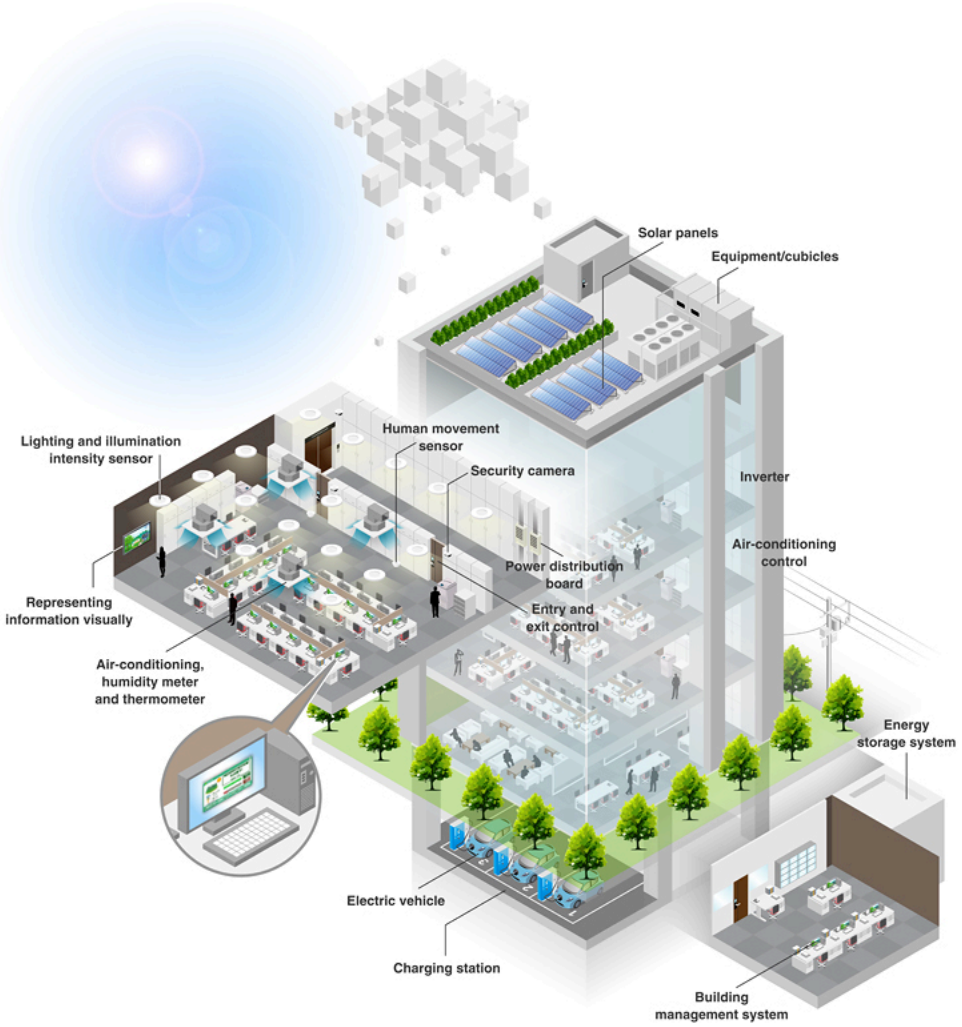


Figure 2.4: A smart building [164].

aid in the tuning of building systems and operations schedules. It applies Support Vector Regression (SVR) on historical energy use of the building, along with temperatures and wet-bulb humidity data from the building's interior and exterior, in order to predict performance for each day. The building information collected from sensors can serve as input to the ML predictor. This does not require knowledge of the building's physical properties, such as size, heating, ventilation, air conditioning (HVAC) or electrical systems. It employs time-delay coordinates as a representation of past data in order to create the feature vectors for Support Vector Machine (SVM) training. The experiments show that the predictive model closely

approximates the actual values of energy usage with some seasonal and occupant-specific variability. The dependence of the data on the day of the week makes the model easily applicable to different types of buildings with minimal adjustments.

Predictive Building Energy Optimization

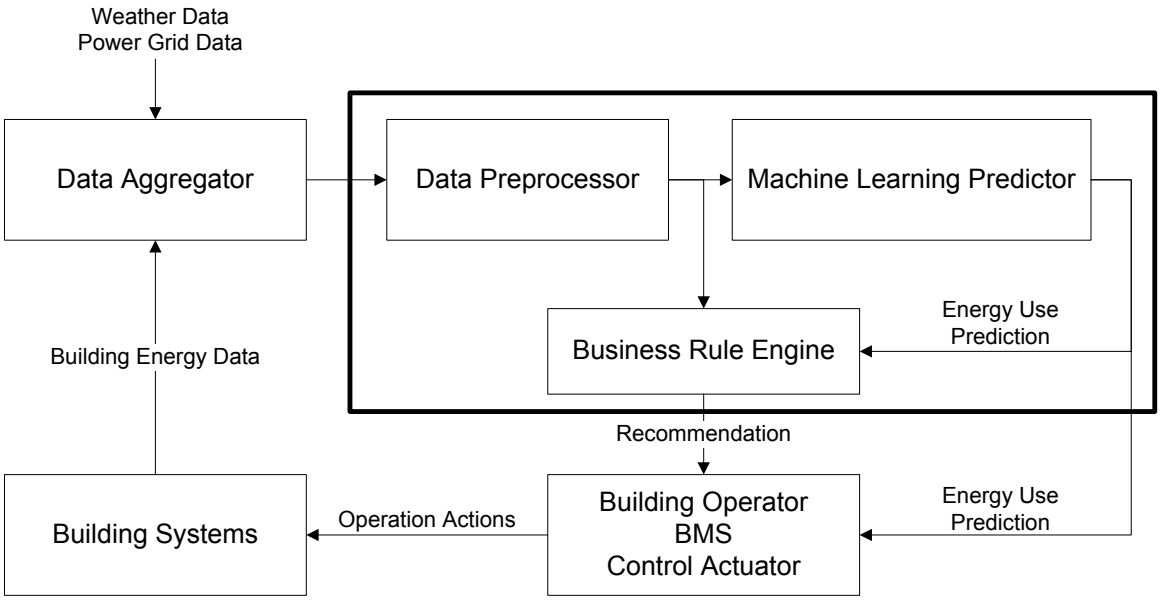


Figure 2.5: Predictive building energy optimization.

As illustrated in Figure 2.5, predictive building energy optimization starts with data aggregation and preprocessing. External data, such as weather and power grid data, is combined with building energy data in the data aggregator, which passes the aggregated data to the data preprocessor for cleaning, formatting and normalization. The ML predictor uses historic energy use data as training data to build a model, which is then used to predict energy use in the present. This prediction is then passed to the building management and business rule engine. The business rule engine processes the aggregated data and the ML prediction in order to generate a set of recommended operation actions. The building management (*i.e.*, building operators or BMS or automatic control actuators) can then take action on the building systems, such as adjusting its HVAC schedule and set-points to achieve more

efficient building operation. The modified building data will then be fed back to the data aggregator, thereby closing the loop.

Building Energy, Weather, Power Grid Data

Building energy use is measured by total electricity consumption over a period of time, typically kilowatt-hours (kWh) per month. The kilowatt-hour is most commonly known as a billing unit for energy delivered to consumers by electric utilities. The energy demand of a building is the rate of energy consumption by the building. Because energy use fluctuates during the week due to tenant activities and building operation schedule, energy demand is a more fine-grained measure of building energy use than the aggregate kilowatt-hours consumed during the whole period.

Large buildings commonly use Building Management Systems (BMS) to manage the interior environment and control mechanical and electrical equipment such as ventilation, lighting, power systems, fire systems and security systems. BMS provides a way to retrieve building energy-related data, such as data readings from sub-meters and sensors.

Climate factors include temperature, humidity, pressure, wind and cloud cover. In a highly condensed urban environment like New York City, different areas can have different weather measurements. This is often called *micro-weather*. The most commonly used weather data for New York City are collected from a weather station located in Central Park, because of its accuracy and stability. Historic hourly weather data can be obtained from some public websites, such as National Climatic Data Center [162] and Weather Underground [234].

Relative humidity and dew point temperature are also important weather data for buildings. Relative humidity is the ratio of the partial pressure of water vapor in the air to the saturated vapor pressure of water under given temperature and pressure conditions. Dew point is the temperature at which the air can no longer hold all of its water vapor, such that some of the water vapor must condense into water. The dew point is always lower than

(or equal to) the air temperature. If the air temperature cools to the dew point, or if the dew point rises to equal the air temperature, then dew begin to form. When the dew point temperature equals the air temperature, the relative humidity is 100%.

Power grid data from utilities include electrical load, peak load, fluctuating electricity pricing during the day and power failure warning. The power grid data from the utilities is often communicated electronically via client web portal or email.

Data Preprocessing

The data preprocessor receives various data streams from the data aggregator and restructures them so that all the data fits the format required by the ML predictor and business rule engine. For the specific predictive modeling technique used in the experiments, all the data needs to be normalized to a value between 0 and 1 for equal weighting.

	Energy Demand	Temperature	Dew Point Temperature	Pressure	Wind Speed	Humidity
Test Set	?	0.65	0.52	0.40	0.72	0.68
Training Set	0.65	0.63	0.50	0.38	0.75	0.71
	.	.	.	.	.	.
	.	.	.	.	.	.
	.	.	.	.	.	.

Figure 2.6: Sample training and test dataset.

As illustrated in Figure 2.6, the training set is used to build the predictive model (*i.e.*, a function that can be used for predicting unknown values). The test set includes data for every column except energy demand, which needs to be predicted. The training set is laid out in descending order, such that one hour before prediction is top-most, two hours before prediction is next, and so forth. Some data rows at the bottom of the training set will lack 'prior value' data due to the ordering system, and those rows are ignored. For any given column, such as temperature, it is possible to expand it such that multiple prior temperatures over a sequential series of time-points become properties of the same data row. In this way,

it is possible to construct a normalized dataset with time-delayed coordinates for use by the ML predictor.

ML Predictive Modeling

Predictive analytics or predictive modeling deals with extracting information from data and using it to predict future trends and behavior patterns. An SVM is a supervised learning method for predictive modeling. It constructs a hyperplane or set of hyperplanes in a high or infinite-dimensional space, which can be used for classification, regression or other tasks [225]. An SVR is a version of SVM for regression [66]. The model produced by SVR depends only on a subset of the training data, because the cost function for building the model ignores any training data close to the model prediction.

A Gaussian radial basis function (RBF), $K(X_i, X_j)$ is selected as the SVM kernel function:

$$K(X_i, X_j) = e^{-(\epsilon\|X_i - X_j\|)^2}, (\epsilon > 0).$$

The RBF kernel nonlinearly maps samples into a higher dimensional space and can handle the case where the relationship between class labels and attributes is nonlinear [65]. It is a nonlinear kernel function and allows the algorithm to fit the maximum-margin hyperplane in a transformed feature space. The nonlinear, dynamic nature of the influence of weather and other data on energy demand in building systems excludes the possibility of using a linear kernel.

In order to measure how well future outcomes are likely to be predicted by the model, the coefficient of determination R^2 is used. Frequently used in the context of statistical models, the main purpose of R^2 is a measurement of the prediction of future outcomes on the basis of other related information. This statistical model accounts for the proportion of variability in a dataset [211].

$$R^2 = 1 - \frac{SS_{err}}{SS_{tot}} = 1 - \frac{\sum_i (y_i - f_i)^2}{\sum_i (y_i - \bar{y})^2},$$

where values y_i are the observed values and values f_i are the modeled values or predicted values in the dataset. SS_{err} is called the residual sum of squares and SS_{tot} is called the total sum of squares. The closer the R^2 is to 1, the more accurate the predicted values are and the better the predictive model is.

Business Rule Engine

The business rule engine receives aggregated data and ML prediction output. It then applies the business knowledge that supports rules, constraints, priority, mutual exclusion, preconditions and other functions onto the data to derive executable recommendations such as work schedule and preventive actions. The business rule engine consists of a BPM (Business Process Management) component and a BRM (Business Rules Management) component. Both components interact with each other responding to events or executing business judgments that are defined by business rules.

The set of business rules is initially defined and incrementally improved by experienced building operators and property managers. It includes both forwarding-chaining (*e.g.* IF something happens THEN do something) and backward-chaining rules (*e.g.*, IF it is intended to achieve this goal THEN something has to happen). These collected rules can also serve as the learning metrics for the more advanced adaptive stochastic controller (ASC) driven by approximate dynamic programming (ADP) to derive action or policy recommendations [11].

Some other research has also been done to predict and analyze the energy demand of buildings. Dong *et al.* used SVM to predict building energy consumption in a tropical region [65]. The DOE-2 model, created by the U.S. Department of Energy, uses physical aspects of the building such as construction materials to predict its energy needs [206].

The ML approach, on the other hand, makes exclusive use of operational building data to model building energy demand. A measurement and actuation profile for building information based on sensor systems was discussed by Dawson-Haggerty *et al.* [56]. These measurements can add to the data available for the ML system.

Previous Work in Smart Building Reliability

Some prior research has been done on smart buildings, especially in the embedded system, wireless sensor networks and energy efficiency. Kleissl and Agarwal treated modern buildings entirely as a cyber-physical energy system and examined the opportunities presented by the joint optimization of energy use by its occupants and information processing equipment [124]. Schor *et al.* presented a web services-based approach to integrate resource constrained sensor and actuator nodes into IP-based networks to achieve automatic service discovery and zero configuration [202].

Agarwal *et al.* described a presence sensor platform that can be used for accurate occupancy detection at the level of individual offices to achieve energy efficiency of building operation [2]. Savvides *et al.* described how cyber-physical systems that provide rapid access to information and decision-making can enable intelligent buildings to autonomously interact with the power grid [200].

For reliability of building systems, Schein and Bushby developed a rule-based system-level fault detection and diagnostic method for HVAC systems [201]. Alekseeva *et al.* did reliability analysis and comparison of long-distance HVAC and HVDC power transmission lines [6]. Kamilaris and Pitsillides examined the use of request queues as a mechanism to manage the communication with embedded devices for smart homes to achieve enhanced reliability and fault tolerance [119]. Irwin *et al.* studied using home automation protocols, such as X10 and Insteon for accurate electric load monitoring in smart buildings [104]. These approaches have different applicable domains than my approach.

2.3 Summary

This chapter describes motivation of the research, some previous work and the applications in the domain of interest including smart building and smart power grid CPS.

Chapter 3

Automated Online Evaluation for CPS

In this chapter, I will describe the automated online evaluation (AOE) approach for improving reliability for cyber-physical systems. For empirical study, I will describe prototype implementation and evaluation of an example of AOE architecture called the Autonomic Reliability Improvement System (ARIS).

In the following section, I will describe an overview. In section 3.2, I will describe my approach of automated online evaluation, followed by the system architecture and implementation in section 3.3 and 3.4. In section 3.5, I will describe my empirical study before the conclusion in section 3.6.

3.1 Overview

A common type of cyber-physical system meets the following criteria: it can process large amounts of data; employ software as a system component; run online continuously; maintain an operator-in-the-loop because of human judgment and accountability requirements for safety-critical systems [54]. Systems that meet these criteria include power grids, building systems, energy systems, transportation systems, defense systems and factory automation systems. These systems do not operate in a controlled environment, and must be robust to unexpected conditions and adaptable to subsystem failures [54]. It is often not possible to

perform robust tests on these cyber-physical systems prior to actual deployment because the physical devices are so expensive that they cannot be replicated in a testing lab and the external environments are impossible to replicate. Thus, it is imperative to have a mechanism that can continuously evaluate the deployed system during runtime in the field to ensure that it is performing reliably.

3.2 Automated Online Evaluation

To meet these challenges, I introduce *automated online evaluation* (AOE), which is a data-centric runtime monitoring and reliability evaluation approach that works in parallel with the cyber-physical system to perform continuous assessment at multiple stages along the system workflow and provide operator-in-the-loop feedback for reliability improvement.

3.2.1 Model

As illustrated in Figure 3.1, automated online evaluation (AOE), consisting of data quality analysis and self-tuning, works in parallel with the CPS and the operator-in-the-loop. AOE performs continuous assessment on the data received from the CPS at multiple stages along the system workflow and provides operator-in-the-loop feedback so that actions can be taken for improving system reliability of CPS. This approach enables ongoing evaluation of data from cyber-physical systems. For example, abnormal input and output data can be detected and flagged based on data quality analysis. As a result, alerts can be sent out that enable the operator-in-the-loop to take actions and make changes to the system in order to minimize system downtime and maximize system reliability.

One technique employed by AOE is *data quality analysis*, wherein computational intelligence is applied to evaluate data quality in an automated and efficient way. AOE also makes use of *self-tuning*, automatically self-managing and self-configuring the evaluation system to ensure that it adapts itself to both changes in the system and feedback from the

operator. This self-tuning continuously adapts the evaluation system itself (not the CPS) to ensure its proper function, which leads to a more robust, accurate and efficient evaluation system.

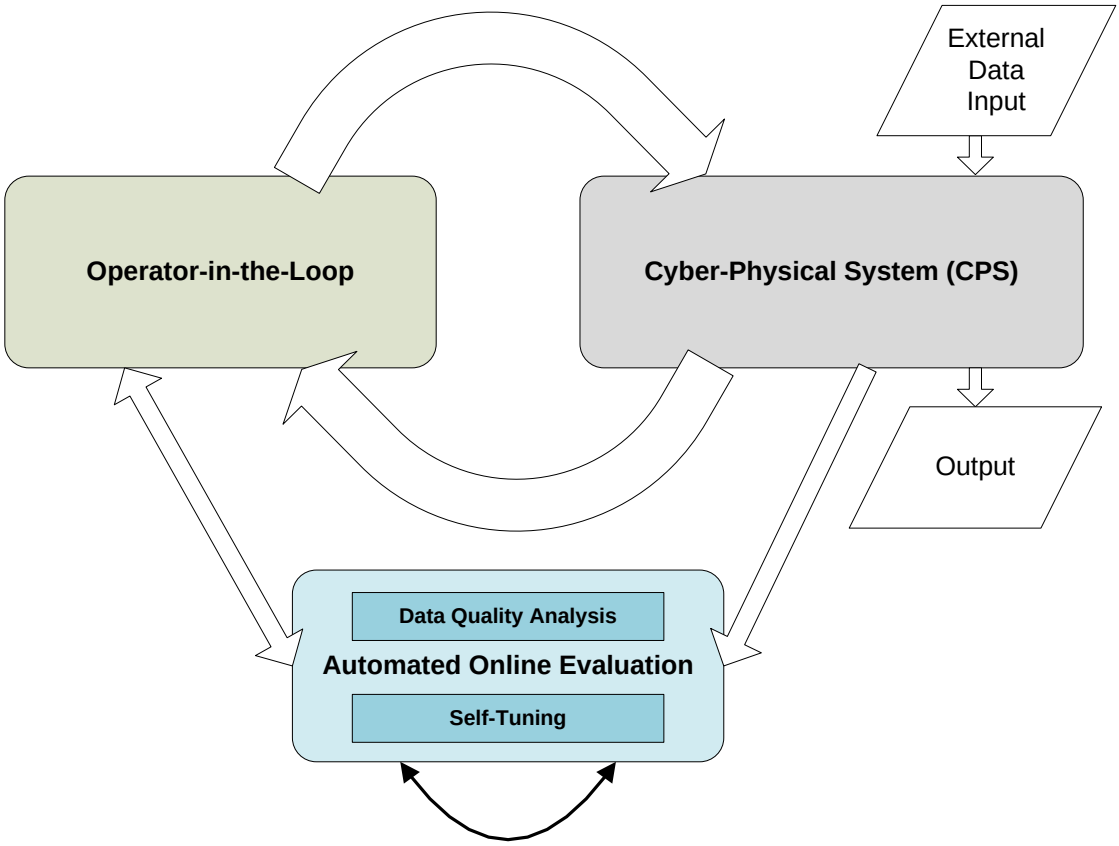


Figure 3.1: Automated online evaluation.

System Agnostic

This approach is system agnostic in that it only looks into the availabe CPS data and it can work with different systems without requiring any special adaptations or prior knowledge about the specific systems. This design attribute requires the approach works comparably well across more than one type of CPS so that it can be applied to a broad range of different

types of CPS.

3.2.2 Assumptions

The approach has two assumptions. First, it assumes that the data from the CPS can be obtained in parallel with the running CPS without causing much disturbance to the CPS. This requirement is usually met in the typical CPS of our interest, as described in 3.1.

Second, the approach assumes that the operator-in-the-loop is able to exercise some change actions onto the CPS following the reliability improvement feedback generated by the AOE. Although this is usually the case for the typical CPS of our interest, there are some types of CPS that do not allow any kind of actions taken by the human operators. For example, some medical cyber-physical systems have tightly coupled fully autonomic computational and physical components which do not enable human operator to take any actions in between.

3.2.3 Data Quality Analysis

AOE uses computational intelligence to perform data quality analysis in an automated and efficient way and thereby ensure that the running system performs reliably. This computational intelligence is enabled by machine learning, data mining, statistical and probabilistic analysis and other intelligent techniques. In a cyber-physical system, data collected from the system (*e.g.*, sensor data points, software bug reports, system status logs and error reports) are often stored in databases. AOE analyzes this data so that useful information on system reliability, such as erroneous data or abnormal system states, can be obtained. This reliability-related information is in turn directed to system operators so that proper actions can be taken–in some cases, proactively based on predictive results–to ensure proper and reliable execution of the system.

Data Quality Dimensions

The data quality dimensions have been studied in prior research [231, 232, 116, 170, 181, 229]. Table 3.1 lists some data quality dimensions AOE evaluates to derive useful information and recommendation for reliability improvement. This list only includes those aspects that have potential impact on system reliability.

Data Quality Dimension	Description
Accuracy	Are all the data within the acceptable range? Do data fit into the usual profile?
Accessibility	Do data exist? Can data be accessed?
Timeliness	Do data arrive on time?
Completeness	Is there any data gap?
Representation Consistency	Are all the data in the same format?

Table 3.1: Data quality dimensions.

Among all the dimensions of the data quality, accuracy of the data is one of the most common problematic aspects and the hardest ones to evaluate. The following sections will describe some examples of data quality analysis techniques that can be used by AOE to evaluate the accuracy of the data for CPS reliability improvement, including:

1. Thresholds
2. Profiling and Anomaly Detection
3. Automated Diagnosis

Identifying Data Quality Issue Using Thresholds

The thresholds define the normal working range for specific data-points. Usually operators determine the thresholds based on heuristics from operation and recommendations by the system providers. If data readings exceed these thresholds at either the lower or upper bound, the data record will be flagged as anomalous and a corresponding warning will be communicated back to the operator electronically.

Identifying Data Quality Issue Using Profiling and Anomaly Detection

Anomaly detection is used to find data instances that are unusual and do not fit any established pattern or profile. This type of anomaly is different than exceeding the thresholds. It concentrates on modeling normal behavior in order to identify atypical data-points. Data anomaly does not necessarily indicate that failure has already occurred. For the cyber-physical systems of interest in this study, time-series data usually arrive continuously in parallel at a varied pace.

One example implementation for identifying data quality issue can process the continuously updated data-streams to detect anomalies for single data-points, using a customized incremental Local Outlier Factor (LOF) algorithm [182]. The algorithm uses the k-nearest neighbor [52, 48] on each inserted data record to instantly compute LOF value, which is the degree to which a data record represents an outlier or an indicator of abnormality. Figure 3.2 shows the general framework for insertion of a data record and computing its LOF value. A sudden increase in LOF value indicates that a data record is likely to be an outlier. LOF values for existing data records can be updated on the fly if necessary. Because each data series represents an individual data source with low data dimensionality, such as a sensor's reading of (time, value) tuples, the incremental LOF algorithm is computationally efficient.

Furthermore, multiple LOF value time series can be processed for different data sources and displayed in parallel via a *sparkline graph*, a type of information graphic characterized by its small size and high data density [220], for more fine-grained checks. As shown in Figure 3.3, data series B and C both experience a sudden increase of LOF value at around the 38th hour after the start of observation. This spike indicates a strong likelihood of a data anomaly at that time-point. This visualization technique provides an easy way to obtain additional verification of a data anomaly. It is also a useful communication channel to help the operator understand where issues are arising.

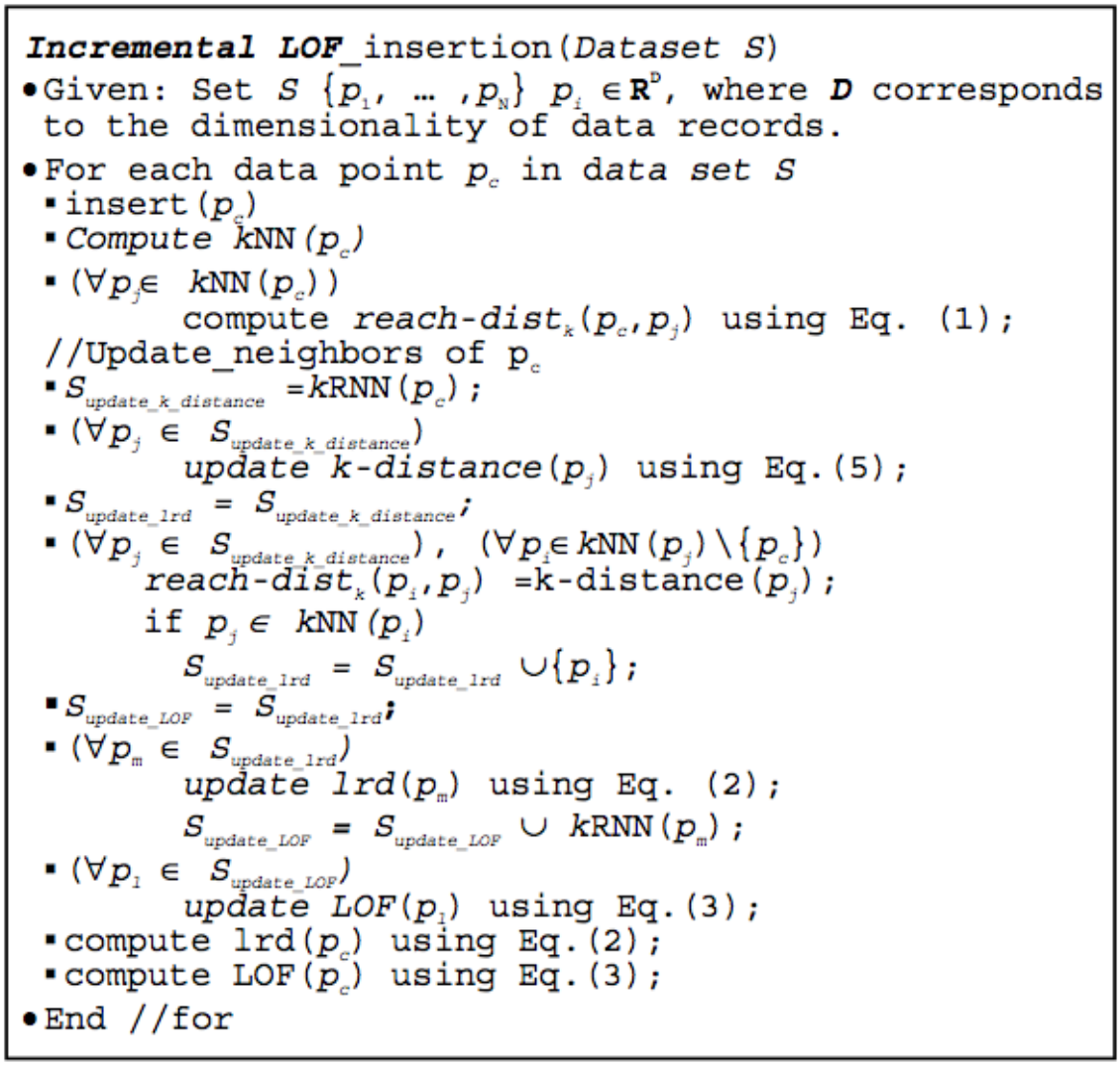


Figure 3.2: General framework for insertion of a data record and computing its LOF value via incremental LOF algorithm [182].

Analyzing Data Quality Issue Using Automated Diagnosis

After a data anomaly is detected, further automated diagnosis or reasoning is needed to infer what physical and computational/software component the data anomaly relates to, what reliability issue this anomaly might cause, and the recommended action (or work order) for the operator to take in order to correct the problem. This can be solved as a supervised learning problem and a classification model trained on existing data can predict unknown

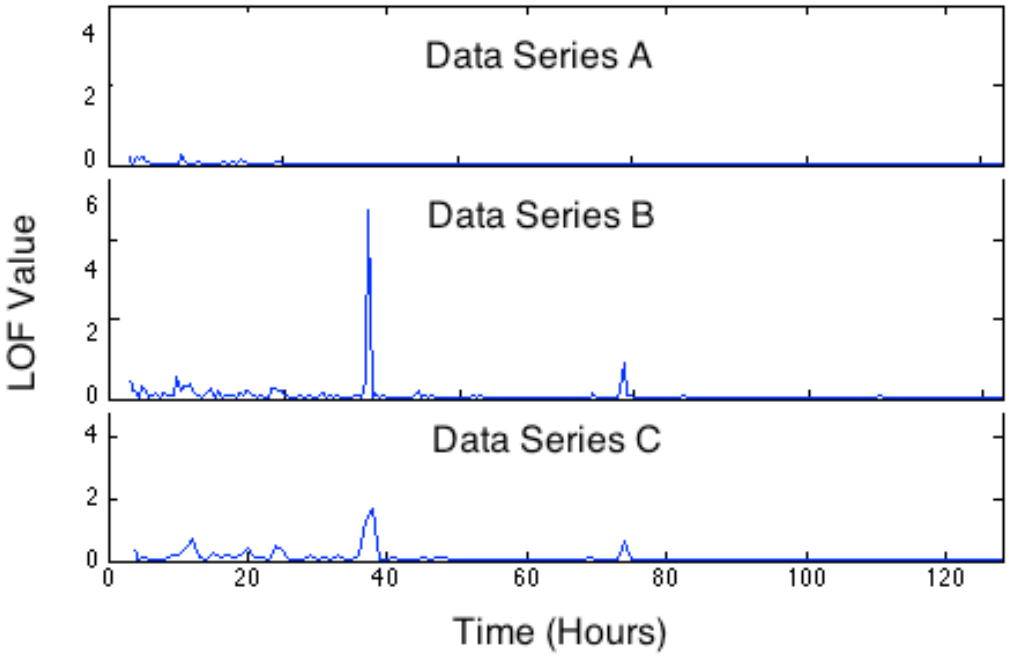


Figure 3.3: LOF value time series displayed using sparkline graph.

values (*i.e.*, the component having issues and the corresponding corrective/preventive action).

For example, Support Vector Machines (SVMs) [225, 51] can be used as the classifier. SVMs formulate the classification modeling process as a quadratic minimization problem and find hyperplanes in a high-dimensional space that separate data instances of different categories while maximizing the margins between categories. First, a set of historic data records (*e.g.*, each one with N attributes) is used as training data to build a linear SVM model as a classifier. For a new data record with one unknown data field A (*i.e.*, $N-1$ attributes available and one attribute or class label unknown), the trained SVM model and the available $N-1$ attributes are used to predict the value of the unknown A field for this data record. In cases where multiple data fields need to be determined for a data record (*e.g.*, $N-M$ attributes available and M attributes unknown), the SVM model and the available $N-M$ available attributes are used to predict the unknown fields one by one.

Using SVM classification as the basis for data anomaly diagnosis has some advantages

over rule-based reasoning systems, *i.e.*, a type of systems that use a list of rules, an inference engine, temporary working memory and user interface to store and manipulate knowledge in order to interpret information in a useful way [93]. SVM classification does not require a lot of prior knowledge of the system because it works solely based on the data itself. Rule-based systems require derivation of the rules, including both forwarding-chaining rules (*e.g.*, IF something happens THEN do something) and backward-chaining rules (*e.g.*, IF I want to achieve this goal THEN something has to happen), based on extensive heuristics and in many cases expert domain knowledge. Also, SVM classification is adaptive based on updated training data, while rule-based logics are often rigid and not easy to change. In some unexpected real-world situations, rule-based systems are often unable to reach any conclusion whereas machine-learning approaches may be able to derive partially useful information for the operator, such as a rank list with scores based on probability and susceptibility.

3.2.4 Self-Tuning

Autonomic computing is an approach to self-managed computing systems with minimal human interference [100] and refers to the self-managing characteristics of distributed computing resources, adapting to unpredictable changes whilst hiding intrinsic complexity to operators and users. *Self-tuning* is an aspect of autonomic computing. A self-tuning system is capable of optimizing its own internal running parameters in order to maximize or minimize the fulfillment of an objective function; typically the maximization of efficiency or error minimization.

AOE also makes use of self-tuning to self-manage and self-configure the evaluation system itself (not the CPS) to ensure that it adapts itself to changes in the system and feedback from the operator. This self-tuning is used to improve accuracy, efficiency and robustness of data quality analysis, and also minimizes the burden imposed on the operator.

Self-tuning is used to improve the automated online evaluation. One example is how to

make the evaluation system automatically adapt to the anomaly of input data, such as the seasonality of the weather data, as shown in Figure 3.4 and 3.5. The self-tuning process is important to make sure the evaluation system can learn from the system changes or operator's feedback and self-manage the evaluation system.

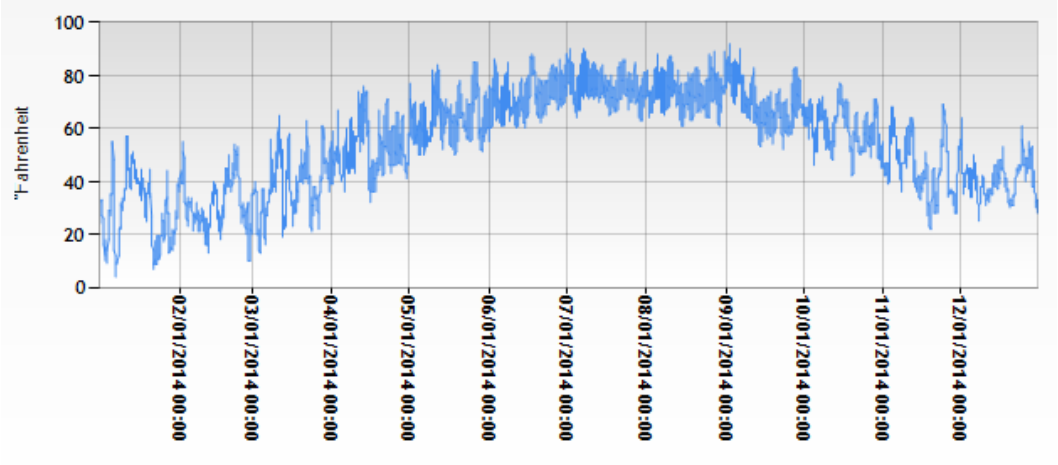


Figure 3.4: Outside temperature in Central Park, New York City in 2014.

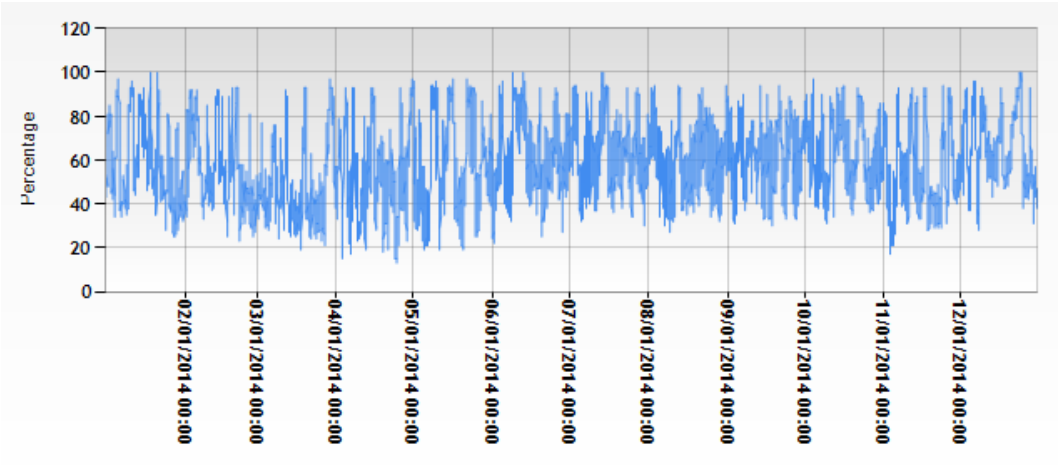


Figure 3.5: Outside humidity in Central Park, New York City in 2014.

As illustrated in Figure 3.6, self-tuning employs a Measure-Identify-Tune (MIT) process in order to achieve the following results:

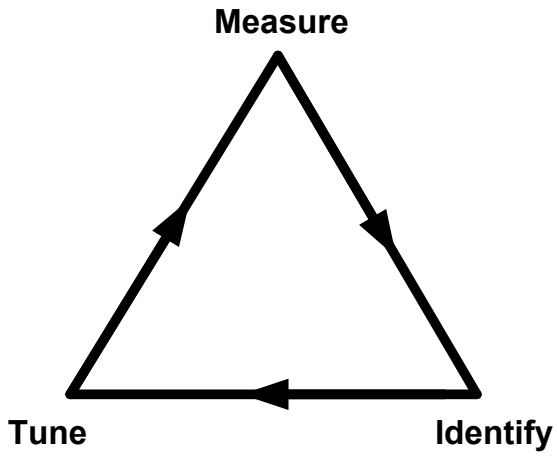


Figure 3.6: MIT self-tuning process.

- use performance metrics such as R^2 (coefficient of determination), ROC (receiver operating characteristic) and AUC (area under the curve) to measure and improve the accuracy of the data analysis models [242]
- use statistical trend detection and curve fitting, such as Weibull distribution and parameters estimation [246, 190], to reduce variability and eliminate overshoot
- prioritize updates from operators and adjust system parameters such as set-points, thresholds and machine learning model parameters when abnormal exogenous situations happen in order to reduce false alarms
- use dynamic load balancing and failover switch, which applies to the parallel processing of the large amount of time series data coming from different data sources, to maximize efficiency and reliability

An example self-tuning approach employs the following techniques:

1. data classification, which helps to determine or predict some unknown data based on historic data.

2. redundancy checking, which helps to determine if the data instance is a duplicate of some prior data.

3. trend detection, which helps to find the trend pattern for the data set.

These techniques help make self-tuning possible, which in turn self-manages the evaluation system to improve system reliability. The following subsections will describe the details for each of the techniques.

Data Classification

In a live system, the input data sometimes may have missing values. To determine or predict these unknown data based on historic data helps automated online evaluation to work more accurately, which leads to better evaluation. The data classification can be solved as a supervised learning problem. By training a classification model on existing data, the missing values can be predicted. Support Vector Machines (SVM) and other machine learning methods can be used as the classifier [225, 51].

Redundancy Checking

Redundant data often leads to duplicate processing and even skewed or abnormal results. It is helpful for self-tuning to effectively detect the data redundancy so that proper adjustment to evaluation system can be done accordingly, which leads to improved system reliability because of less skewed and abnormal results. For redundancy checking, dataset can be represented in a vector space model (*i.e.*, term vector model), an algebraic model for representing text documents as vectors of identifiers, such as index terms [198]. The similarity between two data instances can be measured based on Cosine similarity, *i.e.*, the Cosine of the angle between the two vectors that represent these two data instances, as shown in the following formula:

$$Distance_{COS}(a,b) = \frac{\sum_i a_i \times b_i}{\sqrt{\sum_i a_i^2} \times \sqrt{\sum_i b_i^2}},$$

where a and b represent two vectors. Its result equals 1 when the angle between two vectors is 0 (*i.e.*, two vectors are pointing in the same direction), and its result is less than 1 otherwise.

In addition to Cosine similarity, all prior data instances can be ranked based on their relevance to the new instance using probability distribution. Kullback-Leibler (*i.e.*, KL) divergence [53, 147] is an effective relevance metric that assumes each data instance in a high dimensional feature space is characterized by a probability distribution. KL divergence measures the dissimilarity between two probability distributions, as shown in the following formula:

$$D_{KL}(a||b) = \sum_{t \in V} P(t|M_a) log \frac{P(t|M_a)}{P(t|M_b)},$$

where M_a and M_b represent the probability distributions for vector a and b respectively. V is the vocabulary of all terms and t is a term in V. KL divergence measures how bad the probability distribution M_a is at modeling M_b.

Trend Detection

To detect data trend is important for self-tuning to adjust the evaluation system effectively, which leads to improved system reliability. For example, the change of the data trend curve may indicate overall system state change, which requires self-tuning to act on the evaluation system. One way to model the trend pattern is using Weibull distribution [190], which provides the basis for trend detection and analysis. First, historic data is used to fit the Weibull function and derive the λ and k parameters. Then for any given time t, the number of instances that may happen during that t-th time period can be estimated using the Weibull's density function $f(t)$. Similarly, the instantaneous incidence rate can be estimated using

the hazard function $h(t)$. Other semiparametric approach may be used to provide similar estimation [245].

3.3 Architecture

An example of AOE architecture called the Autonomic Reliability Improvement System (ARIS) is illustrated as a seven-step process in Figure 3.7. ARIS evaluates the cyber-physical system via three stages of data quality analysis (steps 1-3): first, evaluation of the input data; second, evaluation of the data output; and third, evaluation of feedback from the cyber-physical system.

Step 1: The initial evaluation checks to see if the input data meets the quality specifications pre-defined by the application developer and the system operator. Examples of data quality specification include data existence, up-to-date, conforming to certain distribution, time-synchronization across different sources, variation and pattern.

Step 2: The output data evaluation checks the quality of the results of the application. For example, for a machine learning-based prediction system, data output quality relates to the accuracy or confidence level of the prediction. For a non-machine learning-based system, such as a building energy management system, the quality of the data output relates to the extent to which results can be used to guide subsequent actions (*e.g.*, building energy use adjustment).

Step 3: The evaluation of the feedback from the cyber-physical system checks the outcome resulting from the previous steps. This evaluation is important to ensure that the data output in fact leads to the desired system outcome.

In step 4, the results from the data quality analysis are directed to a user interface for system operators, who may take control or recovery actions when abnormal and erroneous situations happen. These actions ensure proper execution of the system and lead to improved system reliability.

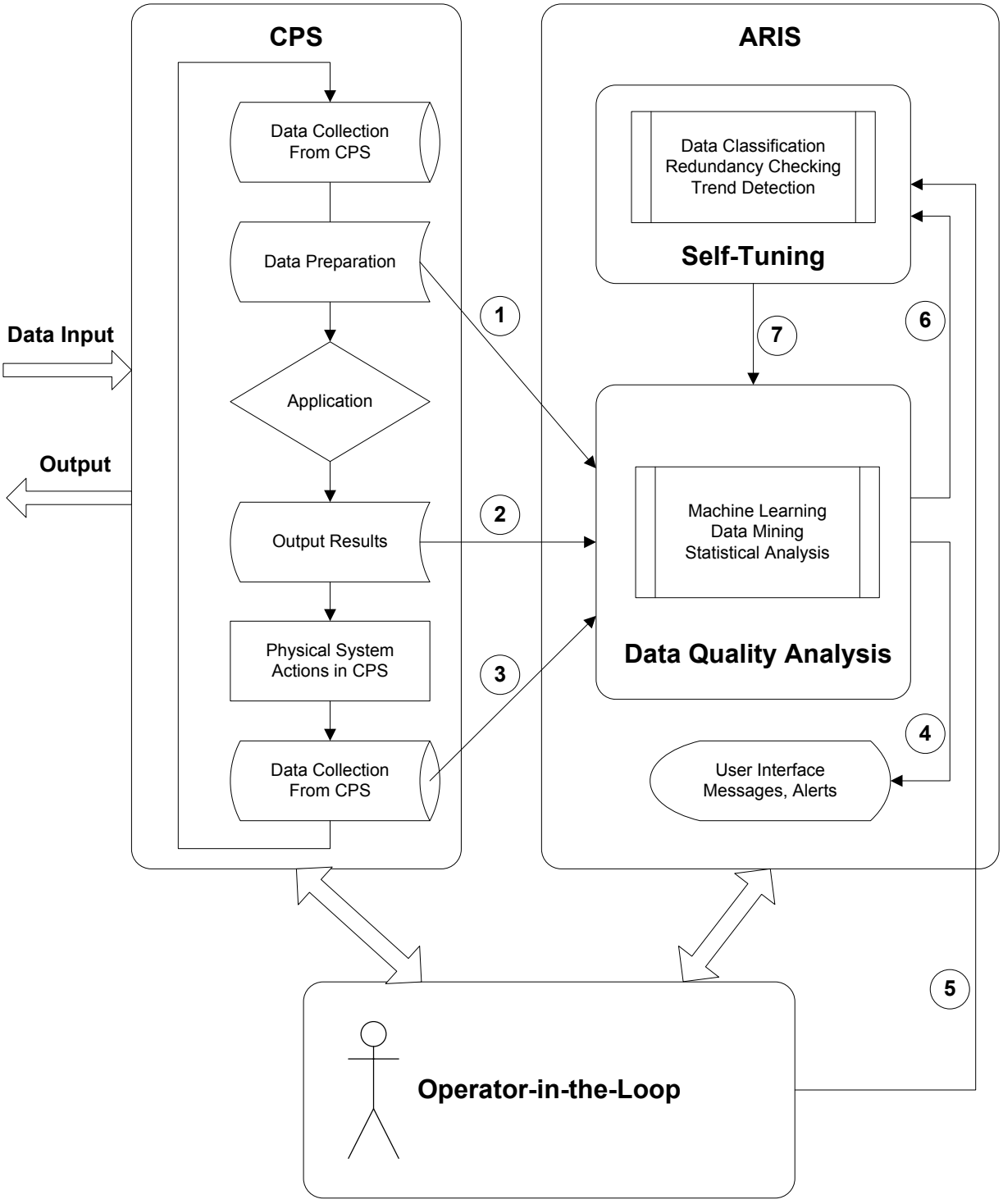


Figure 3.7: ARIS system architecture.

At steps 5 and 6, the self-tuning component receives feedback from both the operator-in-the-loop and changes in the system.

Finally, in step 7, the self-tuning component self-manages and self-configures the evaluation system based on the feedback from the operator and the changes in the system. This self-tuning adapts the evaluation system to ensure proper functioning, which leads to a more robust evaluation system and improved system reliability. The ARIS is autonomic while the CPS is not necessarily autonomic.

To further illustrate the proposed architecture, here is a hypothetical use case wherein multiple steps and actions were managed using ARIS. A *Building Management System (BMS)* is a type of cyber-physical system consisting of both software and hardware components that controls and monitors a building's mechanical and electrical equipment, such as ventilation, lighting, power systems, fire systems and security systems. The building energy control system is an important component of the BMS that reads data feeds representing internal and exogenous conditions (*e.g.*, temperature, humidity, electrical load, peak load, fluctuating electricity pricing and building work schedule) and takes control actions (*e.g.*, adjust lighting, turn on/off the air-conditioning and shut off partial elevators) accordingly. Building operators usually have the ability to change or override control actions taken by the BMS to accommodate special situations such as severe weather or changes in the building's work schedule.

To ensure that the building energy control system works reliably, input data, output data (*i.e.*, control actions) and the result of actions taken are evaluated using ARIS (Figure 3.7, steps 1–3). In one example scenario, a malfunction of the digital thermostat caused a temperature reading to stay at a fixed level without changing for a long time. The building energy control system was designed to accept any value within a certain temperature range and would not be able to handle this type of input data error (*i.e.*, constant temperature). In contrast, ARIS's intelligent data quality analysis component can quickly detect this type of input data error (Figure 3.7, step 1), and give feedback to the building operator (Figure 3.7,

step 4). After receiving an automated notification from ARIS, the building's operator can then take appropriate action.

In another example scenario, building management notifies the operator of the need to keep the building fully functioning for a special, one-time-only event during the coming weekend. The operator then notifies ARIS about the abrupt change (Figure 3.7, step 5). The self-tuning component of the ARIS takes this signal and uses it to adjust data quality analysis (Figure 3.7, steps 6 and 7), thus avoiding possible false-positive system warnings due to the abnormal energy use data during this specific weekend.

3.4 Implementation

I have developed a prototype application of ARIS using Java programming language, MATLAB [87], and some machine learning libraries such as Weka [237]. The standalone application is able to run on various operating systems with Java Virtual Machine [173] enabled and has been tested on Windows Server 2008 R2 and Ubuntu Linux Server 12.04.2 [221]. It mainly uses a JDBC [172] database connection component to access relational database. There is no dependency on other third party or off-the-shelf software component.

Software Design

As shown in Figure 3.8, the software consists of a secure IP-based data connector, a data quality analysis and self-tuning module with back-end database, several feedback mechanisms (including alert emails, warning messages, and reports,) and a user interface enhanced by real-time visualization.

3.5 Empirical Studies

In this section, I will describe empirical studies including evaluation methodology and experiments.

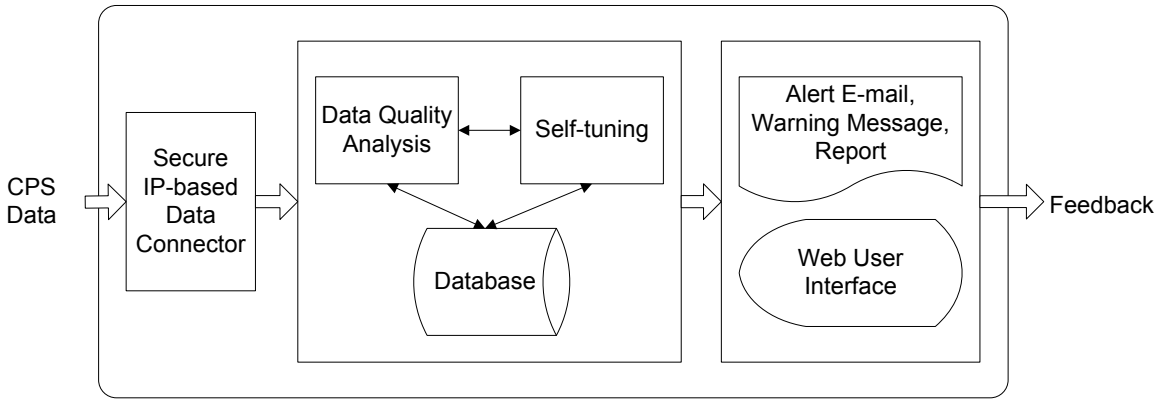


Figure 3.8: ARIS software components and data flow.

3.5.1 Evaluation Methodology

The experiments are conducted not only in a controlled lab environment, but also in some real-world cyber-physical systems. Some reliability metrics are used to quantitatively measure the system reliability improvement.

Reliability Metrics

Reliability may be measured in different ways depending on the particular situation [189]. The following are some commonly used reliability metrics for CPS [241]. These metrics are also used in my experiments.

- *Failure rate* is defined as the total number of failures within an item population divided by the total time expended by that population during a particular measurement interval under stated conditions [139].

- *Mean time between failures (MTBF)* is the mean expected time between system failures. It is the predicted elapsed time between inherent failures of a system during operation [113].

- *Mean time to failure (MTTF)* is sometimes used instead of MTBF in cases where a system is replaced after a failure, since MTBF denotes time between failures in a

system that is then repaired.

- *Mean time to repair (MTTR)* is the mean time required to repair a failed component or device.

- *Availability or mission capable rate* is the proportion of time a system is in functioning condition. It takes repair and restart times into account and is relevant for non-stop continuously running systems. This is also called system uptime ($x\%$). Using a simple representation, it can be calculated as a ratio of the expected value of the uptime of a system to the aggregate of the expected values of up and down time.

- *Rate of fault occurrence* reflects failure rate in the system. It is useful when system has to process a large number of similar requests that are relatively frequent.

- *Probability of failure on demand* is the probability system will fail when a service request is made. It is useful when requests are made on an intermittent or infrequent basis.

- *Power-on hours (POH)* is the length of time (in hours), during which electrical power is applied to a device.

- *Availability at time t* is the probability that the item is able to function at time t [189].

- *Survival probability* is the probability that the item will not fail in a time interval $(0, t]$ [189].

3.5.2 Controlled Experiments

Experimental Requirements

The controlled experiments are based on lab or benchmark environment. The experimental data includes the data that is available to general public via Internet download while the

software environments are the ones that are commonly used. **These controlled experiments are important for proving the hypotheses. They are intended to prove *first and second hypotheses* described in section 1.7, *i.e.*, effectiveness of the automated online evaluation approach using data quality analysis and self-tuning.** The controlled experiments in the lab are good for initial proof-of-concept prior to real-world deployment and experiments.

Methodology

As illustrated in Figure 3.9, in any experiment, two independent cyber-physical systems are used in parallel: one without the ARIS implemented and the other one with ARIS. Both systems are supplied faulty input data. And then measurement and validation are performed to compare both systems' reliability to see if they can continue reliable execution without problem. The faulty conditions in output data and physical system effect are also simulated for evaluating the system reliability of the two cyber-physical systems.

The cyber-physical systems used in the experiments are smart building HVAC (Heating, Ventilation, and Air Conditioning) start-up control system, a subsystem of BMS (Building Management System). Operator bases on space temperature and start-up time forecasting to decide start-up time and BMS target temperature set-points.

Figure 3.10 shows a typical summer day's space temperature and start-up time forecasts for a commercial building in New York City. The colored horizontal bands indicate the desired space temperature range during work hours. I used a prototype ARIS system implemented as described in section 3.4 for the experiments. Figure 3.11 shows the experiment workflow.

Target

As illustrated in Figure 3.12, with faults injected into CPS data, three experiments are conducted to compare the system reliability (R_0) for baseline CPS without any additional

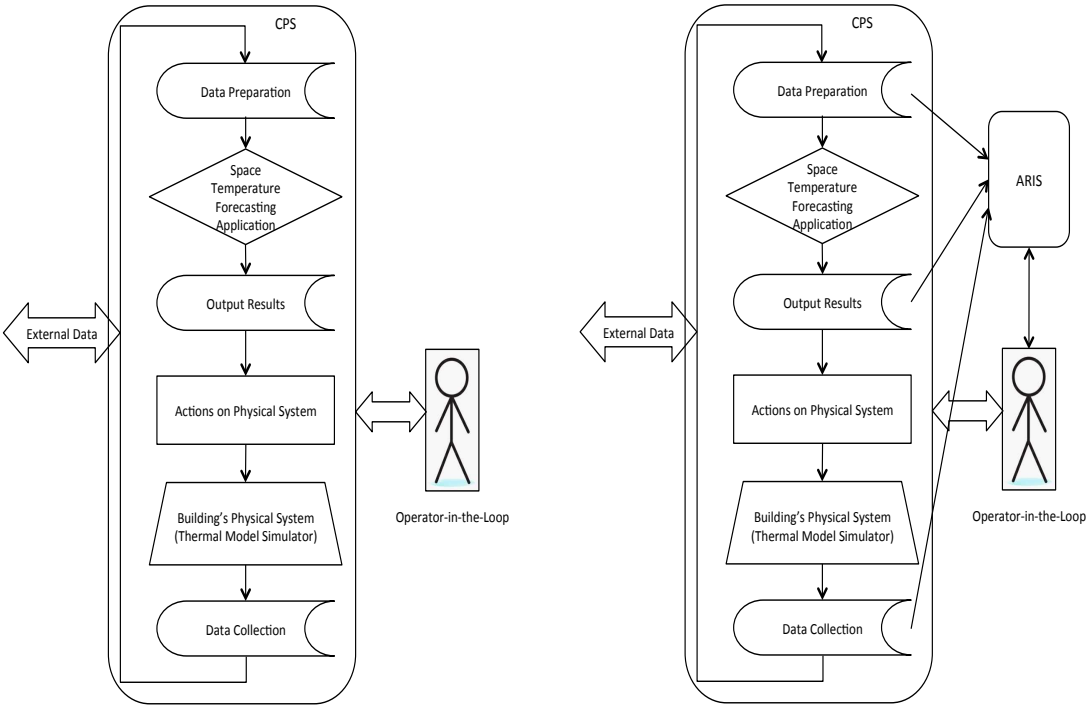


Figure 3.9: Experimental setup of two parallel CPS, one without ARIS and one with ARIS.

technique, system reliability (R_1) for CPS with ARIS, and system reliability (R_2) for CPS with some state of the art technique. The target is to validate that reliability metric $R_1 > R_0$ and $R_1 > R_2$. The specific reliability metric for this illustration is *Mean time between failures (MTBF)*, as defined in section 3.5.1, which shows on average how long before another failure may happen.

The first target of the controlled experiments is to compare the system reliability between the CPS without ARIS and the CPS with ARIS to validate the system reliability improvement brought by the implementation of ARIS.

The second target of the controlled experiments is to compare the system reliability between the CPS without ARIS and the CPS with some state of the art technique. According to the best of my knowledge, the state of the art technique for this specific domain is the rule-based control system defined by ASHRAE (American Society of Heating, Refrigerating and Air-Conditioning Engineers) Standard 90.1-2013, which lists latest requirements for

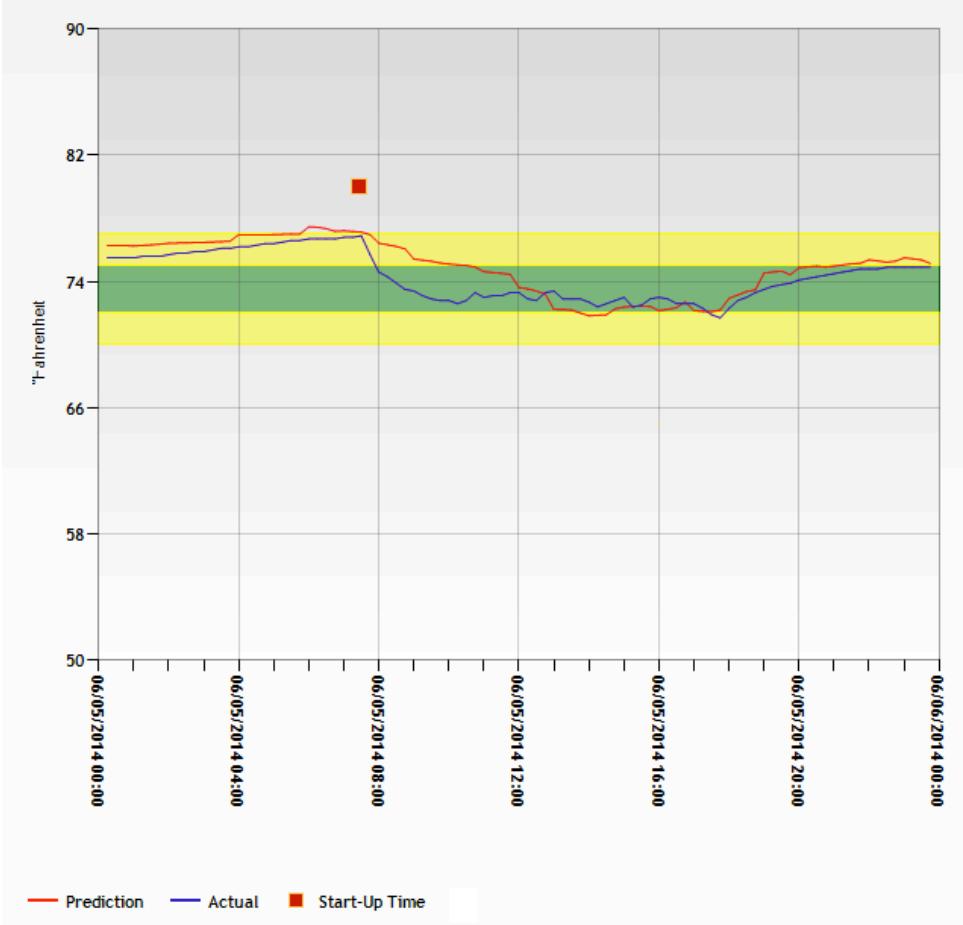


Figure 3.10: Space temperature and start-up time forecasts.

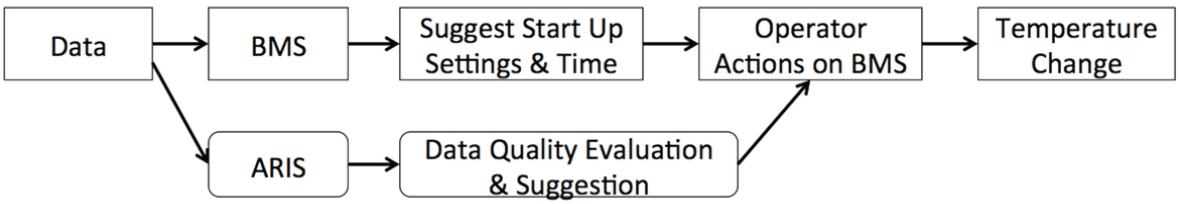


Figure 3.11: Experiment Workflow.

HVAC and Service Water Heating systems, along with insights on how to comply during building design and construction [99]. In my experiments, the ARIS is compared with the rule-based system to see which one can lead to better system reliability. This type of comparative study shows the advancement of the state of the art research by the approach.

Figure 3.13 illustrates the workflow of the rule-based system implemented and used in

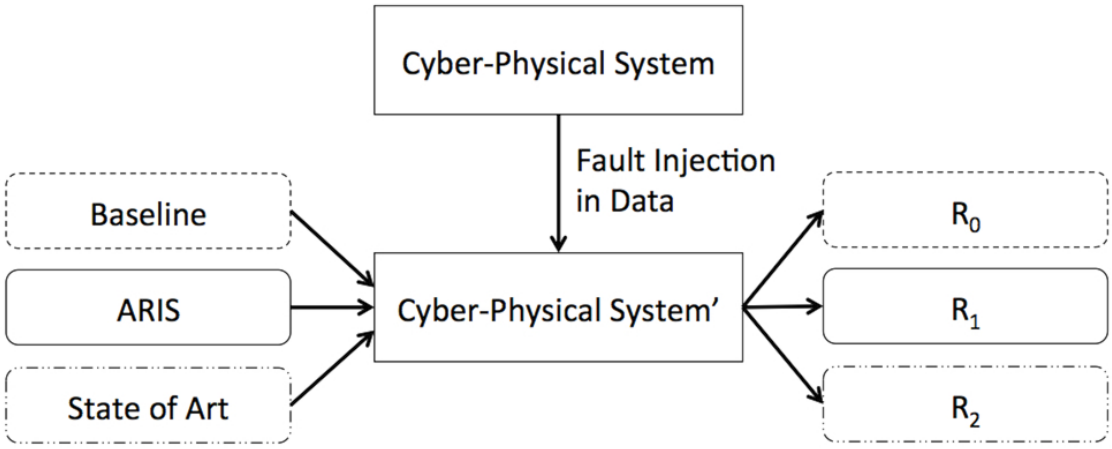


Figure 3.12: Comparison of experiments.

my experiments. The operator uses rules to determine whether to take the suggested actions or not. Specifically, the rules include: if the historic similar weather day's start-up time is over 30 minutes away from the forecasting results, then take the past action ignoring the suggestion; and if the historic similar weather day's temperature set-points are over 10 degrees away from the forecasting results, then take the past action ignoring the suggestion. These rules help to reduce some system failures, mostly those extreme cases where results happen to be too far away from the historic normal values. This kind of rule-based system is commonly used by building operations.

In my experiments, the similar weather day is selected using a Euclidean distance-based ranking upon past three years of data [62, 21]. First, historic days with the same season, which is determined by the same month or the adjacent months, the same day of week, and the same holiday status are chosen. Second, calculate the $n-$dimensional Euclidean distance between the weather forecast for the coming day and the weather of the historic days selected in the first step. Last, the distance scores are sorted at an ascending order and the top one with the lowest score is selected as the similar weather day. As an example, if $p = (p_1, p_2, ..., p_n)$ and $q = (q_1, q_2, ..., q_n)$ are two points in Euclidean $n-$space with p_i and

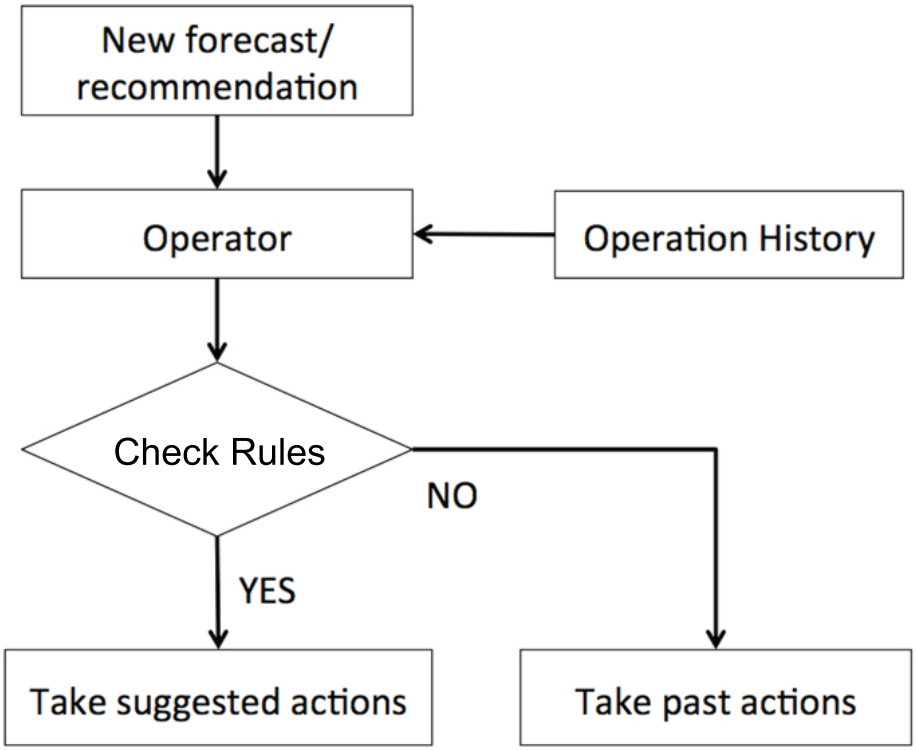


Figure 3.13: Rule-based system used in the experiment.

q_i to be the day's highest temperature, lowest temperature, highest dew point temperature, lowest dew point temperature, highest humidity, lower humidity, and wind speed, then the distance (d) from p to q, or from q to p is given by the formula:

$$\mathrm{d}(\mathbf{p},\mathbf{q}) = \mathrm{d}(\mathbf{q},\mathbf{p}) = \sqrt{(q_1-p_1)^2+(q_2-p_2)^2+\cdots+(q_n-p_n)^2}$$

$$= \sqrt{\sum_{i=1}^{n}(q_i-p_i)^2}.$$

Definition of Failure

As defined in section 1.1, *failure* is the inability of a system or component to perform its required function within the specified performance requirements. It is the manifestation of a fault in the system or human error. In my controlled experiments, *failure* is defined

as the erroneous result, *e.g.*, out of range, produced by the forecasting system and also physical system's failure to meet operation and lease requirement, which often requires interior temperature to be within certain range, *e.g.*, 70 to 77 degrees Fahrenheit (°F), during work hours.

Data

Data used by the smart building start up control system includes *internal data* such as building energy data, space temperature, equipment settings, occupancy, and *exogenous data* such as weather information, day of the week, holiday, and utilities data. Most of the data are time series data with every 15 minutes update frequency.

Large buildings commonly use Building Management Systems (BMS) to manage the interior environment and control mechanical and electrical equipment such as ventilation, lighting, power systems, fire systems, access control (also as people counter) and security systems. BMS provides a way to retrieve building energy-related data, such as data readings from sub-meters and sensors. Building energy use is measured by total electricity consumption over a period of time, typically kilowatt-hours (kWh) per month. The kilowatt-hour is most commonly known as a billing unit for energy delivered to consumers by electric utilities. The energy demand of a building is the rate of energy consumption by the building; because energy use fluctuates during the week due to tenant activities and building operation schedule, energy demand is a more fine-grained measure of building energy use than the aggregate kilowatt-hours consumed during the whole period.

Climate factors include temperature, humidity, pressure, wind and cloud cover. In a highly condensed urban environment like New York City, different areas can have different weather measurements. This is often called *micro-weather*. The most commonly used weather data for New York City are collected from a weather station located in Central Park, because of its accuracy and stability. Historic hourly weather data can be obtained from some public websites, such as National Climatic Data Center [162] and Weather Underground

[234]. Relative humidity and dew point temperature are also important weather data for buildings. Relative humidity is the ratio of the partial pressure of water vapor in the air to the saturated vapor pressure of water under given temperature and pressure conditions. Dew point is the temperature at which the air can no longer hold all of its water vapor, such that some of the water vapor must condense into water. The dew point is always lower than (or equal to) the air temperature. If the air temperature cools to the dew point, or if the dew point rises to equal the air temperature, then dew begin to form. When the dew point temperature equals the air temperature, the relative humidity is 100%.

Power grid data from utilities include electrical load, peak load, fluctuating electricity pricing during the day and power failure warning. The power grid data from the utilities are often communicated electronically via client web portal or email.

Fault Injection

In my experiments, the input and output data anomaly are simulated using fault injection, a software testing technique for improving the coverage of a test by introducing faults to test code paths or mutations to code or data [17, 37, 238, 239]. In the following, I will describe three types of faults injected in my experiments. These three types of faults are the most common types of faults in the real-world environments [55, 250]. Each day's data is susceptible to all three types of faults. All the faults are simulated automatically using SQL scripts with adjustable parameters because all these operations are executed against data tables inside database, *i.e.*, Microsoft SQL Server. Each type of faults is likely to be injected by the automatic process.

The first type of faults injected is **data gaps**, which means some data for certain period are missing or nonexistent. Figure 3.14 illustrates an example fault injection with missing data. In this example, the wind speed for period from January 17, 2014 to January 24, 2014 have null value.

The second type of faults injected is **constant values**, which is usually unreasonable in

	Date	MAX_Temp	MIN_Temp	MAX_Dew	MIN_Dew	MAX_Hum	MIN_Hum	WindSpeed
1...	2014-01-14 00:00:00.000	51.1	45	45	36	90	59	4.92173913043478
1...	2014-01-15 00:00:00.000	44.6	39.2	39.9	33.8	93	66	3.5
1...	2014-01-16 00:00:00.000	41	35.2	39.2	26.1	93	61	4.6
1...	2014-01-17 00:00:00.000	44.6	32.7	28.4	15.8	76	46	NULL
1...	2014-01-18 00:00:00.000	39.2	28.4	33.8	17.6	100	56	NULL
1...	2014-01-19 00:00:00.000	37.4	24.8	21.2	10.4	74	44	NULL
1...	2014-01-20 00:00:00.000	44.6	33.8	28.4	17.6	65	39	NULL
1...	2014-01-21 00:00:00.000	35.6	12.2	15.8	3.2	100	35	NULL
1...	2014-01-22 00:00:00.000	17.6	6.8	3.2	-5.8	67	41	NULL
1...	2014-01-23 00:00:00.000	19.4	8.6	3.2	-5.8	62	42	NULL
1...	2014-01-24 00:00:00.000	19.4	10.4	-0.4	-9.4	53	28	NULL
1...	2014-01-25 00:00:00.000	28	17.6	19.9	-0.4	81	45	6.93333333333333
1...	2014-01-26 00:00:00.000	33.1	17.1	19	-2.9	68	36	7.34736842105263
1...	2014-01-27 00:00:00.000	44.1	21	25	-5.1	61	25	8.38260869565217

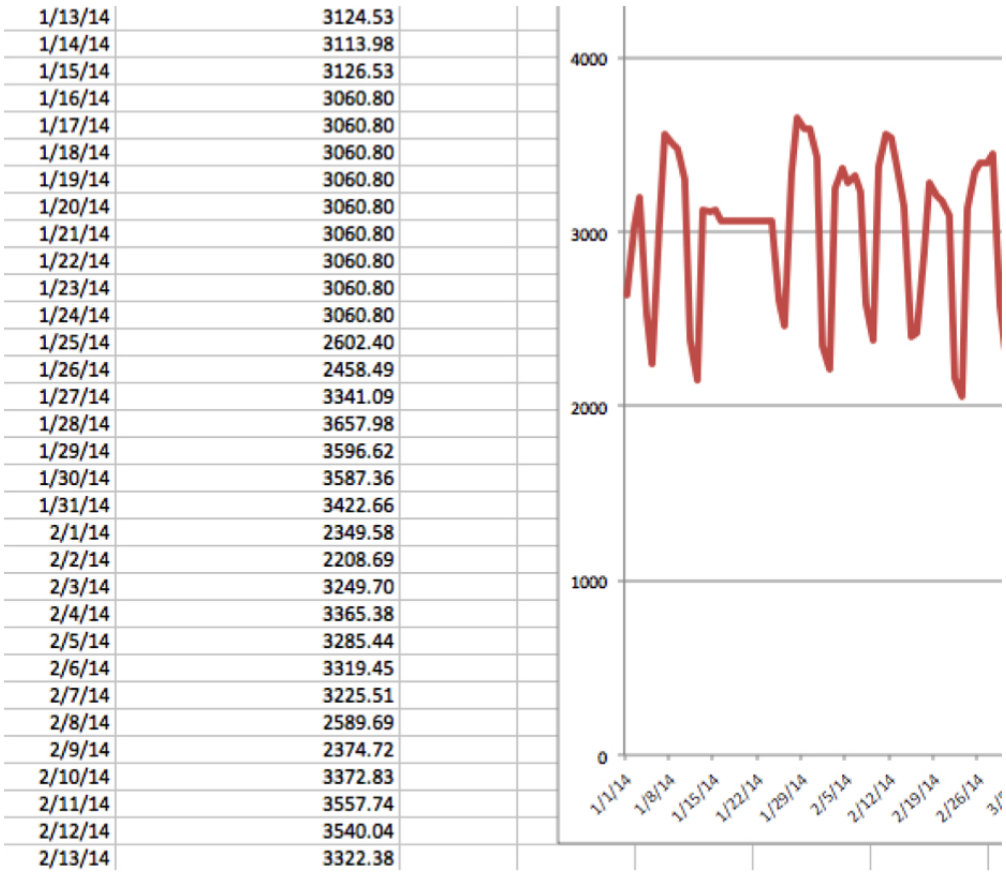
Figure 3.14: Fault injection with data gap.

real-world environment, *e.g.*, for some sensor reading. Figure 3.15 illustrates an example fault injection with constant values. The electric meter reading for the period from January 16 to January 24, 2014 remains exactly the same as 3060.80, which is erroneous. The chart at the right hand side shows the horizontal line for this period.

The third type of faults injected is **random values** simulated using data randomization. This type of faults happen when unreasonable spikes occur during the trendy movement. Figure 3.16 illustrates an example fault injection with random values. At around 10am, the reading dropped to zero and then went back to the normal.

Operator Actions

Operator actions in my experiments include starting up HVAC system (on/off) and adjusting BMS (building management system) temperature set-points points in the heating/cooling system based on the forecast produced by the space temperature forecasting/recommendation application. The operator usually follows the forecasts/recommendations to start up the building's HVAC, in this case cooling, and set target space temperature set-points. If the operator takes incorrect action, the building system would not work properly and may lead to excess cooling or heating, or even system blowout.



1/13/14	3124.53
1/14/14	3113.98
1/15/14	3126.53
1/16/14	3060.80
1/17/14	3060.80
1/18/14	3060.80
1/19/14	3060.80
1/20/14	3060.80
1/21/14	3060.80
1/22/14	3060.80
1/23/14	3060.80
1/24/14	3060.80
1/25/14	2602.40
1/26/14	2458.49
1/27/14	3341.09
1/28/14	3657.98
1/29/14	3596.62
1/30/14	3587.36
1/31/14	3422.66
2/1/14	2349.58
2/2/14	2208.69
2/3/14	3249.70
2/4/14	3365.38
2/5/14	3285.44
2/6/14	3319.45
2/7/14	3225.51
2/8/14	2589.69
2/9/14	2374.72
2/10/14	3372.83
2/11/14	3557.74
2/12/14	3540.04
2/13/14	3322.38

Figure 3.15: Fault injection with constant value.

The assumption of the controlled experiments is that operator usually follows the forecasts/recommendations to take actions. And in the case of ARIS giving different recommendation, the operator takes the actions suggested by the ARIS.

Physical System Simulator

As illustrated in Figure 3.17, a building's heating and cooling system consists of many different physical components such as incoming (return) air fan, outgoing air fan, heating coils, cooling units, and pumps etc.

In my experiments, the effects of the output results and the operator actions on the cyber-physical system are simulated using a thermal response model simulator [137]. The simulator can represent the physical components of the cyber-physical system. Buildings do not respond instantaneously to fluctuations in heat input, whether from external (e.g.

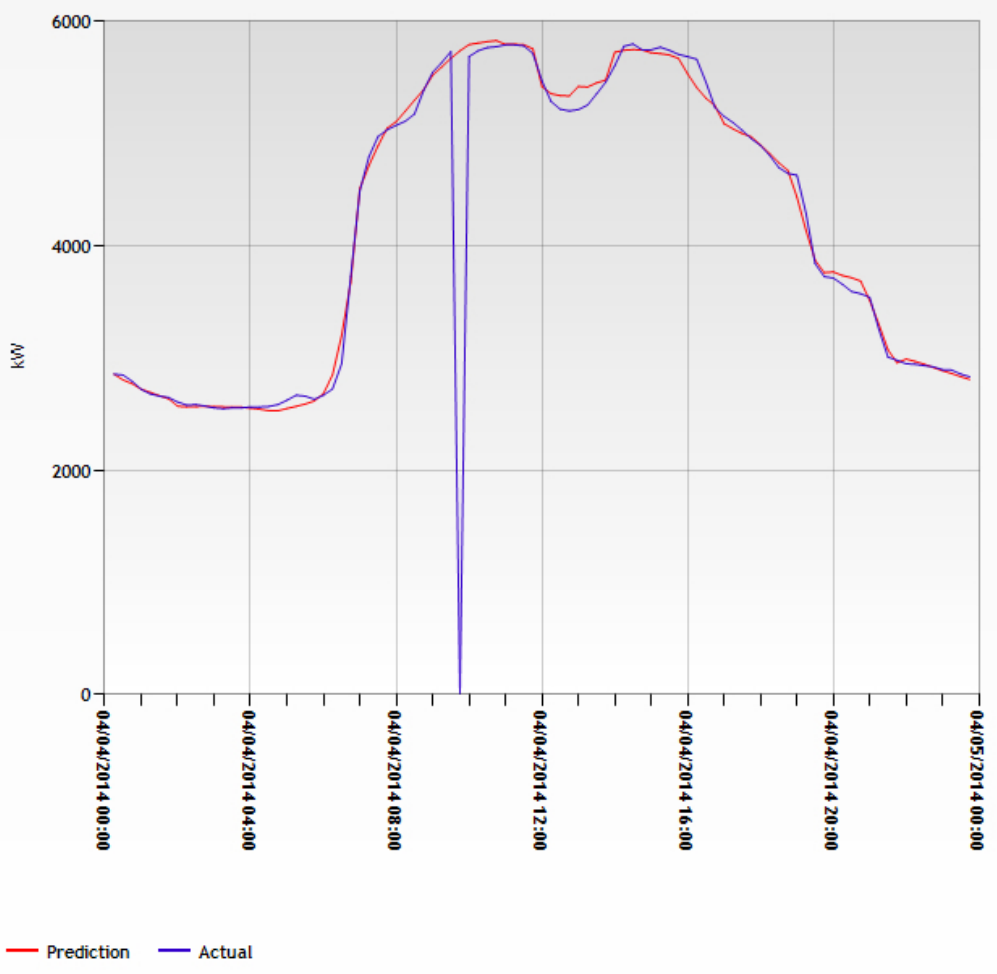


Figure 3.16: Fault injection with random spike.

sunlight, outside temperature) or internal sources such as occupancy level and human activity. The rate at which a building or parts of a building, such as a floor, will heat up or cool down depends largely on its thermal mass and the capacity of the HVAC system. The degree to which comfort conditions can be achieved depends not only on internal ambient temperatures, but also on the temperature of the internal masses.

Thermal response model can be used to accurately estimate space temperature change, *i.e.*, effects on the physical system, using following formula [137]:

$$T = T_0 - \Delta T(1 - e^{-\frac{t}{RC}}),$$

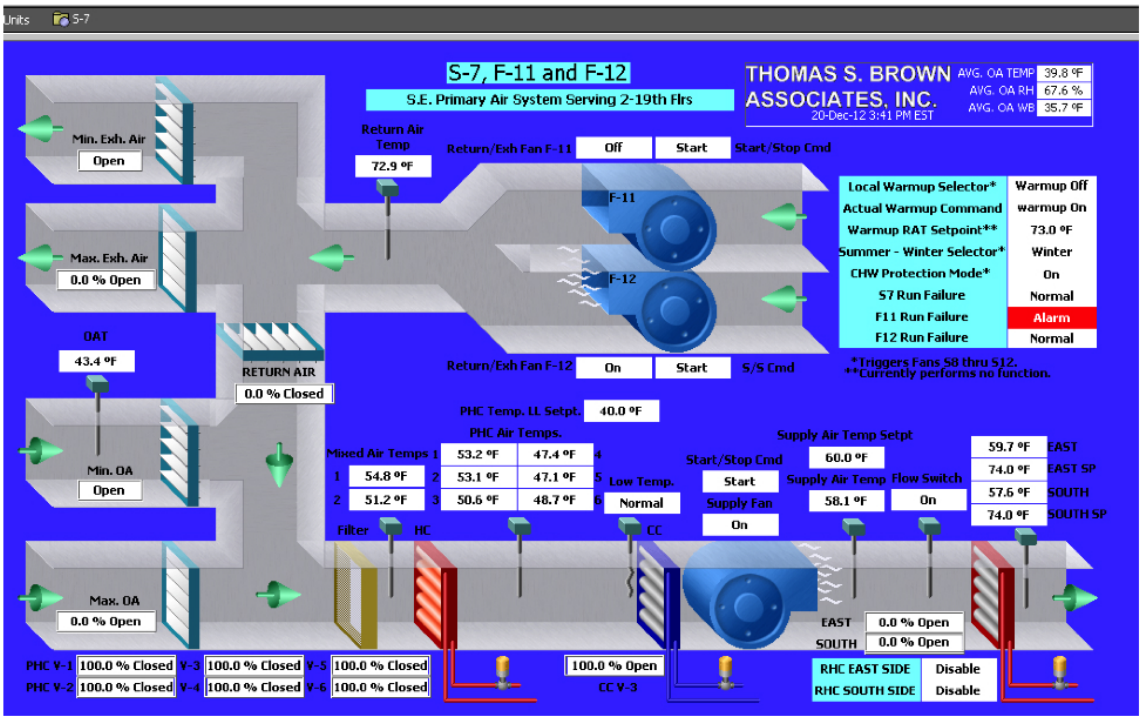


Figure 3.17: An example building heating and cooling system.

where T is the estimated temperature, T_0 is the temperature at start-up time, ΔT is the difference between T_0 and the steady state temperature indicated by the BMS target temperature set-points, t is the time from when the chiller/heater is turned on, RC is a building specific constant.

An example simulation result for a building's temperature change using thermal response model is illustrated in Figure 3.18, which shows that the model's output closely resembles the real outcome in the physical system. The prediction, in red line, in this case is the simulation results using the thermal response model. This figure illustrates the closeness between the simulation temperature results and the actual temperature. This simulator was implemented by X. Li using python and its source code is in the report [137].

Reliability Metrics

In the control experiments, three reliability metrics as defined in section 3.5.1 are used to measure the system reliability:

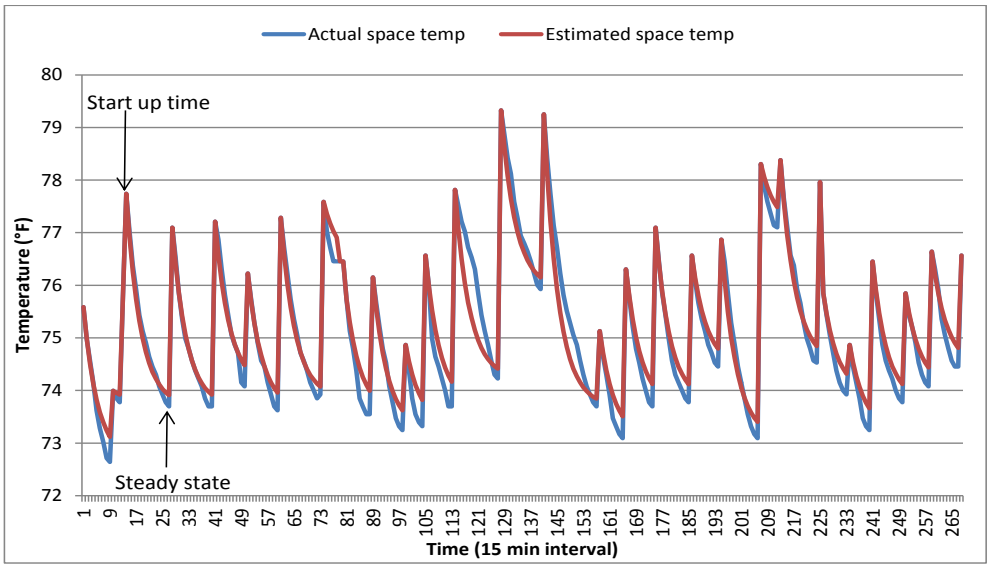


Figure 3.18: Thermal response model estimation for a building's northeast quadrant of second floor in summer days[137].

- *Number of failures* during the experiment period
- *Failure rate*, which equals number of failures divided by total days of test
- *Mean time between failures (MTBF)*, which equals 1/failure rate, shows on average how many days before another failure may happen

A Running Example Test Case

As illustrated in Figure 3.19 and 3.20, the data quality issues, in this case some abnormal data by fault injection such as data gaps for certain period, led to incorrect prediction results for the CPS. These predictions, in red line, are the output of the forecasting system and can be incorrect due to data quality issues. As people can see from Figure 3.19, because the start-up time forecasts for October 7 and October 8 are too early, the excess cooling and incorrect temperature set-points drive the room temperature too low, *i.e.*, below 70 °F, which are considered failures in this experiment.

While, in the case of Figure 3.20, ARIS is able to inform operator to ignore the forecasts and take proper actions based on recommendations from ARIS. Because ARIS evaluates

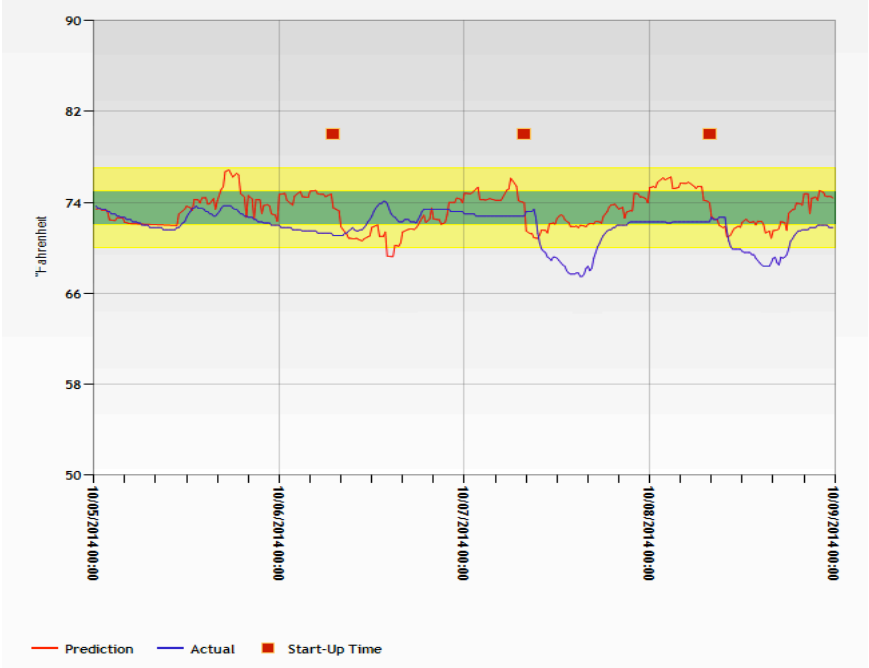


Figure 3.19: CPS space temperature and start-up time forecasting system without ARIS.

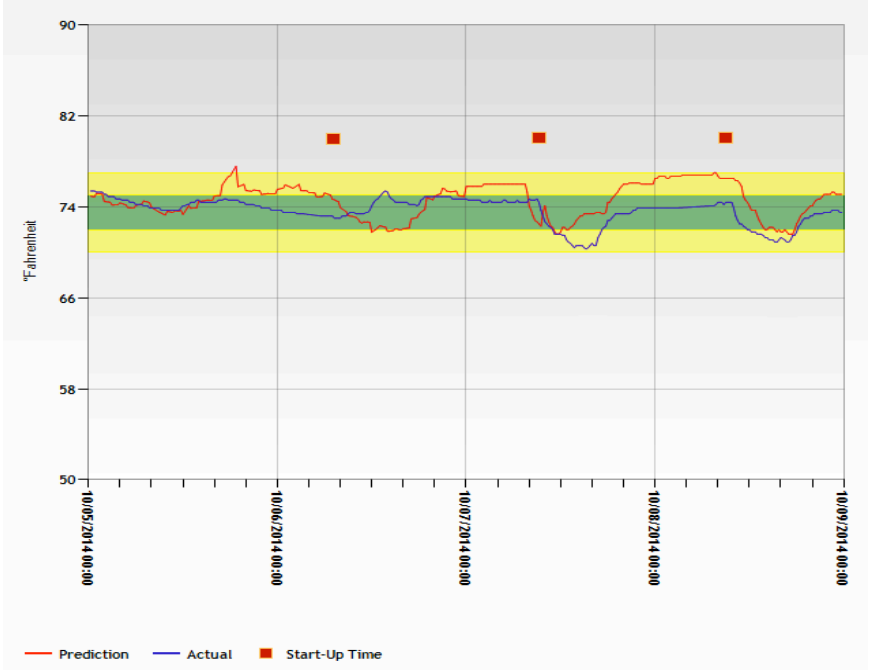


Figure 3.20: CPS space temperature and start-up time forecasting system with ARIS.

data quality on the fly and informs operator some results are erroneous and should be ignored, the CPS with ARIS was able to better cope with the data quality issues and the operator's actions are more accurate, which in turn produced better results and improved

system reliability.

Results of Data Quality Analysis

Table 3.2 shows some statistical results of three experiments using the reliability metrics described in section 3.5. These results are based on experiments of simulating one building's HVAC start-up control system using historic data for the period from July 1, 2014 to October 1, 2014. The CPS with ARIS implemented has improved reliability based on lower failure instances, lower failure rate and longer MTBF. Figures 3.21, 3.22 and 3.23 further illustrates that the CPS with ARIS has the best system reliability comparing to the baseline and rule-based system.

Experiment	#Failure	Failure rate	MTBF (days)
Baseline	12	0.130	7.7
Rule-based system	6	0.065	15.4
ARIS	1	0.011	90.9

Table 3.2: Comparison of results.

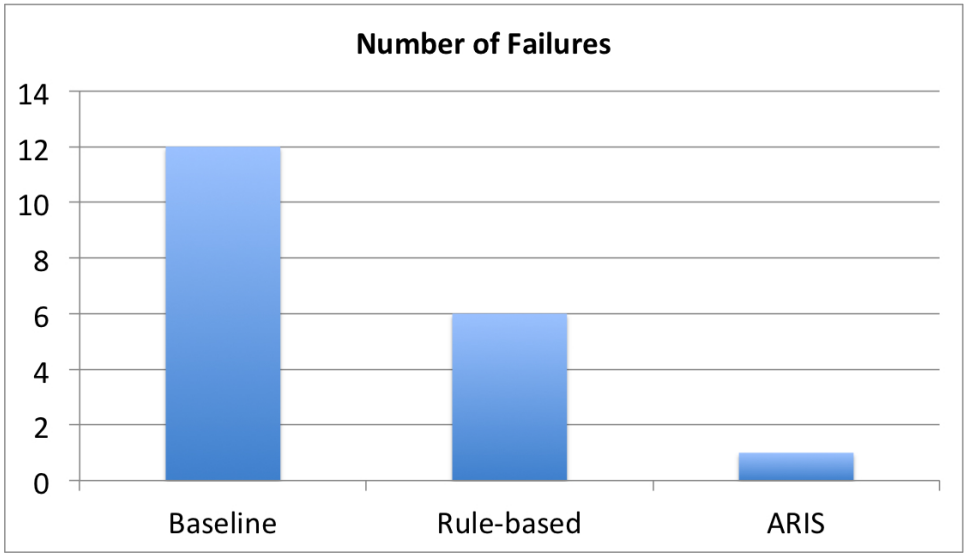


Figure 3.21: Comparison of number of failures.

Figure 3.24, 3.25 and 3.26 further illustrate the system performance from July 1, 2014 to October 1, 2014 for baseline CPS, CPS with rule-based system, and CPS with ARIS. In

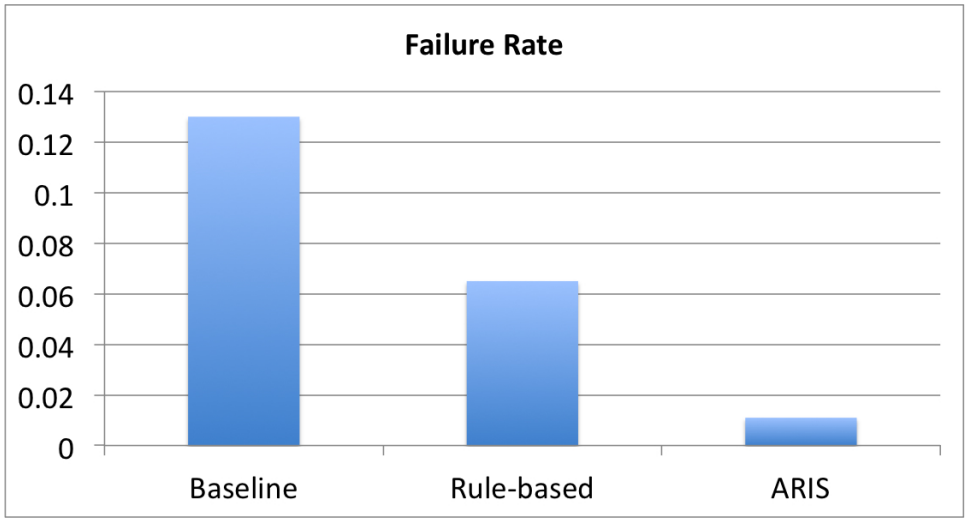


Figure 3.22: Comparison of failure rate.

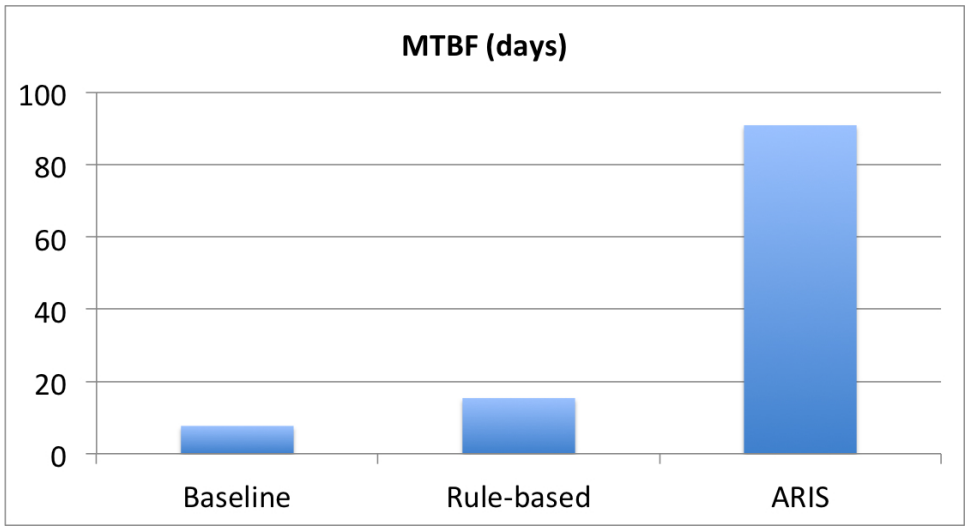


Figure 3.23: Comparison of MTBF.

these three charts, the two yellow bands range from 70 to 77 degrees Fahrenheit (°F) and the green band ranges from 72 to 75 degrees Fahrenheit (°F). The blue lines, consisting of individual data points, that are out of yellow bounds (*i.e.*, either above or below) indicate the incidence of failure. Each failure incidence may have multiple data points. But since they are referring to the same failure, they are counted as one incidence of failure. Because we only consider office hours required by the building lease and do not take into account of non-working hours (*i.e.*, weekends and time outside of 8:00 am to 6:00 pm during work days), the data shown are not contiguous.

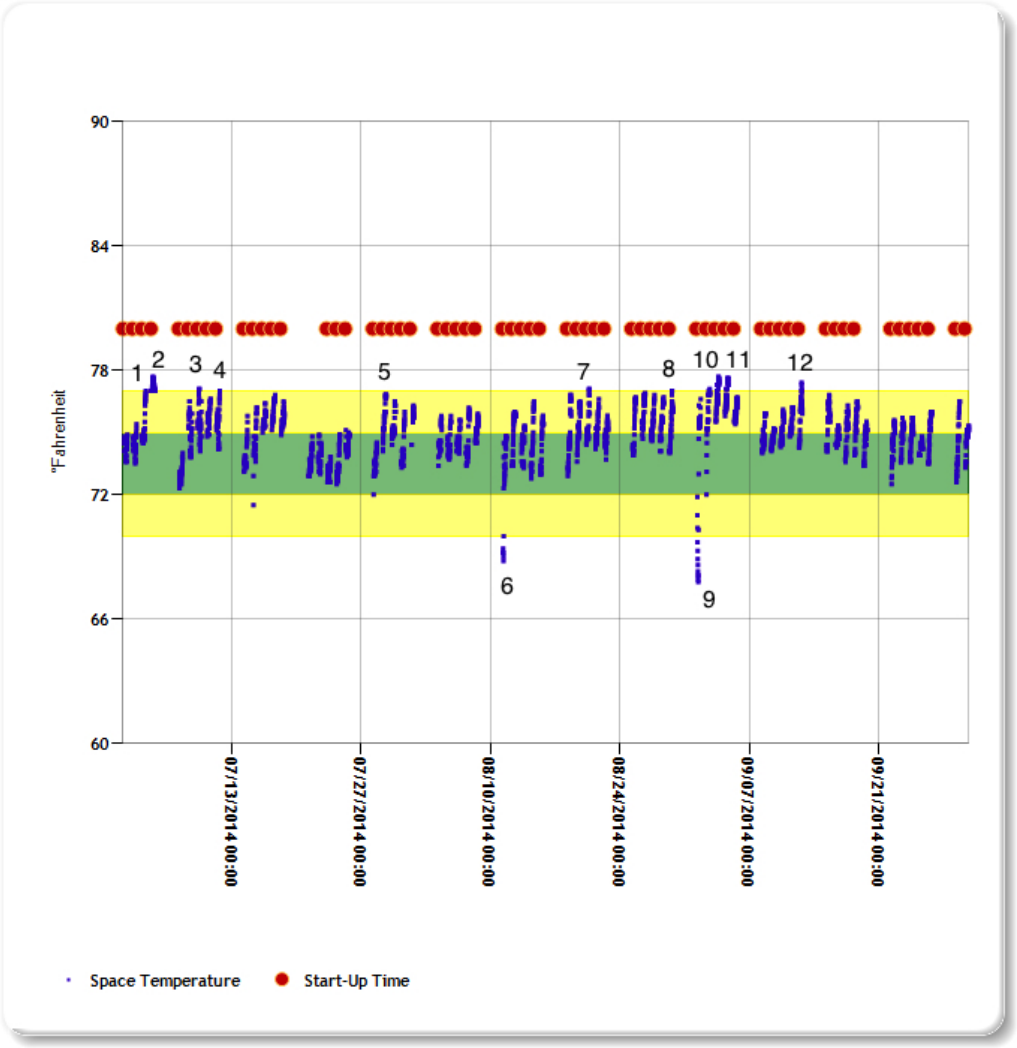


Figure 3.24: CPS space temperature and start-up time forecasting system (baseline) without ARIS (from July 1, 2014 to October 1, 2014).

The results are similar in my experiments using datasets from buildings with different BMS (Building Management System) types, *e.g.*, Schneider Andover system [12], Siemens APOGEE system [34] and Johnson Controls METASYS system [35]. Table 3.3 lists the failure rate comparison for three different buildings. The results show that the system with ARIS has smallest failure rate, hence most reliable. The experiment also demonstrates that the approach is agnostic to any type of BMS system. The main reason for this agnostic property is that ARIS focuses on processing of the data itself with regards to its quality for the specific circumstance disregarding other non-data related information. In this experiment,

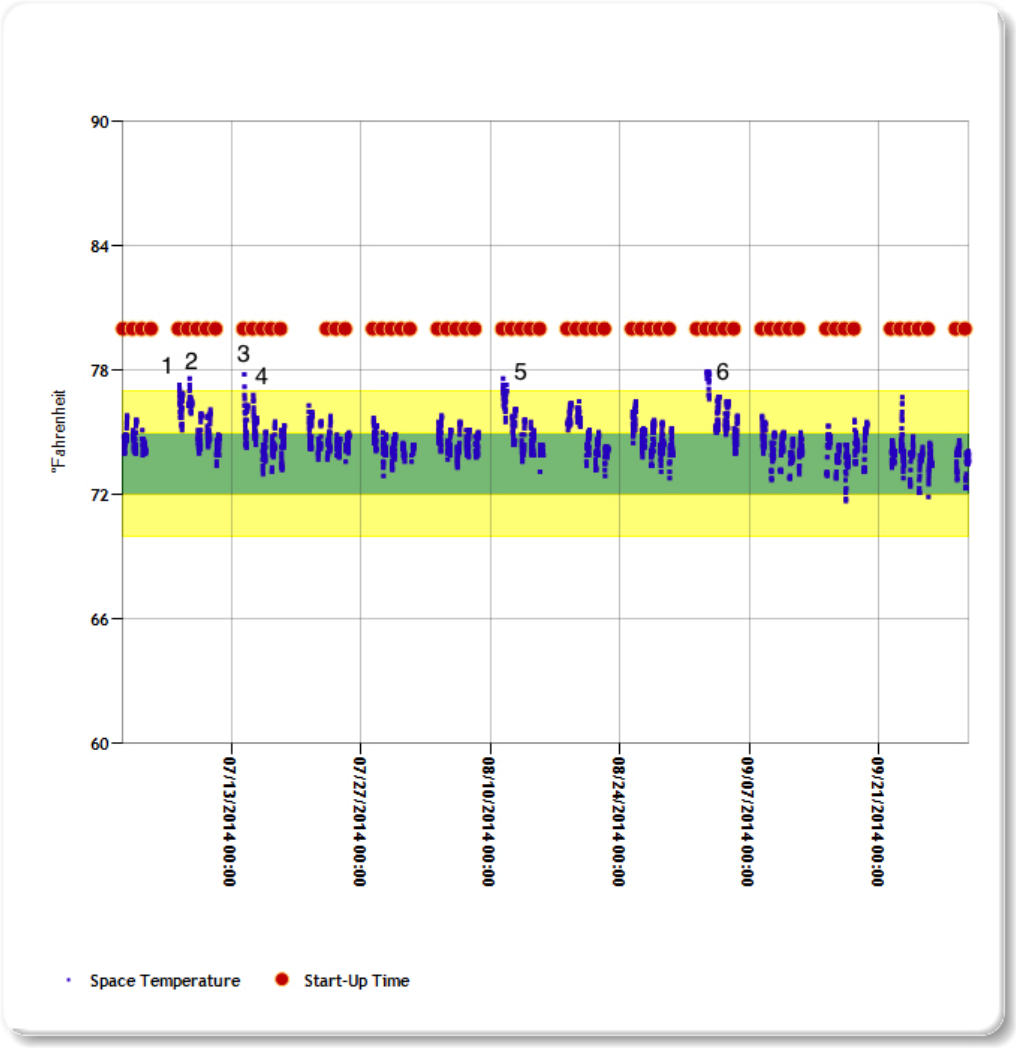


Figure 3.25: CPS space temperature and start-up time forecasting system (rule-based) without ARIS (from July 1, 2014 to October 1, 2014).

different BMS systems may be drastically different in their hardware and communication protocols, *e.g.*, METASYS uses proprietary communication channel while Andover and APOGEE use standard BACnet protocol [92], their final data repositories (mostly time series dataset) look very much the same after formatting.

Furthermore, as illustrated in Figure 3.27, with amount of faults injected into CPS data increasing, *i.e.*, fault density increasing, reliability for each system has different trending pattern. In the figure, system reliability (R_0) denotes system reliability, measured with MTBF (Mean Time Between Failures), for baseline CPS without any additional technique, system

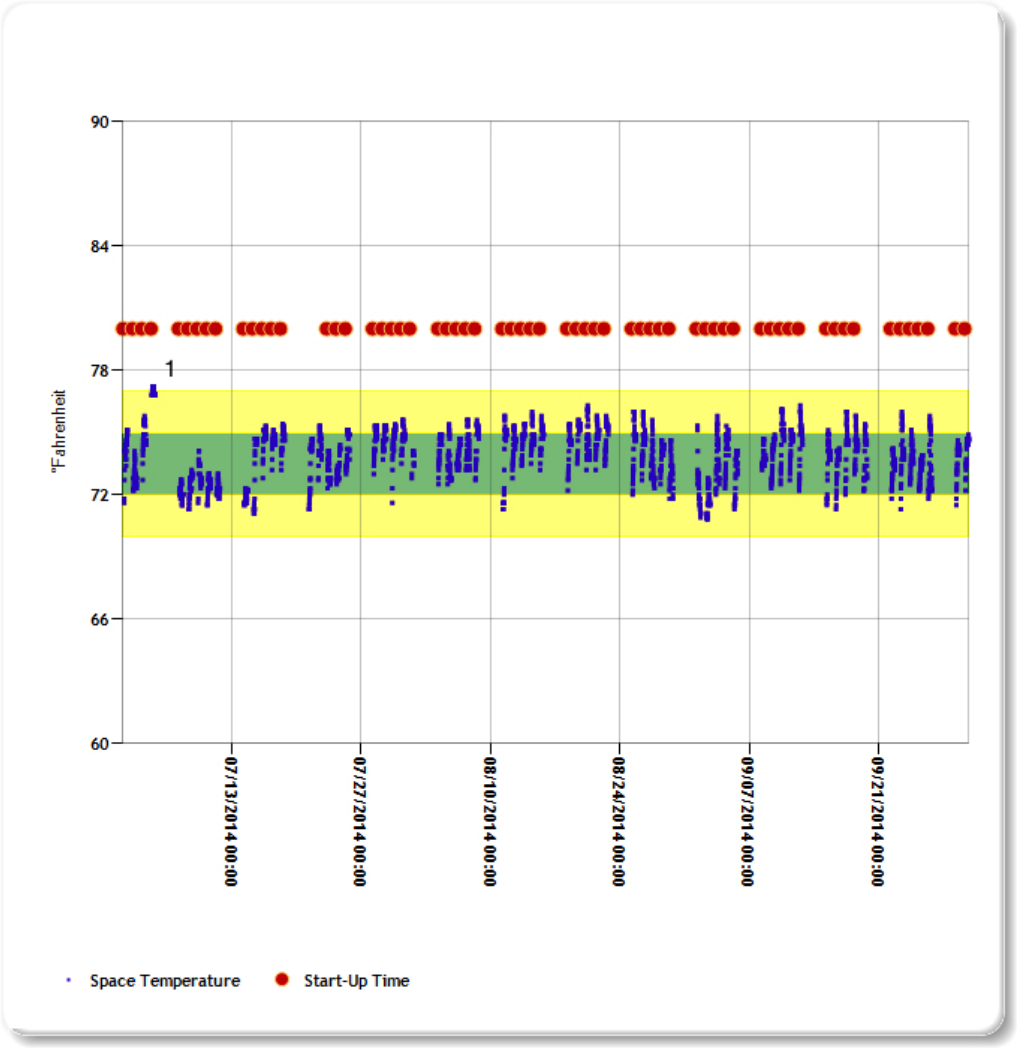


Figure 3.26: CPS space temperature and start-up time forecasting system with ARIS (from July 1, 2014 to October 1, 2014).

Experiment	Failure rate for building with Andover	Failure rate for building with APOGEE	Failure rate for building with METASYS
Baseline	0.130	0.119	0.152
Rule-based system	0.065	0.054	0.087
ARIS	0.011	0.022	0.032

Table 3.3: Comparison of results.

reliability (R_1) for CPS with ARIS, and system reliability (R_2) for CPS with rule-based technique. The result further validates that reliability metric $R_1 > R_0$ and $R_1 > R_2$ under different level of fault density. Table 3.4 lists the experimental results of reliability measured in MTBF (days) and Table 3.5 lists their corresponding failure rates.

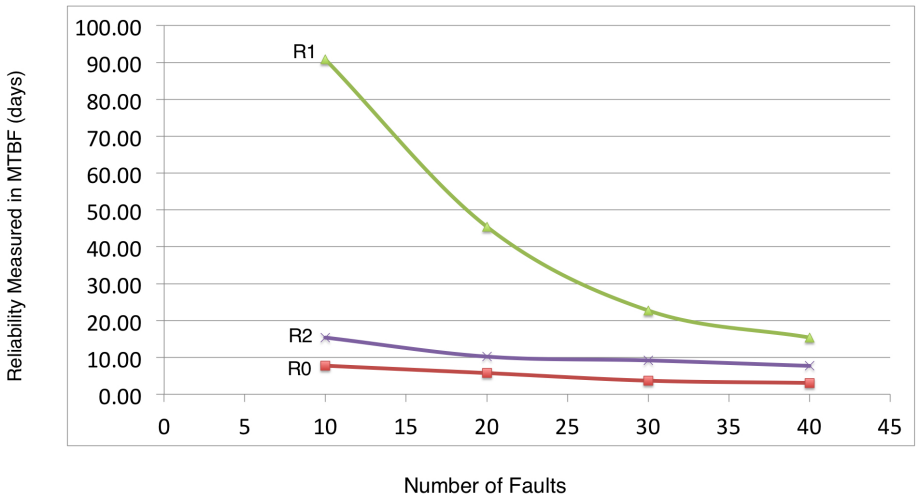


Figure 3.27: Comparison of reliability (measured in MTBF) trend versus fault density.

Number of faults	R_0	R_1	R_2
10	7.69	90.91	15.38
20	5.75	45.45	10.20
30	3.68	22.73	9.17
40	3.07	15.38	7.69

Table 3.4: Comparison of reliability (measured in MTBF) versus fault density.

Number of faults	Failure Rate$_0$	Failure Rate$_1$	Failure Rate$_2$
10	0.130	0.011	0.065
20	0.174	0.022	0.087
30	0.272	0.044	0.109
40	0.326	0.065	0.130

Table 3.5: Comparison of failure rate versus fault density.

Results of Self-Tuning

In these controlled experiments, self-tuning uses performance metric MAPE (Mean Absolute Percentage Error) [213] to measure accuracy of the data analysis models. Self-tuning also uses parameters estimation and statistical trend detection to improve the accuracy of the data quality analysis. Please note that this self-tuning applies to ARIS system itself, not to tune the whole CPS.

In my experiments, self-tuning is used to determine parameters for optimal outcome of a

Support Vector Regression (SVR) algorithm. Specifically, a nu-SVR is used with following parameters, implemented using the libsvm library [138]:

- -s (svn_type to be nu_SVR)
- -k (kernel function to be RBF–radial basis function)
- -c (cost)
- -g (gamma value in kernel function)
- -n (nu value of nu-SVR)

Furthermore, performance metric MAPE (Mean Absolute Percentage Error) is used to measure accuracy of the data quality analysis. As listed below, the accuracy not only depends on the parameters of the model, but also the input data range. Thus, the parameter tuning and trend detection are important to ensure the better performance of the data quality analysis.

- 5.8813 % using September of 2008-2013 and August 2014 as training data to predict data quality trend for September 2014
- 6.6870 % using September of 2009-2013 and August 2014 as training data to predict data quality trend for September 2014
- 6.8176 % using September of 2010-2013 and August 2014 as training data to predict data quality trend for September 2014
- 7.0826 % using September of 2011-2012, 2013 and August 2014 as training data to predict data quality trend for September 2014
- 7.5949 % using September of 2012-2013 as training data to predict data quality trend for September 2014

Summary

In summary, my experiments show that the data quality analysis and self-tuning techniques are effective. Comparing to the baseline and the CPS with state of the art rule-based system, CPS with ARIS has better system reliability. Also, the ARIS system can employ techniques such as accuracy measurement, model parameter estimation and trend detection to perform self-tuning. This self-tuning leads to an self-adaptive evaluation system that works better under system changes and operator feedback, which leads to improved system reliability.

Threat to Validity

In this section, I will discuss some potential threats to validity. First of all, people may ask whether the rule-based system employed in the experiment is sufficient and representative of the state of the art. According to my research, the ASHRAE (American Society of Heating, Refrigerating and Air-Conditioning Engineers) Standard 90.1-2013 [99] is the latest standard in this field and it is widely used by industry. The rule-based system used in my experiment closely resembles the common practice adopted by building operations. There are more complex rule-based systems for building control, mostly used for equipment diagnostics but not for routine day to day operations.

Secondly, people may question whether the fault injection techniques used are good representation of the real-world problems. During design phase of my controlled experiments, a lot of historic data has been analyzed to derive the pattern of the faults that are common for the cyber-physical systems being studied. The faults injected to the CPS data are representative of the common data issues in the CPS of this domain.

Lastly, people may question whether the experiment can be used to reach a general conclusion. In my experiments, ARIS system using data quality analysis and self-tuning is an independent component. It is agnostic to the CPS being implemented and processes the CPS data, in this case, building's internal data such as space temperature, equipment settings, occupancy, and exogenous data such as weather information, day of the week, holiday, and

utilities data on the fly. Also, my experiments show that using different kinds of buildings' data lead to same result in terms of system reliability improvement. Thus, it is reasonable to generalize this controlled experiment applies to other CPS in the domain of interest.

3.5.3 Real-World Experiments

Another important set of experiments is conducted in some real-world environments where unpredictable conditions often happen. The real-world experiments are important for validating the proposed approach because the deployed cyber-physical systems will not be operating in a controlled environment, and must be robust to unexpected conditions and adaptable to subsystem failures. **These real-world experiments demonstrate the approach is applicable to different cyber-physical systems in the real world and support the *third hypothesis* described in section 1.7, *i.e.*, the approach is efficient and does not add too much overhead.**

For the real-world experiments, I experimented with the prototype system on two types of cyber-physical systems: smart power grid CPS and smart building CPS.

- Smart power grid, as described in section 2.2.1, is an important type of cyber-physical system that is becoming more unreliable as more and more complex electrical, computing and communication components are added. The ML-based electrical component ranking systems are used for preventive maintenance of the power grid. I applied the ARIS system on some smart power grid ML system to prove that the proposed approach can in fact improve system reliability for cyber-physical systems and it does not incur too much extra cost. In the following section 3.5.4, I will describe some studies of the experiments on a smart power grid ML system.

- Smart building, as described in section 2.2.2, is another important type of cyber-physical system that is becoming increasingly complex and unreliable. The sensors and meters in the building collect various types of status and usage data for the

building operators and energy control systems to operate the building with the help of building management system (BMS). The predictive building energy optimization system applies machine learning (ML) to historical and real-time building data and other exogenous information to improve the efficiency of building systems. I deployed and experimented with the ARIS system on a smart building BMS system and ML system to prove the proposed approach is effective and efficient in improving system reliability. In the following sections 3.5.5 and 3.5.6, I will describe some studies of the experiments on smart building BMS system and ML system .

3.5.4 Study #1: Experiments on Smart Power Grid ML System

In the following subsections, I present a study on improving smart power grid system reliability using data quality analysis. First, I will describe some background information on power grid reliability. Then I will describe the NOVA system, a prototype implementation of the ARIS system, followed by experimental results and analysis. Note that this NOVA system is not self-tuning.

To improve power grid system reliability using ML based preventive maintenance, it requires objective evaluation of the machine learning and data mining software to ensure they are running as expected, the quality of the data input and output, and the consequential benefits, *i.e.*, physical system improvements, after the actions recommended by the machine learning and data mining systems have been taken. For this purpose, I have developed NOVA system, a prototype implementation of ARIS system, that is able to provide such a data quality analysis and reliability evaluation [242, 243].

NOVA conducts an automated and integrated evaluation at multiple stages along the workflow of the smart power grid CPS. There are three steps provided through a unified user interface, as illustrated in Figure 3.28: first, evaluation of the input data; second, evaluation of the machine learning and data mining output; third, evaluation of the system's performance improvement. The results from Step 1, 2 and 3 are eventually directed to a centralized

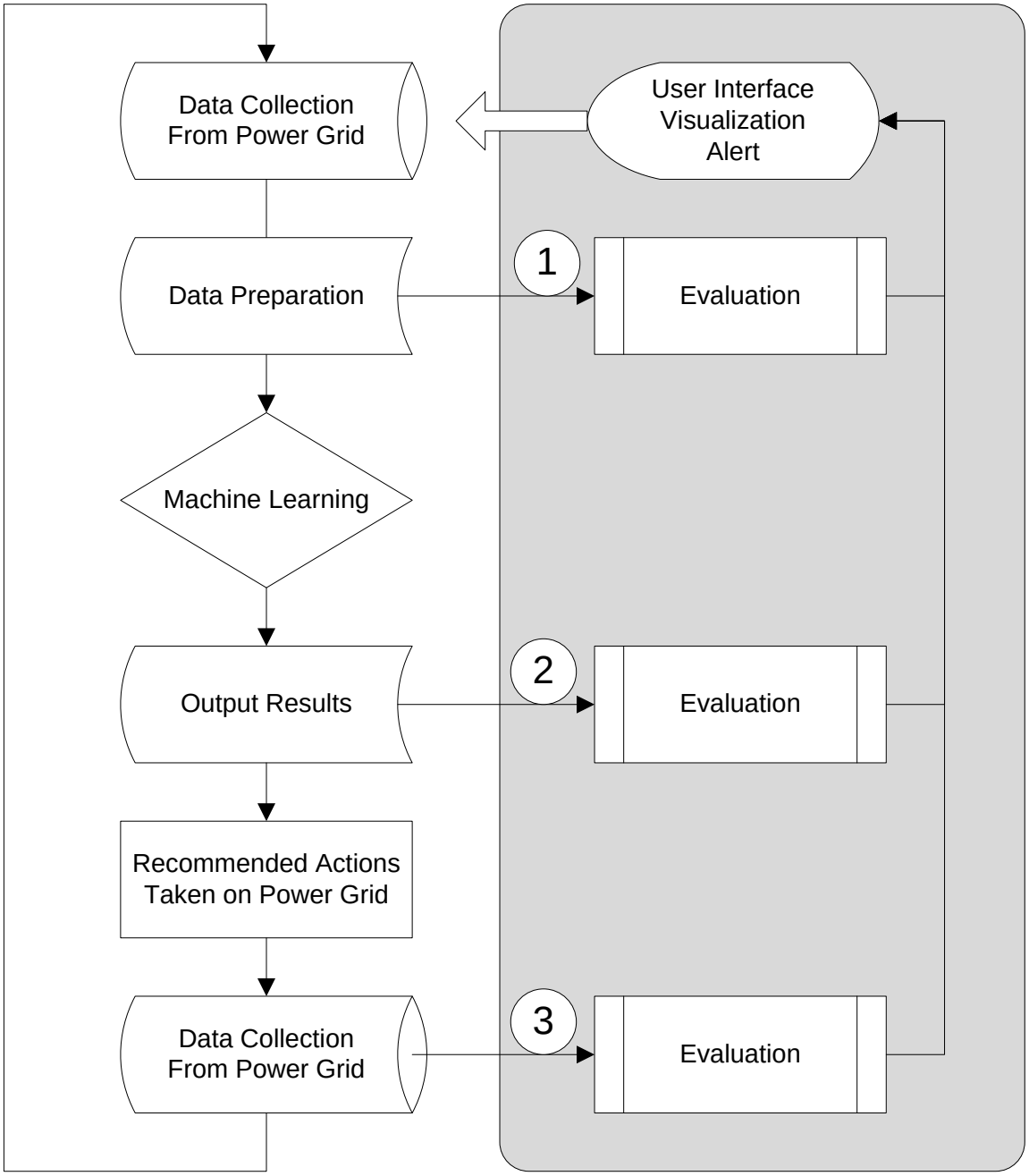


Figure 3.28: Design and workflow of NOVA for smart power grid ML system.

software dashboard for operator-in-the-loop to take actions. When abnormal results trigger pre-defined thresholds at any step, warning messages are dispatched automatically.

I implemented NOVA in evaluating MartaRank and ODDS feeder-ranking systems and

analyzed the experimental results. In the following subsections, I will describe the details of each evaluation stage and demonstrate useful summarization charts for each step.

In order for a system to perform as expected, the input data sets have to meet the pre-defined quality specifications. The evaluation process first uses *data constraints and checks* to see whether the required data exist and are up to date. Then the evaluation process conducts some more fine-grained checks, for example by using a *sparkline graph*, which is a type of information graphic characterized by its small size and high data density [220]. These checks would help researchers to correlate the changes in the input data sets with the variations of machine learning and data mining results, so that further study may be done to improve machine learning and data mining accuracy, thus leading to better rankings/actions and improved system reliability. As illustrated in Figure 3.29, in the sparkline time series graph, for the one-day period preceding an actual outage, among ten feeder attributes—maximum scaled voltage, number of joints, number of cables, peak load, etc.— being plotted, some attributes show varied patterns (*e.g.*, Attribute 1, 2, 5, 6, 7, and 10), while others are constant (*e.g.*, Attribute 3, 4, 8, and 9). These patterns may be used to improve machine learning and data mining results. For example, it may be possible that the constant attributes can be avoided so that only varied attributes are used as input data, which simplifies and improves the processing of the machine learning and data mining.

The ML system's output is a ranked list of components ordered by their susceptibility to failures. To evaluate the output data quality, I use Receiver Operator Characteristic (*ROC*) curves, and accompanying rank statistics such as the Area Under the Curve (*AUC*). The AUC is equal to the probability that a classifier will rank a randomly chosen positive instance higher than a randomly chosen negative one [31, 75]. It is in the range of [0, 1], where an AUC of 0.5 represents a random ordering, and an AUC of close to 1.0 represents better ranking with the positive examples (*i.e.*, correctly predicted examples) at the top and the negative ones at the bottom. Figure 3.30 illustrates one typical ROC curve for a feeder-ranking with AUC equals 0.768. The description for each data point (*e.g.*, 17M96

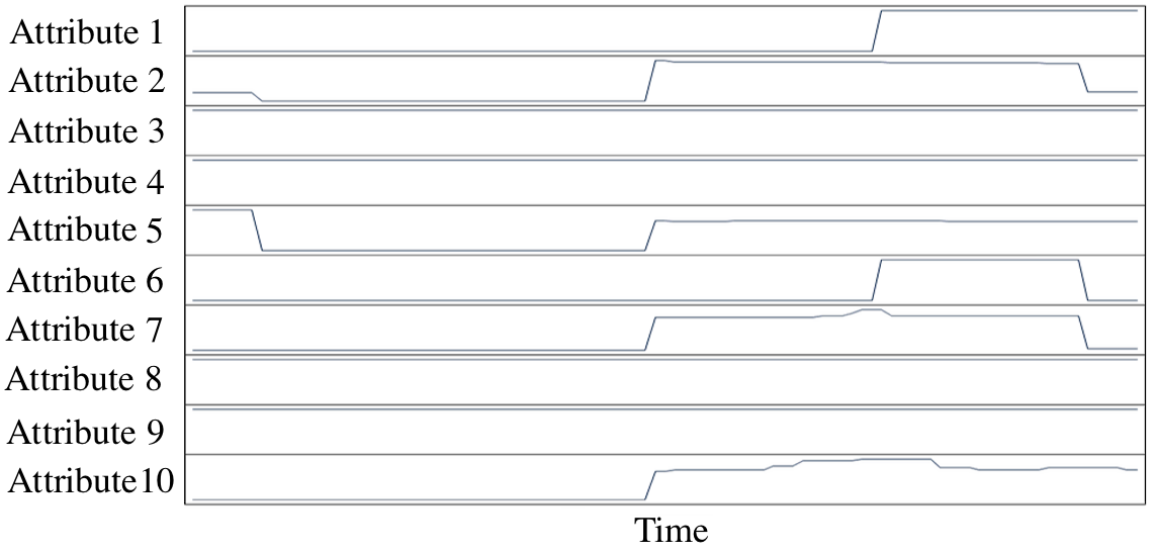


Figure 3.29: Sparkline graph for attributes data.

(511)) stands for feeder name (*e.g.*, 17M96) and its ranking (*e.g.*, 511). When the AUC is bad, *i.e.* close to 0.5, the operator is informed that the output results are close to randomness so that the operator can use alternate factors for decision-making accordingly.

The ranking systems generate new models continuously, so the evaluation is presented as a time series of AUC values as shown in Figure 3.31. The black series in the figure shows the AUC time series of ODDS and the gray series shows the ones for MartaRank, both for the time period from May 2010 to November 2010. My experiments show that MartaRank and ODDS feeder-ranking systems have comparable overall performance according to the AUC. The better the AUC results, the more accurate the component rankings are, which leads to better preventive maintenance results in improving system reliability.

After the machine learning and data mining outputs ranking results, the feeders ranked with highest susceptibility to failure are usually treated with a higher priority. The final stage of the evaluation is to validate that the recommended actions are in fact leading to the expected power system improvement, *i.e.*, fewer outages and longer time between failures. For a longer time, a log(cumulative outages) versus log(time) chart is useful for seeing the changes in the time interval between failures. This graphical analysis is also called a *Duane*

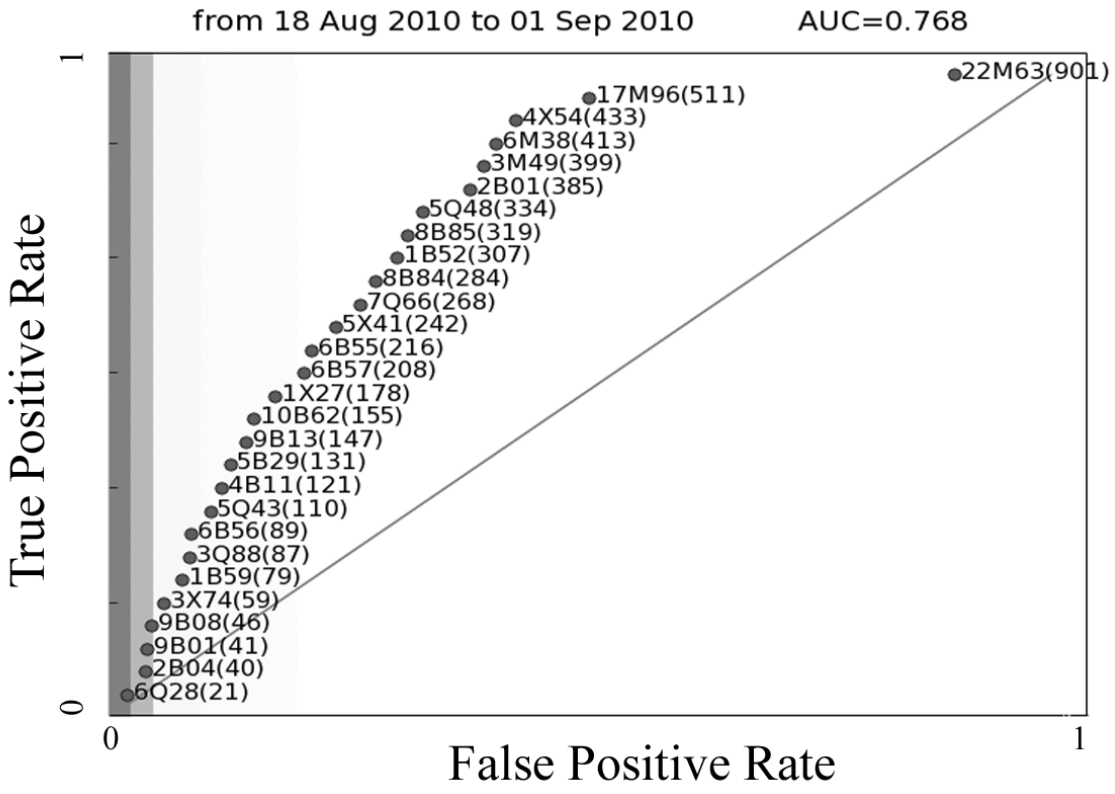


Figure 3.30: ROC curve.

plot, which is a log-log plot of the cumulative number of failures versus time [84], shown in Figure 3.32. The changing slope of the regression lines of the cumulative outages shows the improved rate of outages. If the failure rate had not changed, this log-log plot would show a straight line. The inflection period around 250th days is primarily due to the implementation of the system.

To summarize the above key steps of the NOVA system as described above, Table 3.6 lists the evaluation targets and main techniques (*e.g.*, methods, metrics, charts) used at each evaluation stage.

NOVA system has been implemented in evaluating two feeder-ranking systems in New York City's power grid since 2007. Some of its newer features were added from 2007 to 2010. New York City has over two thousand feeders. One experimental result I concluded from the evaluation using NOVA is the increasing MTBF (*Mean Time Between Failures*), *i.e.*, lower failure rate and better system reliability, for most networks. *Mean Time Between Failures*

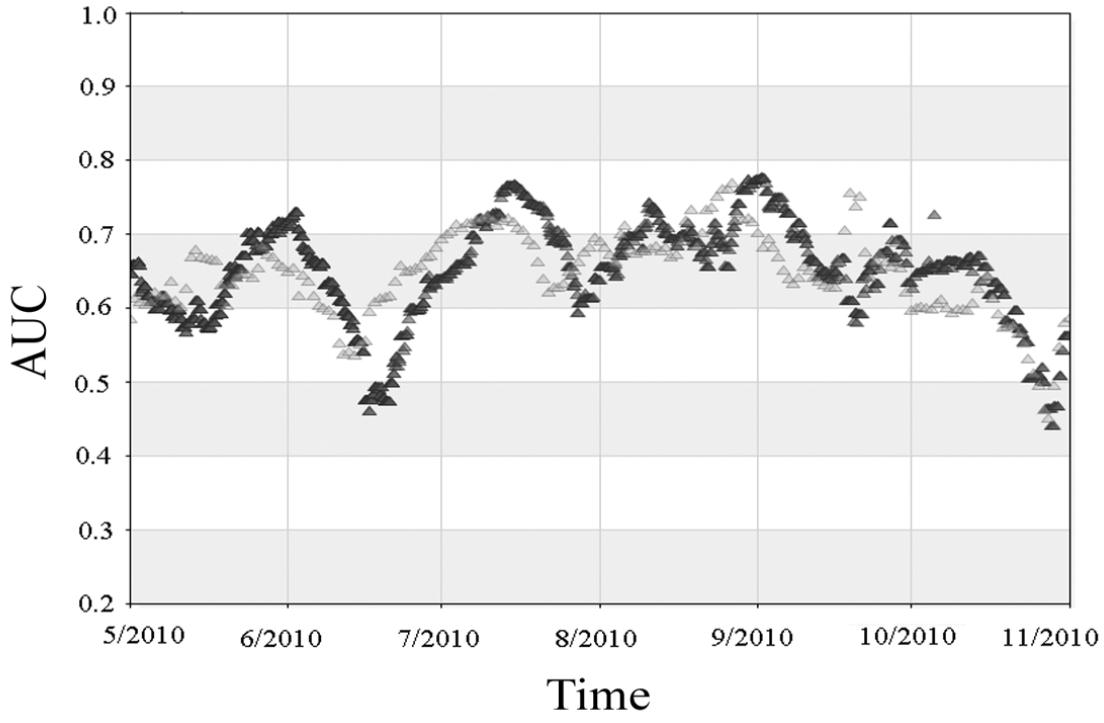


Figure 3.31: AUC cyclicity graph.

Step	Evaluation target	Methods, metrics, charts
1	Input data	Sparkline graph, data checks and constraints
2	Machine learning and data mining results	ROC curve, AUC time series
3	Physical system improvements	Duane plot, MTBF, failure rate, linear regression
	Unified user interface	Dashboard, charts, triggers, warning messages, alert emails

Table 3.6: Summary of techniques used in evaluation.

(MTBF) is the predicted elapsed time between inherent failures of a system during operation [113]. Figure 3.33 illustrates MTBF time series for all the feeders in a specific network for the period from 2002 to 2009 and the linear regression. On average, the MTBF for feeders in this network are improving over time. The MTBF improvement after deployment of NOVA in 2007 was better than pre-deployment period, as the black regression line shown in

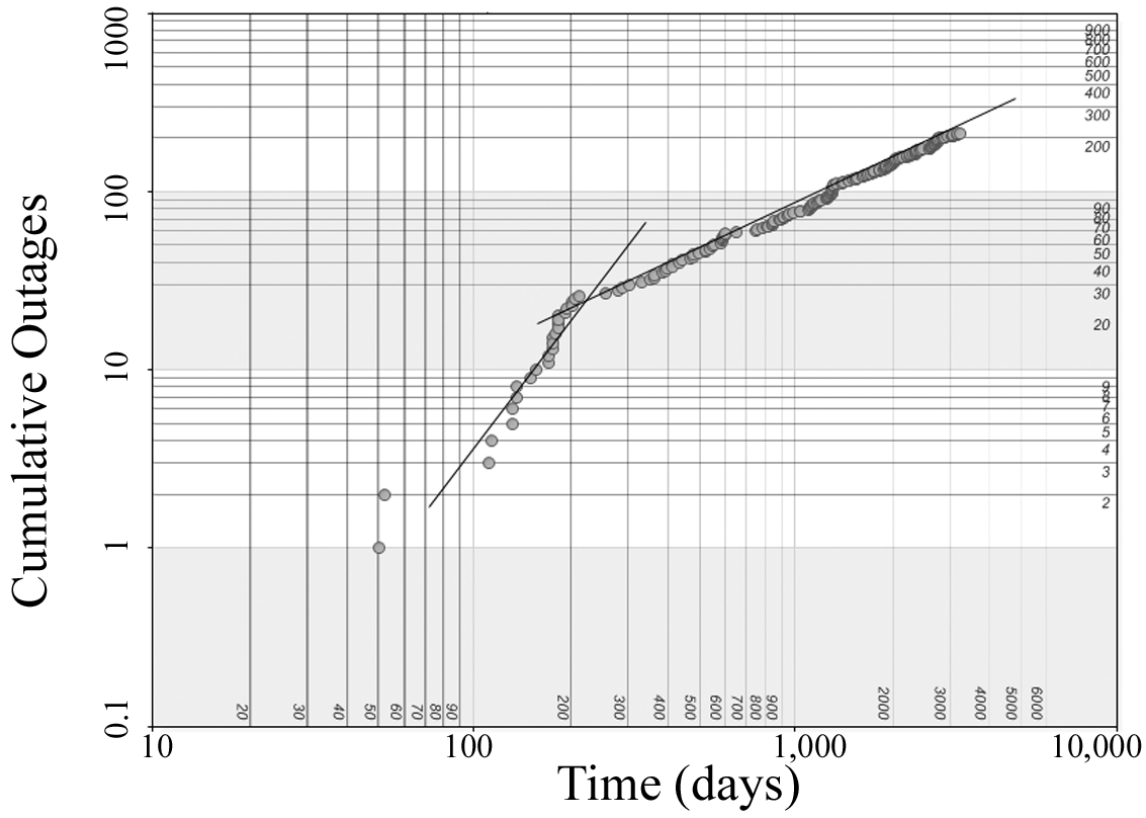


Figure 3.32: Cumulative system outages versus time log-log chart.

the graph.

Figure 3.34 illustrates the MTBF differences between year 2002 and year 2009 for each network. The bars with values above zero indicate MTBF improvements. The majority of the networks saw significant increase of MTBF. More than ten percent of the approximately 2000 feeders in the city have been serviced or replaced according to their rankings. The preventive maintenance of these highly ranked error-prone feeders improved power grid system reliability.

Table 3.7 lists the total number of feeder failures in the city from year 2005 to year 2009. The accelerated decreasing number of feeder failures shows fewer outages of the power network.

In summary, the experiments on New York City's power grid indicate that the NOVA

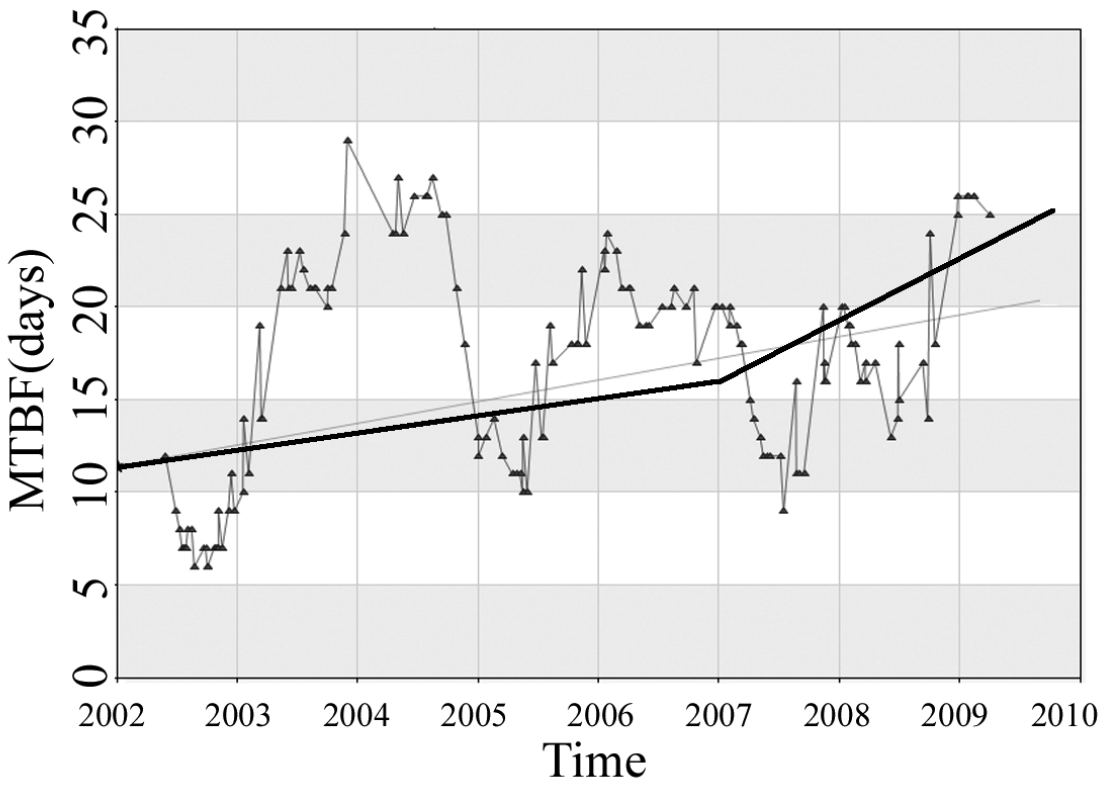


Figure 3.33: MTBF versus time and linear regression.

Year	Number of Feeder Failures	Yearly Decrease
2005	1612	
2006	1547	65
2007	1431	116
2008	1239	192
2009	1009	230

Table 3.7: Number of feeder failures in the city.

system contributes to the system reliability improvement for the smart power grid CPS, based on MTBF and number of feeder failures metrics. The system runs efficiently in parallel with the CPS. It does not add much overhead to the power grid CPS.

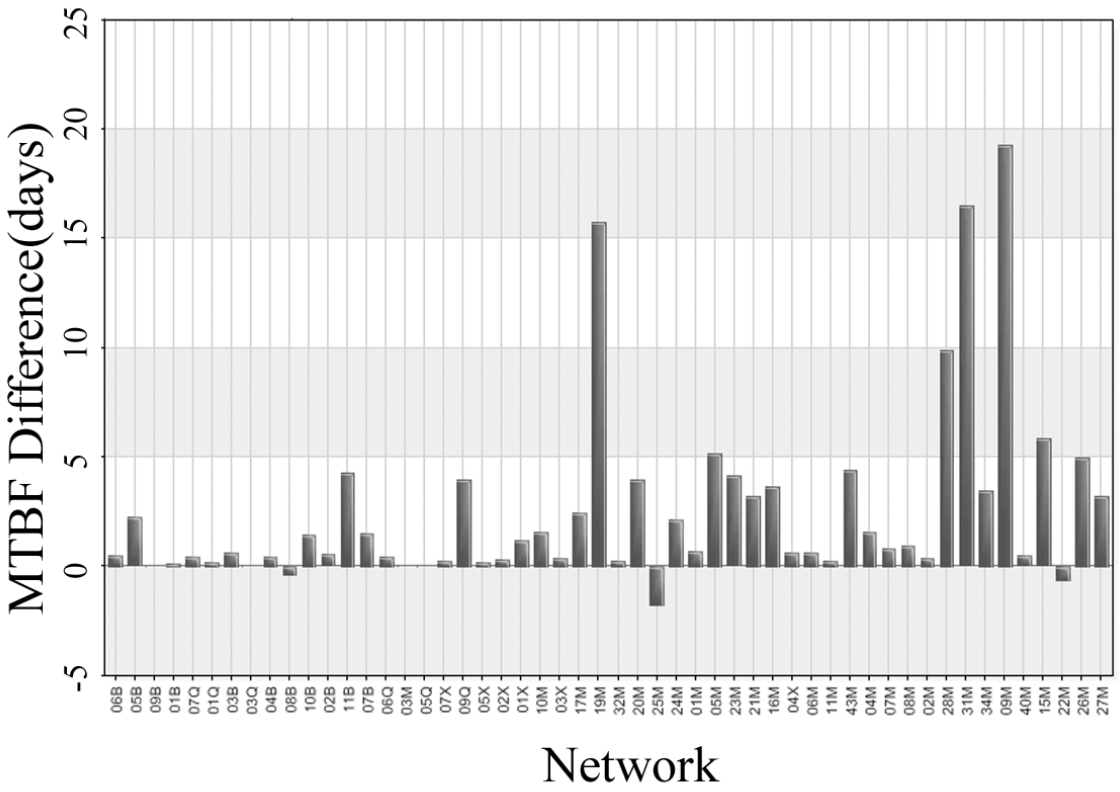


Figure 3.34: MTBF difference for each network.

3.5.5 Study #2: Experiments on Smart Building BMS System

To ensure that the smart building BMS system works reliably, an automated online evaluator monitors the building's internal and external conditions (*e.g.*, temperature, humidity, electrical load, peak load, fluctuating electricity pricing and building work and maintenance schedules) control actions (*e.g.*, adjusting lighting, turning on/off the AC/heat and shutting off elevators) and the results of those actions. This evaluator employs intelligent real-time data quality analysis components to quickly detect anomalies, such as malfunctions of digital thermostats that interfere with temperature reading or introduce variances from normal HVAC set-points, and sends feedback to building management, who can then take appropriate preventive or corrective actions. My experiments show that this automated online evaluator is responsive and effective in further ensuring that building systems continue to run reliably.

I implemented a prototype ARIS application at a large commercial building, which is a 634 foot (193 m) tall skyscraper in midtown Manhattan, New York City. Designed by Emery Roth & Sons and completed in 1969, the building has 44 floors and more than 1.8 million square feet of tenant space. Approximately 5,000 people work in the building, and there are about 1,000 visitors to the building daily. The owner and property manager is one of the largest private real estate companies in New York City. Building management has installed a state-of-the-art building energy monitoring system and BMS, which provided a live building dataset for ARIS.

As shown in Figure 3.35, ARIS worked with the BMS in parallel and processed the live data feeds via a remote data link. In my experiments, I connected ARIS to the building's various intelligent systems directly using a secure IP-based data connector. This setup simplifies the data collection and communication processes.

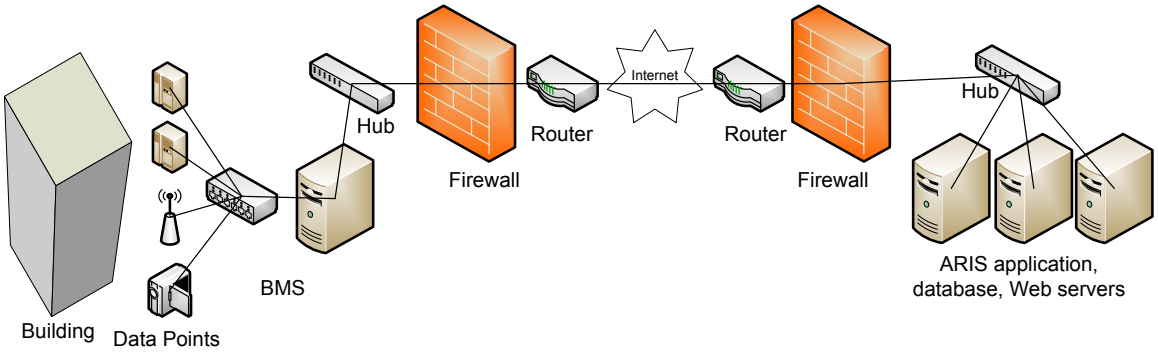


Figure 3.35: Experimental setup.

ARIS efficiently identified a large number of suspicious data anomalies obtained from 2,480 building data sources, mostly sensors, over a six-month period (December 2011 to May 2012). The relevant sensors and SCADA (supervisory control and data acquisition) data sources were investigated with the help from the building operators and engineers. The results confirmed that the majority of the issues identified were in fact caused by system failures such as BMS software errors or equipment malfunctions. Figures 3.36-3.39 present some example time-series visualization charts for selected data sources.

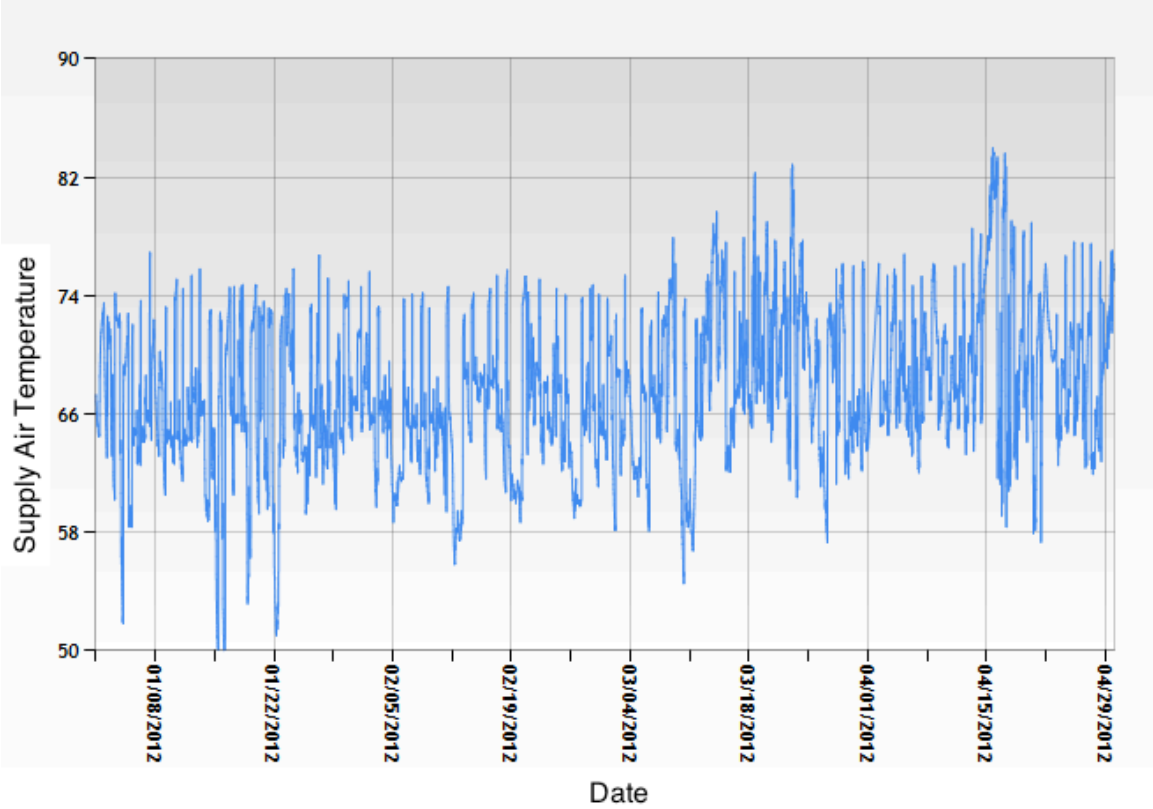


Figure 3.36: Supply air temperature time series.

Figure 3.36 shows out-of-bounds supply air temperatures that are lower than 50 °F or higher than 80 °F. After these abnormal behaviors are detected and flagged by ARIS, the building's operator can take proper control actions to maintain normal operation of the building's cooling system and ensure that service will not be disrupted. Through its SVM classification-based diagnosis, ARIS also recommends corrective actions to the operator as part of a work order management system.

As shown in Figures 3.37-3.38, the drop in maximum energy demand and steam demand around January 1 coincide with the building system shutdown during New Year's Eve and subsequent reactivation after the holiday. This kind of dip would normally be detected as anomalous behavior and a warning would be triggered and sent to building management from the automated online evaluator. However, the self-tuning capability allows the building's

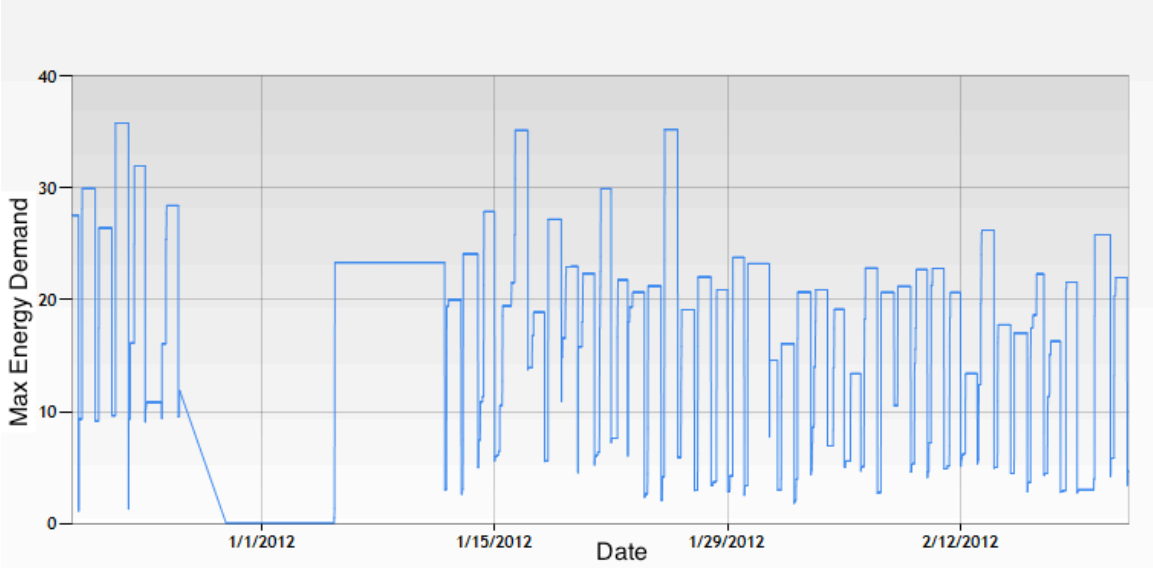


Figure 3.37: Maximum energy demand time series.

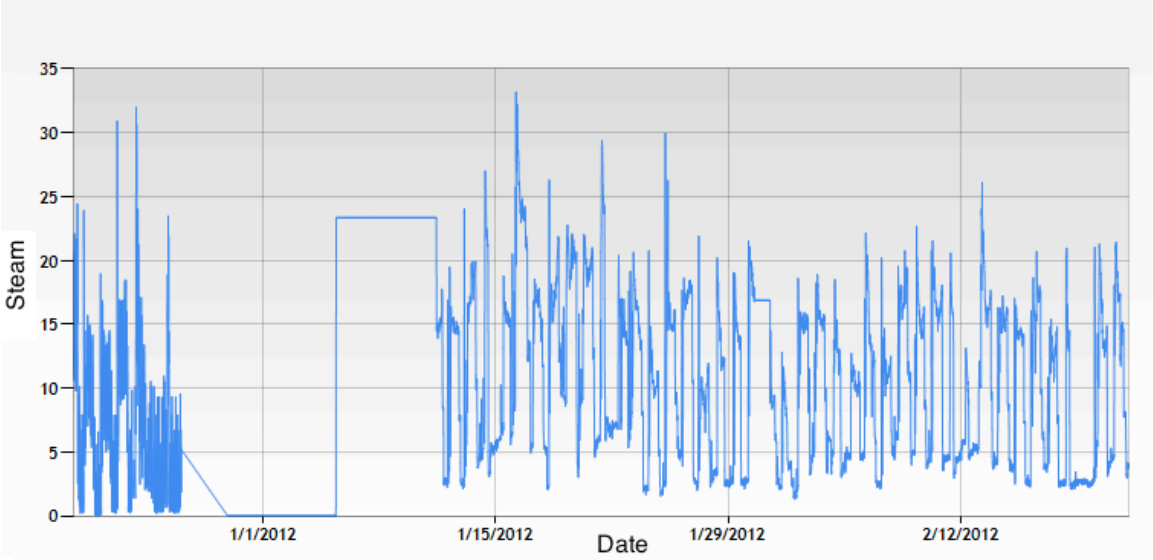


Figure 3.38: Maximum steam demand time series.

operator to notify ARIS about this abrupt schedule change to avoid the generation of unnecessary warnings.

In summary, the experiments showed that ARIS is effective in identifying and analyzing data anomalies in the smart building BMS systems for system reliability improvement. The

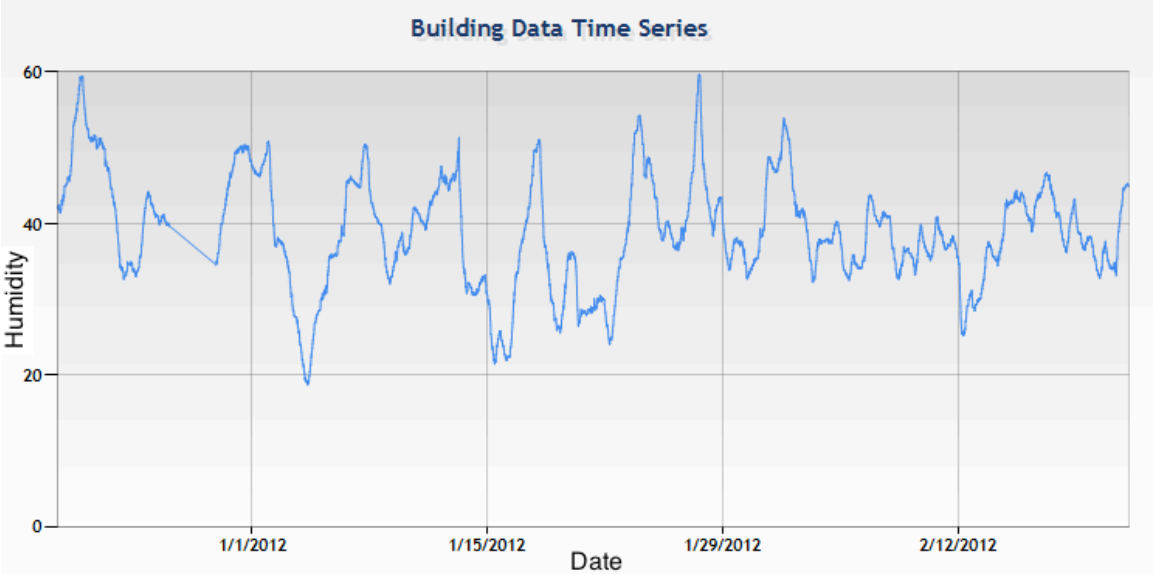


Figure 3.39: Building internal wet-bulb humidity time series.

system runs independently from the smart building BMS system. It runs efficiently and does not add much overhead to the smart building BMS system.

3.5.6 Study #3: Experiments on Smart Building ML System

I further implemented a prototype ARIS application for a smart building ML system, *i.e.,* predictive building energy optimization system as described in section 2.2.2, at a large commercial building in midtown Manhattan, New York City [244, 209]. The building's regular hours are 7:00 AM to 7:00 PM Monday through Friday, and 8:00 AM to 1:00 PM on Saturdays. The estimated energy cost of running the HVAC system of the building for an hour amounts to approximately $2,000 to $2,500 in 2011. The building uses electricity, steam and natural gas supplied by Con Edison, the main utilities company in New York City, for heating and cooling in the building. Management has installed a state-of-the-art energy monitoring system, which provides an archived data log of energy demand that can be used for predictive building energy optimization.

As illustrated in Figure 3.40, the automated online evaluation system receives data at

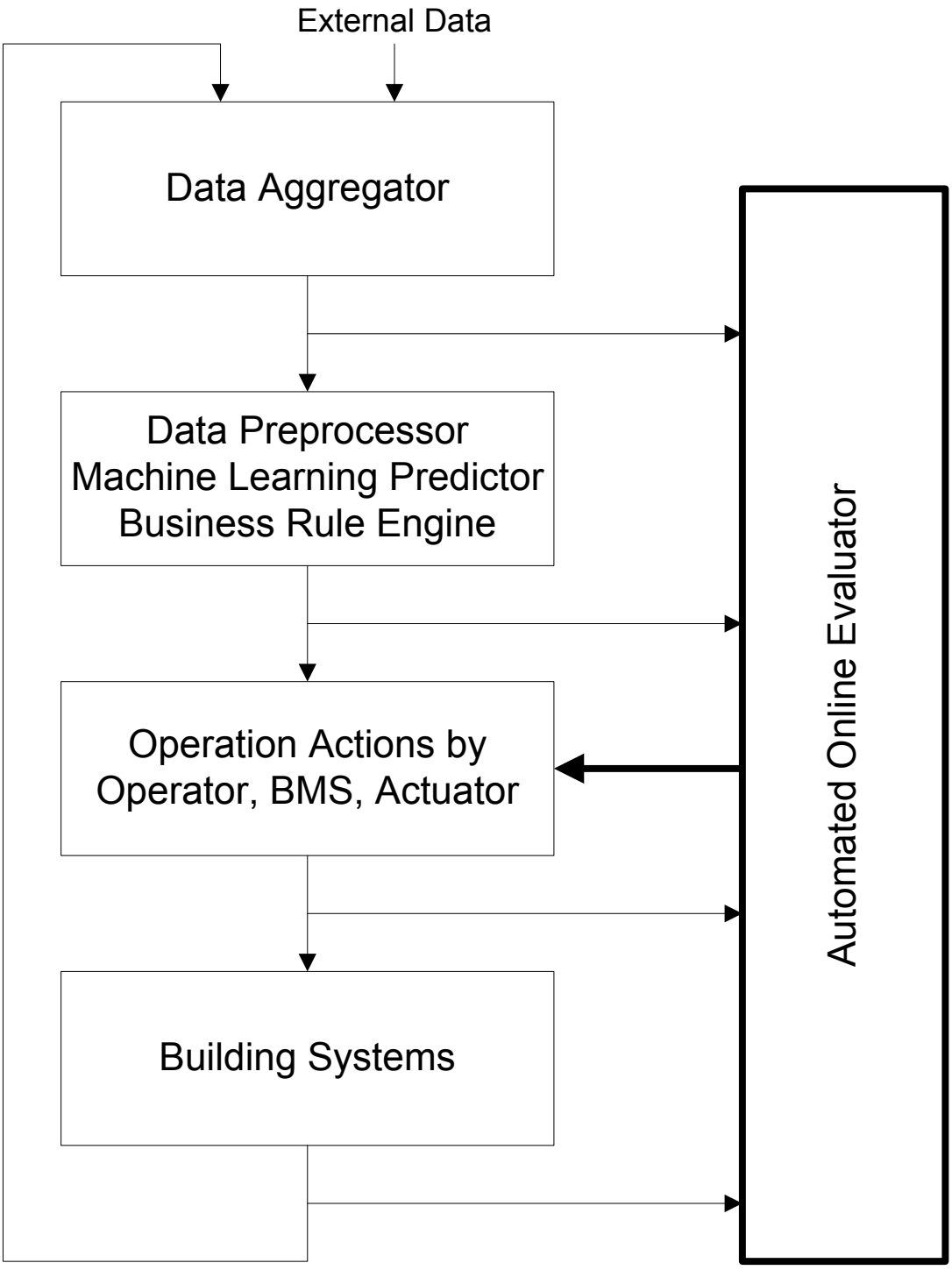


Figure 3.40: Design and workflow of ARIS for smart building ML system.

multiple stages in the ML system workflow. The evaluator employs intelligent real-time data quality analysis components to quickly detect data anomalies (*e.g.*, malfunctions of digital thermostats that interfere with temperature reading or introduce variances from normal expected HVAC set-points) and gives feedback to building management, who can then respond appropriately.

More than 10 suspicious data anomalies were identified for over a two-month period (December 2011 to January 2012) and investigated the related data sources.

The following will describe the self-tuning of the ARIS to adapt the data quality analysis to the changing data patterns including seasonality changes.

In order to identify the ML model parameters (*i.e.*, specifically C and γ values), and number of time delays that yield the most accurate and efficient model, a step-wise search method was used. The step-wise method works by running regressions using values of different orders of magnitude for a specific parameter, calculating the R^2 value to assess accuracy, then evaluating on finer scales until the appropriate value is established. The same method was used for variable selection of C, γ and time delay values, where the test file incorporating real values as classifiers in order to compare the model's accuracy at predicting for those values.

Based on the results of the R^2 statistical tests, the best combination of variables for a February regression would be to use one year of energy data. For May, the best combination of variables would be two years of energy and temperature. While these statistical tests proved the accuracy of these models, two years of energy, temperature and humidity were used for all regressions.

It was important to include those variables in the creation of the model. A model using fewer variables produces smooth, highly cyclical curves, while the addition of more variables creates curves with more noise and statistically poorer fits. However, the inclusion of more variables allows the model to adapt more dynamically to changes in weather that occur within a single day or week, and it aids the model in predicting minimal and maximal energy

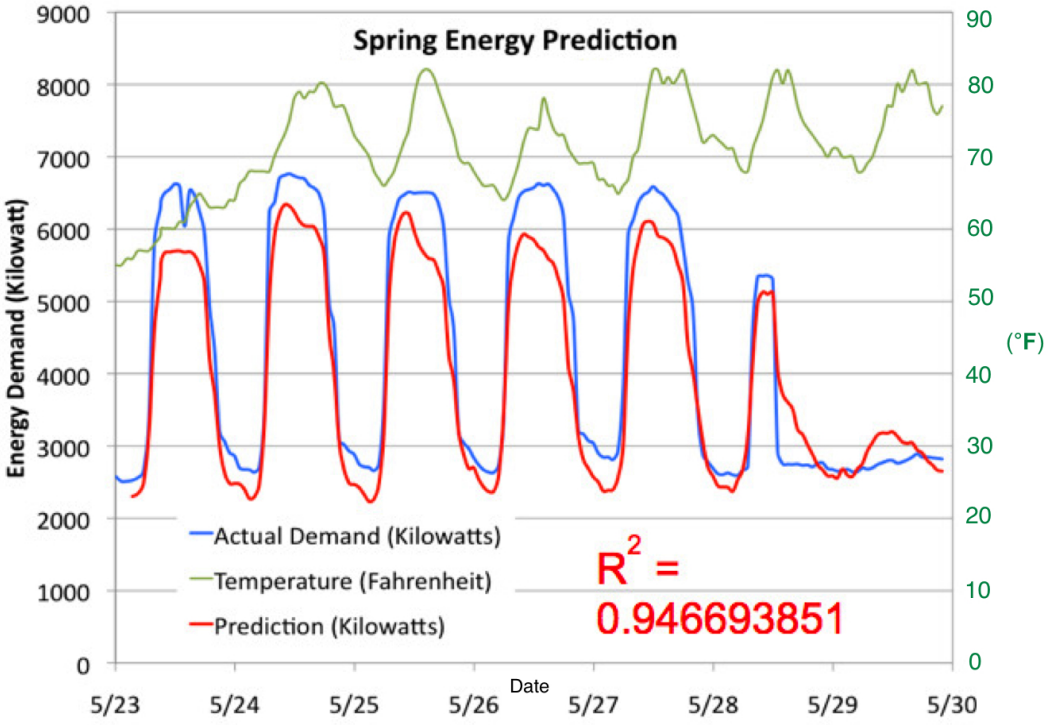


Figure 3.41: Predicted versus actual energy demand in May 2011 [209].

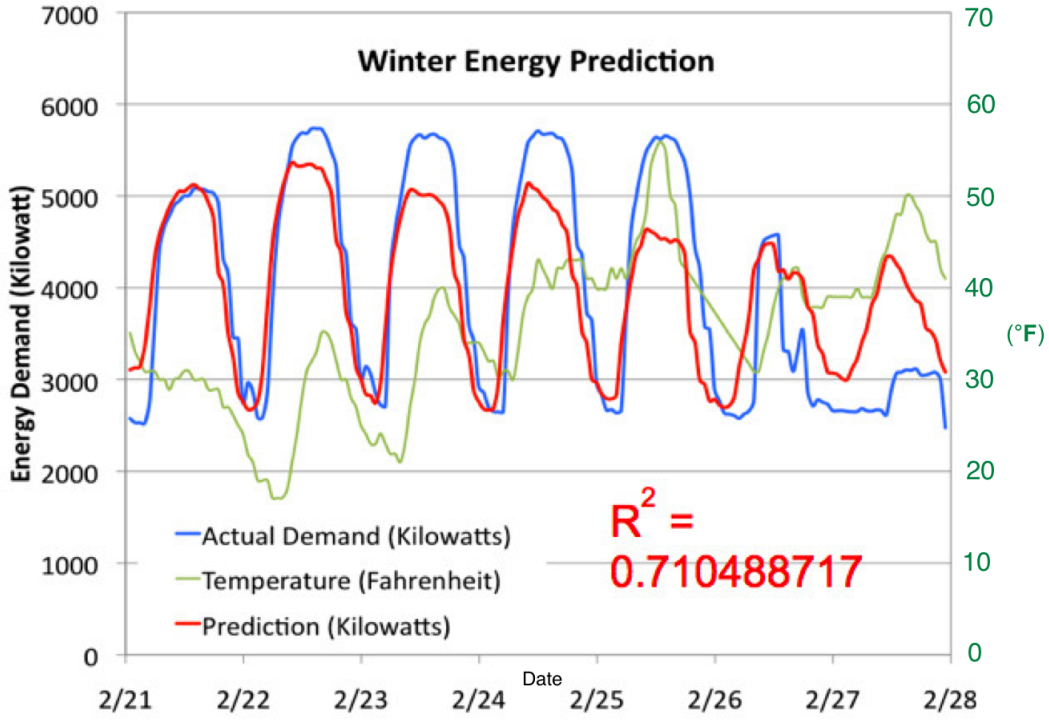


Figure 3.42: Predicted versus actual energy demand in Feb 2011 [209].

demand values.

Figures 3.41 and 3.42 show regression results of SVR prediction versus actual energy demand for two different five-month datasets at different times of the year. The spring graph is closer to the actual energy consumption of the building, with an R^2 value of about 0.95, while the winter graph is less accurate, with an R^2 of about 0.71. The likely reason for the less accurate winter regression is that the SVR predictive model may need additional features in its dataset in order to better handle low winter temperature values, which cause increased energy demand for heating.

In summary, the experiments showed that the ARIS is effective in ensuring that the smart building ML system continues to run reliably and the self-tuning component can adapt the ARIS to the changing data patterns such as seasonality changes. The system runs independently from the smart building ML system. It runs efficiently and does not add much overhead to the smart building ML system.

3.6 Summary

This chapter presents automated online evaluation AOE that performs data quality analysis using computational intelligence and self-tuning techniques to improve system reliability for cyber-physical systems that process large amounts of data, employ software as a system component, run online continuously and maintain an operator-in-the-loop. My experiments with the ARIS system, a prototype architecture and implementation of AOE, in smart power grid CPS and smart building CPS have demonstrated that this approach is effective and efficient. The data-dependence of this system makes it easily applicable to different types of cyber-physical systems, and the open expandable architecture also enables the incorporation of new data quality analysis and self-tuning techniques.

Chapter 4

Cloud-Based Reliability Assurance Framework for CPS

One limitation of the ARIS architecture and implementation described in sections 3.3 and 3.4 is that it is not scalable or economical for dealing with certain types of CPS where large volumes of data need to be processed in parallel within a short period of time. In order to conduct efficient and cost-effective automated online evaluation for data-intensive CPS, I will describe a cloud-based reliability assurance framework called COBRA in this chapter. Using the language similar to the definition of quality assurance [78], *reliability assurance* is defined as the planned and systematic activities implemented in a system so that reliability requirements for a product or service are fulfilled.

In the following section, I will describe an overview with some background information on data-intensive computing and cloud computing. In section 4.2, I will describe the COBRA framework, followed by architecture in section 4.3. In section 4.4, I will describe implementation and some applicable cloud computing environments. Then in section 4.5, I will describe the empirical studies through controlled experiments and an application of the framework on a smart building CPS before my conclusion in section 4.6.

4.1 Overview

For the purpose of this thesis, data-intensive computing is a class of computing applications which often use a data parallel approach to processing large volumes of data, typically terabytes or petabytes in size and commonly referred to as Big Data [107]. Computing applications which devote most of their execution time to computational requirements are deemed compute-intensive and typically require small volumes of data, whereas computing applications which require large volumes of data and devote most of their processing time to I/O and manipulation of data are deemed data-intensive [155]. The challenge of data-intensive computing is to provide the hardware architectures and related software systems and techniques which are capable of transforming ultra-large data into valuable knowledge. Data-intensive applications are well suited for large-scale parallelism over the data and also require an extremely high degree of fault-tolerance, reliability, and availability [82].

Many CPS are data-intensive, requiring large volumes of data and devoting much of their processing time to I/O and data manipulation [155]. BMS and smart grid control systems are examples of typical data-intensive CPS; a BMS processes large amounts of data streams captured by sensors installed throughout the building, while smart grids process many continuous data sources from electrical and computational components.

Because of their data-intensive characteristics, complexity and the unpredictable running environment, it is often hard to estimate and improve the reliability of a data-intensive CPS prior to deployment. During the actual use phase, runtime evaluation can be used. This works in parallel with the CPS, continuously conducting automated online evaluation at multiple stages along the system workflow and providing operator-in-the-loop feedback for reliability improvement [240]. But reliability assurance using only local computing resources is often impossible, unscalable or too expensive for data-intensive CPS.

Thus, a cloud-based approach might be a good solution. According to the definitions by the NIST [151], cloud computing is a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources (*e.g.*, networks,

servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction.

This chapter presents a cloud-based reliability assurance framework called COBRA, which stands for ClOud-Based Reliability Assurance. COBRA employs cloud-based runtime evaluation to conduct reliability assurance for data-intensive CPS. It mitigates the limitations imposed by the large amounts of data transfer required for cloud-based processing using data serialization and messaging systems. Its data quality analysis processes use scheduler and elastic load-handling on demand to achieve a degree of scalability, responsiveness and cost-effectiveness that is not possible with traditional approaches such as local server clustering. COBRA makes use of self-tuning to manage and configure the evaluation system to ensure that it adapts itself to changes in the system and exogenous conditions whilst hiding intrinsic complexity from operators and users. Furthermore, a set of performance metrics is used to evaluate the performance of COBRA.

I have developed a prototype COBRA system, which I implemented and used in real-world experiments with a BMS in a New York City building. The evaluation results showed that it is effective, efficient, scalable and easy to implement. COBRA can also be tested in a simulated environment using historic data or induced reliability issues, including failures caused by fault-injection, human error or abnormal environmental variables.

4.2 COBRA Framework

COBRA framework employs cloud-based runtime evaluation with a feedback loop to achieve reliability assurance for data-intensive CPS. It consists of data transportation between a CPS data source and a cloud data store with load balancing and failover switch, parallel data quality analysis with self-tuning, autonomic process management with elastic scaling, performance and reliability metrics. Performance metrics will be described in section 4.5.1. A set of reliability metrics has been described in section 3.5.1.

4.2.1 Data Transportation

One challenge associated with moving data-intensive computation to the cloud is relocating large amounts of data efficiently and in a timely fashion from the local environment to the cloud computing environment [18, 19]. Figure 4.1 illustrates two methods to achieve this. A typical dataset in the applications of interest consists of a large amount of continuous time series data records in $\{id, timestamp, value\}$ tuple format.

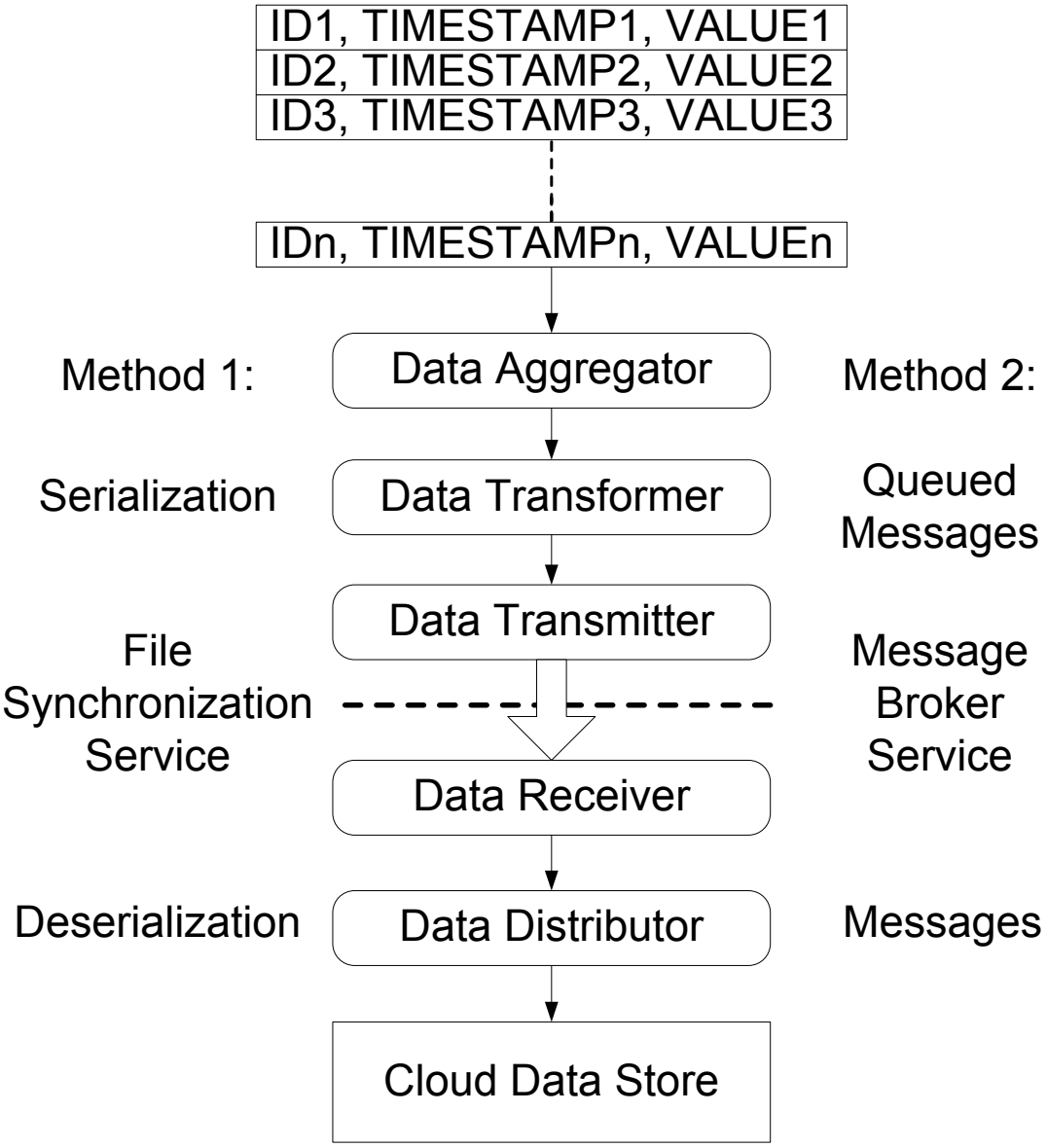


Figure 4.1: Data transportation.

The first method involves serialization during the data transformation stage after time-stamped data has been aggregated. Then, a file synchronization service such as *rsync* [194] can transfer the serialized files to the cloud server. Some file synchronization services are capable of encrypting and compressing the data being transferred, which provides additional security and efficiency. After the files have been transferred, they will be transformed back to data records through deserialization. These data records are then saved to the cloud data store for further processing. This method is suitable for transferring large data records, including binary data such as medical images.

The second method utilizes a message broker service installed at the data transmitter and receiver ends to transfer queued messages generated by the data transformer. Some commonly used message broker service software includes IBM Websphere MQ [102], Apache ActiveMQ [15] and RabbitMQ [227]. The received messages are split and transformed back to the time series data record for storage in the cloud data store. The data store can be written sequentially on a disk for processing. This step can be managed by systems such as Apache Hadoop [16]. This method is suitable for transferring data records with textual or numeric values.

Load Balancing and Failover Switch

At the data transportation stage, self-tuning is used to achieve dynamic load balancing and failover switch, which applies to the parallel processing of large amounts of time series data coming from different sources, in order to maximize efficiency and reliability. Dynamic load balancing automatically distributes the data traffic load across multiple active server instances. When a sudden burst of data traffic causes bandwidth usage to exceed a predefined threshold, additional cloud computing resources will be requested and allocated for handling the heavier load. In the case of some instances failing to respond, the data traffic will be rerouted to healthy instances by the failover switch.

4.2.2 Parallel Data Quality Analysis

After data relocation, data quality analysis is used to process the cloud data store for reliability assurance. Because of the data-intensive and continual processing requirements, the analysis algorithm needs to be lightweight and responsive, and should not require large amounts of data storage in each time window. Furthermore, the processing algorithm should be able to be launched and terminated by the process scheduler.

COBRA uses computational intelligence to perform data quality analysis in an automated and efficient way, thereby ensuring that the running system performs as reliably as possible. This computational intelligence is enabled by machine learning, data mining, statistical and probabilistic analysis and other intelligent techniques. In a CPS, data collected from the system (*e.g.*, sensor data-points, software bug reports, system status logs and error reports) are stored in databases. COBRA analyzes this data so that useful information on system reliability, such as erroneous data or abnormal system states, can be obtained. This reliability-related information is in turn directed to system operators so that proper actions can be taken–in some cases, proactively based on predictive results–to ensure proper and reliable execution of the system. The following sections describe some data quality analysis techniques used by COBRA.

Data Anomaly Detection and Diagnosis

Data anomaly detection and diagnosis used by COBRA are similar to the techniques described in the section 3.2.3.

Self-Tuning

Self-tuning is used to automatically evaluate performance and adjust algorithm parameters for data quality analysis in order to improve the accuracy, efficiency and robustness of analysis. It also minimizes the burden imposed on the administrator. Some examples of self-tuning uses are as follows:

- Use metrics such as R^2 (coefficient of determination), ROC (receiver operating characteristic) and AUC (area under the curve) to measure and improve accuracy of data analysis models. [242]

- Use statistical trend detection and curve fitting, such as Weibull distribution and parameters estimation [246, 190], to reduce variability and eliminate overshoot.

- Adjust system parameters such as set-points, thresholds and machine learning model parameters when abnormal exogenous situations happen in order to reduce false alarms.

4.2.3 Autonomic Process Management

To manage large numbers of parallel data analysis processes, COBRA employs autonomic process management for launching processes based on the workload and scaling the process pool dynamically to ensure efficient processing. Process schedulers elastically launch appropriate numbers of processes for the data feeds and recycle processes that are no longer needed.

Process Scheduling

Similar to the Process Control Block (PCB) [58] or Process Descriptor in an operating system, each process in the COBRA system is represented by a data structure. It contains basic information about the process including:

- Process identification, which is the unique identifier assigned by the process scheduler.

- Process status, which indicates the current status of the process: READY, RUNNING, BLOCKED or SUSPENDED.

- Process state, which contains information about what kind of resources the process has used and for how long. These include information about the specific time series

dataset being processed.

Process creation is illustrated in Figure 4.2. The process scheduler first checks the cloud data store to see if there is any unhandled dataset, which is a new time series data stream identified by a unique dataset identifier. The process scheduler then checks if any idle processes can be reassigned to this new dataset. If none are available, the scheduler launches a new process and updates the process's state with relevant information including the dataset being assigned. This dynamic scheduling system makes decisions on the fly in order to make better use of resources.

Process termination can be triggered by various causes. Because COBRA is designed for continual runtime reliability evaluation, the process does not have a predetermined execution time. As long as the dataset associated with a process is coming in, that process continues. Process termination typically happens due to an error or fault condition. This includes lack of system resources such as memory or I/O failure due to an unexpected situation. To recover a failed process, the process scheduler recreates a new process through the steps described above.

Elastic Scaling

An important feature of autonomic process management is elastic scaling that automatically increases and decreases computing resources and running processes to maintain optimal system performance. *Scalability* is defined as the ability of a system, network or process to handle a growing amount of work in a capable manner, or its ability to be enlarged to accommodate such growth [27, 98]. Elastic scaling modulates the use of computer resources dynamically to meet a variable workload [179].

Elastic scaling in COBRA is enabled by a two-step self-tuning process. The first step is to scale up, or scale vertically. At this stage, the process scheduler allocates more resources, such as CPU, memory and storage, to handle increasing numbers of parallel processes. When the maximum number of processes in a computing instance, which can be a virtual

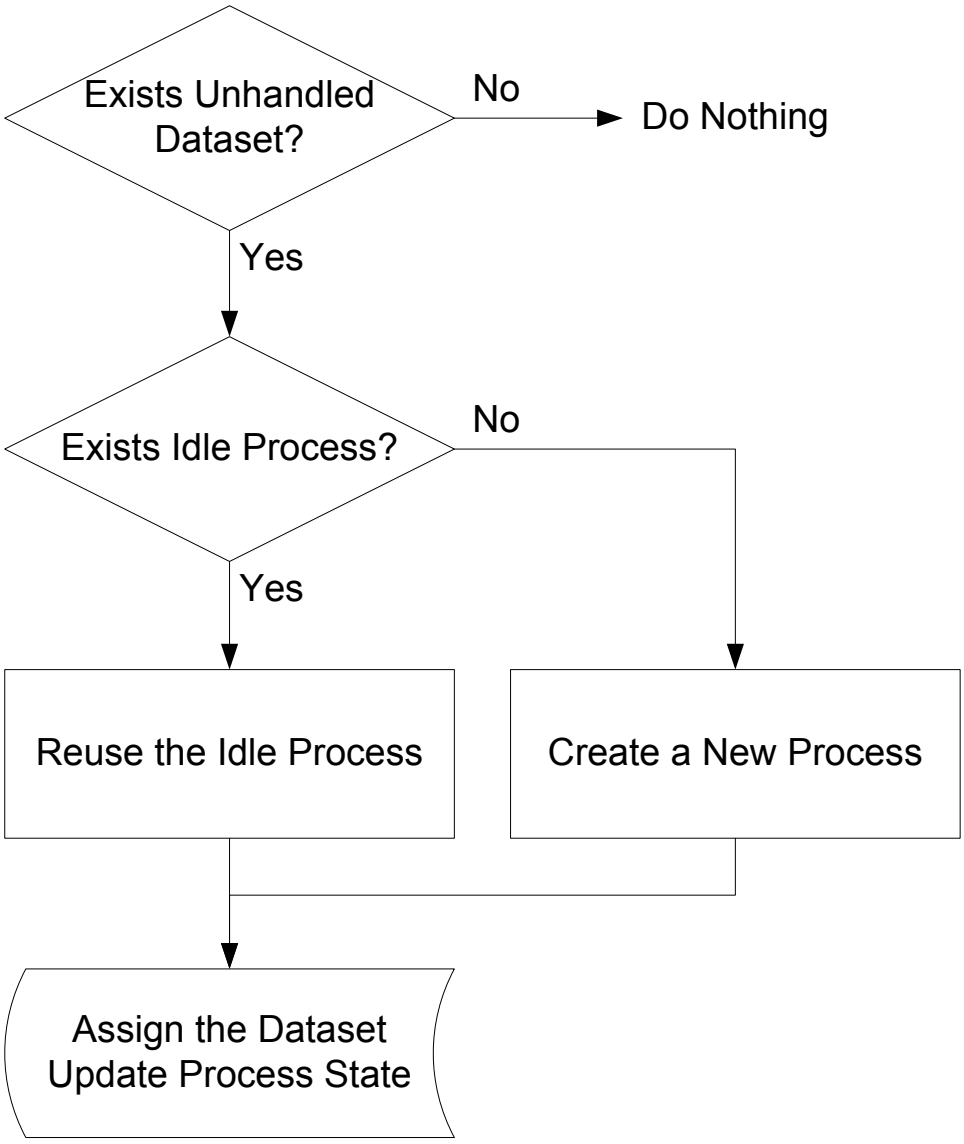


Figure 4.2: Process creation.

machine or cluster running in the cloud computing environment, is reached, the process scheduler sends a request to cloud computing infrastructure such as Amazon Cloud Service [7] and OpenStack [171] for additional system resources. This step is also called scaling up, or scaling horizontally.

4.3 Architecture

As illustrated in Figure 4.3, COBRA's architecture includes the following components: data aggregator and transformer, data transmission, data distributor, cloud data store, parallel processors, master scheduler, self-tuner, user interface and alerts /triggers, performance and reliability metrics.

Step 1: The data collected from the CPS, including sensor measurements, software logs, system alarms and environmental state information, are transferred from the local environment to the cloud computing servers through a multiple-stage data transportation process, as described in section 4.2.1.

Step 2: The process scheduler dynamically launches and manages data quality analysis processes, which conduct continuous processing of the cloud data store. Self-tuning is used to adjust system resources such as the size of the process pool and settings such as the parameters of the data processing algorithm in order to achieve higher efficiency and accuracy. A set of performance metrics, which will be described in section 4.5.1, is measured on the fly.

Step 3: The results of these parallel processes are communicated back to the human operator or an automatic actuator, which can take corrective and preventive actions on the CPS to ensure system reliability. As a separate comparison, a set of reliability metrics as described in section 3.5.1 is measured directly on the CPS on the fly.

4.4 Implementation

I have implemented a prototype COBRA system using Java programming language. The architecture is flexible and easy to implement. It can be implemented using different programming languages, such as Java and Python, and supports various cloud computing environments.

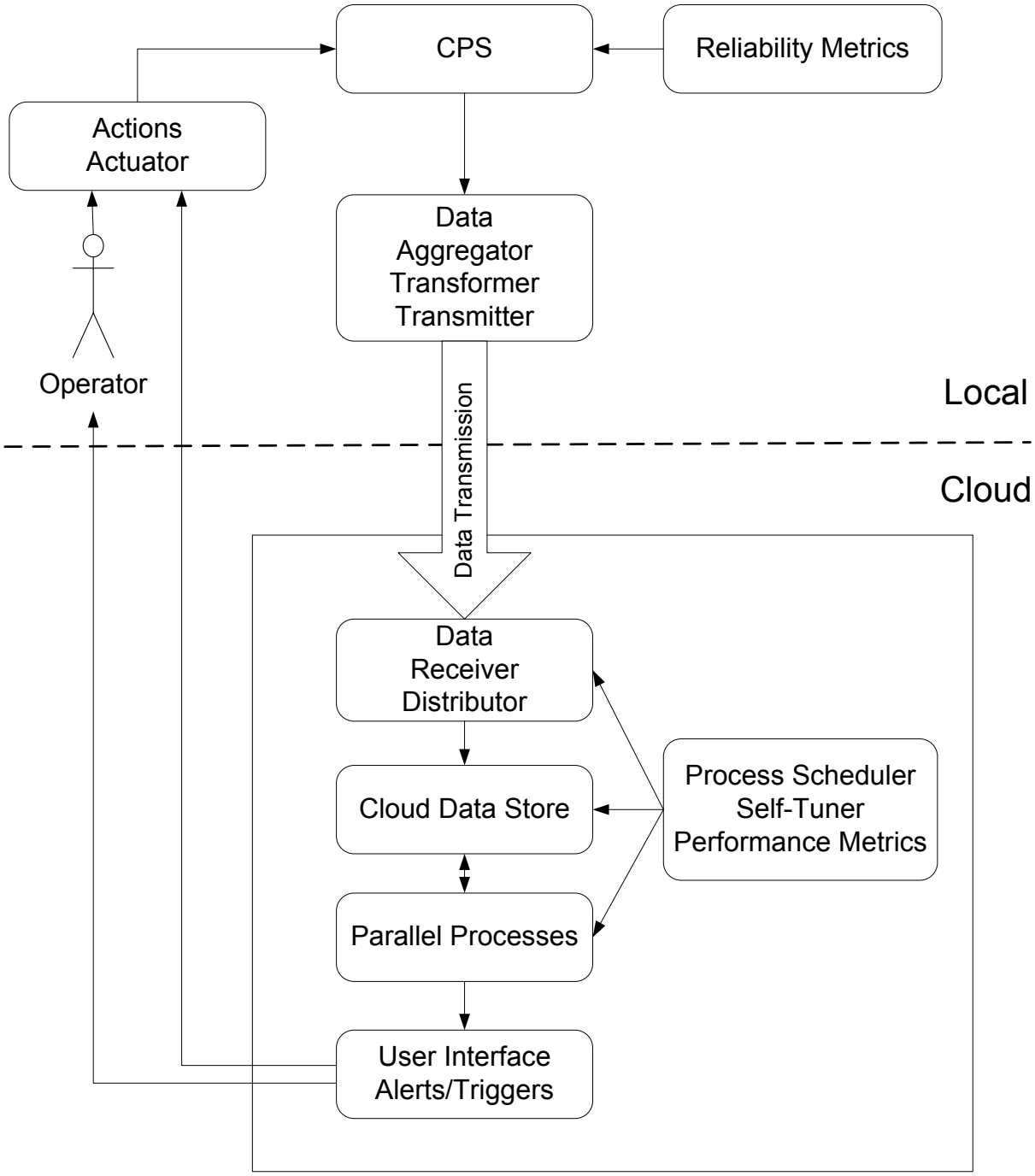


Figure 4.3: COBRA system architecture.

4.4.1 Software Design

As shown in Figure 4.4, the software consists of a data receiver and distributor, a data quality analysis process, a process scheduler, a self-tuning and performance metrics module with a

cloud data store, several feedback mechanisms (including alert emails, warning messages and reports) and a user interface with real-time visualization.

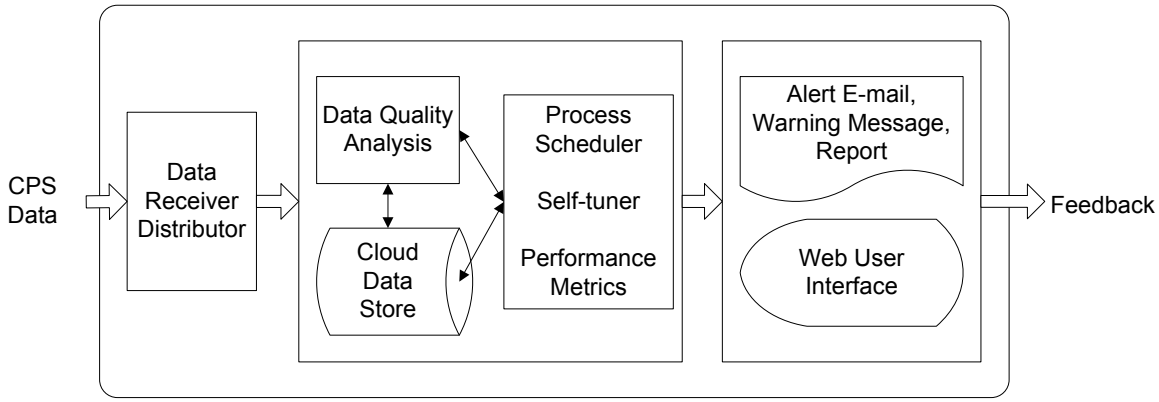


Figure 4.4: COBRA software components and data flow.

For handling real-time time series data, I used a message broker service for communicating queued messages from the local environment to the cloud environment, which is the second data transportation method illustrated in Figure 4.1, In my implementation, I installed a message broker server at both the data transmitter and data receiver ends.

As shown in Figure 4.5, the web interface provides visualization for users. Alert emails, warning messages and reports are generated automatically to provide CPS operators with reliability-related information with low latency.

4.4.2 Cloud Computing Environments

In order to be flexible and applicable for different implementations, COBRA does not depend on any specific cloud computing environment. The following compares some applicable cloud computing services available for COBRA implementation. Amazon Elastic Compute Cloud (EC2) and private cloud are the environments tested in my experiments.

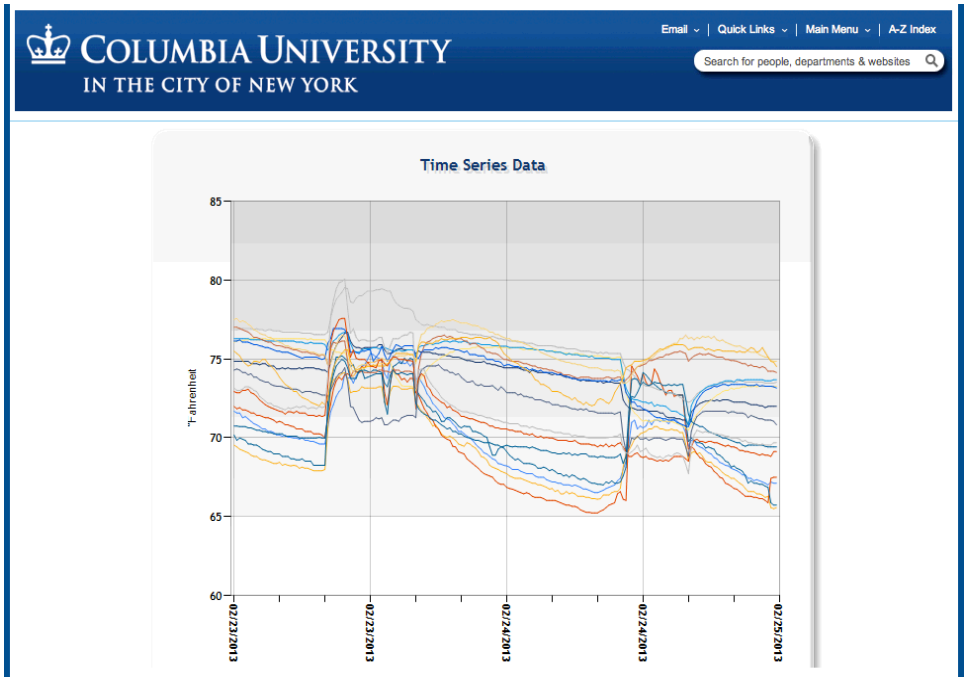


Figure 4.5: User interface example of many different floor temperatures over two days and three nights in an office building.

Amazon Web Services

Amazon Web Services [7] is a full-service cloud computing environment, including cloud-based computing, network, content delivery, storage, database, deployment, management and application services. The services that are useful for COBRA implementation include Amazon Elastic Compute Cloud (EC2), which provides scalable virtual private servers and can be the cloud-based server for COBRA implementation; Amazon Simple Storage Service (S3), which provides Amazon Web Services-based storage; Amazon Simple Queue Service (SQS), which provides a hosted message queue for web applications; Amazon CloudWatch, which provides monitoring for cloud resources and applications and can be used for implementing COBRA's elastic scaling; and Amazon Simple Notification Service (SNS), which provides hosted multiprotocol "push" messaging for applications and can be used for user notification.

OpenStack-Based Services

OpenStack [171] is an open-source infrastructure as a service (IaaS) cloud computing initiative jointly launched by Rackspace [187] and NASA in 2010. The goal of the OpenStack project is to enable any organization to create and run cloud computing services on standard hardware. The cloud-based infrastructure software was developed on the Linux operating system. As illustrated in Figure 4.6, it includes computing, networking, storage, queue, scheduler, hypervisor, dashboard, authentication and image services. The computing, storage and dashboard services can be used directly for implementing COBRA because OpenStack's interfaces are open and standardized.

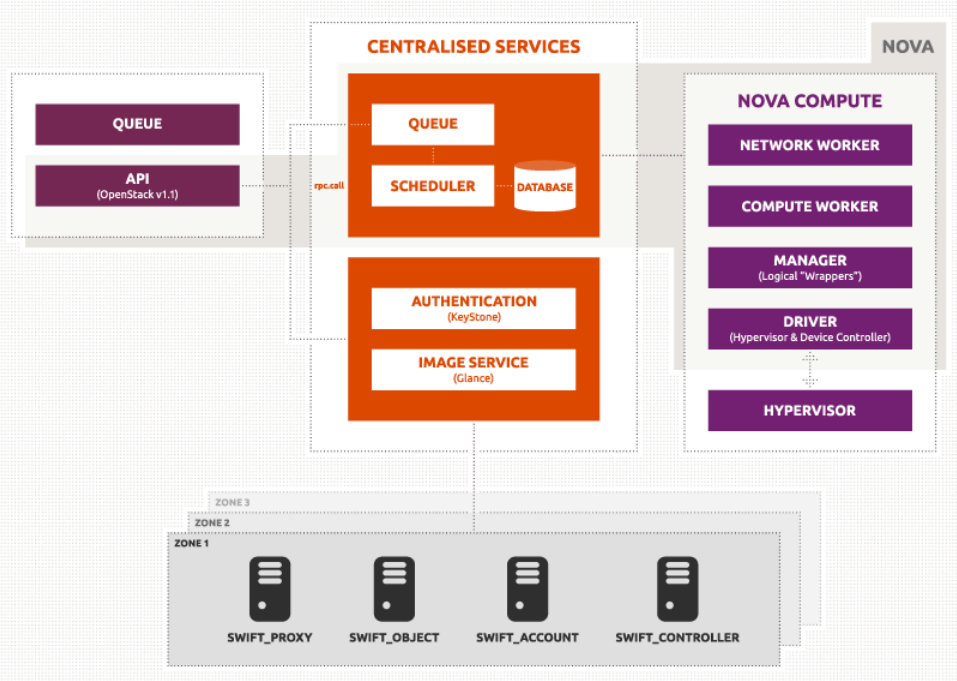


Figure 4.6: OpenStack logical architecture [171].

Windows Azure

Windows Azure [154] is a platform as a service (PaaS) and infrastructure as a service (IaaS) cloud computing environment created by Microsoft for building, deploying and manag-

ing applications and services through a global network of Microsoft-managed datacenters. Services include web, virtual machine, data management, business analytics, identity, messaging, media and mobile services. The virtual machine, data management and messaging services are applicable for implementing COBRA.

Google App Engine

Google App Engine [89] is a platform as a service (PaaS) cloud computing environment for developing and hosting web applications in Google-managed data centers. Applications are sandboxed and run across multiple servers. App Engine offers automatic scaling for web applications as the number of requests increases. Compared to Amazon and OpenStack-based cloud services, Google App Engine is less flexible and has fewer features provided for customized development, but it is still possible to implement COBRA on Google App Engine.

Private Cloud

A private cloud [235] is a cloud computing platform that is implemented within the corporate firewall, under the control of the IT department. A private cloud is designed to offer the same features and benefits of public cloud systems, but removes a number of objections to the cloud computing model, including control over enterprise and customer data, worries about security, and issues connected to regulatory compliance. The implementation of COBRA depends on the specific cloud services enabled by the private cloud environment.

4.5 Empirical Studies

In this section, I will describe evaluation methodology and some empirical studies, in which I applied the prototype COBRA system to some data-intensive CPS to process parallel data continuously for reliability assurance.

4.5.1 Evaluation Methodology

The reliability metrics described in section 3.5.1 and the following performance metrics are used to evaluate the COBRA system quantitatively.

Performance Metrics

Performance metrics are used to evaluate different aspects of the COBRA system including data transportation, data quality analysis, and autonomic process management.

The following are some *scalability metrics* for throughput and resource usage:

- Throughput (Inbound/Outbound) in kbits/second, a measurement of bandwidth utilization.
- Traffic (Inbound/Outbound) in megabytes, a measurement of total data size.
- Process creation rate, which is the number of processes created within a time unit.
- Maximum number of processes before scale-out, which is the maximum allowable number of processes in the computing instance before requesting additional resources.
- CPU % utilization by process, an indicator of CPU use by a process.
- Memory % utilization by process, an indicator of memory use by a process.
- Data store % utilization by process, an indicator of data store use by a process.

The following are some *responsiveness metrics*:

- Average processing time, which gives an estimate of time needed for the algorithm to process a dataset of specific size for a predefined time period.
- Average processing time using self-tuning in adjusting algorithm parameters, which shows the performance improvement when self-tuning is applied.

- Connection duration (TCP Client/Server) in seconds, a data transfer performance metric related to latency.

The following are some other metrics:

- False positive ratio, also known as the false alarm ratio, which refers to the probability of falsely rejecting the null hypothesis (*i.e.*, a general or default position or state) for a particular test.
- Active connections, which is the total number of live connections.
- Allocated server processes, which is the number of allocated processes in the computing instance.
- Number of active processes, which is the total number of processes running.
- Connection requests (TCP Client/Server), the total number of the connection requests.

4.5.2 Real-World Experiments

I evaluated the prototype COBRA system using a smart building BMS system at a large commercial building in New York City. A BMS is a typical data-intensive CPS consisting of both software and hardware components that control and monitor a building's mechanical and electrical equipment, such as ventilation, lighting, power systems, fire systems and security systems. The building energy control system is an important component of the BMS that reads data feeds representing internal and exogenous conditions (*e.g.*, temperature, humidity, electrical load, peak load, fluctuating electricity pricing and building work schedule) and takes control actions (*e.g.*, adjusting lighting, turning on/off the air-conditioning and shutting off partial elevators) accordingly. Building operators usually have the ability to change or override control actions taken by the BMS to accommodate special situations such as severe weather or changes in the building's work schedule. I used a BMS at a large office building in New York City for this study.

4.5.3 Experiments on Smart Building BMS System

The experimental setup is illustrated in Figure 4.7. Data collected from the building sensors and BMS software are aggregated, transformed and transferred to servers in the cloud, where these data are processed on the fly. The results are being sent back to the operator, who can take actions to ensure reliable system operation.

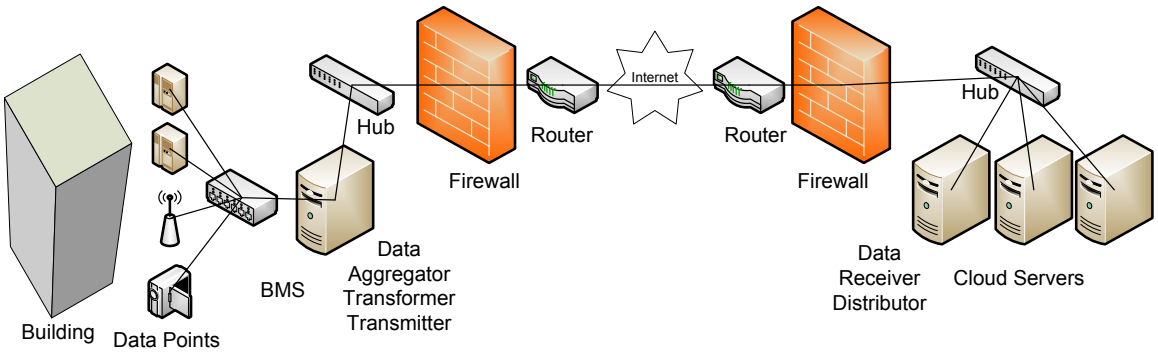


Figure 4.7: Experimental setup.

The following are some experimental results.

Data Transportation

There are a total of 2,600 live time series datasets with each dataset represents a single data source in the BMS such as a sensor's measurement, software log and other system information.

Table 4.1. shows the total data size (inbound/outbound) during each "push" update for different data sampling frequencies. The results show that as the system gets more close to real-time data collection, the volume of data traffic increases significantly. This is an example of the challenges in moving the data-intensive processing from local to cloud environment.

Table 4.2 lists the approximate time that is required for data transportation of 100 time series datasets, each with a typical size, during each data "push". It shows that even with a

Sampling Frequency	Traffic (in megabytes)
Every 15 minutes	10.9
Every 5 minutes	32.8
Every 60 seconds	164.1
Every 5 seconds	1,969.6

Table 4.1: Traffic (inbound/outbound).

high speed connection, the data relocation takes significant time if the number of datasets increases.

Number of datasets	100
750 Kbps	1,450
1.5 Mbps	730
3.0 Mbps	360
10.0 Mbps	100

Table 4.2: Connection duration (in seconds).

Data Quality Analysis

Data quality analysis processing time varies according to the type of datasets and the state of the data inside. Table 4.3 shows some examples of results of average processing time for HVAC (heating, ventilation, and air conditioning), electric meter and BMS software log datasets, along with their false positive ratios. The more uniform the real-time data is, the less time the data analysis processing takes and the more accurate it tends to be.

	HVAC	Meter	Log
Processing time (in seconds)	60	55	120
False positive ratio	2%	5%	8%

Table 4.3: Average processing time and false positive ratio.

Process Management

Table 4.4 lists some examples of results for different performance metrics related to the process management. In my experiments, the memory and data store utilization are often higher than the CPU % of processor time. It indicates that the dataset size and locality would more easily lead to performance bottleneck than would the processing power of the cloud computing instance.

Allocated server processes	100
Number of active processes	35
Maximum number of processes	250
CPU utilization	35%
Memory utilization	60%
Data store utilization	80%

Table 4.4: Maximum number of processes.

Table 4.5 and Figure 4.8 show that the increase in the number of processes does not lead to an exponential increase in the processing time. The average processing time is always below an upper bound t_0 at 70.

# of processes	1	5	10	20	50	80
Processing time (in seconds)	45	48	55	60	64	65

Table 4.5: Processing time versus number of processes.

Figure 4.9 shows a snapshot of an example of system resource usages.

Effectiveness

Table 4.6 lists some reliability issues related to HVAC, BMS software, environmental issues and detector errors that were identified and verified before corrective actions taken by the operators or autonomous actuators.

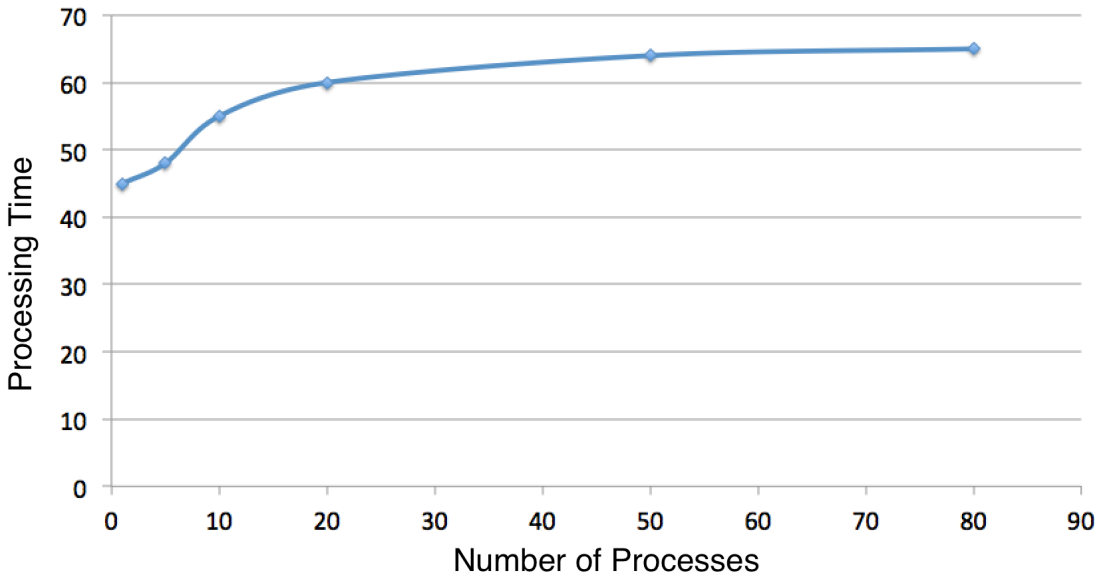


Figure 4.8: Processing time versus number of processes.

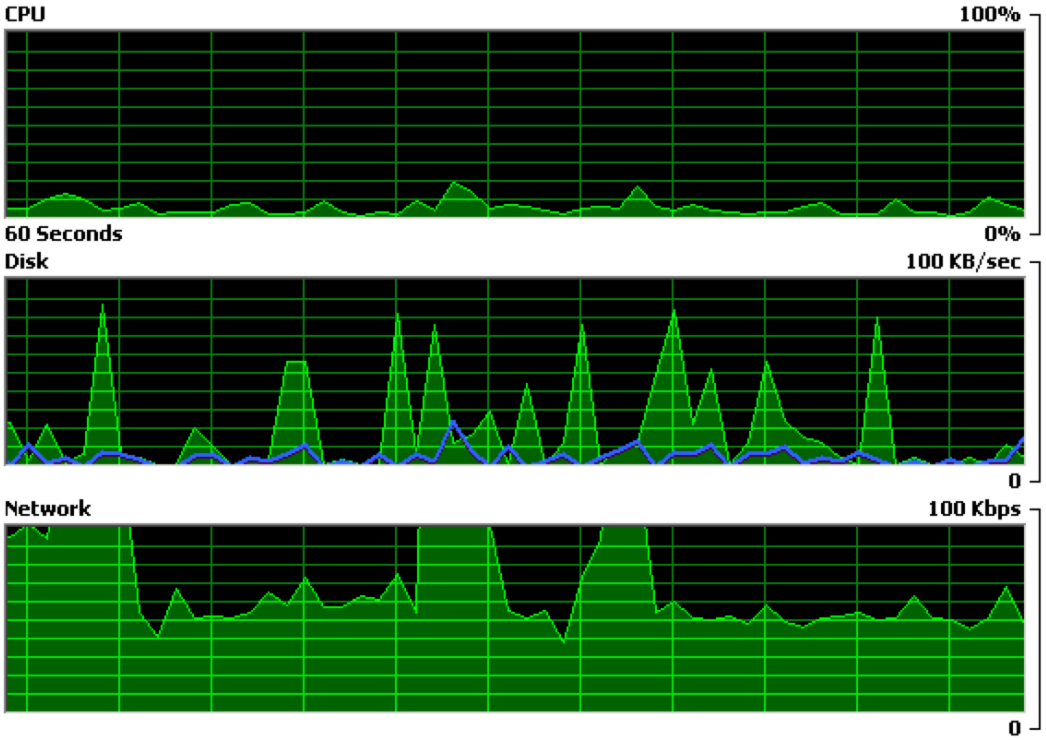


Figure 4.9: Example system resource usages.

Validity

I estimated reliability metrics such as MTBF metrics to measure the improvement or deterioration of system reliability. After failure incidence time series data is collected, I

Issue Type	HVAC	BMS	ENV	Sensor
# of reliability issues	33	25	5	14

Table 4.6: Number of reliability issues identified.

estimated reliability metrics as described in section 3.5.1. The weekly failure rates over six months are charted in Figure 4.10. Also, the linear regression $y = -0.0229x + 1.1169, R^2 = 0.03829$ shows the improved results over time. In this case, $\theta = -0.0229$ using failure rate as the reliability measurement.

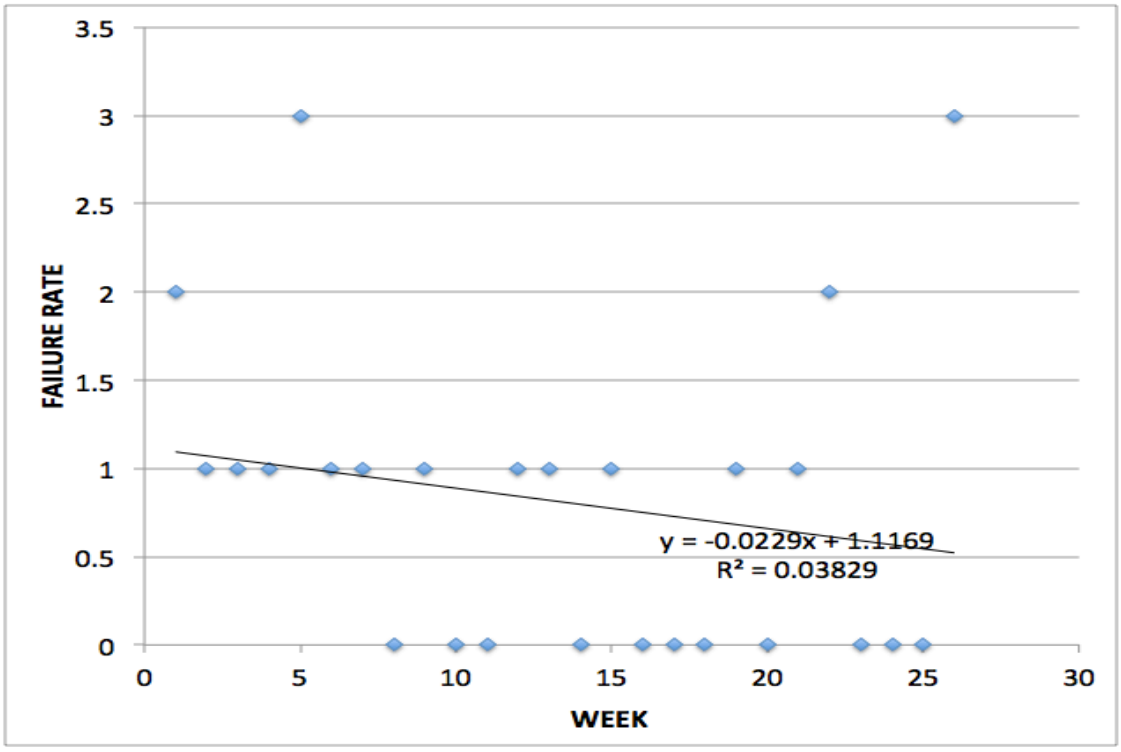


Figure 4.10: Weekly failure rate.

Although it is reasonable to argue that improved CPS reliability may be due to other factors in the complex system, such as better maintenance or friendlier running environment, the number of reliability issues identified and subsequently fixed are strong evidence of the validity of the COBRA framework.

Summary

In summary, the experiments show the approach is effective and efficient. It is also scalable to process data-intensive CPS for reliability improvement. Furthermore, the approach can be implemented relatively easily using various cloud computing services readily available.

4.5.4 Limitations

The prototype implementation uses a disjoint process model; *i.e.*, I assume that processes can be run independently of each other and that the order in which processes are executed is not important. Hence, it treats every streaming data series from each source (*e.g.*, sensor data point) separately. It does not take into consideration correlation or association between separate data sources. In many cases, reliability issues might have causal or clustering effects. For example, a failure in one software component might also cause other adjacent or dependent components to fail. These inter-data-series correlations need to be handled in future work.

4.6 Summary

This chapter presents COBRA, a cloud-based reliability assurance framework for data-intensive CPS. COBRA provides automated multi-stage runtime reliability evaluation along the CPS workflow using data relocation services, a cloud data store, data quality analysis and process scheduling with self-tuning to achieve scalability, elasticity and efficiency. A set of performance metrics is used to evaluate the system performance on the fly. A prototype of COBRA has been implemented and experimented with on the BMS of a large office building in New York City. The experiments show that it is effective, efficient, scalable and easy to implement.

Chapter 5

Reliability Benchmark Framework for CPS

Improving CPS reliability requires an objective measurement, estimation and comparison of the CPS system reliability. Previously in section 3.5.1, I described some reliability metrics used in the experiments. In order to provide a generic way to compare and benchmark system reliability for CPS and to extend the approach described in the prior sections, I will further describe a generic reliability benchmark framework called FARE (Failure Analysis and Reliability Estimation) for CPS in this chapter.

In the following section, I will give an overview. In section 5.2, I will describe the motivation for this framework. In section 5.3, I will describe the FARE framework consisting of a CPS reliability model, selection of testing environment, failure analysis and reliability estimation. For empirical evaluation, I will present my implementation in section 5.4, experiments and results in section 5.5. Finally, I will conclude in section 5.6.

5.1 Overview

Prior researches have proposed reliability benchmarks for some specific CPS such as wind power plants and wireless sensor networks. There were also some prior researches on the

components of CPS including software and some specific hardware. But there isn't any reliability benchmark framework for CPS in general, according to the best of my knowledge. This chapter describes FARE (Failure Analysis and Reliability Estimation), a framework for benchmarking the reliability of cyber-physical systems. The FARE framework provides a set of methods and metrics on failure analysis, data quality measurement and monitoring, operational availability measurement and reliability estimation for benchmarking CPS reliability.

The advantages of the FARE framework include a more general and accurate representation of the CPS reliability; additional reliability metrics; CPS-specific holistic system reliability; emphasis of actual use and continual evaluation. The framework is extensible for accommodating new reliability measurement techniques and metrics. It not only provides a retrospect evaluation and estimation of the CPS system reliability using past data, but also provides a mechanism for continuous monitoring and evaluation of CPS reliability for runtime enhancement.

For empirical study, I implemented the FARE framework as a software application and applied it on a smart building management system for a large commercial building in New York City. My experiments showed that FARE is easy to implement, accurate for comparison and can be used for building useful industry benchmarks and standards after accumulating enough data.

5.2 Motivation

5.2.1 Reliability Benchmarking

Some prior work has been done on benchmarking system dependability for computer systems and software suites. In their book [120], Kanoun and Spainhower collected some dependability benchmarks for computer systems developed by industry and academia and explained the various principles and concepts of dependability benchmarking. The

DBench (Dependability Benchmarking) project by the European community's Information Society Technologies developed a conceptual framework and an experimental environment for benchmarking the dependability of COTS (commercial off-the-shelf) components and COTS-based systems [106].

TPC defined the TPC-C specifications and metrics for benchmarking transaction processing and database performance including integrity through simulating a complete computing environment where a population of users executes transactions against a database [219]. Cooper *et al.* presented the Yahoo! cloud serving benchmark (YCSB) framework for comparison of the performance, scalability and availability of cloud data serving systems [49]. These approaches do not provide a general means for benchmarking reliability of cyber-physical systems.

5.2.2 Component Reliability versus System Reliability

Through a study of flight computing architectures and related avionics components for launch vehicles for NASA future missions, Chen *et al.* stated [43] the limitations of using component reliability to represent system reliability in their paper. Their conclusion was that component reliability analysis and system reliability analysis need to be evaluated at the same time, and that the limitations of each analysis and the relationship between the two need to be fully understood to ensure mission success.

5.3 FARE Framework

The FARE framework consists of a CPS reliability model, the selection of evaluation environment using a decision tree, failure analysis including failure detection and diagnostics, and reliability estimation using metrics and statistical modeling.

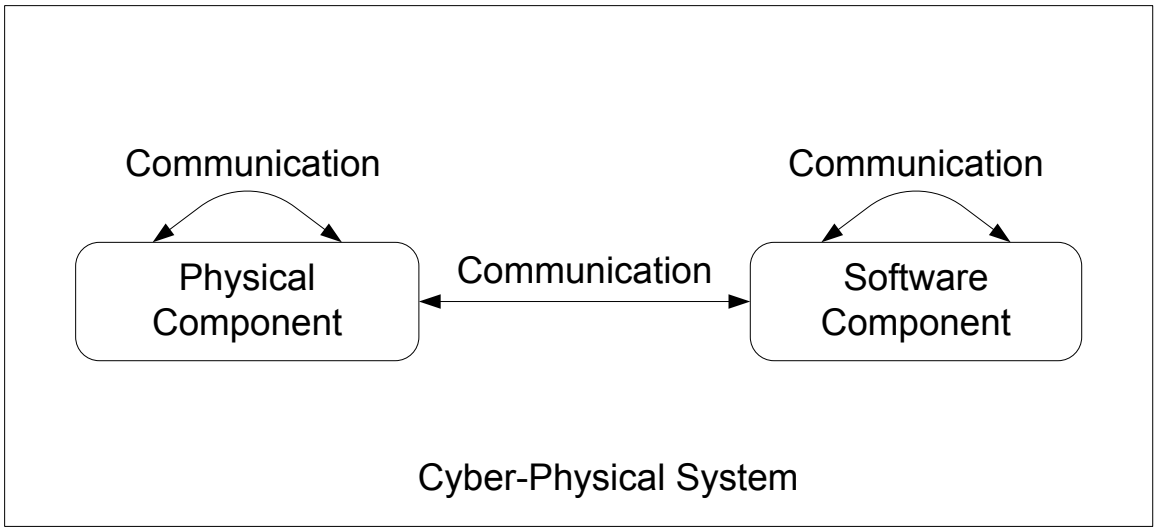


Figure 5.1: CPS reliability model.

5.3.1 CPS Reliability Model

In engineering, computing and communication have become a 'universal system integrator' for physical systems making computer based engineering systems the major source of industrial innovation [216]. Composition of CPS is used as the basis of CPS reliability model in the FARE framework. As illustrated in Figure 5.1, a simple CPS reliability model consists of physical components or hardware, cyber components or software, and communication among them. At the system level, CPS reliability can be measured or estimated and it is an integration of different components' reliability.

5.3.2 Selection of Evaluation Environment

Reliability estimation depends on the results or data collected from the tests or the actual use of the system. Figure 5.2 illustrates the decision tree approach in selecting different evaluation environments where the failure data will be collected. The top-level category determines whether the results are based on tests in a lab environment or actual use in the operational environment. In a lab environment, some tests are based on a life test that simulates the actual running environment. Life tests include Highly Accelerated Life Test

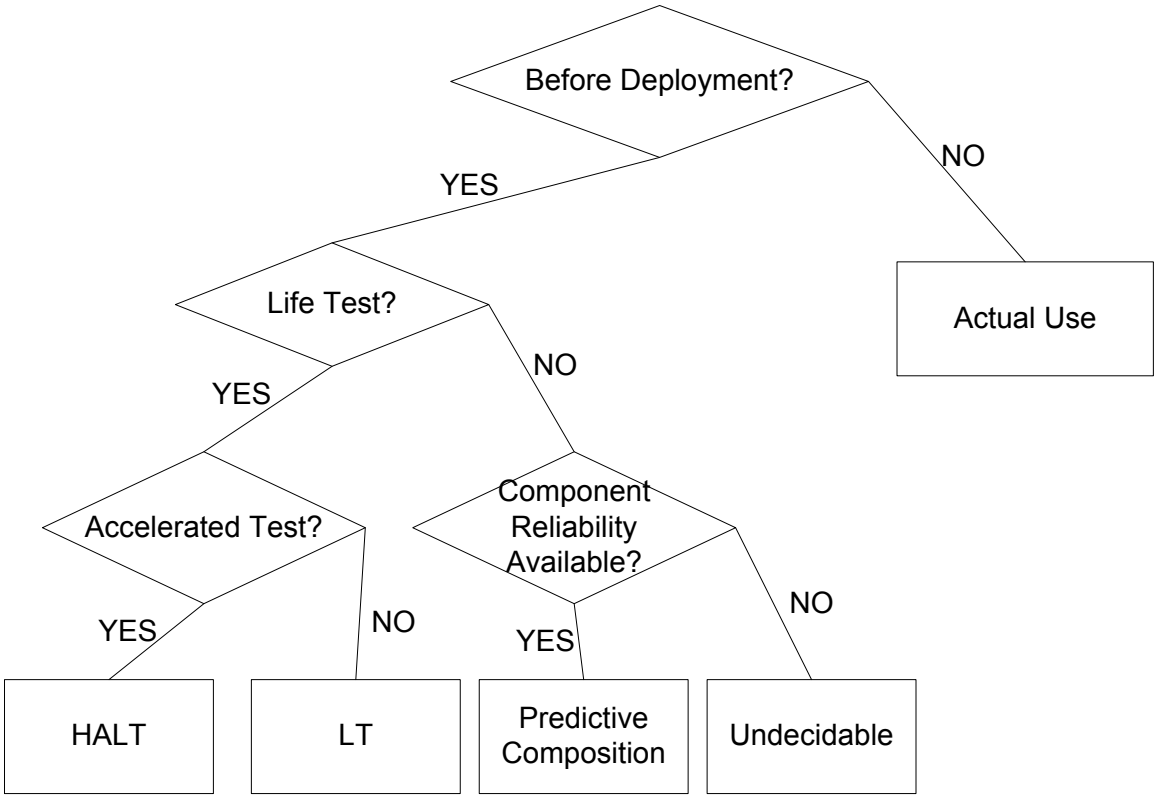


Figure 5.2: Decision tree for selecting evaluation environment.

(HALT) and Life Test (LT) in a normal pace. The HALT is similar to stress test that creates a situation such that failure is more likely to happen. It is a method based on physics-of-failure, an approach to reliability assessment based on modeling and simulation that relies on understanding the physical processes contributing to the appearance of the failures [149].

Without a life test, in a lab environment, components' reliability data can be used to construct the whole system's reliability. Although there are many ways to do the compositional reliability, those estimates are often not indicative or accurate for representing the whole system reliability. One reason is the communication failure that is often not easy to be incorporated in these models [95].

Reliability estimation in actual use employs continual failure data processing to enable rolling estimates that are often useful in system performance monitoring, especially for human operators of these systems. There are several advantages for employing reliability estimation in actual use. The first advantage is that it enables real-time feedback to the

operators or systems so that corrective actions can be implemented in a manual or autonomic fashion. Second, it enables large long-running systems such as power grids or smart buildings to be continuously monitored for reliability improvement or degradation. Those systems are often not possible to be simulated in a lab environment due to their complexity and the unpredictable running environment. The continual reliability estimates can be further used to construct a reliability profile for the system under study. The reliability profiling may show reliability changes in relation to different factors such as seasonality and usage pattern.

Not every type of CPS can be evaluated during actual use. For example, medical systems need to be properly tested for reliability prior to their use in the medical operations. Lab testing for these systems is needed.

5.3.3 Failure Analysis

Failure analysis includes failure detection and diagnostics. As illustrated in Figure 5.3, failure detection provides information for further diagnostics, along with domain knowledge and heuristics.

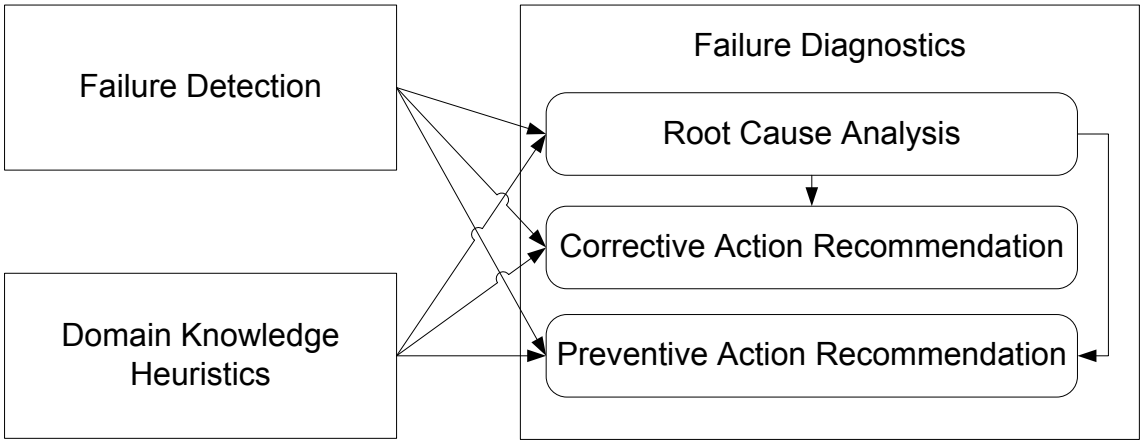


Figure 5.3: Failure analysis.

Rating	Description of Detection
1	Almost certain to detect
2	Very high chance of detection
3	High chance of detection
4	Moderately high chance of detection
5	Medium chance of detection
6	Low chance of detection
7	Slight chance of detection
8	Remote chance of detection
9	Very remote chance of detection
10	No chance of detection; no inspection

Table 5.1: Rating for detection of failure.

Failure Detection

The failure manifestation can be used as a proxy to system failure. For example, an out of range measurement indicates a system failure. Failure might be induced by the external environment, a human mistake or an internal system fault. Automated anomaly detection techniques such as those using machine learning and data mining can be used for more intelligent detection of failures. Table 5.1 lists the ratings for detection of failure [77].

Failure Diagnostics

Failure diagnostics process the detected failure data using root cause analysis techniques, corrective and preventive action recommendation techniques with possible help from domain knowledge and heuristics. Various machine learning techniques can be used in failure diagnostics [42, 191].

- Root cause analysis (RCA) [115, 146] is used to classify failure type, analyze its nature and mechanism.
- Corrective action recommendation is used to correct the current failure and avoid future recurrence of the same type of failure.

Rating	Severity Description
1	The effect is not noticed by customer
2	Very slight effect noticed by customer, does not annoy or inconvenience customer
3	Slight effect that causes customer annoyance, but they do not seek service
4	Slight effect, customer may return product for service
5	Moderate effect, customer requires immediate service
6	Significant effect, causes customer dissatisfaction; may violate regulation or design code
7	Major effect, system may not be operable; elicits customer complaint; may cause injury
8	Extreme effect, system is inoperable and a safety problem. May cause severe injury.
9	Critical effect, complete system shutdown; safety risk
10	Hazardous; failure occurs without warning; life threatening

Table 5.2: Rating for severity of failure.

- Preventive action recommendation is used to prevent the occurrence of a certain potential failure before it happens. Cost versus benefits can be a factor in determining preventive action such as replacement or inspection of the components.

Failure Severity and Impact

To evaluate failure severity and impact, the U.S. military developed Failure Mode Effects Analysis (FMEA) [77] in the 1940s. FMEA and its standards were further developed by aerospace, automotive and other industries. Table 5.2 lists the ratings for failure severity.

Preventive Maintenance

For CPS with high reliability requirement such as critical infrastructure like power grid, *preventive maintenance*, *i.e.,* maintenance of equipment or systems before fault occurs, is often employed to improve system reliability. For power companies to benefit from the use of knowledge discovery methods and statistical machine learning for preventive maintenance, Rudin *et. al* introduced a general process for transforming historical electrical

grid data into models that aim to predict the risk of failures for components and systems [197]. These models can be used directly by power companies to assist with prioritization of maintenance and repair work. Specialized versions of this process are used to produce 1) feeder failure rankings, 2) cable, joint, terminator, and transformer rankings, 3) feeder Mean Time Between Failure (MTBF) estimates, and 4) manhole events vulnerability rankings. The process in its most general form can handle data sources that are historical (static), semi-real-time or real-time, incorporate state-of-the-art machine learning algorithms for prioritization ranking, and include an evaluation of results via cross-validation and blind testing.

5.3.4 Reliability Estimation

Reliability may be measured in different ways depending on the particular situation [189]. Reliability can be estimated using a qualitative or a quantitative method. Some systems' reliability cannot be estimated quantitatively due to various reasons such as lack of failure data. For these systems, qualitative method using heuristics may be applicable.

FARE framework primarily employs quantitative methods for reliability estimation. The following are some commonly used reliability metrics that are also applicable to the FARE framework:

- *Failure rate* is defined as the total number of failures within an item population, divided by the total time expended by that population, during a particular measurement interval under stated conditions [139].
- *Mean time between failures (MTBF)* is the mean (expected) time between system failures.
- *Mean time to failure (MTTF)* is sometimes used instead of MTBF in cases where a system is replaced after a failure, since MTBF denotes time between failures in a system, which is then repaired.

- *Mean time to repair (MTTR)* is the mean time required to repair a failed component or device.

- *Availability or mission capable rate* is the proportion of time a system is in a functioning condition. This is also called system uptime ($x\%$). Using a simple representation, it can be calculated as a ratio of the expected value of the uptime of a system to the aggregate of the expected values of up and down time,

- *Availability at time t* is the probability that the item is able to function at time t [189].

- *Survival probability* is the probability that the item does not fail in a time interval $(0, t]$ [189].

- *Likelihood of recovery* is equal to the probability of the observed system recovery given a particular measurement interval under stated conditions [40].

- *Extent of recovery* is the degree to which the system has recovered [50].

Additionally, I introduce three new reliability measurement metrics in the FARE framework in order to provide coverage for some specific evaluation scenarios:

- *Theta* is the rate of reliability change over time. If MTBF is used as the reliability measurement, then Theta can be calculated as

$$\Theta(t) = \frac{MTBF(t) - MTBF(t + \Delta t)}{\Delta t}.$$

  On a MTBF versus time scatter plot chart, Theta indicates the slope of the linear regression. In a long running continual evaluation environment, Theta provides a useful indicator of the reliability improvement or degradation over time.

- *Vega* is the rate of reliability change over a selected variable, which can be any factor of interest. Similar to Theta, it is a derivative measurement of the reliability for better

indication of the reliability improvement or degradation with respect to a specific variable.

- *Cross-sectional failure percentage (CSFP)* is the percentage of total failed items within an item population in use at a specific time t. This may look similar to failure rate or availability at time t. But they are not the same. In a simple form, failure rate is the total number of failures divided by the total time. Availability at time t is the probability that a single item does not fail at time t. Cross-sectional failure percentage is the total number of failed items divided by the total number of items in use at a given time t. It is a ratio based on actual measurement. This metric is especially useful for a large system that has a large number of subsystems or components running in parallel.

To give some further explanation, I use $\lambda(t)$ to denote the failure rate at time t, and $R(t)$ to denote the reliability function (or survival function), which is the probability of no failure before time t. Then the failure rate is:

$$\lambda(t) = \frac{R(t) - R(t + \Delta t)}{\Delta t \cdot R(t)}.$$

As Δt tends to zero, the above λ becomes the instantaneous failure rate, which is also called hazard function (or hazard rate) $h(t)$:

$$h(t) = \lim_{\Delta t \to 0} \frac{R(t) - R(t + \Delta t)}{\Delta t \cdot R(t)}.$$

A failure distribution $F(t)$ is a cumulative failure distribution function that describes the probability of failure up to and including time t:

$$F(t) = 1 - R(t), t \geq 0.$$

For system with a continuous failure rate, $F(t)$ is the integral of the failure density function

$f(t)$:

$$F(t) = \int_0^t f(x)\,\mathrm{d}x.$$

Then the hazard function becomes

$$h(t) = \frac{f(t)}{R(t)}.$$

For the Weibull [236, 190] failure distribution, the failure density function $f(t)$ and cumulative failure distribution function $F(t)$ are

$$f(t;\lambda,k) = \begin{cases} \frac{k}{\lambda}(\frac{t}{\lambda})^{k-1} e^{-(t/\lambda)^k}, & t \geq 0 \\ 0, & t < 0 \end{cases}$$

$$F(t;\lambda,k) = \begin{cases} 1 - e^{-(t/\lambda)^k}, & t \geq 0 \\ 0, & t < 0 \end{cases}$$

where $k > 0$ is the shape parameter and $\lambda > 0$ is the scale parameter of the distribution. The hazard function when $t \geq 0$ can be derived as

$$h(t;\lambda,k) = \frac{f(t;\lambda,k)}{R(t;\lambda,k)} = \frac{f(t;\lambda,k)}{1 - F(t;\lambda,k)} = \frac{k}{\lambda}\left(\frac{t}{\lambda}\right)^{k-1}.$$

A value of $k < 1$ indicates that the failure rate decreases over time. A value of $k = 1$ indicates that the failure rate is constant (*i.e.*, k/λ) over time. In this case, the Weibull distribution becomes an exponential distribution. A value of $k > 1$ indicates that the failure rate increases with time.

The mean time to failure is given by

$$MTTF = \int_0^\infty t \cdot f(t)\,\mathrm{d}t = \int_0^\infty R(t)\,\mathrm{d}t.$$

If $MTTR$ is known, then the availability is

$$Availability = \frac{MTTF}{(MTTF + MTTR)}.$$

5.4 Implementation

I developed a prototype FARE system for this study using Java programming language and MATLAB.

Software Design

As illustrated in the software architecture diagram in Figure 5.4, the software includes a data preprocessor, a failure detector, a reliability estimator with metrics and profiler, and a user interface along with data output component. It was designed with modular components so that it can be used along with or embedded in another larger system.

5.5 Empirical Studies

In this section, I will describe some empirical studies.

5.5.1 Evaluation Methodology

The reliability metrics described in section 3.5.1 and the following performance metrics are used to evaluate the FARE system quantitatively.

5.5.2 Real-World Experiments

I evaluated the prototype FARE system using a smart building BMS (building management system) system at a large commercial building in New York City. A BMS is a type of CPS consisting of both software and hardware components that controls and monitors a

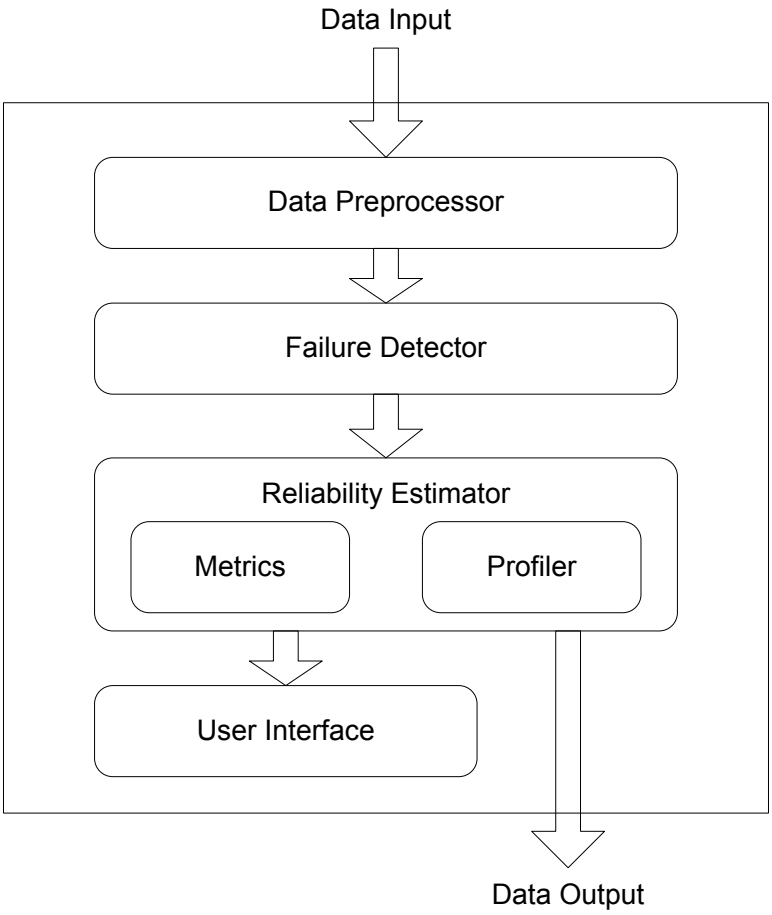


Figure 5.4: FARE software components and data flow.

building's mechanical and electrical equipment, such as ventilation, lighting, power systems, fire systems and security systems. The building energy control system is an important component of the BMS that reads data feeds representing internal and exogenous conditions (*e.g.*, temperature, humidity, electrical load, peak load, fluctuating electricity pricing and building work schedule) and takes control actions (*e.g.*, adjusts lighting, turns on/off the air-conditioning and shuts off partial or all elevators) accordingly. Building operators usually have the ability to change or override control actions taken by the BMS to accommodate special situations such as severe weather or changes in the building's work schedule.

5.5.3 Experiments on Smart Building BMS System

Figure 5.5 illustrates the experimental setup. The building's BMS's software collects various data sources and stores them in the local BMS database. I established a data transmission link between the BMS server and the remote server where FARE software is installed and running.

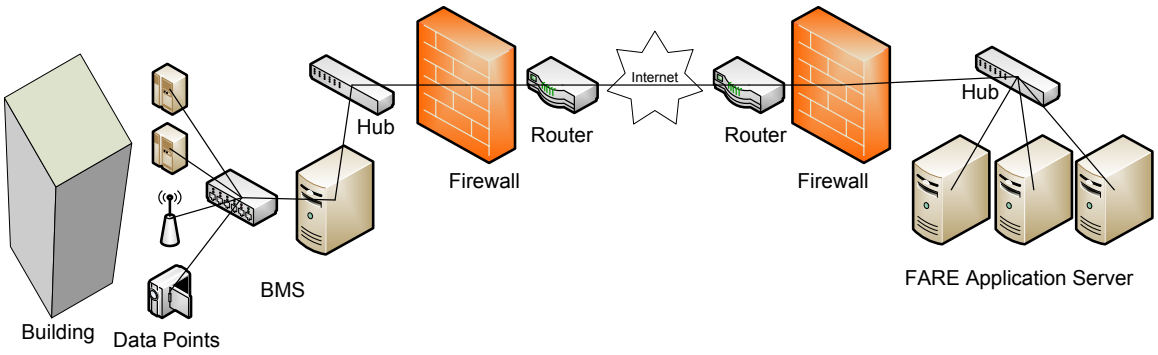


Figure 5.5: Experimental setup.

Determination of the failure trigger condition depends on the data sources being used. In my experiments, FARE software first processes the collected data using the failure criteria to obtain a time series dataset of failure instances. This data is then processed by the FARE software to obtain reliability estimates on the fly.

The following are some experimental results.

Failure Detection

Figure 5.6 shows an example BMS time series data for six months starting from July 1, 2012 to January 1, 2013. As a simple threshold failure detection method, the data points with a value above 80 or below 65 were determined to be nonconformity or failure.

Similarly, Figure 5.7 shows an example BMS electricity time series data for two months starting from August 1, 2013 to September 30, 2013. The chart shows the actual and predicted usage of electricity, along with the system start-up and ramp-down time. Using the simple threshold failure detection method, the actual usage data points with value below

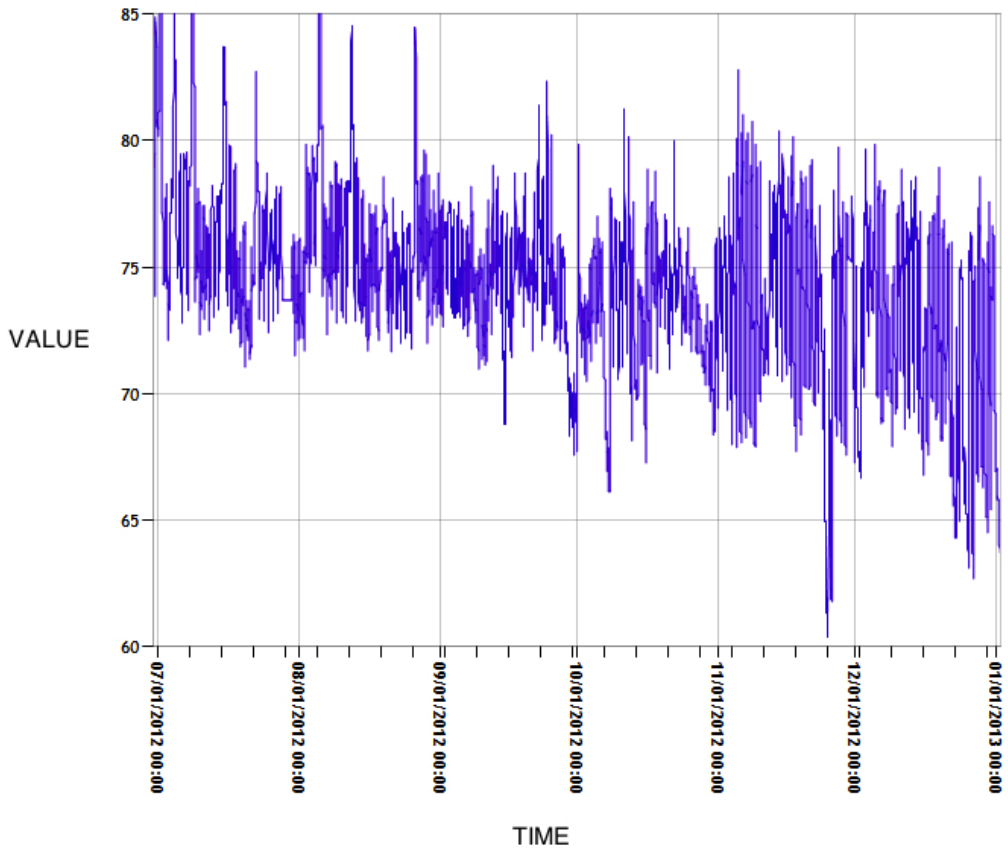


Figure 5.6: BMS time series data.

300 kilowatts (kW) were determined to be nonconformity or failure.

Reliability Estimation

After the failure incidence time series data is collected, FARE then estimates reliability metrics as described in section 5.3.4. To follow the example described in Figure 5.6, the weekly failure rates for the six months are listed and charted in Figure 5.8. Also, the linear regression $y = -0.0229x + 1.1169, R^2 = 0.03829$ shows the improved results over the time. In this case, the Theta equals -0.0229 using failure rate as the reliability measurement.

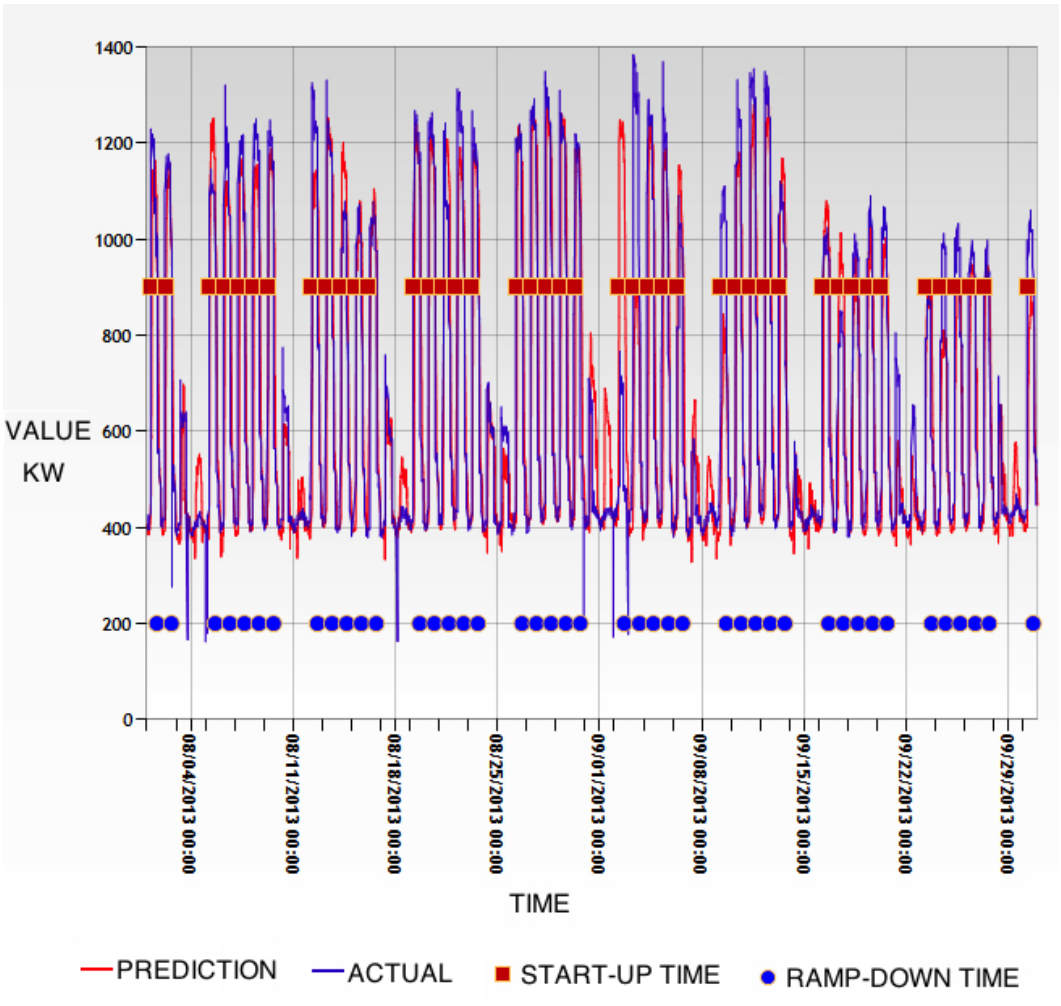


Figure 5.7: BMS electricity time series data.

Summary

In summary, the experiments show the approach is effective for benchmarking reliability of CPS. Furthermore, the approach can be implemented relatively easily during the use phase of the CPS to enable continuous failure analysis and reliability estimation. The framework can also be used to decide the most appropriate evaluation environment following a decision tree.

Week	Failure Rate
1	2
2	1
3	1
4	1
5	3
6	1
7	1
8	0
9	1
10	0
11	0
12	1
13	1
14	0
15	1
16	0
17	0
18	0
19	1
20	0
21	1
22	2
23	0
24	0
25	0
26	3

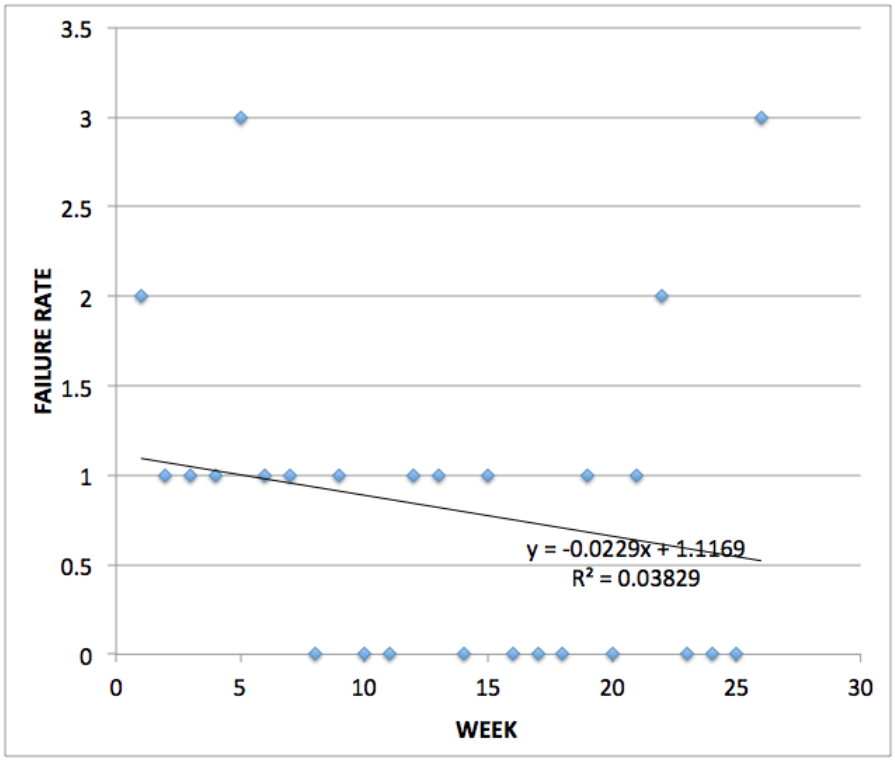


Figure 5.8: Weekly failure rate.

5.6 Summary

This chapter presents FARE (Failure Analysis and Reliability Estimation), a framework for benchmarking reliability of cyber-physical systems. The framework employs a generic CPS reliability model, a set of methods and metrics on the evaluation environment selection, failure analysis, and reliability estimation for benchmarking CPS reliability. It not only provides a retrospect evaluation and estimation of the CPS system reliability using the past data, but also provides a mechanism for continuous monitoring and evaluation of CPS reliability for runtime enhancement. The framework is generic and can accommodate new reliability measurement techniques and metrics. My empirical evaluation demonstrated that FARE is easy to implement, accurate for comparison and can be used for building useful industry benchmarks.

Chapter 6

Related Work

Some related work has been discussed in the prior chapters. In this chapter, I will describe additional prior research and compare them with my approach.

In the following section, I will describe some related work on automated online evaluation. In section 6.2 and 6.3, I will describe some related work on data quality analysis and self-tuning. In section 6.4 and 6.5, I will describe some related work on cloud computing and data-intensive computing. Finally in section 6.6, I will describe some related work on failure analysis and reliability estimation.

6.1 Automated Online Evaluation

In section 1.5, I mentioned some prior work on achieving CPS reliability through system design, usually prior to the deployment of the CPS in the field. My approach aims to improve the reliability of CPS using automated online evaluation. It employs a runtime evaluation with feedback loop for deployed CPS running under the real-world unpredictable environment. Thus, these "reliability by design" researches are complementary to my approach.

6.1.1 Runtime Evaluation of ML System

In section 3.5.6, I described the application and experiments of the ARIS system on the smart building ML system. Then in section 3.5.4, I gave examples each of the steps of automated online evaluation for smart power grid ML system, using NYC power grid data. Depending on specific data and operational goals, there may be many ways to perform one of the three evaluations; the key point is that all of these three types of evaluation must be present. In machine learning and data mining, only the second type of evaluation–output data quality–is typically considered, and even that evaluation is mainly considered in static settings (without the element of time).

Langley's seminal paper "Machine Learning as an Experimental Science" made empirical study an indispensable aspect of machine learning research [131]. Since that time, many challenges in experimental machine learning have been identified. For instance, a more recent survey of Japkowicz reviewed shortcomings in current evaluation methods [109]. Through using ARIS on the New York City power grid, I have also been able to identify new challenges (*e.g.*, the AUC cyclicity challenge). In machine learning, the goal is often to optimize the criteria used for evaluation. ARIS suggests a much more ambitious set of evaluations than what is usually performed in machine learning and data mining experiments, potentially leading to a much broader way to consider and design machine learning systems, and hopefully leading to improvements in power grid operations.

Murphy *et al.* have done research on verification of machine learning programs from software testing perspective [161]. My approach does not verify the internal correctness of the machine learning or data mining component. ARIS treats the machine learning and data mining process as a black-box module and conducts evaluation according to its external specifications. This leaves the quality assurance of the machine learning and data mining software module to the machine learning researchers and software developers or testers.

6.1.2 Runtime Evaluation for Improving Reliability and Security

Mitchell and Chen described a probability model based on stochastic Petri nets [180] to describe the behavior of the CPS in the presence of both malicious nodes exhibiting a range of attacker behaviors, and an intrusion detection and response system (IDRS) for detecting and responding to malicious events at runtime [157]. Vaseashta *et al.* described some vulnerabilities and countermeasures for sensor network [226]. Walters *et al.* gave a general overview on wireless sensor network security: obstacles, requirements, attacks, and defenses [228]. My approach does not address the malicious attack in the context of security, but the automated online evaluation approach and the data quality analysis techniques I described may be expanded to handle some security related challenges.

In order to deal with the problem that software products are often released with missing functionality or errors that result in failures in the field, Bowring *et al.* described an approach of monitoring deployed software using software tomography. This approach splits monitoring tasks across many instances of the software, so that partial information can be collected from users by means of light-weight instrumentation and merged to gather the overall monitoring information [29]. Elbaum and Hardojo also did an empirical study of profiling strategies for released software and their impact on testing activities [70]. These perpetual testing [156] studies are similar to my approach in terms of continuous evaluation after the systems are deployed in the field, but they target only software systems.

6.2 Data Quality Analysis

Some prior research has been done on data quality analysis. Ballow and Pazer presented a general model to assess the impact of data and process quality upon the outputs of multi-user information-decision systems. The data flow/data processing quality control model was designed to address several dimensions of data quality at the collection, input, processing and output stages [22]. Wang and Strong developed a framework that captures the aspects

of data quality that are important to data consumers. A two-stage survey and a two-phase sorting study were conducted to develop a hierarchical framework for organizing data quality dimensions [232]. Pipino *et al.* presented subjective and objective assessments of data quality, as well as simple ratio, min or max operators, and weighted average–three functional forms that can help in developing data quality metrics in practice [181]. Their work does not employ computational intelligence for data quality analysis.

Data mining finds its increased adoption and application in software engineering in recent years. Gegick *et al.* performed text mining of bug reports to identify security issues [85]. Hassan and Xie described the concept of software intelligence and the future of mining software engineering data [94]. Xie *et al.* presented a general overview of data mining for software engineering and described an example of duplicate bug detection using vector space-based similarity [248]. Wang *et al.* also described an approach to detect duplicate bug reports using both natural language and execution information [233].

My redundancy checking engine uses both probability distribution-based KL divergence and vector space-based Cosine similarity ranking, instead of only vector space-based similarity. Furthermore, my approach provides a similarity ranking list that can be used for search, instead of only Yes and No on duplication check. Gegick *et al.* presented text mining of bug reports to identify security issues [85]. Their work aims to identify security problems such as buffer overflow through mining the bug reports. Their purpose and techniques are different from my approach.

6.2.1 Data Anomaly Detection

Data anomaly or outlier detection has been a popular research area for many years due to its broad application such as network intrusion detection for computer security [133]. In their survey paper [39], Chandola *et al.* provided a structured and comprehensive overview of the research on anomaly detection and identified the advantages and disadvantages of different techniques in each category.

For processing stream data, Breunig *et al.* introduced the concept of local outlier factor (LOF), which is a degree, instead of a binary choice, of being an outlier, and assigned it to each object. They gave a detailed formal analysis showing that LOF has many desirable properties. As mentioned in section 3.2.3 and 4.2.2, Pokrajac *et al.* proposed an incremental LOF algorithm, appropriate for detecting outliers in data streams [182]. I used this algorithm in the online anomaly detection. Pokrajac *et al.* further developed an incremental version of connectivity-based outlier factor (COF) algorithm and discuss its computational complexity [183]. Tan *et al.* introduced Streaming Half-Space-Trees (HS-Trees), a fast one-class anomaly detector for evolving data streams, which requires only normal data for training and works well when anomalous data are rare [217]. My approach does not limit itself to any specific type of anomaly detection technique.

6.2.2 Data Anomaly Diagnosis

Although equally important as data anomaly detection, data anomaly diagnosis received much less attention within the research community. With the exponential increase of data volume in different kinds of 'smart' devices and CPS, the automatic analysis of data and diagnosis of data anomalies are becoming more and more important.

Lakhina *et al.* proposed a method to diagnose network-wide traffic anomalies based on a separation of the high-dimensional space occupied by a set of network traffic measurements into disjoint subspaces corresponding to normal and anomalous network conditions [129, 130]. Yang *et al.* developed a diagnosis technique that uses standard monitoring data to determine which related changes in behavior may cause anomalies in grid computing environment [249]. McIntosh *et al.* described a loosely-coupled, semi-automated diagnosis system to diagnose thermal hotspots in a data center to substantially reduce the time, tedium and expertise for analyzing hotspot anomalies that sometimes occur due to excessive workload or equipment failures [150]. Their work is intended for a specific type of system and uses a different diagnosis approach than my machine learning based approach.

6.3 Self-Tuning

In their paper "The Vision of Autonomic Computing," Kephart and Chess from IBM Thomas J. Watson Research Center proposed autonomic computing as a solution to the almost impossible difficulty of managing complex current and planned computing systems, which require integrating several heterogeneous environments into corporate-wide computing systems that extend into the Internet [122]. Kaiser *et al.* have retrofitted autonomic computing onto legacy systems externally, without any need to understand or modify the code, and in many cases even when it is impossible to recompile [117, 174]. As presented in my approach, autonomic computing can be further extended and applied on the complex CPS running in an unpredictable environment and processing increasingly large amount of data.

Chaudhuri and Narasayya discussed self-tuning database systems and automated physical database design in the AutoAdmin project at Microsoft Research [41]. Sullivan demonstrated in his Ph.D. thesis that probabilistic reasoning and decision-making techniques can be used as the foundation of an effective, automated approach to software tuning [212]. Herodotou *et al.* introduced a self-tuning system for Hadoop database systems and big data analytics. The system adapts to user needs and system workloads to provide good performance automatically, without any need for users to understand and manipulate the many tuning knobs in Hadoop [96]. I employed self-tuning in the various aspects of my approach including the ARIS system and COBRA framework. My experiments demonstrated its effectiveness.

6.4 Cloud Computing

NIST cloud computing model is composed of five essential characteristics (*i.e.*, on-demand self-service, broad network access, resource pooling, rapid elasticity, and measured service), three service models (*i.e.*, Software as a Service (SaaS), Platform as a Service (PaaS), and Infrastructure as a Service (IaaS)), and four deployment models (*i.e.*, private cloud,

community cloud, public cloud, and hybrid cloud) [151]. The COBRA framework takes advantage of the cloud model's characteristics by using the Platform as a Service (PaaS) in either public or private cloud.

Armbrust *et al.* gave a Berkeley view of cloud computing and listed top 10 obstacles and opportunities for cloud computing [18, 19]. One of these obstacles is the data transfer bottlenecks. In my COBRA framework, I described my solution to this problem within the context of CPS data. Another obstacle mentioned is the difficulty in scaling quickly. COBRA framework employs self-tuning to achieve elastic scaling.

As a type of cloud based monitoring system, Amazon CloudWatch [7] provides monitoring for cloud resources such as Amazon EC2, Amazon database instances and the applications customers run on Amazon Web Services. Developers and system administrators can use it to collect and track metrics, gain insights and react immediately to keep their applications and services running smoothly. COBRA can connect to CloudWatch by generating and passing through custom metrics. The service provided by CloudWatch can simplify the implementation of COBRA.

6.5 Data-Intensive Computing

Large scalable data management and data-intensive computing has been a research area for many years and much research has focused on large scale data management in a traditional enterprise setting. In their 1998 article, Moore *et al.* described data-intensive computing and digital libraries [158]. Cannataro *et al.* defined applications that explore, query, analyze, visualize, and, in general, process very large scale data sets as *data intensive applications* [36].

Bryant called for data-intensive scalable computer (DISC) systems, which differ in fundamental ways from existing high-performance computing (HPC) systems, for scientific computing applications because these applications must accumulate and manage massive

datasets, as well as perform sophisticated computations over these data [32]. Szalay *et al.* presented the architecture for a three tier commodity component cluster designed for a range of data intensive computations operating on petascale data sets called GrayWulf to meet the challenge for traditional supercomputing architectures that maximize FLOPS, *i.e.*, FLoating-point Operations Per Second, since CPU speed has surpassed IO capabilities of HPC systems and clusters [215]. These approaches do not take advantage of cloud computing's capabilities in order to meet the challenges of data-intensive computing.

Some prior work has been done on the convergence of data-intensive computing and cloud computing. Agrawal *et al.* presented some novel challenges in the cloud computing that must be addressed to ensure the success of data management solutions in the cloud environment [3]. Bicer *et al.* described the challenge of data analysis in a scenario where data is stored across a local cluster and cloud resources; proposing a software framework to enable data-intensive computing using cloud bursting, *i.e.*, using a combination of compute resources from a local cluster and a cloud environment to perform Map-Reduce type processing on a data set that is geographically distributed [26].

Moretti *et al.* presented an abstraction entitled All-Pairs that enables campus computing grids to provide end users with high-level abstractions that allow for the easy expression and efficient execution of data-intensive workloads such as applications in biometrics, bioinformatics, and data mining [159]. Garcia *et al.* presented a campus-based Large Scale Data Facility (LSDF) with adequate storage space and a directly attached analysis farm with value added services for the big scientific data-sets using the mixed Hadoop, Open Nebula Cloud environments, a metadata repository, community specific metadata schemes, graphical tools, and APIs [83]. Similar to my COBRA framework, these approaches try to meet the challenges of data-intensive computing by employing the cloud computing environment, either public, private or hybrid ones, and techniques.

6.6 Failure Analysis and Reliability Estimation

6.6.1 Failure Analysis

Failure analysis has been a popular research area for many years. Stamatis described the theory and execution of Failure Mode Effect Analysis (FMEA), a design tool used to systematically analyze postulated component failures and identify the resultant effects on system operations [210]. Robitaille categorized and described some common corrective actions in his handbook [192]. de Visser *et al.* described failure severity in new product development processes of consumer electronics [57]. Sheldon and Jerath described assessing the effect of failure severity, coincident failures and usage-profiles on the reliability of embedded control systems [204]. Appendix A describes a method of reliability estimation using a semiparametric model with Gaussian smoothing, instead of Weibull and exponential failure distribution. The evaluation of the method used some power grid CPS failure data. These works target only some specific systems while my FARE framework is applicable to general cyber-physical systems.

6.6.2 Component Reliability Assessment

Some prior research has been done on component reliability. Pechet and Nash gave a comprehensive review of the predictive methods for predicting the reliability of hardware electronic equipment [178]. He *et al.* described a theoretical framework for analyzing communication reliability using frequency domain analysis and reliability calculus [95]. Appendix B describes BugMiner, a software reliability analysis technique employing data mining on bug reports. Appendix C describes a mutation technique for constructing subtle software bugs, which can be used for software reliability analysis. These works are complementary to my FARE framework.

6.6.3 System Reliability Assessment

The failure reporting, analysis and corrective action system (FRACAS) [208] was developed by the US government and first introduced for use by the US Navy and all Department of Defense agencies in 1985. The method calls for systematic failure data collection, management, analysis and corrective action implementation. Its process is a disciplined closed loop failure reporting, analysis and corrective action system. FRACAS is typically used in an industrial environment to collect data, record and analyze system failures, mostly as an offline postmortem analysis for reporting purpose. My approach employs online continual runtime evaluation to achieve reliability assurance. Further, FRACAS deals with actual failures that have happened, while my approach aims to prevent failures from happening as well as correct them through intelligent data quality analysis and action recommendation.

Chapter 7

Conclusion

In this chapter, I will describe the contributions, research accomplishments, future work and conclusion of my thesis.

7.1 Contributions

There are three major contributions in this thesis work. The first contribution is automated online evaluation (AOE), which is a data-centric runtime monitoring and reliability evaluation approach that performs data quality analysis using computational intelligence and self-tuning techniques to improve system reliability for cyber-physical systems that process large amounts of data, employ software as a system component, run online continuously and maintain an operator-in-the-loop. I have presented an example architecture of AOE called autonomic reliability improvement system (ARIS). My implementation and experiments with ARIS in smart building CPS and smart power grid CPS have demonstrated that this approach is effective and efficient. The data-dependence of this system makes it easily applicable to different types of cyber-physical systems, and the open expandable architecture also enables the incorporation of new data quality analysis and self-tuning techniques. A list of items under this contribution is as follows:

- A system evaluation approach called automated online evaluation that is able to improve system reliability for cyber-physical systems in the domain of interest as described in section 1.2. The approach employs data quality analysis and self-tuning. It enables online reliability assurance of the deployed systems that are not possible to perform robust tests prior to actual deployment because of physical and cost constraints.

- A prototype implementation of the approach, *i.e.*, ARIS system, and experimental demonstration of the approach using ARIS in some controlled experiments as well as real-world environments.

- A new technique of data quality analysis using computational intelligence and its application in this type of evaluation system for cyber-physical system.

- A new demonstration of applying self-tuning in this type of evaluation system for cyber-physical systems.

- A study on applicability of the approach on other domains in order to show that the approach can be potentially adapted and extended for use in improving system reliability for a much broader range of large-scale real-world online cyber-physical systems.

The second contribution is COBRA, a cloud-based reliability assurance framework for data-intensive CPS. COBRA provides automated multi-stage runtime reliability evaluation along the CPS workflow using data relocation services, a cloud data store, data quality analysis and process scheduling with self-tuning to achieve scalability, elasticity and efficiency. A set of performance metrics is used to evaluate the system performance on the fly. A prototype of COBRA has been implemented and experimented with on a smart building management system. The experiments show that it is effective, efficient, scalable and easy to implement.

The third contribution is FARE, a framework for benchmarking reliability of cyber-physical systems. The framework provides a CPS reliability model, a set of methods and metrics on evaluation environment selection, failure analysis, and reliability estimation for benchmarking CPS reliability. It not only provides a retrospect evaluation and estimation of the CPS system reliability using the past data, but also provides a mechanism for continuous monitoring and evaluation of CPS reliability for runtime enhancement. The framework is extensible for accommodating new reliability measurement techniques and metrics. My empirical evaluation demonstrated that FARE is easy to implement, accurate for comparison and can be used for building useful industry benchmarks.

7.2 Research Accomplishments

In addition to the main contributions described above, my research accomplishments include a list of publications and software applications.

7.2.1 List of Publications

For this thesis work, I have completed following papers:

Papers related to AOE, *i.e.*, automated online evaluation for CPS as described in section 3:

- Leon Wu and Gail Kaiser. An Autonomic Reliability Improvement System for Cyber-Physical Systems. In Proceedings of the IEEE 14th International Symposium on High-Assurance Systems Engineering (HASE), October 2012 [240].

- Leon Wu, Gail Kaiser, David Solomon, Rebecca Winter, Albert Boulanger, and Roger Anderson. Improving Efficiency and Reliability of Building Systems Using Machine Learning and Automated Online Evaluation. In Proceedings of the Eighth Annual

IEEE Long Island Systems, Applications and Technology Conference (LISAT), May 2012 [244].

- Leon Wu, Gail Kaiser, Cynthia Rudin, and Roger Anderson. Data Quality Assurance and Performance Measurement of Data Mining for Preventive Maintenance of Power Grid. In Proceedings of the ACM SIGKDD 2011 Workshop on Data Mining for Service and Maintenance, August 2011 [242].

- Leon Wu, Gail Kaiser, Cynthia Rudin, David Waltz, Roger Anderson, Albert Boulanger, Ansaf Salleb-Aouissi, Haimonti Dutta, and Manoj Pooleery. Evaluating Machine Learning for Improving Power Grid Reliability. In ICML 2011 Workshop on Machine Learning for Global Challenges, July 2011 [243].

Papers related to FARE, *i.e.*, a reliability benchmark framework for CPS as described in section 5:

- Leon Wu and Gail Kaiser. FARE: A Framework for Benchmarking Reliability of Cyber-Physical Systems. In Proceedings of the Ninth Annual IEEE Long Island Systems, Applications and Technology Conference (LISAT), May 2013 [241].

- Leon Wu, Boyi Xie, Gail Kaiser, and Rebecca Passonneau. BugMiner: Software Reliability Analysis Via Data Mining of Bug Reports. In Proceedings of the 23th International Conference on Software Engineering and Knowledge Engineering (SEKE), July 2011 [246].

- Leon Wu, Timothy Tervinen, Gail Kaiser, Roger Anderson, Albert Boulanger, and Cynthia Rudin. Estimation of System Reliability Using a Semiparametric Model. In Proceedings of the IEEE EnergyTech 2011 (EnergyTech), May 2011 [245].

7.2.2 List of Software Applications

For empirical studies, I have developed and experimented with the following software applications:

- ARIS, an autonomic reliability improvement system as described in section 3.3.
- COBRA, a cloud-based reliability assurance framework as described in section 4.3.
- FARE, a reliability benchmark framework as described in section 5.4.

7.3 Future Work

In this section, I will describe future work for different aspects of this thesis work.

7.3.1 Autonomize AOE for CPS

For future work on automated online evaluation as described in section 3.2, one potential direction is to further offload the work by human system operators, thus closing the feedback loop, and instead employ some automated software controllers with actuators or robots that can take actions as human operators on the cyber-physical systems. In this way, the whole cyber-physical system can be fully autonomic with self-management, self-configuration and self-healing. Some prior work has been done for closed-loop control. Li *et al.* introduced a modeling framework for identifying dynamic models of systems that are under feedback control with closed-loop conditions and produced a joint representation including both the plant and controller models in state space form [136]. This kind of study may be referenced for developing a fully autonomic CPS reliability improvement system in the future.

7.3.2 Security for Autonomic System

The future research of enabling fully autonomic CPS reliability improvement operation without human intervention will have to take into consideration other factors such as security

and responsibility. If an Internet or cloud computing-based approach will be used, then the cybersecurity will have to be considered as an additional system requirement. In their paper "Security as a New Dimension in Embedded System Design," Kocher *et al.* provided a unified and holistic view of embedded system security by analyzing the typical functional security requirements for embedded systems from an end-user perspective [125]. Romero-Mariona *et al.* described some techniques for developing cybersecurity requirements [193]. These prior works can be useful for developing future secure reliability improvement systems.

7.3.3 Improvement of Data Quality Analysis

For data quality analysis, one future work is to go beyond the disjoint model. As mentioned in section 4.5.4, one limitation in the prototype implementation of the COBRA framework is the assumption of the disjoint model. The current work assumes that processes can be run independently of each other and that the order in which processes are executed is not important. Hence, it treats every streaming data series from each source (*e.g.*, sensor data point) separately. It does not take into consideration correlation or association between separate data sources. In many cases, reliability issues might have causal or clustering effects. For example, a failure in one software component might also cause other adjacent or dependent components to fail. These inter-data-series correlations need to be handled in future work.

Furthermore, new data quality analysis and self-tuning techniques can be explored and experimented with in the future work. The ARIS system and COBRA framework are not limited to any specific type of evaluation techniques or implementation programming languages. With a broad spectrum of computational learning algorithms and data analysis techniques available, new methods may be experimented with and employed to improve the reliability improvement system.

7.3.4 Building a CPS Reliability Benchmark Database

My experiments for the FARE framework as described in section 5.5 are not extensive due to the limited access to the CPS and the amount of data available. My real-world experiments used smart power grid systems and smart building systems. Other types of sensor-based CPS include wireless sensor networks and other types of intelligent control CPS include autonomous automotive systems, medical monitoring, process control systems, distributed robotics, and automatic pilot avionics. One possible future work would be to apply the approach to some additional types of CPS with some larger datasets. This would be especially useful for benchmarking evaluation so that a benchmarking results database can be developed for further analysis and comparison by research and industry communities. A successful benchmarking system for reference is the TPC Benchmark C for online transaction processing (OLTP) [219].

7.3.5 Privacy Protection for CPS Users

In the case of CPS involving private information about human subjects, such as medical systems dealing with patients' personal health information, privacy protection will become another important system requirement. Improving system reliability for this CPS should not violate the privacy policy. For example, when a CPS encounters data anomalies or system failures, the reliability assurance process should not collect or transmit sensitive information that is beyond the CPS's operating state for reliability purpose. In their survey paper [45], Choi *et al.* discussed the challenges associated with privacy in health care in the electronic information age based on the Health Insurance Portability and Accountability Act of 1996 (HIPAA) Privacy and Security Rules [223]. They examined the storing and transmission of sensitive patient data in the modern health care system and discussed current security practices that health care providers institute to comply with HIPAA Security Rule regulations. This kind of study will have to be incorporated for those types of CPS that involve human private information.

7.4 Conclusion

In this thesis, I presented automated online evaluation (AOE), a data-centric runtime monitoring and reliability evaluation approach that performs data quality analysis using computational intelligence and self-tuning techniques to improve system reliability for cyber-physical systems that process large amounts of data, employ software as a system component, run online continuously and maintain an operator-in-the-loop. AOE employs data quality analysis and self-tuning techniques. My experiments with ARIS, a prototype implementation of AOE, in smart building CPS and smart power grid CPS have demonstrated that this approach is effective and efficient.

Additionally, in order to conduct efficient and cost-effective automated online evaluation for data-intensive CPS, I presented COBRA, a cloud-based reliability assurance framework using data relocation services, a cloud data store, data quality analysis and process scheduling with self-tuning to achieve scalability, elasticity and efficiency. A set of performance metrics is used to evaluate the system performance on the fly. My experiments on a smart building BMS shows that it is effective, efficient, scalable and easy to implement.

Finally, in order to provide a generic way to compare and benchmark system reliability for CPS and to extend the approach described above, I further presented FARE, a framework for benchmarking reliability of cyber-physical systems. The framework provides a CPS reliability model, a set of methods and metrics on evaluation environment selection, failure analysis, and reliability estimation for benchmarking CPS reliability. It not only provides a retrospect evaluation of the CPS system reliability using the past data, but also provides a mechanism for continuous monitoring and evaluation of CPS reliability for runtime enhancement. My empirical evaluation demonstrated that FARE is easy to implement, accurate for comparison and can be used for building useful industry benchmarks.

This work on improving system reliability for CPS validated the hypotheses outlined in section 1.7, advanced the state of the art in the CPS reliability research, expanded the body of knowledge in this field, and provided some useful studies for further research.

Chapter 8

Bibliography

[1] A. T. Acree, T. A. Budd, R. A. DeMillo, R. J. Lipton, and F. G. Sayward. Mutation analysis. Technical Report GIT-ICS-79/08, Georgia Institute of Technology, Atlanta, Georgia, 1979.

[2] Y. Agarwal, B. Balaji, R. Gupta, J. Lyles, M. Wei, and T. Weng. Occupancy-driven energy management for smart building automation. In *Proceedings of the 2nd ACM Workshop on Embedded Sensing Systems for Energy-Efficiency in Building*, BuildSys '10, pages 1–6, New York, NY, USA, 2010. ACM.

[3] D. Agrawal, S. Das, and A. El Abbadi. Big data and cloud computing: current state and future opportunities. In *Proceedings of the 14th International Conference on Extending Database Technology*, EDBT/ICDT '11, pages 530–533, New York, NY, USA, 2011. ACM.

[4] M. Albu and G. T. Heydt. On the use of RMS values in power quality assessment. *IEEE Transactions on Power Delivery*, 18(4):1586–1587, 2003.

[5] J. Aldrich, E. G. Sirer, C. Chambers, , and S. J. Eggers. Comprehensive synchronization elimination for java. In *Science of Computer Programming*, pages 47(2–3):91–120, 2003.

[6] N. D. Alekseeva, A. V. Bubnova, V. S. Chudny, and N. N. Tikhodeev. Reliability analysis and comparison of long-distance HVAC and HVDC power transmission lines. In *Proceedings of 2002 International Conference on Power System Technology (PowerCon '02)*, volume 1, pages 375–379 vol.1, 2002.

[7] Amazon. Amazon web services, 2013. http://aws.amazon.com.

[8] S. M. Amin. U.S. electrical grid gets less reliable. *IEEE Spectrum*, page 80, January 2011.

[9] P. K. Andersen, O. Borgan, R. D. Gill, and N. Keiding. *Statistical Models Based on Counting Processes*. Springer, 1995.

[10] R. N. Anderson. Building the energy internet. *Economist*, March 11th 2004.

[11] R. N. Anderson, A. Boulanger, W. B. Powell, and W. Scott. Adaptive stochastic control for the smart grid. *Proceedings of the IEEE*, 99(6):1098–1115, June 2011.

[12] Andover Continuum single source solution for HVAC and electronic access control. http://www.schneider-electric.com/products/ww/en/1200-building-management-system /1210-building-management-systems/6823-andover-continuum/, 2014.

[13] ANSI/IEEE. *Standard Glossary of Software Engineering Terminology*. ANSI/IEEE, 1991.

[14] ANSI/ISO/ASQ. *Q9001-2000 Quality Management Systems–Requirements*. ANSI/ASQ, 2001.

[15] Apache. Apache ActiveMQ, 2013. http://activemq.apache.org.

[16] Apache. Apache Hadoop, 2013. http://hadoop.apache.org.

[17] J. Arlat, M. Aguera, L. Amat, Y. Crouzet, J.-C. Fabre, J.-C. Laprie, E. Martins, and D. Powell. Fault injection for dependability validation: a methodology and some applications. *Software Engineering, IEEE Transactions on*, 16(2):166 –182, feb 1990.

[18] M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, I. Stoica, and M. Zaharia. A view of cloud computing. *Communications of the ACM*, 53(4):50–58, April 2010.

[19] M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. H. Katz, A. Konwinski, G. Lee, D. A. Patterson, A. Rabkin, I. Stoica, and M. Zaharia. Above the clouds: A berkeley view of cloud computing. Technical Report UCB/EECS-2009-28, Electrical Engineering and Computer Sciences, University of California at Berkeley, 2009.

[20] J. Armstrong, R. Virding, C. Wikström, and M. Williams. *Concurrent Programming in ERLANG, Second Edition*. Prentice Hall, 1993.

[21] AvMet. An approach for identifying weather events in the national airspace system with similar meteorological characteristics and/or similar impacts on air traffic. Technical report, AvMet Applications, 2011.

[22] D. P. Ballou and H. L. Pazer. Modeling Data and Process Quality in Multi-Input, Multi-Output Information Systems. *Management Science*, 31(2):150–162, 1985.

[23] S. Bapat, W. Leal, T. Kwon, P. Wei, and A. Arora. Chowkidar: Reliable and scalable health monitoring for wireless sensor network testbeds. *ACM Transactions on Autonomous and Adaptive Systems*, 4(1):3:1–3:32, February 2009.

[24] S. Barnum, S. Sastry, and J. A. Stankovic. Roundtable: Reliability of embedded and cyber-physical systems. *IEEE Security & Privacy*, 8(5):27 –32, sept.-oct. 2010.

[25] H. Becker and M. Arias. Real-time ranking with concept drift using expert advice. In *Proceedings of the 13th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD)*, pages 86–94, New York, NY, USA, 2007. ACM.

[26] T. Bicer, D. Chiu, and G. Agrawal. A framework for data-intensive computing with cloud bursting. In *2011 IEEE International Conference on Cluster Computing (CLUSTER)*, pages 169–177, 2011.

[27] A. Bondi. Characteristics of scalability and their impact on performance. In *Proceedings of the 2nd international workshop on Software and performance*, WOSP '00, pages 195–203, New York, NY, USA, 2000. ACM.

[28] A. Bose. Models and techniques for the reliability analysis of the smart grid. In *2010 IEEE Power and Energy Society General Meeting,*, pages 1–5, 2010.

[29] J. Bowring, A. Orso, and M. J. Harrold. Monitoring deployed software using software tomography. In *Proceedings of the 2002 ACM SIGPLAN-SIGSOFT workshop on Program analysis for software tools and engineering*, PASTE '02, pages 2–9, New York, NY, USA, 2002. ACM.

[30] J. S. Bradbury, J. R. Cordy, and J. Dingel. Mutation operators for concurrent java (j2se 5.0). In *Proceedings of the Second Workshop on Mutation Analysis (Mutation '06)*, page 11. IEEE Computer Society, 2006.

[31] A. P. Bradley. The use of the area under the ROC curve in the evaluation of machine learning algorithms. *Pattern Recognition*, 30(7):1145–1159, July 1997.

[32] R. E. Bryant. Data-intensive scalable computing for scientific applications. *Computing in Science Engineering*, 13(6):25–33, 2011.

[33] Bugzilla. http://www.bugzilla.org, 2011.

[34] Building Automation Software Siemens APOGEE. http://w3.usa.siemens.com/buildingtechnologies/us/en/building-automation-and-energy-management /apogee/pages/apogee.aspx, 2014.

[35] Building Management Johnson Controls. http://www.johnsoncontrols.com/content/us/en/products/building_efficiency/products-and-systems/building_management.html, 2014.

[36] M. Cannataro, D. Talia, and P. K. Srimani. Parallel data intensive computing in scientific and commercial applications. *Parallel Computing - Parallel data-intensive algorithms and applications*, 28(5):673–704, May 2002.

[37] J. V. Carreira, D. Costa, and J. G. Silva. Fault injection spot-checks computer system dependability. *Spectrum, IEEE*, 36(8):50 –55, aug 1999.

[38] R. Carver. Mutation-based testing of concurrent programs. In *Proceedings of the International Test Conference*, pages 845–853, 1993.

[39] V. Chandola, A. Banerjee, and V. Kumar. Anomaly detection: A survey. *ACM ACM Computing Surveys*, 41(3):15:1–15:58, July 2009.

[40] S. Chandra and P. Chen. The impact of recovery mechanisms on the likelihood of saving corrupted state. In *Proceedings of the 13th International Symposium on Software Reliability Engineering*, ISSRE '02, pages 91–, Washington, DC, USA, 2002. IEEE Computer Society.

[41] S. Chaudhuri and V. Narasayya. Self-tuning database systems: a decade of progress. In *Proceedings of the 33rd international conference on Very large data bases*, VLDB '07, pages 3–14. VLDB Endowment, 2007.

[42] M. Chen, A. X. Zheng, J. Lloyd, M. I. Jordan, and E. Brewer. Failure diagnosis using decision trees. In *Proceedings of the 1st International Conference on Autonomic Computing (ICAC)*, pages 36–43, New York, NY, USA, 2004. IEEE Computer Society.

[43] Y. Chen, A. M. Gillespie, M. W. Monaghan, M. J. Sampson, and R. F. Hodson. On component reliability and system reliability for space missions. In *2012 IEEE International Reliability Physics Symposium (IRPS)*, pages 4B.2.1–4B.2.8, 2012.

[44] S.-E. Choi and E. C. Lewis. A study of common pitfalls in simple multi-threaded programs. In *Proceedings of the 31st SIGCSE Technical Symposium on Computer Science Education*. ACM, 2000.

[45] Y. B. Choi, K. E. Capitan, J. S. Krause, and M. M. Streeper. Challenges associated with privacy in health care industry: Implementation of hipaa and the security rules. *Journal of Medical Systems*, 30(1):57–64, February 2006.

[46] E. M. Clarke, B. Krogh, A. Platzer, and R. Rajkumar. Analysis and verification challenges for cyber-physical transportation systems. In *National Workshop for Research on High-confidence Transportation Cyber-Physical Systems: Automotive, Aviation & Rail*, Washington, DC, November 2008.

[47] Con Edison. Smart grid initiative, 2013. http://www.coned.com/publicissues/smartgrid.asp.

[48] D. Coomans and D.L. Massart. Alternative k-nearest neighbour rules in supervised pattern recognition : Part 1. k-nearest neighbour classification by using alternative voting rules. *Analytica Chimica Acta*, 136(0):15–27, 1982.

[49] B. F. Cooper, A. Silberstein, E. Tam, R. Ramakrishnan, and R. Sears. Benchmarking cloud serving systems with ycsb. In *Proceedings of the 1st ACM symposium on Cloud computing*, SoCC '10, pages 143–154, New York, NY, USA, 2010. ACM.

[50] P. Cornwell, W. Overman, and C. Ross. Extent of recovery from neonatal damage to the cortical visual system in cats. *J Comp Physiol Psychol.*, pages 92(2):255–70, April 1978.

[51] C. Cortes and V. Vapnik. Support-vector networks. In *Machine Learning*, page 20. Springer, 1995.

[52] T. Cover and P. Hart. Nearest neighbor pattern classification. *IEEE Transactions on Information Theory*, 13(1):21–27, 1967.

[53] T. M. Cover and J. A. Thomas. *Elements of Information Theory*. Wiley, 1991.

[54] CPS Steering Group. Cyber-physical systems executive summary. In *CPS Summit 2008*, March 2008. http://varma.ece.cmu.edu/Summit/.

[55] M.A. Cuguero, M. Christodoulou, J. Quevedo, V. Puig, D. Garcia, and M.P. Michaelides. Combining contaminant event diagnosis with data validation/reconstruction: Application to smart buildings. In *2014 22nd Mediterranean Conference of Control and Automation (MED)*, pages 293–298, June 2014.

[56] S. Dawson-Haggerty, X. Jiang, G. Tolle, J. Ortiz, and D. Culler. sMAP – a simple measurement and actuation profile for physical information. In *Proceedings of the 8th ACM Conference on Embedded Networked Sensor Systems (SenSys'10)*, November 2010.

[57] I. M. de Visser, L. Yuan, and G. Nagappan. Understanding failure severity in new product development processes of consumer electronics products. In *Management of Innovation and Technology, 2006 IEEE International Conference on*, volume 2, pages 571–575, 2006.

[58] H. M. Deitel. *An introduction to operating systems*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2nd edition, 1990.

[59] M. Delamaro, M. Pezzé, A. M. R. Vincenzi, and J. C. Maldonado. Mutant operators for testing concurrent java programs. In *XV Simpósio Brasileiro de Engenharia de Software*, pages 272 – 285, Rio de Janeiro, RJ, Brasil, 2001.

[60] R. A. DeMillo, D. S. Guindi, W. M. McCracken, A. J. Offutt, and K. N. King. An extended overview of the mothra software testing environment. In *Proceedings of the Second Workshop on Software Testing, Verification, and Analysis*, pages 142–151, 1988.

[61] R. A. DeMillo, R. J. Lipton, and F. G. Sayward. Hints on test data selection: Help for the practicing programmer. In *Computer*, pages 11(4):34–41, 1978.

[62] E. Deza and M. M. Deza. *Encyclopedia of Distances*. Springer Berlin Heidelberg, 2009.

[63] L. Doherty and D. A. Teasdale. Towards 100% reliability in wireless monitoring networks. In *Proceedings of the 3rd ACM international workshop on Performance evaluation of wireless ad hoc, sensor and ubiquitous networks*, PE-WASUN '06, pages 132–135, New York, NY, USA, 2006. ACM.

[64] A. D. Dominguez-Garcia. Reliability modeling of cyber-physical electric power systems: A system-theoretic framework. In *2012 IEEE Power and Energy Society General Meeting*, pages 1–5, 2012.

[65] B. Dong, C. Cao, and S. E. Lee. Applying support vector machines to predict building energy consumption in tropical region. *Energy and Buildings*, 37, 2005.

[66] H. Drucker, C. J. C. Burges, L. Kaufman, A J. Smola, and V N. Vapnik. Support vector regression machines. *Advances in Neural Information Processing Systems 9, NIPS 1996*, pages 155–161, 1997.

[67] H. Dutta, D. Waltz, A. Moschitti, D. Pighin, P. Gross, C. Monteleoni, A. Salleb-Aouissi, A. Boulanger, M. Pooleery, and R. Anderson. Estimating the time between failures of electrical feeders in the new york power grid. In *Next Generation Data Mining Summit : Dealing with the Energy Crisis, Greenhouse Emissions, and Transportation Challenges (NGDM)*, 2009.

[68] D. J. Eck. Chat, 2006. http://math.hws.edu/eck/cs124/s06/lab11/index.html.

[69] Eclipse. Eclipse.org, 2010. http://www.eclipse.org.

[70] S. Elbaum and M. Hardojo. An empirical study of profiling strategies for released software and their impact on testing activities. In *Proceedings of the 2004 ACM SIGSOFT international symposium on Software testing and analysis*, ISSTA '04, pages 65–75, New York, NY, USA, 2004. ACM.

[71] Electrotek Concepts, Inc. PQView: example PQView charts - RMS variation analysis, 2013. http://www.pqview.com.

[72] S. Ertekin, C. Rudin, and T. McCormick. Predicting power failures with reactive point processes. In *Proceedings of AAAI Late Breaking Track*, 2013.

[73] Y. Eytani and S. Ur. Compiling a benchmark of documented multi-threaded bugs. In *Proceedings of the 18th International Parallel and Distributed Processing Symposium (IPDPS '04)*, page 266, 2004.

[74] E. Farchi, Y. Nir, and S. Ur. Concurrent bug patterns and how to test them. In *Proceedings of the 17th International Symposium on Parallel and Distributed Processing*. IEEE Computer Society, 2003.

[75] T. Fawcett. An introduction to roc analysis. *Pattern Recognition Letters*, 27:861–874, 2006.

[76] A. Faza, S. Sedigh, and B. McMillin. Integrated cyber-physical fault injection for reliability analysis of the smart grid. In Erwin Schoitsch, editor, *Computer Safety, Reliability, and Security*, volume 6351 of *Lecture Notes in Computer Science*, pages 277–290. Springer Berlin / Heidelberg, 2010.

[77] American Society for Quality. http://asq.org/learn-about-quality/process-analysis-tools/overview/fmea.html, 2013.

[78] American Society for Quality. http://asq.org/learn-about-quality/quality-assurance-quality-control/overview/overview.html, 2013.

[79] Federal Smart Grid Task Force. Smart grid basics, 2010. http://www.smartgrid.gov/basics.

[80] U.S.-Canada Power System Outage Task Force. Interim report: Causes of the august 14th blackout in the united states and canada, 2003.

[81] Y. Freund, R. D. Iyer, R. E. Schapire, and Y. Singery. An efficient boosting algorithm for combining preferences. In *Proceedings of the 15th International Conference on Machine Learning (ICML)*, 1998.

[82] B. Furht and A. Escalante. *Handbook of Data Intensive Computing*. Springer, 2011.

[83] A. O. Garcia, S. Bourov, A. Hammad, J. Van Wezel, B. Neumair, A. Streit, V. Hartmann, T. Jejkal, P. Neuberger, and R. Stotzka. The large scale data facility: Data intensive computing for scientific experiments. In *2011 IEEE International Symposium on Parallel and Distributed Processing Workshops and Phd Forum (IPDPSW)*, pages 1467–1474, 2011.

[84] O. Gaudoin, B. Yang, and M. Xie. A simple goodness-of-fit test for the power-law process, based on the duane plot. *IEEE Transactions on Reliability*, 52(1):69–74, March 2003.

[85] M. Gegick, P. Rotella, and T. Xie. Identifying security bug reports via text mining: An industrial case study. In *Proceedings of the 7th IEEE Working Conference on Mining Software Repositories (MSR)*, pages 11–20, Cape Town, May 2010.

[86] S. Ghosh. Towards measurement of testability of concurrent object-oriented programs using fault insertion: a preliminary investigation. In *Proceedings of the Second IEEE International Workshop on Source Code Analysis and Manipulation*, pages 17–25, 2002.

[87] A. Gilat. *MATLAB: An Introduction with Applications 2nd Edition*. John Wiley & Sons., July 2004.

[88] Diana Göhringer, Lukas Meder, Oliver Oey, and Jürgen Becker. Reliable and adaptive network-on-chip architectures for cyber physical systems. *ACM Transactions on Embedded Computing Systems (TECS)*, 12(1s):51:1–51:21, March 2013.

[89] Google. Google app engine, 2013. http://cloud.google.com/appengine.

[90] P. Gross, A. Salleb-Aouissi, H. Dutta, and A. Boulanger. Susceptibility ranking of electrical feeders: A case study. Technical Report CCLS-08-04, Center for Computational Learning Systems, Columbia University, 2008.

[91] P. Gross, A. Salleb-Aouissi, H. Dutta, and A. Boulanger. Ranking electrical feeders of the new york power grid. In *Proceedings of the International Conference on Machine Learning and Applications (ICMLA)*, pages 725–730, 2009.

[92] BACnet Advocacy Group. BACnet: Official website of ASHRAE SSPC 135, 2015. http://www.bacnet.org.

[93] A. Gupta, C. Forgy, A. Newell, and R. Wedig. Parallel algorithms and architectures for rule-based systems. *ACM SIGARCH Computer Architecture News - Special Issue: Proceedings of the 13th annual international symposium on Computer architecture (ISCA '86)*, 14(2):28–37, May 1986.

[94] A. E. Hassan and T. Xie. Software intelligence: Future of mining software engineering data. In *Proceedings of the FSE/SDP Workshop on the Future of Software Engineering Research (FoSER 2010)*, pages 161–166, Santa Fe, NM, November 2010.

[95] W. He, X. Liu, L. Zheng, and H. Yang. Reliability calculus: A theoretical framework to analyze communication reliability. In *2010 IEEE 30th International Conference on Distributed Computing Systems (ICDCS)*, pages 159–168, June 2010.

[96] H. Herodotou, H. Lim, G. Luo, N. Borisov, L. Dong, F. B. Cetin, and S. Babu. Starfish: A self-tuning system for big data analytics. In *In 5th Biennial Conference on Innovative Data Systems Research (CIDR '11)*, pages 261–272, 2011.

[97] R. M. Hierons and M. P. Wiper. Estimation of failure rate using random and partition testing. *Software Testing, Verification & Reliability*, 7(3):153–164, 1997.

[98] M. D. Hill. What is scalability? *SIGARCH Computer Architecture News*, 18(4):18–21, December 1990.

[99] R. H. Howell, W. J. Coad, and H. J. Sauser, Jr. *Principles of Heating, Ventilating and Air Conditioning*. American Society of Heating, Refrigerating and Air-Conditioning Engineers, 7th edition, 2013.

[100] IBM. Autonomic computing, 2011. http://www.research.ibm.com/autonomic/.

[101] IBM. Smarter buildings, 2013. http://www.ibm.com/smarterplanet/us/en/green_buildings/nextsteps/index.html.

[102] IBM. WebSphere MQ, 2013. http://www.ibm.com/software/integration/wmq/.

[103] J. G. Ibrahim, M.-H. Chen, and D. Sinha. *Bayesian Survival Analysis*. Springer, 2001.

[104] D. Irwin, S. Barker, A. Mishra, P. Shenoy, A. Wu, and J. Albrecht. Exploiting home automation protocols for load monitoring in smart buildings. In *Proceedings of the Third ACM Workshop on Embedded Sensing Systems for Energy-Efficiency in Buildings*, BuildSys '11, pages 7–12, New York, NY, USA, 2011. ACM.

[105] ISO. *ISO 8402:1994 Quality management and quality assurance – Vocabulary*. ISO, 1994.

[106] IST. IST dependability benchmarking project - DBench, 2013. http://www.laas.fr/DBench/.

[107] A. Jacobs. The pathologies of big data. *Communications of the ACM*, 52(8):36–44, August 2009.

[108] V. Jagannath, M. Gligoric, S. Lauterburg, D. Marinov, and G. Agha. Mutation operators for actor systems. In *Third International Conference on Software Testing, Verification, and Validation Workshops (ICSTW)*, pages 157–162, 2010.

[109] N. Japkowicz. Why question machine learning evaluation methods (an illustrative review of the shortcomings of current methods). In *2006 AAAI Workshop Evaluation Method for Machine Learning*. AAAI, 2006.

[110] Y. Jia and M. Harman. Constructing subtle faults using higher order mutation testing. In *Proceedings of the Eighth IEEE International Working Conference on Source Code Analysis and Manipulation*, pages 249–258, 2008.

[111] Y. Jia and M. Harman. An analysis and survey of the development of mutation testing. In *IEEE Transactions of Software Engineering*, 2010.

[112] J. Jones and J. Hayes. Estimation of system reliability using a "non-constant failure rate" model. *IEEE Transactions on Reliability*, 50(3):286–288, September 2001.

[113] J. V. Jones. *Integrated Logistics Support Handbook, 2nd Edition*. McGraw-Hill Professional, 1998.

[114] JRP. Miasma, 2010. http://rsb. info.nih.gov/miasma/Miasma.java.

[115] K. Julisch. Clustering intrusion detection alarms to support root cause analysis. *ACM Transactions on Information and System Security*, 6(4):443–471, November 2003.

[116] B. K. Kahn, D. M. Strong, and R. Y. Wang. Information quality benchmarks: product and service performance. *Commun. ACM*, 45(4):184–192, April 2002.

[117] G. Kaiser. Autonomizing legacy systems. In *2002 IBM Almaden Institute Symposium on Autonomic Computing*, April 2001.

[118] J. Kalbfleisch and R. Prentice. *The Statistical Analysis of Failure Time Data*. Wiley-Interscience, 2002.

[119] A. Kamilaris and A. Pitsillides. Using request queues for enhancing the performance of operations in smart homes. In *Proceedings of the 7th ACM workshop on Performance monitoring and measurement of heterogeneous wireless and wired networks*, PM2HW2N '12, pages 1–6, New York, NY, USA, 2012. ACM.

[120] K. Kanoun and L. Spainhower. *Dependability Benchmarking for Computer Systems*. Wiley-IEEE Press, 2008.

[121] W. Karwowski. *International Encyclopedia of Ergonomics and Human Factors*, volume 2. CRC Press, Boca Raton, FL, USA, 2nd edition, 2006.

[122] J. O. Kephart and D. M. Chess. The vision of autonomic computing. *Computer*, 36(1):41–50, 2003.

[123] Linux Kernel. http://bugzilla.kernel.org/, 2011.

[124] J. Kleissl and Y. Agarwal. Cyber-physical energy systems: focus on smart buildings. In *Proceedings of the 47th Design Automation Conference*, DAC '10, pages 749–754, New York, NY, USA, 2010. ACM.

[125] P. Kocher, R. Lee, G. McGraw, and A. Raghunathan. Security as a new dimension in embedded system design. In *Proceedings of the 41st annual Design Automation Conference*, DAC '04, pages 753–760, New York, NY, USA, 2004. ACM.

[126] S. Kubal, J. May, and G. Hughes. Building a system failure rate estimator by identifying component failure rates. In *Proceedings of the International Symposium on Software Reliability Engineering (ISSRE)*, 1996.

[127] H. J. La and S. D. Kim. A service-based approach to designing cyber physical systems. In *Proceedings of the 2010 IEEE/ACIS 9th International Conference on Computer and Information Science*, ICIS '10, pages 895–900, Washington, DC, USA, 2010. IEEE Computer Society.

[128] J. Lafferty and C. Zhai. Document language models, query models, and risk minimization for information retrieval. In *Proceedings of the 24th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 111–119, 2001.

[129] A. Lakhina, M. Crovella, and C. Diot. Diagnosing network-wide traffic anomalies. In *Proceedings of ACM SIGCOMM 2004*, pages 219–230, August 2004.

[130] A. Lakhina, M. Crovella, and C. Diot. Exploring the subspace method for network-wide anomaly diagnosis. In *Proceedings of the ACM SIGCOMM workshop on Network troubleshooting: research, theory and operations practice meet malfunctioning reality*, NetT '04, pages 319–319, New York, NY, USA, 2004. ACM.

[131] P. Langley. Machine learning as an experimental science. *Machine Learning*, 3(1):5–8, 1988.

[132] J-C. Laprie. Dependable computing and fault tolerance: Concepts and teminology. In *Fault-Tolerant Computing, 1995, ' Highlights from Twenty-Five Years'., Twenty-Fifth International Symposium on*, page 2, jun 1995.

[133] A. Lazarevic, L. Ertoz, V. Kumar, A. Ozgur, and J. Srivastava. A comparative study of anomaly detection schemes in network intrusion detection. In *In Proceedings of SIAM Conference on Data Mining*, 2003.

[134] Edward A. Lee. Cyber physical systems: Design challenges. In *International Symposium on Object/Component/Service-Oriented Real-Time Distributed Computing (ISORC)*, May 2008. Invited Paper.

[135] F. Leonardi, A. Pinto, and L. P. Carloni. Synthesis of distributed execution platforms for cyber-physical systems with applications to high-performance buildings. In *2011 IEEE/ACM International Conference on Cyber-Physical Systems (ICCPS)*, pages 215–224, 2011.

[136] P. Li, U. Kruger, and G. W. Irwin. Identification of dynamic systems under closed-loop control. *International Journal of Systems Science*, 37(3):181–195, February 2006.

[137] X. Li. Building thermal response modeling. Technical report, Center for Computational Learning Systems, Columbia University, 2012.

[138] LIBSVM - A Library for Support Vector Machines. http://www.csie.ntu.edu.tw/ cjlin/libsvm/, 2014.

[139] B. Littlewood and J. L. Verrall. A bayesian reliability model with a stochastically monotone failure rate. *IEEE Transactions on Reliability*, R-23(2):108–114, 1974.

[140] Smart Buildings LLC. What is a smart building?, 2013. http://www.smart-buildings.com.

[141] B. Long, R. Duke, D. Goldson, P. Strooper, and L. Wildman. Mutation-based exploration of a method for verifying concurrent java components. In *Proceedings of the 18th International Parallel and Distributed Processing Symposium (IPDPS '04)*, 2004.

[142] B. Long and P. Strooper. A classification of concurrency failures in java components. In *Proceedings of the International Parallel and Distributed Processing Symposium (IPDPS '03)*, page 8, 2003.

[143] P. M. Long and R. A. Servedio. Martingale boosting. In *Eighteenth Annual Conference on Computational Learning Theory (COLT)*, pages 79–94, 2005.

[144] R. H. C. Lopes, I. Reid, and P. R. Hobson. The two-dimensional kolmogorov-smirnov test. In *XI International Workshop on Advanced Computing and Analysis Techniques in Physics Research*, April 2007.

[145] S. Lu, S. Park, E. Seo, and Y. Zhou. Learning from mistakes: a comprehensive study on real world concurrency bug characteristics. In *Proceedings of the 13th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS'08)*. ACM, 2008.

[146] J. P. Magalhães and L. M. Silva. Root-cause analysis of performance anomalies in web-based applications. In *Proceedings of the 2011 ACM Symposium on Applied Computing*, SAC '11, pages 209–216, New York, NY, USA, 2011. ACM.

[147] C. D. Manning, P. Raghavan, and H. Schütze. *Introduction to Information Retrieval*. Cambridge University Press, 2008.

[148] T. Martinussen and T. H. Scheike. *Dynamic Regression Models for Survival Data*. Springer, 2006.

[149] Z. Matic and V. Sruk. The physics-of-failure approach in reliability engineering. In *30th International Conference on Information Technology Interfaces (ITI '08)*, pages 745–750, 2008.

[150] S. McIntosh, J. O. Kephart, J. Lenchner, B. Yang, M. Feridun, M. Nidd, A. Tanner, and I. Barabasi. Semi-automated data center hotspot diagnosis. In *Proceedings of the 7th International Conference on Network and Services Management*, CNSM '11, pages 260–266, Laxenburg, Austria, Austria, 2011. International Federation for Information Processing.

[151] P. Mell and T. Grance. *The NIST Definition of Cloud Computing*. National Institute of Standards and Technology (NIST), Gaithersburg, MD, September 2011.

[152] J. Mi. Bathtub failure rate and upside-down bathtub mean residual life. *IEEE Transactions on Reliability*, 44(3):388–391, 1995.

[153] Microsoft. Asynchronous agents library. http://msdn.microsoft.com/en-us/library/dd492627(VS.100).aspx.

[154] Microsoft. Windows azure, 2013. http://www.windowsazure.com.

[155] A. M. Middleton. *Handbook of Cloud Computing*. Springer, 2010.

[156] E. F. Miller, Jr. Getting quality methods into practice. *ACM Computing Surveys (CSUR) - Special issue: position statements on strategic directions in computing research*, 28(4es), December 1996.

[157] R. Mitchell and I. Chen. Effect of intrusion detection and response on reliability of cyber physical systems. *IEEE Transactions on Reliability*, 62(1):199–210, 2013.

[158] R. Moore, T. A. Prince, and M. Ellisman. Data-intensive computing and digital libraries. *Communications of the ACM*, 41(11):56–62, November 1998.

[159] C. Moretti, B. Hoang, K. Hollingsworth, B. Rich, P. Flynn, and D. Thain. All-pairs: An abstraction for data-intensive computing on campus grids. *IEEE Transactions on Parallel and Distributed Systems*, 21(1):33–46, 2010.

[160] G. S. Mudholkar and D. K. Srivastava. Exponentiated weibull family for analyzing bathtub failure-rate data. *IEEE Transactions on Reliability*, 42(2):299–302, 1993.

[161] C. Murphy, G. E. Kaiser, L. Hu, and L. Wu. Properties of machine learning applications for use in metamorphic testing. In *Proceedings of the 20th International Conference on Software Engineering and Knowledge Engineering (SEKE)*, pages 867–872, July 2008.

[162] National Oceanic and Atmospheric Administration. National climate data center, 2012.

[163] National Science Foundation. NSF/Intel partnership on cyber-physical systems security and privacy (CPS-Security), 2014. http://www.nsf.gov/pubs/2014/nsf14571/nsf14571.htm.

[164] NEC. Smart buildings, 2013. http://www.nec.com/en/global/environment/energy/ building.html.

[165] NERC. *Reliability Considerations from the Integration of Smart Grid*. NERC, December 2010.

[166] NITRD. Winning the future with science and technology for 21st century smart systems. Technical report, Networking and Information Technology Research and Development (NITRD) Program, 2010.

[167] A. J. Offutt. Investigations of the software testing coupling effect. In *ACM Transactions on Software Engineering and Methodology*, pages 1(1):5–20, 1992.

[168] A. J. Offutt, A. Lee, G. Rothermel, R. H. Untch, and C. Zapf. An experimental determination of sufficient mutant operators. In *ACM Transactions on Software Engineering and Methodology*, pages 5(2):99–118, 1996.

[169] A. J. Offutt and R. H. Untch. Mutation 2000: uniting the orthogonal. In *Mutation Testing for the New Century*, pages 34–44, 2001.

[170] J. E. Olson. *Data Quality: The Accuracy Dimension*. Morgan Kaufmann, 1st edition, 2002.

[171] OpenStack. OpenStack open source cloud computing software, 2013. http://www.openstack.org.

[172] Oracle. Java SE technologies – database, 2013. http://www.oracle.com/technetwork/java/javase/jdbc/index.html.

[173] Oracle. The Java Virtual Machine specification, 2013. http://docs.oracle.com/javase/specs/jvms/se7/html/.

[174] J. Parekh, G. Kaiser, P. Gross, and G. Valetto. Retrofitting autonomic capabilities onto legacy systems. *Journal of Cluster Computing*, 9(2):141–159, April 2006.

[175] R. Passonneau, C. Rudin, A. Radeva, and Z. A. Liu. Reducing noise in labels and features for a real world dataset: Application of NLP corpus annotation methods. In *Proceedings of the 10th International Conference on Computational Linguistics and Intelligent Text Processing (CICLing)*, 2009.

[176] R. Passonneau, C. Rudin, A. Radeva, A. Tomar, and B. Xie. Treatment effect of repairs to an electrical grid: Leveraging a machine learned model of structure vulnerability. In *Proceedings of the KDD Workshop on Data Mining Applications in Sustainability (SustKDD), 17th Annual ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, 2011.

[177] R. J. Passonneau, A. Tomar, S. Sarkar, H. Dutta, and A. Radeva. Multivariate assessment of a repair program for a new york city electrical grid. In *11th International Conference on Machine Learning and Applications (ICMLA)*, volume 2, pages 509–514, 2012.

[178] M. G. Pecht and F. R. Nash. Predicting the reliability of electronic equipment. *Proceedings of the IEEE*, 82(7), July 1994.

[179] J. Perez, C. Germain-Renaud, B. Kégl, and C. Loomis. Responsive elastic computing. In *Proceedings of the 6th international conference industry session on Grids meets autonomic computing*, GMAC '09, pages 55–64, New York, NY, USA, 2009. ACM.

[180] J. L. Peterson. Petri nets. *ACM Computing Surveys*, 9(3):223–252, September 1977.

[181] L. L. Pipino, Y. W. Lee, and R. Y. Wang. Data quality assessment. *Communications of ACM*, 45(4):211–218, April 2002.

[182] D. Pokrajac, A. Lazarevic, and L. J. Latecki. Incremental local outlier detection for data streams. In *Proceedings of IEEE Symposium on Computational Intelligence and Data Mining (CIDM'07)*, pages 504–515, 2007.

[183] D. Pokrajac, N. Reljin, N. Pejcic, and A. Lazarevic. Incremental connectivity-based outlier factor algorithm. In *Proceedings of the 2008 International Conference on Visions of Computer Science: BCS International Academic Conference*, VoCS'08, pages 211–223, Swinton, UK, UK, 2008. British Computer Society.

[184] M. Polo, M. Piattini, , and I. García-Rodríguez. Decreasing the cost of mutation testing with second-order mutants. In *Software Testing, Verification and Reliability*, pages 19(2):111–131, 2009.

[185] Apache Project. Bugzilla, 2011. http://issues.apache.org/bugzilla/.

[186] Debian Project. Debian bug report logs – #615000, 2011. http://bugs.debian.org/cgi-bin/bugreport.cgi?bug=615000.

[187] Rackspace. Rackspace open cloud, 2013. http://www.rackspace.com/cloud/.

[188] C. E. Rasmussen and C. K. I. Williams. *Gaussian Processes for Machine Learning*. MIT Press, 2006.

[189] M. Rausand and A. Høyland. *System Reliability Theory: Models, Statistical Methods, and Applications*. Wiley, 2nd edition, December 2003.

[190] S. E. Rigdon and A. P. Basu. Estimating the intensity function of a Weibull process at the current time: Failure truncated case. In *Journal of Statistical Computation and Simulation (JSCS)*, volume 30, pages 17–38, 1988.

[191] I. Rish, M. Brodie, and S. Ma. Accuracy vs. efficiency trade-offs in probabilistic diagnosis. In *Eighteenth National Conference on Artificial Intelligence*, pages 560–566, Menlo Park, CA, USA, 2002. American Association for Artificial Intelligence.

[192] D. Robitaille. *The Corrective Action Handbook*. Paton Press, 2002.

[193] J. Romero-Mariona, H. Ziv, D. J. Richardson, and D. Bystritsky. Towards usable cyber security requirements. In *Proceedings of the 5th Annual Workshop on Cyber Security and Information Intelligence Research: Cyber Security and Information Intelligence Challenges and Strategies*, CSIIRW '09, pages 64:1–64:4, New York, NY, USA, 2009. ACM.

[194] Rsync. Rsync, 2013. http://rsync.samba.org.

[195] C. Rudin, R. Passonneau, A. Radeva, S. Ierome, and D. Isaac. 21st-century data miners meet 19th-century electrical cables. *IEEE Computer*, 44(6):103–105, June 2011.

[196] C. Rudin, R J. Passonneau, A. Radeva, H. Dutta, S. Ierome, and D. Isaac. A process for predicting manhole events in manhattan. *Machine Learning*, 80(1):1–31, 2010.

[197] C. Rudin, D. Waltz, R. N. Anderson, A. Boulanger, A. Salleb-Aouissi, M. Chow, H. Dutta, P. Gross, B. Huang, S. Ierome, D. Isaac, A. Kressner, R. J. Passonneau, A. Radeva, and L. Wu. Machine learning for the new york city power grid. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, May 2011.

[198] G. Salton, A. Wong, and C. S. Yang. A vector space model for automatic indexing. *Communications of the ACM*, 18(11):613–620, 1975.

[199] T. Sanislav and L. Miclea. An agent-oriented approach for cyber-physical system with dependability features. In *2012 IEEE International Conference on Automation Quality and Testing Robotics (AQTR)*, pages 356–361, 2012.

[200] A. Savvides, I. Paschalidis, and M. Caramanis. Cyber-physical systems for next generation intelligent buildings. *SIGBED Review*, 8(2):35–38, June 2011.

[201] J. Schein and S. T. Bushby. A hierarchical rule-based fault detection and diagnostic method for HVAC systems. *HVAC&R Research*, 12(1), January 2006.

[202] L. Schor, P. Sommer, and R. Wattenhofer. Towards a zero-configuration wireless sensor network architecture for smart buildings. In *Proceedings of the First ACM Workshop on Embedded Sensing Systems for Energy-Efficiency in Buildings*, BuildSys '09, pages 31–36, New York, NY, USA, 2009. ACM.

[203] L. Sha and J. Meseguer. Design of complex cyber physical systems with formalized architectural patterns. In Martin Wirsing, Jean-Pierre Banâtre, Matthias Hölzl, and Axel Rauschmayer, editors, *Software-Intensive Systems and New Computing Paradigms*, pages 92–100. Springer-Verlag, Berlin, Heidelberg, 2008.

[204] F. T. Sheldon and K. Jerath. Assessing the effect of failure severity, coincident failures and usage-profiles on the reliability of embedded control systems. In *Proceedings of the 2004 ACM symposium on Applied computing*, SAC '04, pages 826–833, New York, NY, USA, 2004. ACM.

[205] H. P. Simao, H. B. Jeong, B. Defourny, W. B. Powell, A. Boulanger, A. Gagneja, L. Wu, and R. N. Anderson. A robust solution to the load curtailment problem. *IEEE Transactions on Smart Grid*, PP(99):1–11, 2013.

[206] Simulation Research Group, Lawrence Berkeley National Laboratory, University of California. *Overview of DOE-2.2*. University of California, June 1998.

[207] C. Singh and A. Sprintson. Reliability assurance of cyber-physical power systems. In *2010 IEEE Power and Energy Society General Meeting*, pages 1–6, july 2010.

[208] R. Smith and B. Keeter. *FRACAS; Failure Reporting, Analysis, Corrective Action System*. Reliabilityweb.com Press, December 2010.

[209] D. Solomon, R. Winter, A. Boulanger, R. Anderson, and L. Wu. Forecasting energy demand in large commercial buildings using support vector machine regression. Technical Report CUCS-040-11, Department of Computer Science, Columbia University, September 2011.

[210] D. H. Stamatis. *Failure Mode Effect Analysis: FMEA From Theory to Execution*. ASQ Quality Press, 2nd edition, 2003.

[211] R. G. D. Steel and J. H. Torrie. *Principles and Procedures of Statistics*. McGraw-Hill, 1960.

[212] D. G. Sullivan. Using probabilistic reasoning to automate software tuning. Technical report, Harvard University, September 2003.

[213] P. M. Swamidass. MAPE (mean absolute percentage error). In *Encyclopedia of Production and Manufacturing Management*, pages 462–462. Springer US, 2000.

[214] Debian Bug Tracking System. http://www.debian.org/bugs, 2011.

[215] A. S. Szalay, G. Bell, J. VandenBerg, A. Wonders, R. Burns, F. Dan, J. Heasley, T. Hey, M. Nieto-Santisteban, A. Thakar, C. van Ingen, and R. Wilton. Graywulf: Scalable clustered architecture for data intensive computing. In *42nd Hawaii International Conference on System Sciences (HICSS '09)*, pages 1–10, 2009.

[216] J. Sztipanovits. Composition of cyber-physical systems. In *14th Annual IEEE International Conference and Workshops on the Engineering of Computer-Based Systems (ECBS '07)*, pages 3–6, March 2007.

[217] S. C. Tan, K. M. Ting, and T. F. Liu. Fast anomaly detection for streaming data. In *Proceedings of the Twenty-Second international joint conference on Artificial Intelligence*, volume 2 of *IJCAI'11*, pages 1511–1516. AAAI Press, 2011.

[218] TechTarget. Search data management, 2011. http://www.searchdatamanagement.com.

[219] TPC. *TPC Benchmark C, Standard Specification*. Transaction Processing Performance Council, 5.11 edition, February 2010.

[220] E. Tufte. *Beautiful Evidence*. Graphics Press, 2006.

[221] Ubuntu. Ubuntu server, 2013. http://www.ubuntu.com/server.

[222] U.S. Department of Energy. Energy efficiency & renewable energy: Building technologies program, 2012. http://www.eere.energy.gov/buildings/.

[223] U.S. Department of Health & Human Services. Health information privacy, 2013. http://www.hhs.gov/ocr/privacy/.

[224] U.S. Energy Information Administration. Annual energy review 2010, October 2011.

[225] V. N. Vapnik. *The nature of statistical learning theory*. Springer-Verlag, New York, 1995.

[226] A. Vaseashta and S. Vaseashta. A survey of sensor network security. *Sensors and Transducers*, 94(7):91–102, July 2008.

[227] VMware. RabbitMQ, 2013. http://www.rabbitmq.com.

[228] J. P. Walters, Z. Liang, W. Shi, and V. Chaudhary. Wireless sensor network security: A survey, in book chapter of security. In *in Distributed, Grid, and Pervasive Computing, Yang Xiao (Eds*, pages 0–849. CRC Press, 2007.

[229] Y. Wand and R. Y. Wang. Anchoring data quality dimensions in ontological foundations. *Communications of the ACM*, 39(11):86–95, November 1996.

[230] P. Wang, Y. Xiang, and B. R. Dai. A reliable workflow for cyber-physical system components substitution. *International Conference on Computer and Information Technology*, 0:885–891, 2012.

[231] R. Y. Wang, H. B. Kon, and S. E. Madnick. Data quality requirements analysis and modeling. In *Proceedings of the Ninth International Conference on Data Engineering*, pages 670–677, Washington, DC, USA, 1993. IEEE Computer Society.

[232] R. Y. Wang and D. M. Strong. Beyond accuracy: what data quality means to data consumers. *Journal of Management Information Systems*, 12(4):5–33, March 1996.

[233] X. Wang, L. Zhang, T. Xie, J. Anvik, and J. Sun. An approach to detecting duplicate bug reports using natural language and execution information. In *Proceedings of the 30th International Conference on Software Engineering (ICSE)*, pages 461–470. ACM Press, 2008.

[234] Weather Underground. Weather underground: Weather forecast & reports, 2012. http://www.wunderground.com.

[235] Webopedia. Private cloud, 2013. http://www.webopedia.com.

[236] W. Weibull. A statistical distribution function of wide applicability. *ASME Journal of Applied Mechanics*, pages 293–297, September 1951.

[237] I. H. Witten, E. Frank, L. Trigg, M. Hall, G. Holmes, and S. J. Cunningham. Weka: Practical machine learning tools and techniques with java implementations. In *Proceedings of the ICONIP/ANZIIS/ANNES'99 Workshop on Emerging Knowledge Engineering and Connectionist-Based Information Systems*, pages 192–196, 1999.

[238] L. Wu and G. Kaiser. Empirical study of concurrency mutation operators for Java. Technical Report CUCS-041-10, Department of Computer Science, Columbia University, 2010.

[239] L. Wu and G. Kaiser. Constructing subtle concurrency bugs using synchronization-centric second-order mutation operators. In *Proceedings of the 23th International Conference on Software Engineering and Knowledge Engineering (SEKE)*, July 2011.

[240] L. Wu and G. Kaiser. An autonomic reliability improvement system for cyber-physical systems. In *Proceedings of the IEEE 14th International Symposium on High-Assurance Systems Engineering (HASE)*, October 2012.

[241] L. Wu and G. Kaiser. FARE: A framework for benchmarking reliability of cyber-physical systems. In *Proceedings of the Ninth Annual IEEE Long Island Systems, Applications and Technology Conference (LISAT'13)*, May 2013.

[242] L. Wu, G. Kaiser, C. Rudin, and R. Anderson. Data quality assurance and performance measurement of data mining for preventive maintenance of power grid. In *Proceedings of the 17th ACM SIGKDD Workshop on Data Mining for Service and Maintenance*, August 2011.

[243] L. Wu, G. Kaiser, C. Rudin, D. Waltz, R. Anderson, A. Boulanger, A. Salleb-Aouissi, H. Dutta, and M. Pooleery. Evaluating machine learning for improving power grid reliability. In *ICML 2011 Workshop on Machine Learning for Global Challenges*, July 2011.

[244] L. Wu, G. Kaiser, D. Solomon, R. Winter, A. Boulanger, and R. Anderson. Improving efficiency and reliability of building systems using machine learning and automated online evaluation. In *Proceedings of the Eighth Annual IEEE Long Island Systems, Applications and Technology Conference (LISAT'12)*, May 2012.

[245] L. Wu, T. Teräväinen, G. Kaiser, R. Anderson, A. Boulanger, and C. Rudin. Estimation of system reliability using a semiparametric model. In *Proceedings of the IEEE EnergyTech 2011 (EnergyTech'11)*, May 2011.

[246] L. Wu, B. Xie, G. Kaiser, and R. Passonneau. BugMiner: Software reliability analysis via data mining of bug reports. In *Proceedings of the 23th International Conference on Software Engineering and Knowledge Engineering (SEKE)*, July 2011.

[247] B. Xie, R. J. Passonneau, H. Dutta, J. Y. Miaw, A. Radeva, A. Tomar, and C. Rudin. Progressive clustering with learned seeds: An event categorization system for power grid. In *Proceedings of the 24th International Conference on Software Engineering and Knowledge Engineering (SEKE)*, July 2012.

[248] T. Xie, S. Thummalapenta, D. Lo, and C. Liu. Data mining for software engineering. *Computer*, 42(8):55–62, 2009.

[249] L. Yang, C. Liu, J. M. Schopf, and I. Foster. Anomaly detection and diagnosis in grid environments. In *Proceedings of the 2007 ACM/IEEE conference on Supercomputing*, SC '07, pages 33:1–33:9, New York, NY, USA, 2007. ACM.

[250] Y. Zhao, Y. Wang, H. Ren, and M. Ding. Design of a data acquisition system for building electrical fault diagnosis. In *2015 27th Chinese Control and Decision Conference (CCDC)*, pages 3355–3359, May 2015.

Appendix A

Reliability Estimation Using a Semiparametric Model

How to accurately and effectively evaluate system reliability has been a long-time research challenge. One commonly used indictor for system reliability is *failure rate*, which is the frequency with which an engineered system or component fails. To estimate the failure rate, historical failure information and/or testing of a current sample of equipment are commonly used as the basis of the estimation. After these data have been collected, a failure distribution model, *i.e.*, a cumulative distribution function that describes the probability of failure up to and including time t, is assumed (*e.g.*, the exponential failure distribution or more generally, the Weibull distribution) and used to estimate the failure rate.

Our experimental results indicate that using an exponential or Weibull distribution prior may not be as effective for power grid failure modeling as a particular semiparametric model introduced in this work. This semiparametric model does not assume a constant or monotonic failure rate pattern as the other models do. We introduce Gaussian smoothing that further helps the semiparametric model to closely resemble the true failure rate. We applied this method to power network component failure data and compared its blind-test estimation results with the subsequent real failures. We also compared it with other models during these experiments. In all of these cases, the semiparametric model outperformed the other models.

A.1 Background on Reliability Analysis

The failure rate can be defined as the total number of failures within an item population, divided by the total time expended by that population, during a particular measurement interval under stated conditions [139]. We use $\lambda(t)$ to denote the failure rate at time t, and $R(t)$ to denote the reliability function (or survival function), which is the probability of no failure before time t. Then the failure rate is:

$$\lambda(t) = \frac{R(t) - R(t + \Delta t)}{\Delta t \cdot R(t)}.$$

As Δt tends to zero, the above λ becomes the instantaneous failure rate, which is also

called hazard function (or hazard rate) $h(t)$:

$$h(t) = \lim_{\Delta t \to 0} \frac{R(t) - R(t + \Delta t)}{\Delta t \cdot R(t)}.$$

A failure distribution $F(t)$ is a cumulative failure distribution function that describes the probability of failure up to and including time t:

$$F(t) = 1 - R(t), t \geq 0.$$

For system with a continuous failure rate, $F(t)$ is the integral of the failure density function $f(t)$:

$$F(t) = \int_0^t f(x)\,\mathrm{d}x.$$

Then the hazard function becomes

$$h(t) = \frac{f(t)}{R(t)}.$$

A.1.1 Weibull and Exponential Failure Distribution

For the Weibull failure distribution, the failure density function $f(t)$ and cumulative failure distribution function $F(t)$ are

$$f(t; \lambda, k) = \begin{cases} \frac{k}{\lambda}(\frac{t}{\lambda})^{k-1} e^{-(t/\lambda)^k}, & t \geq 0 \\ 0, & t < 0 \end{cases}$$

$$F(t; \lambda, k) = \begin{cases} 1 - e^{-(t/\lambda)^k}, & t \geq 0 \\ 0, & t < 0 \end{cases}$$

where $k > 0$ is the shape parameter and $\lambda > 0$ is the scale parameter of the distribution. The hazard function when $t \geq 0$ can be derived as

$$h(t; \lambda, k) = \frac{f(t; \lambda, k)}{R(t; \lambda, k)} = \frac{f(t; \lambda, k)}{1 - F(t; \lambda, k)} = \frac{k}{\lambda}\left(\frac{t}{\lambda}\right)^{k-1}.$$

A value of $k < 1$ indicates that the failure rate decreases over time. A value of $k = 1$ indicates that the failure rate is constant (*i.e.*, k/λ) over time. In this case, the Weibull distribution becomes an exponential distribution. A value of $k > 1$ indicates that the failure rate increases with time.

A.2 Semiparametric Model with Gaussian Smoothing

We consider the semiparametric estimation of the longitudinal effect of a blip treatment (*i.e.*, a single "all-or-nothing" treatment occurring at a precisely recorded time) on a system with recurring events (*e.g.*, immediately-recoverable failures in a mechanical/electronic system). The estimand is the effect of the most recent blip treatment on the future arrival rate. The

method assumes that the effect of treatment is to scale the underlying rate, and is thus an extension of Cox regression with internal covariates, using the Gaussian process to provide much-needed smoothing.

Although the method applies to any blip treatment, we focus on estimating the effect of an event (failure) on future failures. For example, an association of an event with an immediate increase in failure rate provides a finely-detailed explanation for "infant mortality" which can be compared with parametric models such as the Weibull.

A.2.1 Probability and Regression Model

We assume each of N units is under observation for some interval of time $[0, T]$. The method can be easily adapted to allow for units with missing observation periods (known in advance). Let $\mathbb{T}$ denote the (finite) set of times at which an event occurs. The unit to fail at time t (if any) is denoted as $i(t)$; ties are broken in preprocessing, if necessary, by randomly selecting tied units and shifting their failures by one second. For any unit j under observation at time t denote by $\tau_{t,i}$ the time of the treatment (which is here the time of previous outage). It turns out to be important to remove "unobserved" units (*i.e.* those for which $t - \tau_{t,i}$ is unknown due to left-truncation of the study); thus, the index-set of fully-observed units at time t is given by $\Re(t)$, and commonly called the "risk set." Note that if the mechanism for observation is independent of the treatment and failure processes (*i.e.*, if it is fixed in advance), this does not introduce bias [9]. We consider the non-parametric rate model as follows:

$$\lambda(t; i) = \lambda_0(t)\psi(t - \tau_{t,i});$$
$$\psi(\cdot) = e^{\phi(\cdot)},$$

that is, 20 seconds after treatment the effect will be to make failure $\psi(20) = e^{\phi(20)}$ times more likely.

The full likelihood is then [9]:

$$l(\lambda_0(\cdot), \psi(\cdot)) = \left(\prod_{t\in\mathbb{T}} \lambda_0(t)\psi(t - \tau_{t,i(t)})\right) \times e^{-\int_0^T \sum_{j\in\Re(t)} \lambda_0(t)\psi(t-\tau_{t,j})\mathrm{d}t}.$$

The estimation proceeds in two steps, detailed in Appendix A–2. The λ_0 term is first shown to be estimated as 0 at all times $t \notin \mathbb{T}$. Thus, conditioning on the failure times, the λ_0 term is cancelled out (since it affects all units equally). This allows convenient estimation of $\psi(t) = e^{\phi(t)}$. After the estimation of $\psi(t)$, the λ_0 term may be estimated by a weighted non-parametric estimator (which uses the estimate of ψ). For simplicity, in this paper we fit the λ_0 as a constant (within each network) by using the method of moments (Appendix A–3).

Since only the time since last treatment is tracked, it is implicitly assumed that any prior treatments are immediately "forgotten" by the system upon administration of a new treatment.

The connection between the hazard λ and the distribution function is detailed in Appendix A–1.

The information reduction induced by the Cox framework should be very useful, especially in the Gaussian process setup which scales as $O(p^3)$ in the number of predictors. To achieve further reduction of data for numerical stability and to expedite cross-validation, we "bin" values of $t - \tau_{t,\cdot}$ (which can be viewed as the predictors of $\phi(t - \tau_{t,\cdot})$) into percentiles.

A.2.2 Application

The method is applied to the failure rate of distribution power feeders in three boroughs of New York City (Manhattan, Queens, and Brooklyn). Distribution feeders are the power cables that feed intermediate voltage power in distribution grids. In New York City, underground distribution feeders, mostly 27KV or 13KV, are one of the most failure-prone electrical components in the power grid. The effect of infant mortality and the changing hazard rate are of interest for maintenance scheduling applications.

In our application, $N = 81$ and there are $|\mathbb{T}| = T = 667$ distinct failure times (*i.e.*, 667 total failures are observed among the 81 units).

A.2.3 Preliminary Fit

The model predictions without smoothing are provided in Figure A.1, which shows the failure rate versus time since treatment, and they are clearly overfitted to the data. Since events occur rarely, we have that some $(t - \tau_{t,i})$-bins may be observed only once, associated with a failure, causing a direct estimate of $\psi(\cdot)$ to overestimate. Likewise, many bins will be associated only with the non-failed risk set, and $\psi(\cdot)$ will go to 0. This effect will be more pronounced with a large number of units and rare failures.

A.2.4 Gaussian Process

We apply a Gaussian process prior to the values $\phi(t)$ with a radial basis function. After the standard marginalizing of the prior [188] onto $t \in \mathbb{T}$, the $\phi(t)$ are normally distributed with mean 0 and covariance matrix K with

$$K_{t,t'} = ae^{-(t-t')^2/b}.$$

This marginal prior distribution will be referred to as π. The parameters a, b are the marginal variance and so-called "characteristic time-scale" respectively. We use the parameter values $a = 5$, $b = 1 \cdot 10^3$ based on good performance on the training data. Alternatively, cross-validation on a grid search on these parameters can be used to obtain approximate "point estimates" of a, b.

Details of the fitting process are in Appendix A–4.

Figure A.2 shows the smoothed fit using the Gaussian process prior. It is much better than the unsmoothed fit.

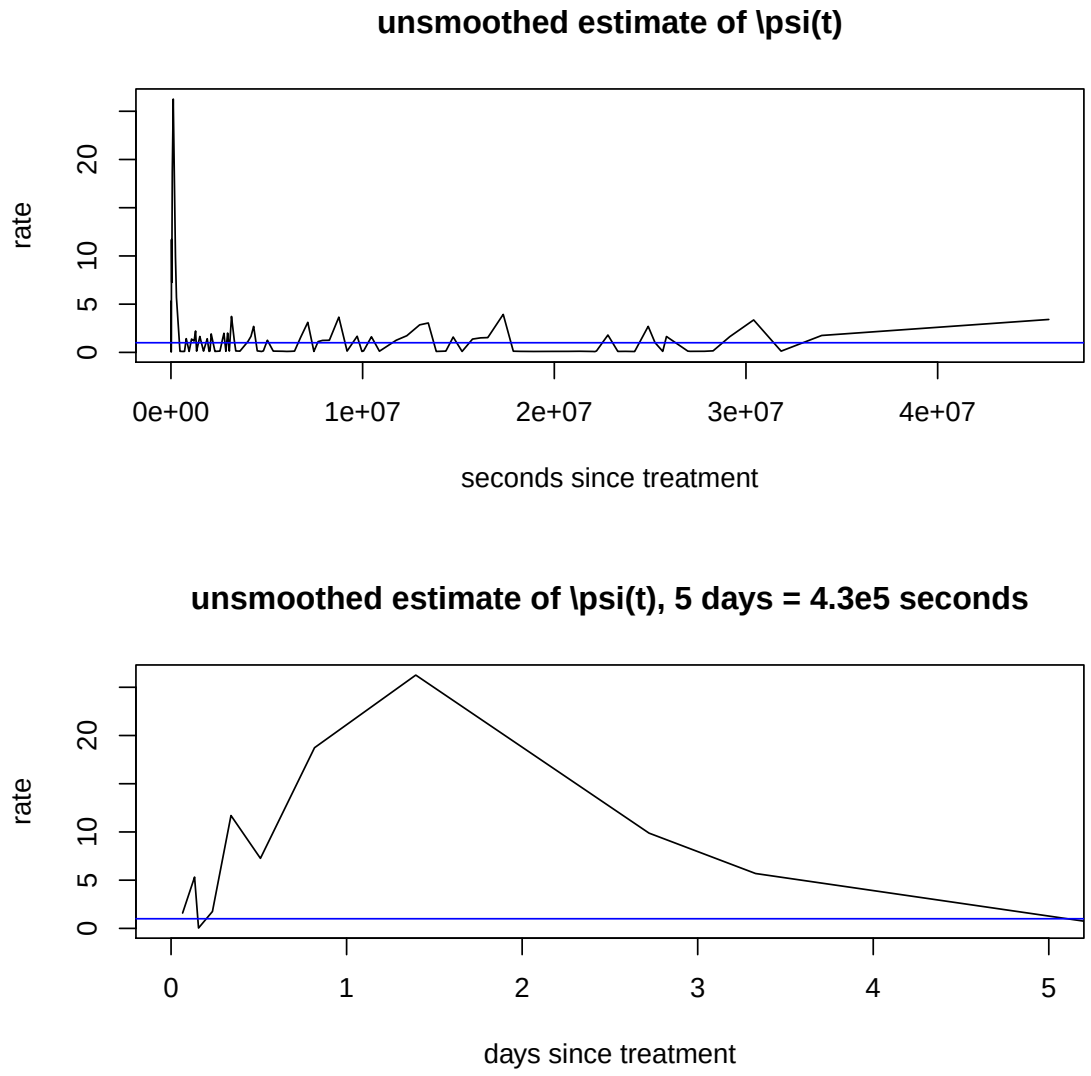


Figure A.1: Preliminary fit.

A.3 Empirical Studies

We implemented the semiparametric model with Gaussian smoothing and applied it to five years of distribution power feeder failure data collected in New York City, as discussed in section A.2.2. We further compared the estimation with what actually happened. We also applied the exponential distribution and Weibull distribution models on the same set of data and compared their results with the results from the semiparametric model.

A.3.1 Experimental Setup

Our experiments consist of three main groups of blind tests. In New York City, the distribution power feeder failures are seasonal. During summer heat waves, more feeder failures are

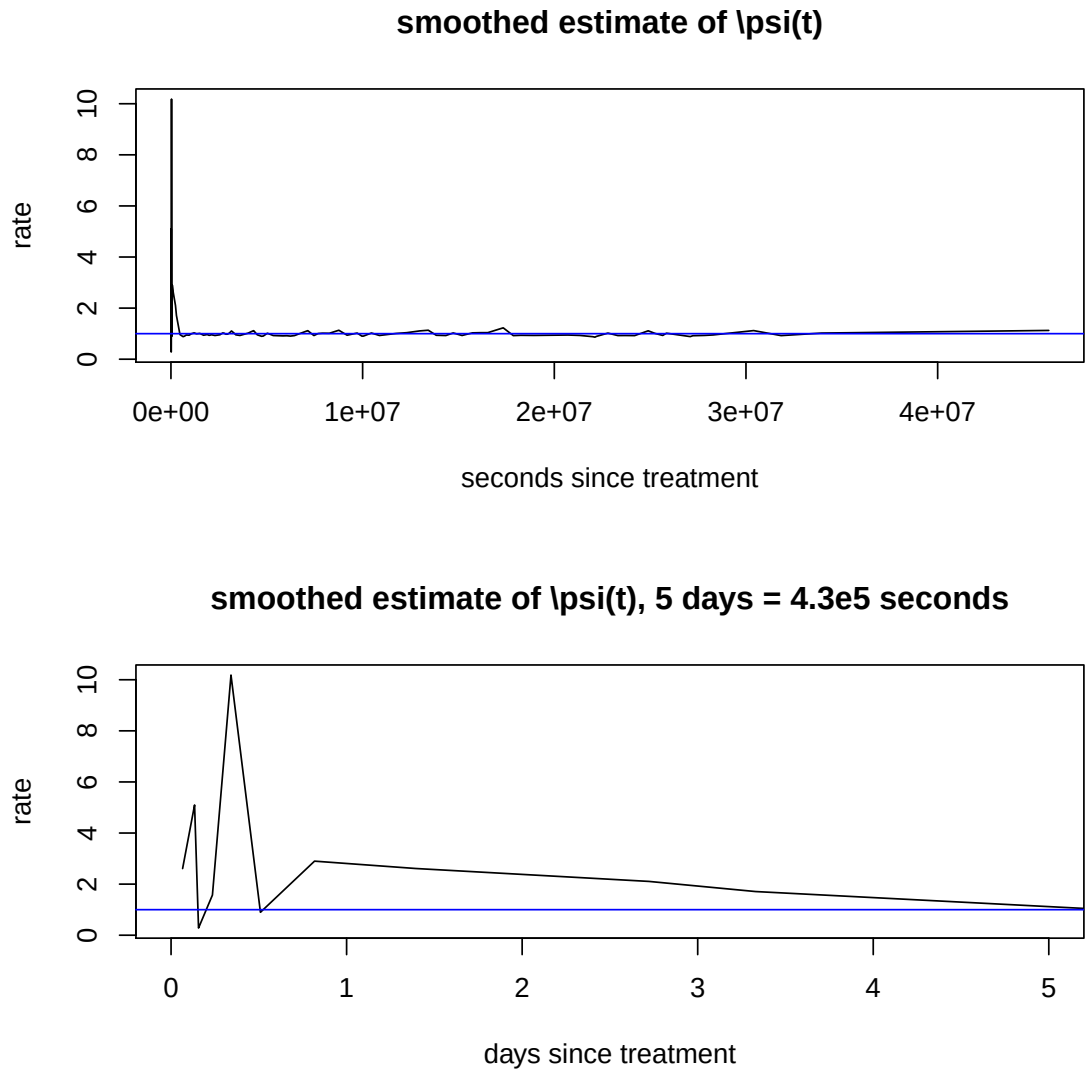


Figure A.2: Smoothed fit.

likely to happen. The three groups are estimates of the failure rate for the summer, winter, and the whole year using the historical data for the first three years, *i.e.*, from year 2006 through 2008. Then we compare these estimates with the actual failure rates measured for the years 2009–2010 using the failure data. We perform similar experiments on the exponential and Weibull models.

A.3.2 Results and Analysis

The results of fitting the model are summarized in Table A.1 (giving the constants) and Figure A.3 (giving the estimated failure rate multiplier $\psi(t)$) for each network.

To analyze the fit of each model, we integrate (numerically for the semiparametric) to convert the hazard estimates to estimates of the cumulative distribution function (see

Network	# of Units	# of Failures	Exponential λ
Queens: 01Q	26	327	75.2
Brooklyn: 01B	29	197	154.12
Manhattan: 02M	26	143	114.1
Network	Weibull k	Weibull λ	Semiparametric λ_0
Queens: 01Q	0.48	42	71.0
Brooklyn: 01B	0.69	120.4	130.0
Manhattan: 02M	0.62	108.0	112.1

Table A.1: Summary of results (units are in days).

Training			
Network	Exponential	Weibull	Semiparametric
Queens: 01Q	0.40	0.19	**0.13**
Brooklyn: 01B	0.25	0.17	**0.14**
Manhattan: 02M	0.27	0.17	**0.12**
Testing			
Network	Exponential	Weibull	Semiparametric
Queens: 01Q	0.35	0.23	**0.20**
Brooklyn: 01B	0.27	0.20	**0.16**
Manhattan: 02M	0.38	**0.31**	0.32

Table A.2: Kolmogorov-Smirnoff test of fit.

section A.2 and Appendix A–1). The resulting model fits are then visually and numerically compared to the empirical distribution function of the data.

The fit of each model is evaluated on the training (2006–08) and test (2009–10) sets using the Kolmogorov-Smirnoff (K-S) statistic [144], which is a distance between the empirical distribution of the cumulative distribution function F, $\hat{F}_{\text{emp}}$, and the F provided by each model fit. Since none of these models are to be considered true, we use the statistic simply as a "measure of fit" on training and holdout data, rather than as a formal hypothesis test. The empirical distribution function is defined as

$$\hat{F}_{\text{emp}}(t) = \frac{1}{T} \sum \mathbb{I}_{t_i < t},$$

with the sum being over all inter-arrival times in the data. The K-S statistic is the maximum absolute discrepancy between the two distributions, defined as

$$\text{KS}(\hat{F}_{\text{emp}}, F) = \sup_t |\hat{F}_{\text{emp}}(t) - F_{\text{model}}(t)|.$$

As expected, the Weibull uniformly performs better than the exponential. Table A.2 shows the K-S test of fit. The semiparametric method uniformly outperforms the Weibull on the training data, and outperforms the Weibull on the holdout test data in Queens and Brooklyn, demonstrating accuracy in prediction. The semiparametric method comes very

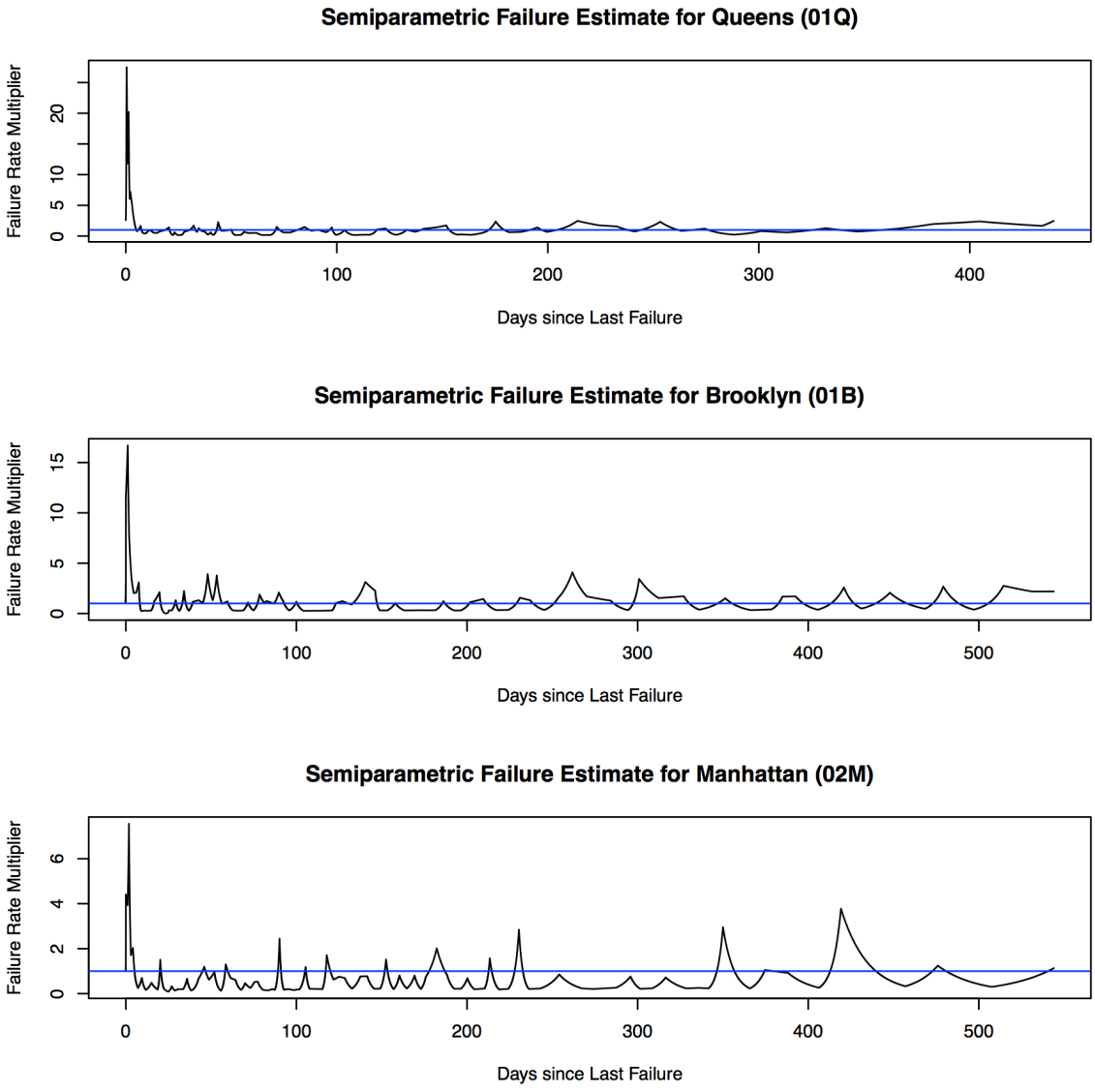


Figure A.3: Semiparametric infant mortality rate estimates.

close to the Weibull in test performance on the Manhattan network which, notably, also exhibits the worst degradation from training to test performance, across all models.

The comparison of the estimation results shows that the failure rate estimates using the semiparametric model are closer to the actual measured inter-arrival times, which means the semiparametric model with Gaussian smoothing is more accurate in estimating the failure rate.

A.4 Related Work

Estimation of system reliability by modeling failure rate has been an active research area. Various estimation models have been proposed for different kinds of systems including

power electrical components, semiconductor chips and boards, and software systems. The exponential, Bayesian, log-normal, and Weibull approaches were popular in prior research. In 1974, Littlewood and Verrall used a Bayesian reliability model to estimate stochastic monotone failure rates [139]. Ibrahim et al. formalized the field of Bayesian survival analysis in 2001 [103]. Rigdon and Basu described a way to estimate the intensity function of a Weibull process [190]. Mudholkar and Srivastava used the exponential Weibull family for analyzing the bathtub failure rate model [160]. In prior sections, we compared our approach with the exponential and Weibull models. Our approach differs from previous Bayesian models in making fewer assumptions on a continuous failure distribution.

Among the failure patterns, the bathtub model and infant mortality are perhaps the most well-studied [152, 160]. To model non-constant failure rates, Jones used a constant failure intensity assumption and exponential failure distribution-based method to do the estimation, and experimented with the method in reliability analysis of digital circuit devices [112]. Our approach does not assume a constant failure rate or a constant failure intensity. The semiparametric model we described is not a modified version of the exponential or Weibull models.

In 1984, Laprie described a mathematical model for the failure behavior of component-based software systems with physical and design faults [132]. Hierons and Wiper researched the estimation of software system failure rate using random and partition testing methods [97]. Kubal et al. proposed a way of estimating software system failure rate based on the failure rates of the underlying components using a Bayesian approach [126]. Although we directly applied our approach to distribution power feeder failures here, our approach can be directly applied to other areas, for instance, software reliability analysis.

A.5 Summary

This appendix presented a new method of estimating failure rate using a semiparametric model with Gaussian process smoothing. The method is able to provide accurate estimation based on historical data and it does not make strong *a priori* assumptions of the failure rate pattern (*e.g.*, constant or monotonic). Our empirical studies of applying such an approach in power system failure data and a comparison of this approach with other existing models show its efficacy and accuracy. This method may also be used in estimating reliability for many other systems, such as software systems or components.

Appendix A–1 Equivalence of Hazard and Distribution Functions

From definition of the hazard function,

$$\lambda(t) = f(t)/(1 - F(t)),$$

and from the definitions of

$$f(t) = -\frac{\partial(1 - F(t))}{\partial t},$$

and finally, from calculus,

$$\frac{\partial \log(f(t))}{\partial t} = \frac{f'(t)}{f(t)}.$$

Therefore:

$$\frac{-\frac{\partial(1-F(t))}{\partial t}}{1-F(t)} = \lambda(t),$$

$$-\frac{\partial \log(1-F(t))}{\partial t} = \lambda(t),$$

$$-\log(1-F(t)) = \int_0^t \lambda(u)\mathrm{d}u,$$

$$1-F(t) = e^{-\int_0^t \lambda(u)\mathrm{d}u},$$

$$F(t) = 1-e^{-\int_0^t \lambda(u)\mathrm{d}u}.$$

Appendix A–2 Marginalizing Times without Failure

We consider the contribution to the likelihood from the observation of no failures between times t_{i-1}, t_i, assuming no censoring and that $\phi(\cdot) < \infty$:

$$L = e^{-\int_{t_{i-1}}^{t_i} \lambda_0(u) \sum_{j\in\Re(u)} e^{\phi(i-\tau_{u,j})}\mathrm{d}u}.$$

Taking the functional derivative of λ_0 at time $s \in (t_{i-1}, t_i)$:

$$\frac{\partial L}{\partial \lambda_0(s)} = \begin{pmatrix} e^{-\int_{t_{i-1}}^{t_i} \lambda_0(u) \sum_{j\in\Re(u)} e^{\phi(i-\tau_{u,j})}\mathrm{d}u} \end{pmatrix} \times \begin{pmatrix} -\lambda_0(s) \sum_j e^{\phi(s-\tau_{s,j})} \end{pmatrix},$$

which is negative for all positive values of $\lambda_0(s)$. Since $\lambda_0 \geq 0$ by definition, the maximum likelihood estimate of baseline hazard is $\hat{\lambda}_0(s) = 0$, which gives the MLE (*i.e.*, Maximum Likelihood Estimation) of failure rate

$$\hat{\lambda}_0(s) \sum_j e^{\phi(s-\tau_{s,j})} = 0.$$

Substituting this into the likelihood, we see that it does not depend on ϕ when there are no failures, reducing the estimation problem to event times. This result, derived more formally [148], is also valid under random censoring, as shown by Cox and given in [9].

Thus, since intervals without failures give no information about ϕ, we can reduce the problem of estimating ϕ to the conditional probability of each *observed unit failing at time t*, given that *some unit failed at time t*, which is:

$$\prod_t \frac{\text{unit } i \text{ fails at } t}{\text{some unit fails at } t}$$

$$= \prod_t \frac{\lambda_0(t) e^{\phi(t-\tau_{t,i})}}{\lambda_0(t) \sum_j e^{\phi(t-\tau_{t,j})}}$$

$$= \prod_t \frac{e^{\phi(t-\tau_{t,i})}}{\sum_j e^{\phi(t-\tau_{t,j})}},$$

which gives the "Cox likelihood" for ϕ at those values $t - \tau_{t,j}$, which are observed.

After the estimate of ϕ is obtained, we can derive an estimate of $\Lambda_0 = \int_0^t \lambda_0$ through the weighted non-parametric Nelson-Aalen estimator [118]. This Λ_0 is smoothed and used directly in computing the test-penalty, or if desired λ_0 may be approximately estimated by differentiating the smoothed version.

Appendix A–3 Fitting λ_0

For simplicity we take the baseline hazard λ_0 to be constant for each network. After estimating ψ, the reliability function is

$$R(t) = e^{-\int_0^t h(t)} = e^{-\lambda_0 \int_0^t \psi(u)\mathrm{d}u},$$

from which the mean time to failure can be computed directly by the so-called layered representation of the expectation (which follows from integration by parts):

$$\mathbb{E}_{\lambda_0}[T] = \int_0^\infty e^{-\lambda_0 \int_0^u \psi(u)\mathrm{d}u} \mathrm{d}t.$$

At this point, the λ_0 is chosen by grid search over numeric approximations of this integral, so that the mean time to failure equals the empirical mean time to failure: $\mathbb{E}_{\lambda_0}[T] = \overline{T}$.

Appendix A–4 Fitting the Gaussian Process

The log-posterior probability is proportional to the sum of the log of the Cox likelihood (l) and the log of the marginalized Gaussian process prior (π):

$$\frac{\partial L}{\partial \lambda_0(s)} = l + \pi = \begin{array}{l} \sum_t \log \phi(t - \tau_{t,i}) - \\ \log \sum_{j \in \Re(t)} \phi(t - \tau_{t,j}) + \\ \left(-\frac{1}{2}\phi^\dagger K^{-1} \phi\right). \end{array}$$

We apply the Newton-Raphson method to find the maximum a-posteriori estimate. The gradient with respect to ϕ is

$$\nabla(l + \pi) = \sum_t \frac{-\psi(t - \tau_{\cdot,t}) + e_{i(t)} s_t}{s_t} + K^{-1}\phi,$$

with Hessian

$$(\nabla\nabla(l+\pi))_{i,j} = \psi(t-\tau_{i,t})\psi(t-\tau_{j,t})/s_t^2 + K^{-1},$$

where

$$s_t = \sum_j \psi(t-\tau_{j,t}),$$

the total hazard of observed units at time t, and $e_{i(t)}$ is the unit basis vector indicating the failed unit at time t, $\delta_{i(t)}$.

The step-size is dynamically adjusted, and is stopped on a relative improvement of the quasi-posterior probability by less than 1.4e − 08.

Appendix B

Software Reliability Analysis Via Bug Mining

Software bugs reported by human users and automatic error reporting software are often stored in some bug tracking tools (*e.g.*, Bugzilla and Debbugs). These accumulated bug reports may contain valuable information that could be used to improve the quality of the bug reporting, reduce the quality assurance effort and cost, analyze software reliability, and predict future bug report trend. In this chapter, I present BUGMINER, a tool that is able to derive useful information from historic bug report database using data mining, use these information to do completion check and redundancy check on a new or given bug report, and to estimate the bug report trend using statistical analysis. My empirical studies of the tool using several real-world bug report repositories show that it is effective, easy to implement, and has relatively high accuracy despite low quality data.

B.1 Introduction

Finding and fixing the faults in software is an indispensable while time-consuming quality assurance task in software development. The definition of *fault* is a programming error that leads to an erroneous result in some programs during execution. A *software bug* is the common term used to describe a fault, error, flaw, mistake, or failure in a program that produces an incorrect or unexpected result, or causes it to behave in unintended ways. When a software bug is identified, it is often reported and recorded into a bug report database using some bug tracking tools so that further analysis or fix can be performed, possibly by a developer or tester. For some real-world software, their bug report databases have accumulated a large amount of historic bug reports. For example, as of February 2011, Debbugs, *i.e.*, Debian bug tracking system, has accumulated 615,000 bug reports [186, 214].

These accumulated bug reports may contain valuable information that could be used to improve the quality of the bug reporting, reduce the cost of quality assurance, analyze software reliability, and predict future bug report trend. One of the challenges in bug reporting is that the bug reports are often incomplete (*e.g.*, missing data fields such as product version or operating system details). Another challenge is that there are often many duplicate bug reports for the same bug. Software developers or testers normally have to

review these redundant bug reports manually, which is time-consuming and cost inefficient.

I developed a tool called BUGMINER that is able to derive useful information from historic bug reports using data mining techniques, including machine learning (*e.g.*, SVM [225, 51]) and natural language processing, and use these information to do completion check through classification and redundancy check through similarity ranking on a new or given bug report. BUGMINER can also perform bug report trend analysis using Weibull distribution [190]. I implemented the tool and experimented it using three real-world bug report repositories including Apache Tomcat [185], Eclipse [69], and Linux Kernel [123]. My experiments demonstrate that it is effective, easy to implement, and has relatively high accuracy despite low quality data.

B.2 Background on Bug Reporting

Bug tracking tools are often developed as a database-driven web application. The web interface allows multiple geographically distributed users to enter the bug reports simultaneously. The backend database stores the records for the reported bugs. Table B.1 lists some main attributes (*i.e.*, data fields or columns) of a typical bug report for Apache Tomcat using Bugzilla [185, 33]. These attributes are meta information of the bug report. The field bug_id is an unique identifier for a distinct bug instance. A bug report is often modified by subsequent reviewers or developers who are trying to verify or fix the bug. Table B.2 lists the additional commentary entries related to the same bug listed in Table B.1. Each new entry (*i.e.*, new long_desc record) records the author name, entry date and time, and the free text description. The entry date and time for the first and last long_desc record, along with the first author's name, are also stored in the main attributes list of the same bug (*i.e.*, creation_ts, delta_ts, and reporter). There is no predefined limit on how many additional commentary entries a bug report can hold. Bug report datasets will be further explained in section B.4.2.

Attribute Name	Sample Value	Attribute Name	Sample Value
bug_id	48892	component	Connectors
creation_ts	2010-03-11 12:10:09	delta_ts	2010-12-14 14:30:22
short_desc	Use URIEncoding...	rep_platform	All
cclist_accessible	1	op_sys	All
classification_id	1	bug_status	NEW
classification	Unclassified	bug_severity	enhancement
product	Tomcat 7	priority	P2
reporter	reporter 1	assigned_to	dev

Table B.1: Main attributes of a bug report

Attribute Name	long_desc 1	long_desc 2	long_desc 3
isprivate	0	0	0
who	reporter 1	reporter 2	reporter 3
bug_when	2010-03-11 12:10:09	2010-04-04 10:18:48	2010-12-14 14:30:22
thetext	Here is a ...	There are ...	For encoding ...

Table B.2: Additional attributes

B.3 Approach

B.3.1 Architecture

Figure B.1 illustrates the architecture of BugMiner. There are two types of bug reporters: human users such as software developers, testers, and end users; automatic error reporting processes that run as a service on users' computers. The bug reporters generate new bug reports and enter the related information via the bug tracking tool's interface. The bug tracking tool then store the new bug report into the bug report database.

BugMiner consists of three data mining and statistical processing engines: automatic completion check engine; automatic redundancy check engine; and bug report trend analysis engine. These three engines process the historic data stored in the bug report database and the new bug report coming in. The results from these engines are then directed to the bug tracking tool so that these results can be reviewed and stored. In the following subsections, I will describe each engine in detail.

B.3.2 Attributes and Feature Selection

BugMiner analyzes bug report data based on two sets of attributes: 1) static meta information, and 2) bag-of-words (*i.e.*, a collection of distinct free text words) attributes. For each bug report in Bugzilla, users need to fill in a predefined set of bug information, as shown in Table B.1. This set of attributes has two characteristics: 1) static: the list of fields is fixed for all types of software products, and those fields are available for all bug reports; 2) meta information: they describe the general information about the bug report but doesn't go to the details of the problem. Bug report analysis based solely on the static and meta information is very limited. In BugMiner, I further include the free text data of a bug report in my analysis.

The free text data usually describes a bug scenario in natural language followed by some sample code. I represent the textual data as a bag-of-words. Each data instance is a high dimensional vector with words being attributes. The values of the attributes are Term Frequency-Inverse Document Frequency (TF-IDF) weight [147], which gives a higher weight on words that are frequent in certain data records but not too common across all records. Stemming, a process of reducing inflected (or sometimes derived) words to their stem or root form, is not necessary because the vocabulary usually doesn't have a variety of morphed forms, and imperfect stemming may bring in additional noisy content unnecessarily. My feature selection also bases on inverse document frequency (IDF) and global term frequency. Words with a low IDF (*e.g.*, stopwords such as 'the' and 'a') are removed because they are too common and lack discriminative power. Words with a very

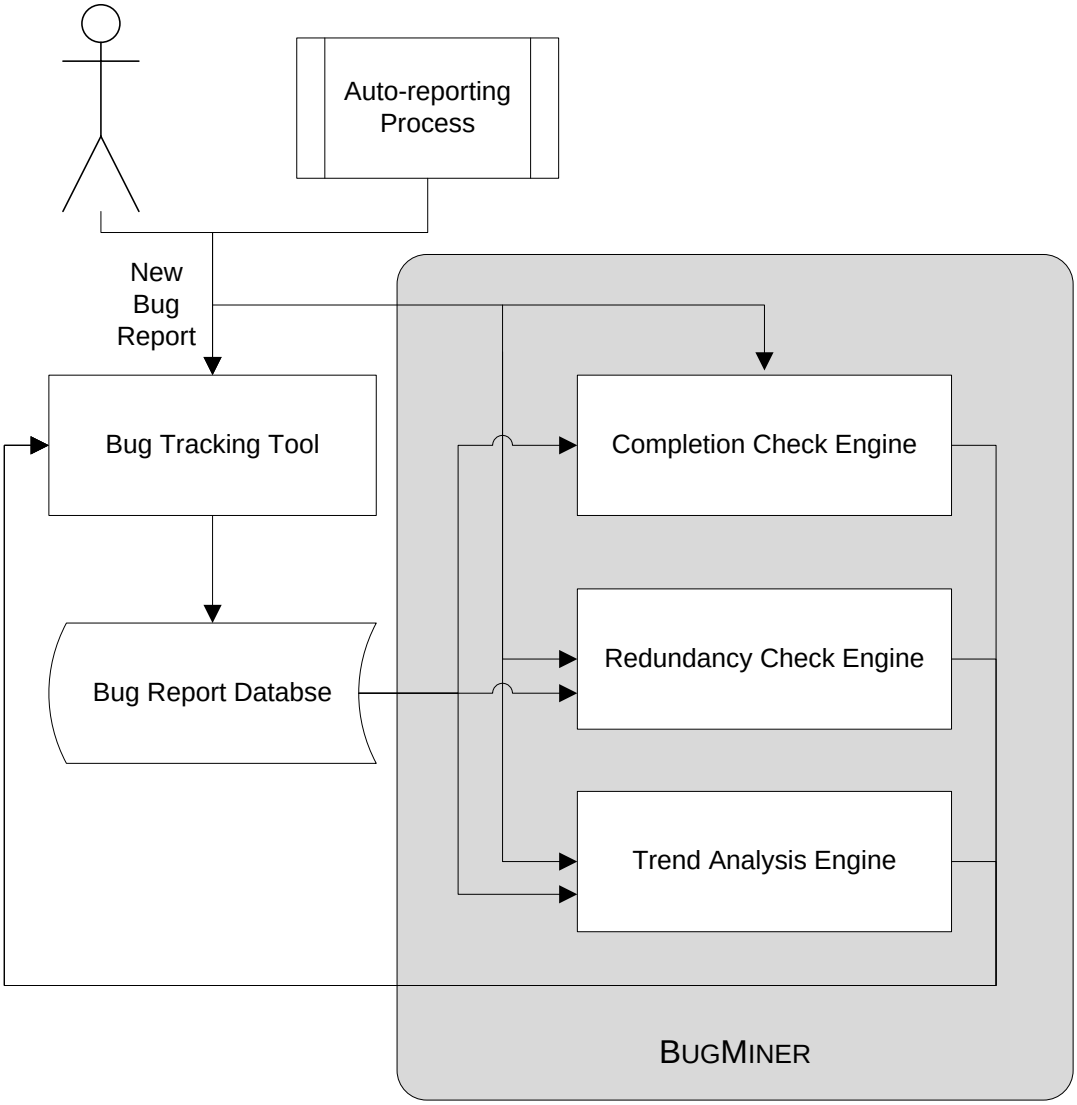


Figure B.1: BUGMINER architecture

low global term frequency are also removed because they are rare and their inclusion leads to a high dimensionality, which may cause "curse of dimensionality" problem in machine learning.

B.3.3 Automatic Completion Check Engine

When a bug report is filed, the bug information submitted are sometimes incomplete (*e.g.*, missing data fields). BUGMINER's automatic completion check engine derives these missing data fields through mining historic data and classification using the partially filled information. It reduces manual effort, keeps the bug report complete, and helps developers and testers to analyze the bug report more precisely.

Classification Using Support Vector Machine

Missing field autocompletion can be solved as a supervised learning problem. By training a classification model on existing data, I can predict the missing values. In BugMiner, I use Support Vector Machines (SVM) as the classifier. SVM is a popular machine learning method because of its high performance. It formulates the classification modeling process as a quadratic minimization problem, and finds hyperplanes in a high dimensional space that separate data instances of different categories, while maximizing the margins between categories.

I first use a set of historic bug reports (*e.g.*, each one with n attributes) as training data to build a linear SVM model. For a new or given bug report with one missing data field a (*i.e.*, $n-1$ attributes filled and 1 attribute missing), I use the trained SVM model as a classifier and the filled $n-1$ attributes to predict the value of the missing a field for this bug report. In the case of multiple data fields are missing for a report (*e.g.*, $n-m$ attributes filled and m attributes missing), I use the SVM model and the $n-m$ filled attributes to predict the missing fields one by one.

B.3.4 Automatic Redundancy Check Engine

A common way of searching a bug report database to find out whether a new or given bug report already exists or not is to use keyword search, which normally uses keyword in combination with some wildcat characters such as '%' and '?' to construct database query string that can be executed on the database table. This kind of search based on keyword matching is often imprecise and may generate a large amount of useless or irrelevant results. The similarity ranking used by BugMiner's automatic redundancy check engine is able to tell whether the new bug report is a duplicate or not more precisely. Furthermore, the similarity ranking can find out the most similar prior bug reports and sort them for the user.

Similarity Ranking Using Cosine Similarity

I represent bug report dataset in a vector space model (*i.e.*, term vector model), an algebraic model for representing text documents as vectors of identifiers, such as index terms [198]. Each bug report is a vector that consists of a list of feature values. As described in section B.3.2, BugMiner uses two sets of features: 1) static meta information; 2) bag-of-words attributes with TF-IDF values.

I measure the similarity between two bug reports based on Cosine similarity, *i.e.*, the Cosine of the angle between the two vectors that represent these two bug reports, as shown in the following formula:

$$Distance_{COS}(a,b) = \frac{\sum_i a_i \times b_i}{\sqrt{\sum_i a_i^2} \times \sqrt{\sum_i b_i^2}},$$

where a and b represent two vectors. Its result equals 1 when the angle between two vectors is 0 (*i.e.*, two vectors are pointing in the same direction), and its result is less than 1 otherwise.

For a new or given bug report, I compute the Cosine similarity value (*i.e.*, *csv*) between this new bug report's vector and all the prior bug reports' vectors, and then rank the *csv* values in an descending order. The historic bug report with the highest *csv* value (*i.e.*, the closest one to 1) is the most similar prior record.

Similarity Ranking Using KL Divergence

In addition to Cosine similarity, I rank all prior bug reports based on their relevance to the new bug report using probability distribution. Kullback-Leibler (*i.e.*, KL) divergence [53, 147] is an effective relevance metric that assumes each data instance in a high dimensional feature space is characterized by a probability distribution. KL divergence measures the dissimilarity between two probability distributions, as shown in the following formula:

$$D_{KL}(a||b) = \sum_{t \in V} P(t|M_a) log \frac{P(t|M_a)}{P(t|M_b)},$$

where M_a and M_b represent the probability distributions for vector a and b respectively. V is the vocabulary of all terms and t is a term in V. KL divergence measures how bad the probability distribution M_a is at modeling M_b. Previous work [128] presents results suggesting that model comparison approach outperforms both query-likelihood and document-likelihood approaches. However, this metric is asymmetric, *i.e.*, $D_{KL}(a||b) \neq D_{KL}(b||a)$. In order to use it as a distance metric, I adopt a symmetrized KL divergence method for similarity ranking, which is defined as:

$$Distance_{KL}(a, b) = \frac{1}{2} D_{KL}(a||b) + \frac{1}{2} D_{KL}(b||a).$$

The result is symmetric and nonnegative. It equals 0 when two distributions are identical. It is bigger than 0 otherwise, and the larger the value the greater their dissimilarity.

For a new or given bug report, I compute the symmetrized KL divergence value (*i.e.*, *kld*) between this new bug report's vector and all the prior bug reports' vectors, and then rank the *kld* values in an ascending order. The historic bug report with the lowest *kld* value (*i.e.*, the closest one to 0) is the most similar prior record.

Is the New Bug Report a Duplicate? What are the Similar Bugs Reported before?

I categorize a new or given bug report into one of the three categories according to the ranked *csv* and *kdl* values, along with the value ranges they fall into:

- If a prior report exists with $csv \geq c\_r_2$ and $kld \leq k\_r_1$, it is highly likely to be a duplicate (or repeat) of a prior report.
- If a prior report exists with $c\_r_1 < csv < c\_r_2$ or $k\_r_1 < kld < k\_r_2$, it has similar prior report.
- If all prior reports have $csv \leq c\_r_1$ and $kld \geq k\_r_2$, it does not have any similar prior report.

The value range parameters (*i.e.*, c_r_1, c_r_2, k_r_1, and k_r_2) can be determined based on heuristics obtained from experiments.

B.3.5 Bug Report Trend Analysis Engine

After major software releases, the number of software bugs tend to increase initially. As these bugs are fixed, the number of bugs gradually decreases, which resembles the "bathtub curve" in reliability engineering. The increase and decrease of the number of bugs normally lead to the similar trend of the number of bug reports. Weibull distribution can be used to model this kind of pattern and provide the basis for trend analysis.

Report Incidence Distribution

For the Weibull distribution, the incidence (*e.g.*, failure or bug report) density function $f(t)$ and cumulative incidence distribution function $F(t)$ are

$$f(t;\lambda,k) = \frac{k}{\lambda}(\frac{t}{\lambda})^{k-1}e^{-(t/\lambda)^k}, \quad t \geq 0,$$

$$F(t;\lambda,k) = 1 - e^{-(t/\lambda)^k}, \quad t \geq 0,$$

where $k > 0$ is the shape parameter and $\lambda > 0$ is the scale parameter of the distribution. The instantaneous incidence rate (or hazard function) when $t \geq 0$ can be derived as

$$h(t;\lambda,k) = \frac{f(t;\lambda,k)}{1 - F(t;\lambda,k)} = \frac{k}{\lambda}\left(\frac{t}{\lambda}\right)^{k-1}.$$

A value of $k < 1$ indicates that the incidence rate decreases over time. A value of $k = 1$ indicates that the incidence rate is constant (*i.e.*, k/λ) over time. In this case, the Weibull distribution becomes an exponential distribution. A value of $k > 1$ indicates that the incidence rate increases with time.

Estimation of Coming Bug Report

I first use historic data to fit the Weibull function and derive the λ and k parameters. Then for any given time t, which is the number of weeks (or other chosen time units such as days or hours) after the starting date, the number of bug reports that may happen during that week can be estimated using the Weibull's density function $f(t)$. The result is an estimate of how many bug reports may happen during the t-th week after the starting event, *e.g.*, a new software release. Similarly, the instantaneous incidence rate can be estimated using the hazard function $h(t)$. These estimates give software developers or testers a baseline for designing the software testing and maintenance plan.

B.4 Empirical Studies

B.4.1 Implementation

I implemented BugMiner in Java using some existing machine learning and statistical analysis tools, including Weka [237] and MATLAB [87].

B.4.2 Bug Report Datasets and Data Processing

I experiment BugMiner on the bug report repositories of three real-world software applications (Apache Tomcat [185], Eclipse [69], and Linux Kernel [123]). Table B.3 lists some statistics of these bug report repositories. For example, the Apache Tomcat dataset contains two product versions—Tomcat 3 and Tomcat 7. The OS is the operating system the software runs on. The components are the functional components of the software.

Software Name	# bug reports	# product	# OS	# components
Apache Tomcat	1525	2	16	16
Eclipse	1674	2	17	13
Linux Kernel	1692	16	1	106

Table B.3: Software and bug report datasets

I first apply pattern matching to extract static meta information, as listed in Table B.1. Then I process free text descriptions using tokenization and bag-of-words feature selection as described in section B.3.2. The dimensionalities of the term feature space range from 4000 to 13,000 depending on the dataset. After the attribute data sources are combined, the final vector space to represent bug report instances includes static meta information and bag-of-words features.

B.4.3 Results and Analysis

My experimental results show that BugMiner is effective in automatic completion check, automatic redundancy check, and bug report trend analysis. The following subsections present the detailed results and analysis.

Classification for Missing Field Autocompletion

For missing field autocompletion, I train classification model on 80% of the data and do blind-test on the remaining 20% of the data. For example, for Apache Tomcat, I use 1220 (or 80%) bug reports as training data and use 305 (or 20%) bug reports as the testing data. Table B.4 lists the classification results for the Tomcat version. The accuracy of the classification on testing instances is 99.02%. This means the automatic completion check engine can determine the product version highly accurately in this case.

TP Rate	FP Rate	Precision	Recall	F-Measure	ROC Area	Class
0.991	0.014	0.996	0.991	0.993	0.989	tomcat 3
0.986	0.009	0.973	0.986	0.98	0.989	tomcat 7
0.99	0.012	0.99	0.99	0.99	0.989	weighted Avg.

Table B.4: Classification results of products

Table B.5 lists the classification results for the operating system version for Tomcat. The accuracy of the classification on testing instances is 68.52%.

TP Rate	FP Rate	Precision	Recall	F-Measure	ROC Area	Class
0.888	0.449	0.758	0.888	0.818	0.719	all
0.356	0.081	0.432	0.356	0.39	0.637	linux
0.087	0.018	0.286	0.087	0.133	0.535	other
0.176	0.014	0.429	0.176	0.25	0.581	solaris
0.786	0.047	0.629	0.786	0.698	0.869	windows xp
0.685	0.294	0.632	0.685	0.647	0.696	weighted Avg.

Table B.5: Classification results of OS versions

Table B.6 lists the classification results for the software component related to the bug report. The accuracy of the classification on testing instances is 53.11%. The results show that it is relatively difficult to accurately determine the problematic component based on the bug reports in this case.

TP Rate	FP Rate	Precision	Recall	F-Measure	ROC Area	Class
0.5	0.007	0.714	0.5	0.588	0.747	auth
0.868	0.067	0.73	0.868	0.793	0.9	catalina
0.2	0.039	0.313	0.2	0.244	0.58	config
0.368	0.037	0.583	0.368	0.452	0.665	connectors
0.5	0.003	0.5	0.5	0.5	0.748	encoding
0.622	0.041	0.676	0.622	0.648	0.79	jasper
0.667	0.003	0.667	0.667	0.667	0.832	manager
0.6	0.154	0.513	0.6	0.553	0.723	servlet
0.1	0.007	0.333	0.1	0.154	0.547	webapps
0.531	0.091	0.535	0.531	0.518	0.72	weighted Avg.

Table B.6: Classification results of components

I also did some experiments on the bug report datasets of Eclipse and Linux Kernel. Table B.7 shows the summary of classification accuracy rates for the datasets tested. As the number of classes increases, the accuracy rate tends to decrease; nevertheless, the accuracy rates (*e.g.*, 53.11% for Tomcat's components) are relatively high if they are compared to the chance baseline (*i.e.*, probability is $1/n$ if there are n possible components).

Software Name	product	OS	components
Apache Tomcat	99.02%	68.52%	53.11%
Eclipse	97.90%	66.47%	67.37%
Linux Kernel	76.33%	N/A	58.88%

Table B.7: Summary of classification accuracy

Similarity Ranking

I first transform the historic training bug reports and the testing bug report to vectors using the vector space model. After the csv and kld value for each training bug report are calculated, all the training bug reports are then sorted in an descending order based on the csv value and in an ascending order based on the kld value. The bug reports at the top of the ranked lists are the most similar ones to the testing bug report.

Based on the heuristics from the experiments, I determine the value range parameters as $c\_r_1 = 0.2, c\_r_2 = 0.9, k\_r_1 = 2.0$, and $k\_r_2 = 10.0$ for Tomcat. Table B.8 lists some sample results for a given bug report #393. From the results, the bug report #393 is highly likely to be a duplicate of some prior reports because there exists historic bug reports with $csv \geq 0.9$ and $kld \leq 2.0$ (*i.e.*, bug report #330 and #296). Furthermore, bug report #228 is likely to be a similar bug report of #393 because it has $0.7 < csv < 0.9$ or $2.0 < kld < 10.0$. To determine whether a new or given bug report is in fact a duplicate usually requires human judgment. My manual verification shows that the similarity ranking results produced by BugMiner are highly accurate despite the low quality data.

bug_id	csv	kld
330	0.928	1.940
296	0.917	0.816
228	0.717	9.868

Table B.8: Similarity ranking results

Trend Analysis

I implement the bug report trend analysis based on the Weibull distribution. I first aggregate the historic data to compute a vector of the time (i-th week) and the number of bug reports whose first reporting date falls in the i-th week. Then a result vector returns the 95% confidence intervals for the estimates of the parameters of the Weibull distribution given the historic vector data. The two-element row vector estimates the Weibull parameter λ and k. The first row of the 2-by-2 matrix contains the lower bounds of the confidence intervals for the parameters, and the second row contains the upper bounds of the confidence intervals.

Table B.9 shows the estimates of the Weibull parameters for Apache Tomcat 3. The value of k is less than 1, which indicates that the incidence rate decreases over time. The related curve fit is illustrated in Figure B.2 and B.3. The starting time, (*i.e.*, the 0 on the x-axis) is the week of August 25, 2000. The curve fit shows that the Weibull distribution closely resembles the actual bug report incidence distribution.

Software	λ	λ_{low}	λ_{high}	k	k_{low}	k_{high}
Tomcat 3	0.3885	0.2280	0.6621	0.2241	0.2041	0.2461
Tomcat 7	6.4941	5.1735	8.1519	1.3077	1.0576	1.6168

Table B.9: Weibull parameter estimates

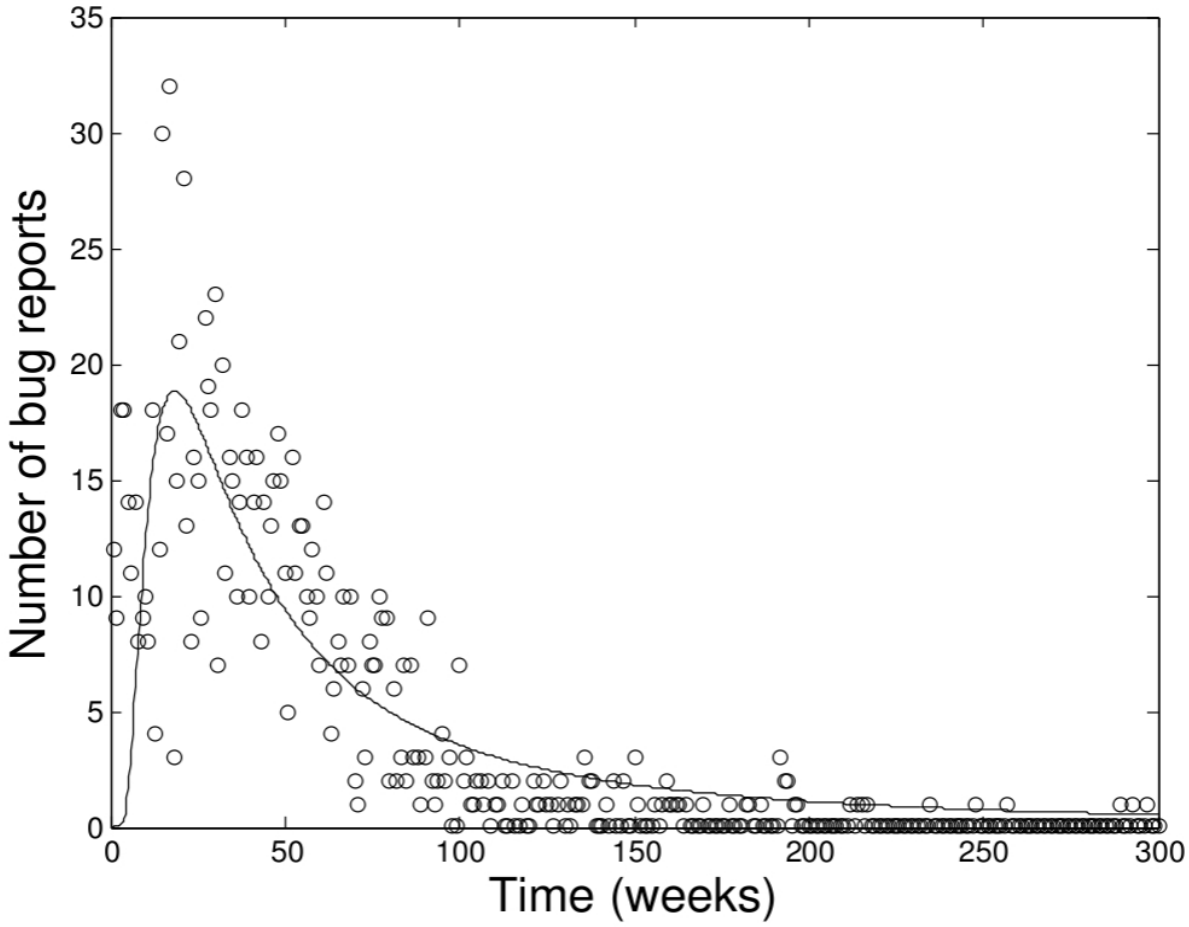


Figure B.2: Weibull fit for Tomcat 3

B.5 Related Work

Some prior studies have been done on applying data mining on software engineering. Hassan and Xie described the concept of software intelligence and the future of mining software engineering data [94]. Xie *et al.* presented a general overview of data mining for software engineering and described an example of duplicate bug detection using vector space-based similarity [248] . Wang *et al.* also described an approach to detect duplicate bug reports using both natural language and execution information [233]. My redundancy check engine uses both probability distribution-based KL divergence and vector space-based Cosine similarity ranking, instead of only vector space-based similarity. Furthermore, my approach provides a similarity ranking list that can be used for search, instead of only Yes and No on duplication check. Gegick *et al.* presented text mining of bug reports to identify security issues [85]. Their work aims to identify security problems such as buffer overflow through mining the bug reports. Their purpose and techniques are different from my approach.

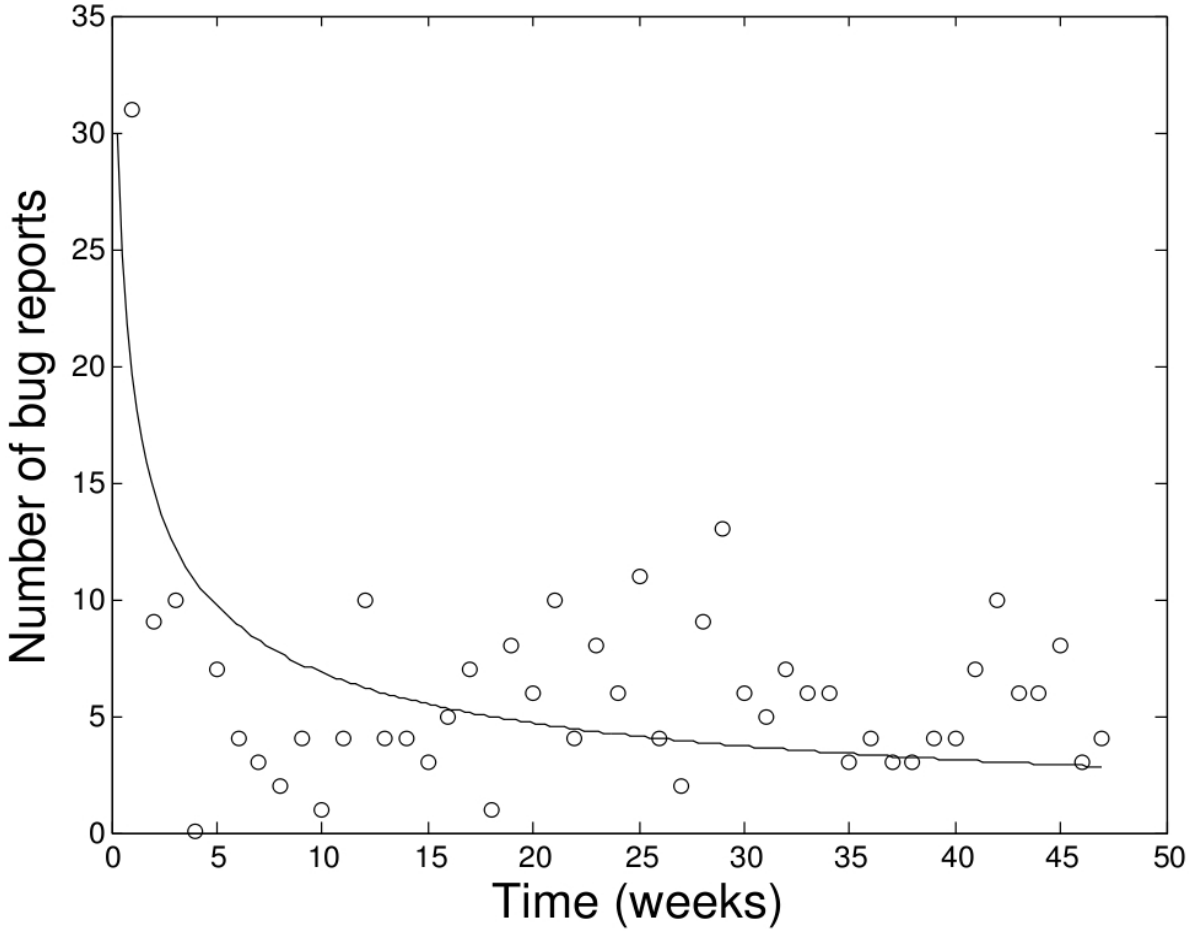


Figure B.3: Weibull fit for Tomcat 7

B.6 Summary

In this appendix, I presented BugMiner, a tool that is able to derive useful information from historic bug report database via data mining, use these information to do completion check and redundancy check on a new or given bug report, and to estimate the bug report trend using statistical analysis. I did empirical studies of the tool using several real-world bug report repositories. The experimental results show that BugMiner is effective, easy to implement, and has relatively high accuracy despite low quality data. BugMiner can be integrated into some existing bug tracking tools or software testing suites for more intelligent and cost-efficient software reliability analysis.

Appendix C

Constructing Software Bugs Using Mutation

Mutation testing applies mutation operators to modify program source code or byte code in small ways, and then runs these modified programs (*i.e.*, mutants) against a test suite in order to evaluate the quality of the test suite. In this appendix, I first describe a general fault model for concurrent programs and some limitations of previously developed sets of first-order concurrency mutation operators. I then present a new mutation testing approach, which employs synchronization-centric second-order mutation operators that are able to generate subtle concurrency bugs not represented by the first-order mutation. These operators are used in addition to the synchronization-centric first-order mutation operators to form a small set of effective concurrency mutation operators for mutant generation. My empirical study shows that our set of operators is effective in mutant generation with limited cost and demonstrates that this new approach is easy to implement.

C.1 Introduction

Mutation testing is a white-box fault-based software testing technique that uses mutants, slightly modified variants of the program source code or byte code, to characterize the effectiveness of a testing suite and locate weaknesses in the test data or program that are seldom or never exposed during normal execution [61]. Mutation testing is based on the *Competent Programmer Hypothesis* and the *Coupling Effect Hypothesis*. The *Competent Programmer Hypothesis* assumes that programmers are competent and normally write programs that are close to perfect; program faults are syntactically small and can be corrected with a few small code modifications [1, 61]. The *Coupling Effect Hypothesis* states that complex bugs in software are closely coupled to small, simple bugs. Thus, mutation testing can be effective in simulating complex real-world bugs [61, 167].

Mutation testing typically involves three stages: (1) Mutant generation, the goal of which is the generation of mutants of the program through inserting bugs. (2) Mutant execution, the execution of test cases against both the original program and the mutants. (3) Result analysis, to check the *mutation score*, *i.e.*, the percentage of nonequivalent mutants that are *killed* by the test suite [184, 167]. A mutant is *equivalent* to the original program if

the mutant and the original program always produce the same output, hence no test case can distinguish between the two [60]. A mutant is considered *killed* by the test suite if the execution result of the mutant is different from the result of the original program [169]. A test data set is said to be *adequate* if its mutation score is 100% [60, 168].

For the first stage, a predefined set of mutation operators are used to generate mutants from program source code or byte code. A *mutation operator* is a rule that is applied to a program to create mutants [169]. Mutants containing one simple fault are called *first-order mutants* and mutants containing two simple faults are called *second-order mutants* [184]. Researchers have developed many sets of mutation operators [169, 111], targeting a variety of programming languages. For example, Delamaro *et al.* and Bradbury *et al.* have proposed different set of mutation operators for concurrent Java programs [59, 30]. My empirical study and analysis shows that some subtle concurrency bugs are not generated by any of these proposed first-order mutation operators. My study further shows that a large portion of these operators are not effective in mutant generation: the majority of the mutants are generated by a small subset of the mutation operators, generally those that are synchronization-centric, *i.e.*, directly relating to the synchronization of different processes or threads. Based on a general fault model for concurrent programs and my analysis of the limitations in prior work, I present my new mutation testing approach, which employs synchronization-centric second-order mutation operators that are able to generate subtle concurrency bugs not represented by the first-order mutation. These operators are used in addition to the synchronization-centric first-order mutation operators to form a small set of effective concurrency mutation operators that can be used in mutant generation. My empirical study shows that our small set of operators is effective in mutant generation with limited cost and demonstrates that this new approach is easy to implement. The initial analysis of the possible implications of our results has potential impact on the Coupling Effect Hypothesis, indicating that possibly the coupling effect is weaker in concurrent programs than in sequential programs.

The remainder of this appendix is structured as follows. In section C.2, I describe a fault model for concurrent programs. In section C.3, I present the limitations of some previous work. In section C.4, I present my new approach. In section C.5, I present some empirical study. Lastly, I discuss some related work before conclusion.

C.2 Fault Model for Concurrent Programs

Testing concurrent programs is difficult. It is generally impossible or impractical to exhaustively test all combinations of input values or cover all possible control or data flow paths in sequential programs but even more so in concurrent programs; nevertheless, test suites can and must be constructed according to various criteria to attempt to find bugs. In order to develop a set of concurrency mutation operators that are able to model subtle concurrency bugs, I employ a general fault model that is based on the concurrency bug patterns and the synchronization mechanisms. The definition of *fault* is a programming error that leads to an erroneous result in some programs during execution.

C.2.1 Concurrency Bug Patterns

Some prior research on concurrency bug patterns has been done. [74] and [142] described taxonomy of common concurrency bugs. [73] compiled a benchmark of concurrency bugs. [44] and [145] described some empirical studies on concurrency bugs. I consolidate the common concurrency faults from these prior researches that I consider for mutation operators to model and present them below.

- Data Race: *Data race* condition happens when multiple threads read and write the same data, and the outcome of the execution depends on the particular order in which the accesses happen [44]. It is also called *thread interference*.
- Memory Inconsistency: *Memory inconsistency* errors occur when different threads have inconsistent views of the same variable.
- Atomicity Violation: *Atomicity violation* error is caused by concurrent execution of multiple threads violating the atomicity of a certain code region [145].
- Deadlock: *Deadlock* happens when multiple threads are blocked forever, waiting for each other.
- Livelock: *Livelock* happens when two threads are busy responding to each other and make no progress.
- Starvation: *Starvation* happens when a thread is unable to gain regular access to shared resources and is unable to make progress.
- Suspension: *Suspension* happens when a thread suspends or waits indefinitely.

C.2.2 Synchronization Mechanism

Concurrent programs rely on synchronization to ensure correct program execution. There are two main synchronization mechanisms: synchronization using shared memory and synchronization using message passing. For programming models that use shared memory synchronization (*e.g.*, Java and C#), the threads communicate primarily by sharing access to fields and the objects reference fields refer to. The synchronization aims to avoid thread interference and memory consistency errors. For programming models that use message passing (*e.g.*, Erlang [20] and Microsoft Asynchronous Agents Library [153]), the concurrent agents or actors in the programs communicate with each other through exchanging messages and use the synchronization to avoid problems in the message communications.

C.3 Limitations of First-Order Concurrency Mutation Operators

To measure the testability of concurrent Java programs, Ghosh described mutation based on two mutation operators that remove the keyword `synchronized` [86]. Long *et al.* tested mutation-based exploration for concurrent Java components [141]. Delamaro *et al.* proposed a set of 15 concurrency mutation operators for Java [59]. Later, Bradbury *et al.* proposed a new set of 24 concurrency mutation operators for Java [30]. The operators they proposed are all first-order mutation operators. I have investigated these mutation operators in my empirical study and identified some of their limitations.

C.3.1 Subtle Concurrency Bugs Are Not Generated

The first important limitation I found is that some subtle concurrency bugs are not generated by any of these proposed first-order mutation operators. This limitation could lead to loss of comprehensive representation of common concurrency bugs by the mutants, thus reducing the reliability of the mutation score that follows. I give two examples in the following subsections.

Data Race Example

The following code fragment from the *LinkedList* program, a Java program from the IBM concurrency benchmark programs repository [73], inserts an element to the end of a list. Another process, not shown here, reads the list. The synchronized method first starts from the top of the list (line 4), then moves to the end of the list via a loop (line 5) before inserts the object `x` to the end (line 6). Suppose we only apply first-order mutation operators (*e.g.*, removing `synchronized` keyword from line 2 or deleting a statement from line 4 to 6), the mutant does not represent a feasible programming error that line 2 is not synchronized and line 5's error causes the node index `itr` does not move to the end of the list properly. The combined error in line 2 and line 5 would potentially cause data race because multiple threads would try to write to the header of the list without synchronization and other threads might read the list at the same time. The outcome of the execution would depend on the thread schedule and which thread made the last call to the method because different threads all try to update the header of the list. By definition, this is a data race condition. To apply either operator independently is not going to create the same fault because under single application of either first-order mutation operator, data race would less likely happen and their mutants would represent different kind of faults.

```
1   /* Inserts element to the end of list */
2   public synchronized void addLast( Object x )
3   {
4       MyListNode itr = this._header;
5       while( itr._next != null ) itr = itr._next;
6       insert(x,new MyLinkedListItr(itr));
7   }
```

Deadlock Example

Incorrect use of synchronization can result in two or more threads waiting for each other to release the locks on the synchronized objects, forming a deadlock circle. As shown in the following example code for money transfer between two accounts, the line 5 and 6 in the original code may be incorrectly programmed in a nested synchronized block, which makes the deadlock possible. For example, two threads with execution of line 9 and 10 simultaneously would lead to deadlock since each thread will be waiting in a circle for the other thread to release required lock. This kind of deadlocks that require changes in more than one place are not generated by any first-order operator.

```
1   void TransferMoney(Acct a, Acct b, int amount) {
2      synchronized(a) {
3         a.debit(amount);
4      }
5       synchronized(b) {
6         b.credit(amount);
7      }
8   }
1   void TransferMoney(Acct a, Acct b, int amount) {
2       synchronized(a) {
3         synchronized(b) {   //first change
4            a.debit(amount);
5            b.credit(amount);   //second change
6         }
7      }
8   }
9   Thread1.run() { TransferMoney(a,b,10); }
10  Thread2.run() { TransferMoney(b,a,20); }
```

C.3.2 A Large Portion of Operators Do Not Generate Any Mutant

My empirical study shows that a large portion of existing mutation operators are not effective in generating mutants. For example, several previously proposed mutation operators for concurrent Java, including MSF, MXC, MBR, RCXC, ELPA, EAN, RSTK, RFU, RXO and EELO [59, 30], did not generate any mutants in my experiments. Some others, including MXT, RNA, RJS, InsNegArg, and ReplTargObj [59, 30], generated very few mutants. In my assessment, over half of the total number of operators are non-performing mutation operators, *i.e.*, operators that do not generate any new mutant. Most of the performing ones are related to mutation of a synchronized method or block.

C.4 Approach

C.4.1 Synchronization-Centric Second-Order Mutation Operators

My new mutation testing approach is based on the fault model and my analysis of the limitations of some previous work. I use synchronization-centric second-order concurrency mutation operators to construct subtle concurrency bugs that are not represented by the first-order mutation. While, a random and brute-force approach without any reduction would lead to $n * n$ second-order mutation operators based on n first-order mutation operators. To reduce the number of second-order mutation operators and mutant execution cost, I employ two steps of reduction. I first choose one of the two first-order mutation operators to be a synchronized method or block related modification and the other first-order operator to perform code changes related to the same synchronized method or block. For example, in concurrent Java, there are five first-order mutation operators related to synchronized

methods [59, 30], I choose two out of the five in the same category. Then I evaluate the chosen two first-order mutation operators to see if their combination can generate mutants that resemble some possible faults due to programming mistakes and only keep those meaningful combinations. This second reduction step through selection based on domain knowledge further reduces the amount of second-order mutation operators and leads to fewer unnecessary or redundant mutants.

After the set of synchronization-centric second-order concurrency mutation operators are chosen, they are combined with the synchronization-centric first-order mutation operators to form a smaller set of mutation operators for mutant generation.

C.4.2 Example Mutation Operators for Java

Table C.1 lists the synchronization-centric second-order concurrency mutation operators for Java, as an example of synchronization using shared memory. I describe each operator with example code in the following subsections.

sync method	RKSN+RSSN	Remove `synchronized` Keyword and a Statement from Synchronized Method
	AKST+MASN	Add `static` Keyword and Modify Argument with Constant to Synchronized Method
	RKSN+MASN	Remove `synchronized` Keyword and Modify Argument with Constant
sync block	RSNB+RSSB	Remove `synchronized` Block and a Statement from Synchronized Block
	MOSB+RSSB	Modify `synchronized` Object and Remove a Statement from Synchronized Block
	MOSB+MVSB	Modify `synchronized` Object and Move Statement(s) Out of Synchronized Block

Table C.1: Second-order concurrency mutation operators for Java

RKSN+RSSN

The RKSN+RSSN operator removes a `synchronized` keyword and a statement from a synchronized method. This operator simulates programming errors that can potentially lead to data race, memory inconsistency, and deadlock. The data race described in section C.3.1 can be constructed by this operator.

```
/* Original Code */
public synchronized void proc(Object A) {
   <statement 1>
   <statement 2>
}
/* RKSN+RSSN Mutant 1 */
public void proc(Object A) {   // sync removed
```

```
    ...// statement removed
    <statement 2>
}
/* RKSN+RSSN Mutant 2 */
public void proc(Object A) {  // sync removed
    <statement 1>
    ...// statement removed
}
```

AKST+MASN

The AKST+MASN operator adds a `static` keyword and modifies an argument with a constant to a synchronized method. This operator simulates programming errors that can potentially lead to data race and memory inconsistency.

```
/* Original Code */
public synchronized void send(String m) {...}
/* AKST+MASN Mutant */
public static synchronized void send(String n) {...}
```

RKSN+MASN

The RKSN+MASN operator removes a `synchronized` keyword and modifies an argument with a constant to a synchronized method. This operator simulates programming errors that can potentially lead to data race and memory inconsistency.

```
/* Original Code */
public synchronized void send(String m) {...}
/* AKST+MASN Mutant */
public void send(String n) {...}
```

RSNB+RSSB

The RSNB+RSSB operator removes the synchronized block and a statement from a synchronized block. This operator simulates programming errors that can potentially lead to data race, memory inconsistency, and deadlock.

```
/* Original Code */
synchronized (this) {
    <statement 1>
    <statement 2>
}
/* RSNB+RSSB Mutant 1 */
... // removed
    ... // removed
    <statement 2>
```

```
...
/* RSNB+RSSB Mutant 2 */
... // removed
   <statement 1>
   ... // removed
...
```

MOSB+RSSB

The MOSB+RSSB operator modifies a synchronized object and removes a statement from a synchronized block. This operator simulates programming errors that can potentially lead to data race and memory inconsistency.

```
/* Original Code */
synchronized(obj1) {
   <statement 1>
   <statement 2>
}
/* MOSB+RSSB Mutant */ // object modified
synchronized(newobj) {
   <statement 1>
   ... // removed
}
/* MOSB+RSSB Mutant */
synchronized(newobj) { // object modified
   ... // removed
   <statement 2>
}
```

MOSB+MVSB

The MOSB+MVSB operator modifies a synchronized object and moves statement(s) out of a synchronized block. This operator simulates programming errors that can potentially lead to data race, deadlock, memory inconsistency, and atomicity violation. The deadlock described in section C.3.1 can be constructed by this operator, *i.e.*, moving two lines of code including the synchronized block.

```
/* Original Code */
synchronized(obj1) {
   <statement 1>
   <statement 2>
   ...
}
/* MOSB+MVSB Mutant 1*/
<statement 1>  // moved
synchronized(newobj) { // object modified
```

```
    <statement 2>
    ...
}
/* MOSB+MVSB Mutant 2*/
<statement 1>  // moved
<statement 2>  // moved
synchronized(newobj) { // object modified
    ...
}
```

C.4.3 Example Mutation Operators for Erlang

Table C.2 lists the synchronization-centric second-order concurrency mutation operators for Erlang, as an example of synchronization using message passing. The CRT (*i.e.*, Change Reference Type) refers to changing a message reference from *Send by Ref* to *Send by Val*, and vice versa [108]. The CST (*i.e.*, Change Synchronization Type) refers to changing a message's synchronization method from *Sync Send* to *Async Send*, and vice versa. Since these mutation operators are self-explanatory, I do not give detailed example code here.

Messaging	CRT+MMP	Change reference type and modify message parameter
	CRT+RMP	Change reference type and reorder message parameter
	CRT+MMN	Change reference type and modify message name
	CRT+MMR	Change reference type and modify message recipient
	CST+MMP	Change sync type and modify message parameter
	CST+RMP	Change sync type and reorder message parameter
	CST+MMN	Change syn type and modify message name
	CST+MMR	Change syn type and modify message recipient
Constraint	CRT+RC	Change reference type and remove constraint
	CRT+MC	Change reference type and modify constraint
	CST+RC	Change syn type and remove constraint
	CST+MC	Change syn type and modify constraint

Table C.2: Second-order concurrency mutation operators for Erlang

C.5 Empirical Study

C.5.1 Implementation

I developed an Eclipse Plug-in [69] called BugGen that is able to automate mutant generation after the specific mutation operator is selected. Eclipse is a popular integrated development environment (IDE) with an extensible plug-in system. Building BugGen as an Eclipse Plug-in leverages the functionalities of the Eclipse and simplifies software development. During my implementation, I found the new set of mutation operators is easy to implement. In my empirical study, I focus on concurrent Java.

C.5.2 Example Programs

I use the following four example programs in my experiments to study mutant generation, as well as the cost and effectiveness of each proposed operator,

- *Webserver*, a Java web server program that supports concurrent client connections and synchronization [5].
- *Chat*, a Java chat program that supports multiple clients exchanging messages [68].
- *Miasma*, a graphical Java applet program from the NIH web-site [114]. It supports synchronization and uses `wait(t)` for prior pixels to be accepted before triggering another one.
- *LinkedList*, a modified Java program from the IBM concurrency benchmark programs repository [73]. The original program was developed to emulate the concurrency bug in using Java linked list, which is a non-synchronized collection.

I select the above example programs because they all employ different concurrency features and these programs are diversified in terms of type, size, coding style, applied field, and developer. These programs are representative in demonstrating common programming practices using concurrent Java. Table C.3 lists some statistical information for each program's source code.

Program Name	lines of code	classes	sync methods	sync blocks
Webserver	125	6	11	2
Chat	482	4	10	2
Miasma	360	1	0	2
LinkedList	421	5	1	1
Total	1,388	16	22	7

Table C.3: Example programs

C.5.3 Mutant Generation Results and Analysis

In my experiments, I apply each of the mutation operators listed in Table C.1, along with the synchronization-centric first-order mutation operators, on the example programs, count the number of mutants generated by each operator for each program, and then examine these mutants. From my experiments, I found that over half of the first-order mutation operators, especially those that are not related to synchronization, are not effective in generating mutants. Synchronization-centric mutation operators generate the majority of the mutants. My quantitative data and summations for each category are recorded in the histogram chart presented in Figure C.1. Details for each operator and the example programs can be found in my technical report [238]. The vertical axis shows the number of mutants. Most synchronization-centric mutation operators, in particular the second-order ones, are effective in mutant generation.

My empirical study demonstrates that the second-order mutation operators generate subtle concurrency bugs not represented by the first-order mutation; my mutant generation

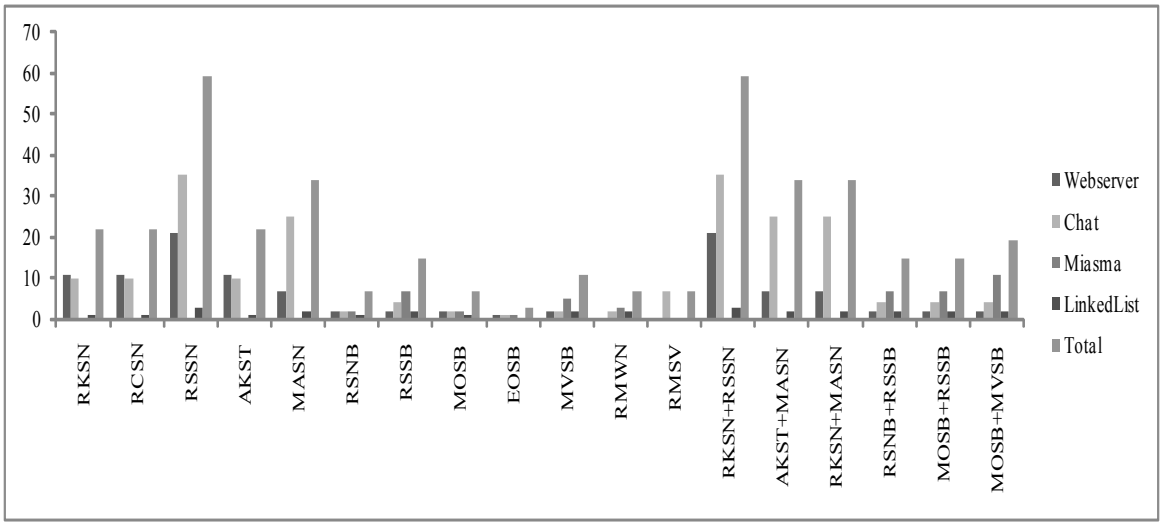


Figure C.1: Number of mutants generated per operator

effort is limited; fewer percentages of equivalent mutants are generated. Second-order operators tend to decrease the percentage of equivalent mutants [184].

C.6 Related Work

Some prior studies have been done on mutation testing for concurrent programs [111]. Carver described deterministic execution mutation testing and debugging of concurrent programs using synchronization-sequence [38]. Researchers have developed many sets of mutation operators [169, 111], targeting a variety of programming languages. Other than the mutation operators for concurrent Java, Jagannath *et al.* have proposed a set of mutation operators for actor programming model [108]. The synchronization-centric second-order mutation operators for message passing presented here also apply to the actor programming model.

For higher-order mutation, Polo *et al.* studied mutation cost reduction using second-order mutants [184]. Jia *et al.* described some general cases of higher-order mutation and related algorithms [110]. In my approach, I used second-order mutation to construct some subtle concurrency faults. By keeping a small number of second-order mutation operators based on synchronization and reduction through domain analysis, I avoided the drastic growth of the number of mutants, thus avoiding higher computing cost in mutant execution. To the best of my knowledge, this work is the first study of the second-order mutation operators specifically for concurrent programs.

C.7 Summary

This appendix first described a general fault model for concurrent programs and some limitations of previously developed sets of first-order concurrency mutation operators. I then

presented my new mutation testing approach, which employs synchronization-centric second-order mutation operators that are able to generate subtle concurrency bugs not represented by the first-order mutation. These operators are used in addition to the synchronization-centric first-order mutation operators to form a small set of effective concurrency mutation operators that can be used in mutant generation. I developed an Eclipse Plug-in called BugGen to automate the mutant generation using these operators. My empirical study showed that our set of mutation operators is effective in mutant generation with limited cost and this new approach is easy to implement. One potential future work is to evaluate some concurrency testing suites using the set of mutation operators.