Dependable Computing Systems Lab: Research Overview

Prepared by: Saurabh Bagchi

https://engineering.purdue.edu/dcsl/

Last Update: June, 2017

1 Executive Summary

The Dependable Computing Systems Lab (DCSL) comprises students (graduate and undergraduate), post-doctoral researchers, and programming staff members within the School of Electrical and Computer Engineering, the Department of Computer Science, and three institute-wide centers – CRISP (Center for Resilient Infrastructures, Systems, and Processes), CERIAS (The Center for Education and Research in Information Assurance and Security) and GE-funded PRIAM.

The broad goal of our research is to design practical dependable distributed systems. Our work is motivated by the fact that systems are increasing in scale, both in terms of the number of executing elements and the amount of data that they need to process and existing dependability techniques are increasingly failing to meet the demands of such scaling. Further, systems are heterogeneous, both in terms of hardware (GPU, DSP, FPGA, etc. coupled with traditional CPUs) and software (software from multiple parties being integrated to provide end-user functionality). The faults that bedevil such systems may be due to accidental (or natural) causes, or malicious (or induced) causes and we deal with both classes. Our work deals with faults by providing the functionality of *detection* (tell quickly that there is something wrong), *diagnosis* (what is the root cause of the failure), *containment* (how to prevent the failure from propagating through the system), and in some cases, *prediction* (of an impending failure, so that proactive mitigation actions can be triggered). The dependability mechanisms must not overly impact the application or the execution environment, either in terms of performance impact or in terms of the level of changes required from them. Our work focuses primarily in the application and in the

middleware software layers, while with embedded wireless devices, we also delve into the low-level firmware.

Changing execution environments

We observe that the kinds of execution environments are changing – from one where all the components are developed in-house, are open source and well-understood to one where they are made up of third-party software components. Some of these components are at least partially opaque to the system owner and interactions amongst the components have many patterns, some of which cannot be enumerated *a priori* (*i.e.*, before deployment of the system). There is a growing amount of non-determinism in the behavior of the systems due to various factors. First, the execution environments are intrinsically noisy, especially at large scales, due to interference from other applications executing on the same environment and unpredicted interactions among the application's components themselves. Second, the number of possible usage scenarios are increasing, with the end user interacting with the system under a variety of conditions (consider for example, a smart phone which the consumer is using for talking, while sending text messages, while a healthcare app is processing data from sensors on the phone). Third, the variety of runtime environments being used result in different kinds of faults, some subtly different and some radically so. For example, with heterogeneous computing, an incorrect mapping of functional blocks of the application to the system blocks (such as, a memory intensive part of code to a GPU), can lead to a performance degradation. Also, systems are being built using multiple programming languages and their interactions lead to new failure modes. For malicious errors, the different usage scenarios give rise to different attack paths.

The dependability solution has to handle these sources of non-determinism, increasing the fraction of failure cases that it handles correctly, while also keeping a limit on the number of correct executions that it incorrectly denotes as faulty. Different distributed systems impose different kinds of resource constraints on the dependability solution, e.g., large-scale computing clusters require that most communication due to the dependability protocols be kept local, with only rare use of global communication operations; embedded wireless networks constrain the volume of communication due to bandwidth and energy budget constraints.

Broad characteristics of our work

Our work fits within this broad universe of applications and execution environments. Our work can be characterized as "systems-y". It derives from solid theoretical underpinnings, adapting them to the domain-specific challenges, and then developing them for use in practical scenarios. Many of our problems involve large volumes of data, in rest and in motion, and we adapt data analytic algorithms to solve our problems. This often involves making existing algorithms distributed, scalable to large system sizes, and capable of dealing with unstructured data. We perform synthetic fault injections to evaluate our solutions and then test them out, as far as practicable, in real deployments and with real workloads. Our collaborations take two broad forms: with academic experts in a specific sub-field of Computer Science and with practitioners who face the dependability challenges in their respective domains. Examples of the former type include our explorations of data mining, machine learning, static analysis, scientific computing, and wireless software development. The latter class comprises collaborations with colleagues from industries and federal labs. Our work has been supported by and adopted by partners at Sandia National Labs (embedded and game-theoretic security), Northrop Grumman (distributed intrusion detection for enterprise class systems), Lawrence Livermore National Lab (reliability for large-scale scientific applications), Argonne National Lab (computational genomics), IBM (mobile computing

workloads migrated to the cloud), Avaya (Voice over IP security), Emnet LLC (wireless mesh network for waste water monitoring), Motorola (multi-hop wireless network for emergency responders), and Lockheed Martin (intrusion tolerance for zero-day attacks).

Our research is structured around three thrusts:

1. Dependability in large-scale applications
2. Strengthening enterprise-class distributed systems
3. Dependability of embedded wireless networks

2 Research Thrust 1: Dependability in Large-Scale Applications

2.1 Problem Statement

Current techniques for resilience are insufficient for exascale systems (i.e., systems capable of executing 10^{18} floating point operations per second), and unless radical changes are made across the entire software stack, exascale systems may never compute reliable scientific results. The available parallelism on exascale systems is expected to increase by 3-5 orders of magnitude over today's petascale systems, driven by increases in on-node concurrency and power density. At that point in the design space, hard and soft failures will be commonplace occurrences. The model of hardware being correct all the time, on all regions of the chip, and forever, will become prohibitively expensive to maintain, in terms of both manufacturing and energy cost. Replication-based approaches have promise, but blind replication of all tasks will halve the available performance on large-scale machines at best, wasting CPU cycles and energy on redundant work.

A targeted approach is needed to allow large-scale runtime systems to isolate regions where faults occur and replicate only those parts of the system. To enable this, we need runtime systems that monitor and analyze their own behavior to determine when to take preventive action. This type of analysis has been investigated before, but existing approaches aggregate system-wide data to a central point for analysis, which is not scalable and time-consuming. Further, existing analyses assume that parallel application behavior is relatively homogeneous across nodes and tasks. Such approaches will be ill-equipped to cope with the pervasive adaptive behavior of large-scale applications and heterogeneity of these platforms.

One emerging challenge is that the notion of correctness is sometimes tied to the characteristics of the data. For example, processing one request may take a millisecond while another may take over a second and both are correct behaviors. It is simply that the data accessed by the requests are different. We therefore have to come up with system models and related detection and diagnosis approaches that can be aware of the characteristics of the data.

2.2 Solution Approach

We have developed AutomaDeD [3], a tool that detects errors based on runtime information of control paths that the parallel application follows and the times spent in each control block. AutomaDeD suggests possible root causes of detected errors by pinpointing, in a probabilistic rank-ordered manner, the erroneous process and the code region in which the error arose. Intuitively, the erroneous tasks often form a small minority of the full set of tasks. Hence, they are outliers when we cluster the tasks, based on their features related to control flow and timing. Further, in the time dimension, the

executions in the first few iterations are more likely to be correct than in later iterations, which we also leverage to determine correct or erroneous labels; else we make use of some labeled correct runs, if available. Our solution approach is based on a detailed understanding and analyses of errors that show up in such large-scale systems, including some that we do on the DOE supercomputers [49, 50].

All existing parallel debugging tools (including our first effort at AutomaDeD) failed to scale to the process counts of today's state-of-the-art systems. Three main factors impede scalability. First, the tools include a centralized component that performs the data analysis. Thus, tools must stream behavioral information from all the processes to this central component so that it can process the information to deter- mine the error and, possibly, its location. Second, the tools require huge amounts of data. While many tools optimize the monitoring part quite well, the cost of shipping all information to the analysis engine and the cost of analyzing the full volume of data remains. While tools such as STAT [4] reduce the data volume that the central component must handle, they still must process the full data in their communication structure. Third, the data structures used to maintain the information are not completely optimized for the operations that need to be performed for error detection and localization, such as comparison of information from processes that belong to the same equivalence class. Small differences in the cost of one operation, though insignificant for hundreds of processes, become significant at larger scales. We address each of these concerns and develop a scalable version of AutomaDeD [5] that runs at scales of tens of thousands of application processes on LLNL's largest clusters and is able to detect and diagnose application problems. We have demonstrated the power of AutomaDeD on real bugs [**Error! Reference source not found.**], including a hard-to-crack bug in a molecular dynamics code [7]. For this, we augmented AutomaDeD with the ability to perform backward slicing, which it did working backward from the point of manifestation of the fault. The fault manifested only with 7,996 or more processes and our tool quickly found the fault — a sophisticated deadlock condition. This tool is now being used in production settings within LLNL and has had an open source release from the LLNL Scalability Team and us, in June 2014 [38]. A sample use case study is presented in a short paper [34].

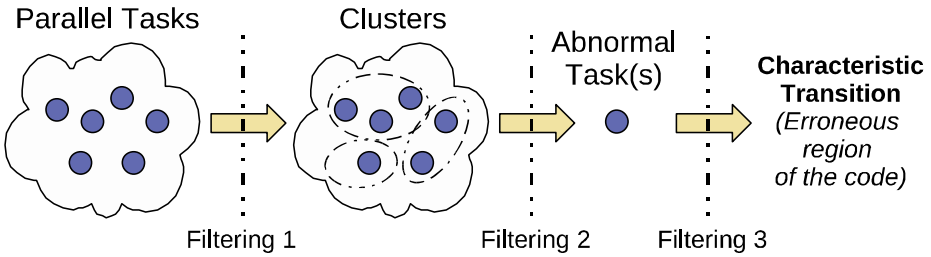

Figure 1. Problem localization with AutomaDeD

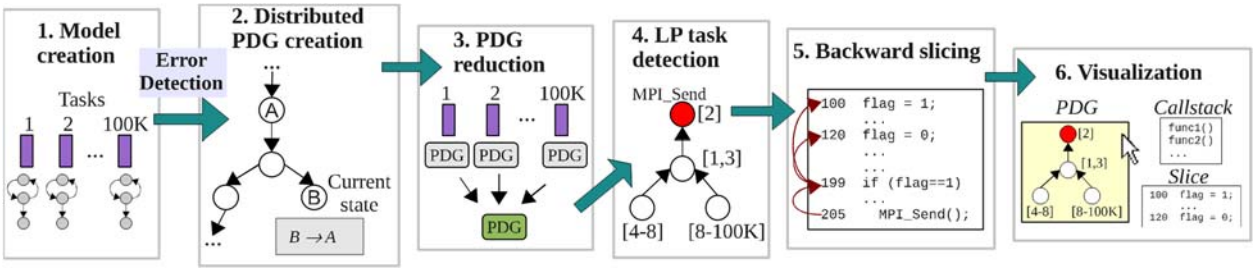

Figure 2. Overview of the workflow for problem localization

Diagnosing hangs and performance slowdowns

Hangs and performance slowdowns are a common and hard-to-diagnose problem in distributed applications. Due to the tight coupling of multiple tasks, a fault in one task can quickly spread to the other tasks, and can cause a large number of tasks, or even the entire application, to hang or to exhibit a slowdown. Previous work [39] proposed the notion of progress of tasks as a useful model to track down the root cause of a hang or a performance slowdown. Intuitively, progress is a partial order that orders tasks based on how much execution a task has made in relation to other tasks. Finding the least-progressed tasks can significantly reduce effort to identify where the fault originated. However, existing approaches to detecting them suffer low accuracy and large overheads [4, 6]; either they use imprecise static analysis or are unable to infer progress dependence inside loops. We have developed a progress-dependence analysis tool, called PRODOMETER [40], which determines relative progress among parallel tasks using *only dynamic analysis*. Our fault-injection experiments suggest that its accuracy and precision are over 90% for most cases and that it scales well up to 16,384 MPI tasks. Further, our case study shows that it significantly helped diagnosing a perplexing error in an LLNL MPI program, which only manifested at large scale. This algorithm has been incorporated into our previously mentioned open source release.

We have made our analysis and attendant tool aware of the characteristics of the data. We have developed a performance anomaly detection tool for closely interacting distributed applications, called GUARDIAN [51, 52]. Statistical modeling is a very powerful approach for making predictions about the behavior of a distributed system. However, the fact that applications' behavior often depends closely on their configuration parameters and properties of their inputs means that statistical models are built by training on only a small fraction of its overall behavior space. It has been well known that a model's accuracy often degrades as application configuration and inputs deviate further from its training set [54, 55], and this makes it difficult to do error detection or diagnosis based on the model's predictions. We have developed a systematic approach to quantify the prediction errors of the statistical models of the application behavior, focusing on extrapolation, where the application configuration and input parameters differ significantly from the model's training set. Given *any* statistical model of application behavior and a data set of training application runs from which this model is built, our technique predicts the accuracy of the model for predicting application behavior on a new run on hitherto unseen inputs. We validate the utility of this method by evaluating it on the use case of anomaly detection for seven mainstream applications and benchmarks. The evaluation demonstrates that our technique can reduce false alarms while providing high detection accuracy compared to a statistical, input-unaware modeling technique.

Scale-dependent bugs

An especially subtle class of bugs in large-scale applications are scale-dependent bugs: while small-scale test cases may not exhibit the bug, the bug arises in large-scale production runs, and can change the result or performance of an application. A simple example of this is a data type overflow that happens say when either a large amount of data is exchanged or data is exchanged among a large number of processes. A popular approach to finding bugs is statistical bug detection, where abnormal behavior is detected through comparison with bug-free behavior. Unfortunately, for scale-dependent bugs, there may not be bug-free runs at large scales and therefore traditional statistical techniques are not viable. We have developed a technique called Vrisha [8], a statistical approach to detecting and localizing scale-dependent bugs. Vrisha detects bugs in large-scale programs by building models of

behavior based on bug-free behavior at small scales. These models are constructed using kernel canonical correlation analysis (KCCA) and exploit scale-determined properties, whose values are predictably dependent on application scale. Then, if the predicted property at the large scale deviates from the observed property, an error is detected. By intelligently reverse mapping the deviation to the original behavioral feature, we can pinpoint the bug to a behavioral feature and from that, to a region of code. In our system called WuKong [37], we are able to pinpoint the source of the bug to a code region, even though the bug is not seen in the small-scale training runs.

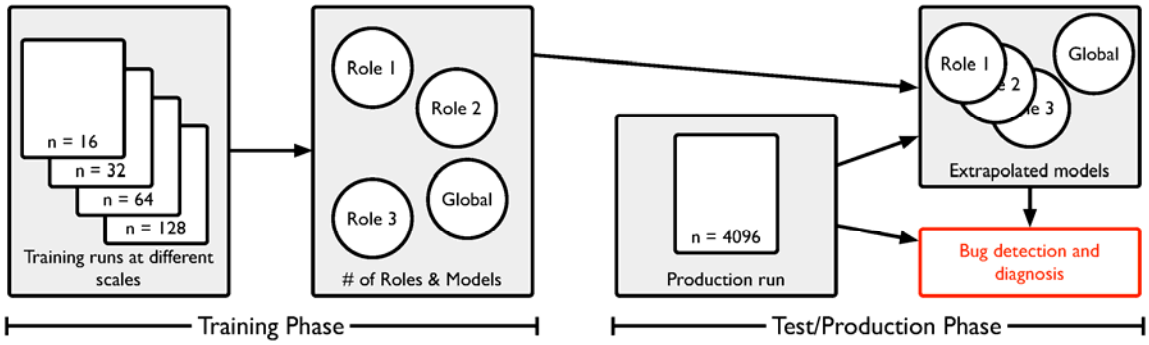

Figure 3. Overview of the operation of Vrisha divided into training runs at small scales, followed by production runs at large scales

2.3 What's Coming Up

We are developing techniques that can operate with different kinds of data at large scales. Sometime the behavior of the application changes in a hitherto hard-to-predict manner when the size of data goes above a certain threshold (e.g., thrashing effects kick in because the cache size is no longer sufficient) and we are equipping our current system to determine how correct behavior should change. To uncover such problems, we are developing a technique based on symbolic execution to determine what input dataset can likely trigger these hard-to-predict behaviors. As mentioned above, a large open problem in this domain is when behavior depends on characteristics of the data, not just its size. For example, an algorithm may execute a short time if it is invoked on a sorted list, but a long time if it is on a perfectly reverse sorted list. We are developing an approach to first determine what the data structure and algorithm specific feature is, and then extract that feature and use it in our existing technique. In another aspect, we are developing performance debugging tool for heterogeneous clusters. The problem is that it is not clear statically which codes, or blocks of a code, should run on an accelerator and which should run on the main cores. Further, when a code block runs on an accelerator, what configuration parameter should be used for the code block depends on the code characteristics as well as the accelerator resources. An incorrect decision can severely hurt performance. Our method is meant localize when such a performance anomaly happens and indicate the probable causes.

3 Research Thrust 2: Strengthening Enterprise-Class Distributed Systems

3.1 Problem Statement

Today's enterprise IT systems mostly run distributed applications. These applications are built out of a large number of software components and run on a variety of hardware platforms. Many of these applications require continuous availability despite being built out of unreliable components or components that are opaque to the system owner. Therefore, system administrators need efficient techniques and practical tools for error detection that can operate online (as the application runs), and that can detect errors and anomalies with small delay—the time between the error manifestation and its detection should be short. Preventing an error from becoming a user-visible failure is a further desirable characteristic. Automatically predicting impending failures based on observed patterns of measurements can trigger prevention techniques, such as microrebooting [9], redirection of further requests to a healthy server, or simply starting a backup service for the data. A third necessary functionality for reliable execution of distributed applications is problem localization, whereby automated techniques can determine if the program is at fault or the infrastructure on which the program is executing. If the program is at fault, then the system can provide localization of the fault to a region of the code, which can then be inspected by the developer for the purpose of implementing a fix.

We make our problem concrete by focusing on application systems from various domains. The first kind we are currently focusing on is approximate computing, which approximates some computation for reducing the processing time and energy cost of the computation. This class has been found to be useful in application domains where some amount of inaccuracy can be tolerated, such as, human perception (video or image processing) or machine learning. One challenge is to determine where in the application some approximation can be applied—approximation in some sensitive part of the computation can throw off the results completely. Another challenge is to determine how aggressively to approximate, *e.g.*, should we skip every other step in the processing loop, or every third step in the processing loop. Finally, if the result of the approximate computation is unacceptable, how do we do the root cause analysis to determine where something went wrong irreparably. The second domain that we are working on is computational genomics applications. Here, the large amounts of data (genomic, metagenomic, epigenomic data) mean that we often see scalability bottlenecks in the algorithms and the errors in input data (due to limitations of the sequencing instruments when processing samples at high rate) can lead to unbounded error propagation leading to scientifically, and clinically, incorrect prognosis. We therefore seek to develop both highly scalable algorithms and techniques to detect and diagnose error propagation cases. We attempt to do these in a manner that is not just specific to each individual algorithm, but also lead to fundamental primitives that are applicable across a swath of algorithms in the application domain.

In this sphere, we also consider maliciously injected errors at enterprise systems. We are focused on attacks to distributed enterprise systems that involve multiple steps, known as *multi-stage attacks* [56]. In these, adversaries compromise outward-facing services and use them as stepping stones to progressively compromise other services, with the ultimate goal to compromise a critical asset. An example would be compromising a web server, then achieve a series of intermediary steps (such as compromising a developer's box thanks to a vulnerable PHP module and connecting to a FTP server with gained credentials) to ultimately connect to a database where user credentials or financial information

are stored. Current detection systems are not capable of analyzing the multi-step attack scenario because they only focus on single steps of this multi-chain process and perform all their inferencing in a "greedy" manner based on the manifestation on that single place where the detector is installed. Further, the security posture is essentially reactive – once the detector finds something, some "greedy" response is taken, such as, disconnecting a TCP connection. The essential problem with this is three-fold – by the time the reactive response is taken, damage is already done; it is too dependent on the fidelity of the detectors; and finally, this security strategy often fails against hitherto unknown attacks, which are also known as *zero-day attacks*.

In this space, another problem that we are working on is unpredictable application performance in virtualized environments. Performance issues arise due to imperfect isolation of hardware resources across multiple VMs. Some resources, such as CPU and memory can be partitioned among VMs with little interference. However, current hypervisors do not isolate low level hardware resources, such as cache and memory bandwidth. Contention for these shared hardware resources leads to variable performance across VMs [41, 42]. We are creating solutions for this problem from the point of view of the end customer, as opposed to the cloud provider (who could change many things such as, the allocation of applications to VMs). More broadly, we are showing how an application can be reconfigured to survive through periods of resource contention. We look to do this in an analytically rigorous and general-purpose manner, across different kinds of server softwares.

3.2 Solution Approach

Today's enterprise-class distributed systems routinely collect a plethora of metrics by monitoring at various layers---system-level, middleware-level, and application-level. Many commercial and open-source tools exist for collecting these metrics, such as HP OpenView, Sysstat, and Ganglia. Examples of useful metrics are: at the system level: CPU, memory, storage, and network-bandwidth usage ; at the middleware level: resource usages in a Java EE container (such as Tomcat or JBoss) or time spent in an MPICH library call; at the application level: number of servlet requests and exceptions, number of JDBC connections, or time spent in a region of the code. A common class of error-detection techniques works as follows. From values of metrics collected during training runs, a model is built up for how the metrics should behave during normal operation. At runtime, a comparison is made between what is indicated by the trained model and what metric values are observed in the system. If there is sufficient divergence between the two, an error is flagged. Further the metrics that cause the divergence are mapped back to code regions that affect these metrics, thus providing a level of fault localization. However, existing approaches toward performing error-detection within a node based on statistical analysis of runtime metrics suffer from one or more of the following problems. First, their models do not consider multiple metrics simultaneously [11,12]. Many software bugs and performance faults are manifested in such a way that the correlations between measurements of different metrics are broken and these bugs are then missed. Second, some models do not consider observations of a metric as a sequence of measurements [13,14]. Many software bugs, for example those related to performance problems, develop a distinctive temporal pattern that can only be captured by analyzing measurements in a sequential manner rather than through instantaneous snapshots of the metric values. Third, the overwhelming majority of techniques do not offer failure prediction. They operate in a reactive mode by flagging alarms when a failure occurs rather than in a proactive mode by anticipating a failure. Failure prediction has been a hot topic in the past few years [15,16,17], however, to the best of our knowledge all the failure-prediction systems suffer from either the first or the second problem (or both).

With these insights, we develop a system called *Augury* [35,36] to perform error detection within a node using three progressively more sophisticated schemes - first, check for thresholds of individual metric values (both lower and upper bounds); next, check that the temporal patterns of the metric values follow the models of normality; and finally, check that the dependencies between metrics are maintained. Also, depending on which metrics are found to cause the deviation from normality, we can localize the fault to the program (application-level metrics) or the infrastructure (system-level or middleware-level metrics).

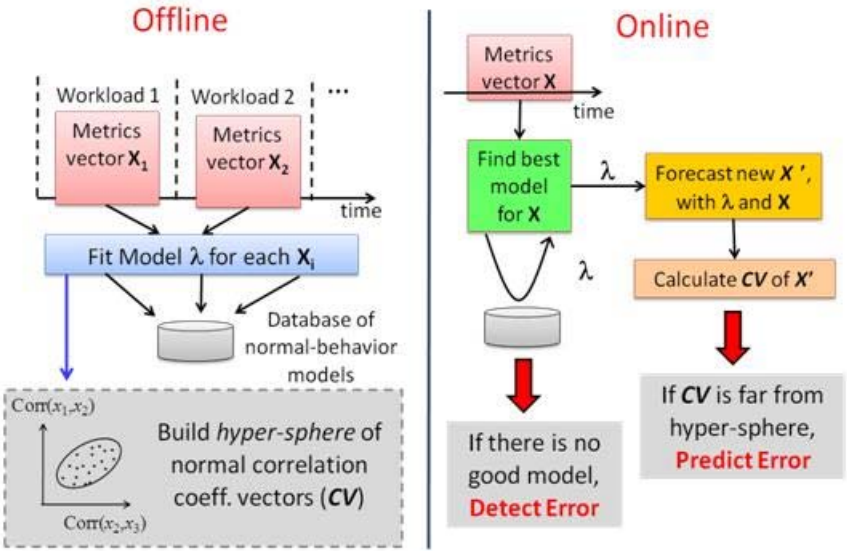

Figure 4. Overview of AUGURY for detection and prediction of errors in distributed enterprise systems

Failure prediction is only a small add-on to the steps described above for detecting anomalous behavior. Augury keeps a per-metric observation window and uses it to forecast future measurements using a selected ARIMA model. Once model parameters have been selected, Augury can perform *s*-step ahead forecasts and perform the error detection using the forecast values, thereby providing prediction. In work with IBM, we have shown [18] how web services that execute on cloud computing infrastructures can make use of such prediction to enable failover and load balancing. In this work, we have used the OSGi framework and built fault-tolerant services in it.

Since the accuracy of prediction of any statistical model is dependent on the fidelity of data that is used to train the model, we have developed a systematic approach, called E-ANALYZER [51, 52], to quantify the prediction errors of the statistical models of the application behavior. Our method focuses on extrapolation, where the application configuration and input parameters differ significantly from the model's training set. Given any statistical model of application behavior and a data set of training application runs from which this model is built, E-ANALYZER predicts the accuracy of the model for predicting application behavior on a new run on hitherto unseen inputs. We validate the utility of this method by evaluating it on the use case of anomaly detection for seven mainstream applications and benchmarks. The evaluation of our anomaly detection system, called GUARDIAN [51], demonstrates that our technique can reduce false alarms while providing high detection accuracy compared to a statistical, input-unaware modeling technique. Our approach gives a way out of the arrogance of claims of a

statistical model and provides a rigorous error quantification (or uncertainty quantification, if you will) that can then be used to make decisions based on a false positive-missed detection consideration.

Approximate computing

We have shown that many applications exhibit execution phase-specific sensitivity towards approximation of the internal subcomputations [57]. Therefore, approximation in certain phases can be more beneficial than others. With this insight we have developed OPPROX, a novel system for application's execution phase-aware approximation. For a user provided error budget and target input parameters, OPPROX identifies different program phases and searches for profitable approximation settings for each phase of the application execution. Our evaluation with five benchmarks (drawn from particle physics, video processing, computer vision, and optimization) and four existing approximation technique shows that when compared to an oracle but phase-agnostic version from prior work [58, 59], our approach on average provides 42% speedup compared to 37% from the oracle version for an error budget of 20% and for a small error budget of 5% provides on average 14% speedup compared to only 2% achieved by the phase-agnostic oracle version. In ongoing work, we have showed further that the approximation needs to be done in a content-aware manner. For example, in video processing, the optimal approximation setting for a hi-def sports scene is likely to be different than that for a standard-def sitting scene with only a few individuals. The challenge that we are taming is how to make the decision *quickly*, even though the decision is made in a content-aware manner. This relies on quickly identifying the relevant characteristics of the input, efficient search through the space of approximation settings, and doing change point detection to decide when a search needs to be re-initiated.

Scalable and reliable computational genomics

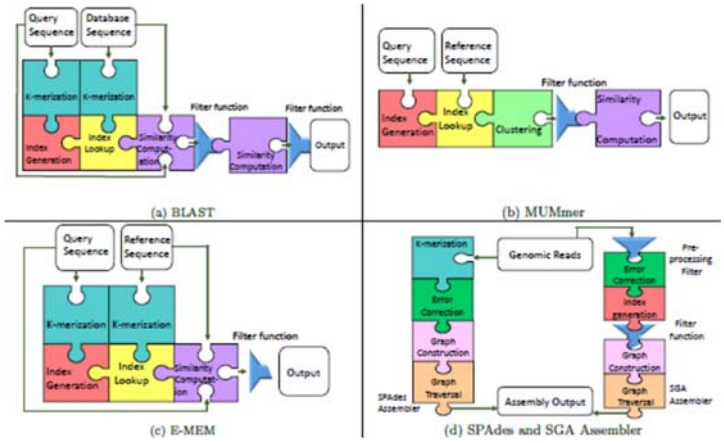

Figure 5. Overview of Genomic Applications in the categories of Local Alignment (BLAST), Whole genome Alignment (MUMmer and E-MEM), and Sequence Assembly (SPAdes and SGA). This shows the kernels, or the commonly recurring and reusable blocks of software functionality. Common kernels are shaded using the same color. Our DSL, Sarvavid, allows an application developer to easily piece together these kernels to create scalable applications. Our repository has efficient implementations of these kernels, for various different backends including conventional processors and accelerators.

From the earliest days of genomic sequencing, computers have been an essential component of genomic data analysis. However, with the cost of sequencing plummeting, newer kinds of genomic data

being generated, and newer questions being asked of the data, we are at a juncture where algorithms and computational tools need to play serious catch-up to keep pace with the rate of sequenced data. The genomes obtained through sequencing projects are at the core of the molecular chemistry of all species in the tree of life. But to understand these recipes, we need to develop novel computational genomics applications (e.g., pattern mining in gene regulatory networks) and we need to scale up existing applications (e.g., genome assembly with long reads) to work with larger and more diverse data sets. Further, the results of the computation need to be resilient to errors in the data (which seem unavoidable due to the nature of the genomic instruments or the data collection process, for metagenomics applications) and to errors in the computational pipeline.

With these goals, we have developed a domain-specific language, called SARVAVID, for computational genomics [60]. We made the observation that the popular bioinformatics applications, across a wide range of application areas, contain a recurring set of software modules, or *kernels* [61, 62]. The availability of efficient implementations of such kernels can improve programmer productivity, and provide effective scalability with growing data. Our DSL SARVAVID provides these kernels as language constructs. SARVAVID comes with a compiler that performs domain-specific optimizations, which are beyond the scope of libraries and generic compilers. Furthermore, SARVAVID inherently supports exploitation of parallelism across multiple nodes. We demonstrate how easy and worthwhile (from a speedup standpoint) it is to port 5 popular genomics applications from 3 areas---local alignment, global alignment, and genome assembly. The re-implemented applications can scale up (to more powerful individual nodes) as well as scale out (to a large number of nodes in a compute cluster).

Multi-stage attacks

For protecting a distributed enterprise system against multi-stage attacks (MSAs), we have developed a solution called the Distributed Intrusion and Attack Detection System (DIADS) [18,20]. DIADS has a central inferencing engine, which has a model of MSAs as attack graphs. DIADS creates a Bayesian Network (BN) out of an attack graph and observable (or evidence) nodes in the attack graph are mapped from sensor alerts (typical sensors are network-based intrusion detection sensors such as Snort and Bro and host-based intrusion detection sensors such as Tripwire). It receives inputs from the sensors and performs inferencing to determine whether a re-configuration of sensors is needed, i.e., whether any new sensor needs to replace an existing sensor, whether the placement of a sensor should be changed, or whether certain rules within a sensor need to be turned on or off. Thus, the inferencing engine has a two-way communication path with the sensors — obtaining alerts from the sensors and then interacting with the sensors once the inferencing is done. If on the basis of current evidence, it determines that a critical asset (also synonymously referred to as a "crown jewel") will imminently be compromised, it determines what further sensors close to the asset should be chosen, or equivalently, what further rules in an already active sensor should be turned on. DIADS can handle dynamism in the protected system (additions of computers, changes to configurations) as well as evolving attacks. Our system is being used in an internal cyber test range at Northrop Grumman and for intrusion detection in our NSF center NEEScomm IT infrastructure [22].

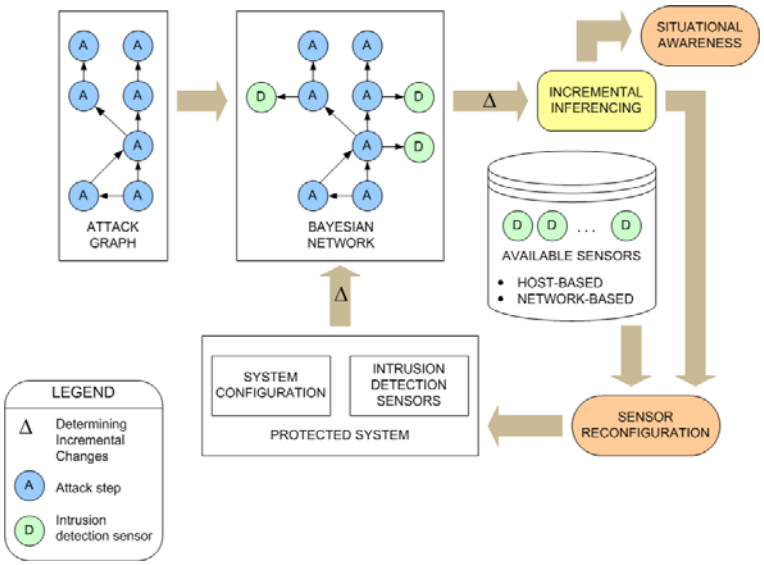

Figure 6. Overview of approach of DIADS to place and configure intrusion detection sensors in a changing enterprise environment

One type of intrusion detection system, called misuse-based detector, uses signatures of attacks to inspect the traffic and flag the malicious activity. But a potential problem faced by these signature-based systems is that as new attacks are created and as new kinds of benign traffic are observed, the signatures need to be updated. The current approach to this process is manual. Consequently, keeping them updated is a Herculean task that involves tedious work by many security experts at organizations that provide the detection software. A big drawback of the signature-based schemes that has been pointed out by many researchers and practitioners [43] is that due to their relatively static nature, they miss zero-day attacks. These are attacks that target hitherto unknown vulnerabilities and consequently, no signature exists for such attacks.

We have developed a technique for the automatic generation of intrusion signatures by mining the vast amount of public data available on attacks [44]. It follows a four-step process to generate the signatures (Figure 7), by first crawling attack samples from multiple public cybersecurity web portals. Then, a feature set is created from existing detection signatures to model the samples, which are then grouped using a biclustering algorithm which also gives the distinctive features of each cluster. Finally the system automatically creates a set of signatures using regular expressions, one for each cluster. We tested our architecture for SQL injection attacks and found our signatures to have a True and False Positive Rates of 90.52% and 0.03%, respectively and compared our findings to other SQL injection signature sets from popular IDS and web application firewalls. Results show our system to be very competitive to existing signature sets, which were manually generated and refined with significant domain expertise. We have also mined a vast amount of phishing email messages to come up with higher-level signatures for phishing campaigns. The unsurprising observation from mining the messages has been that adversaries create new phishing messages from existing templates, by changing, say, the name and details of a natural disaster. Our technique, by reasoning about higher-level features, is able to flag hitherto unseen phishing campaigns.

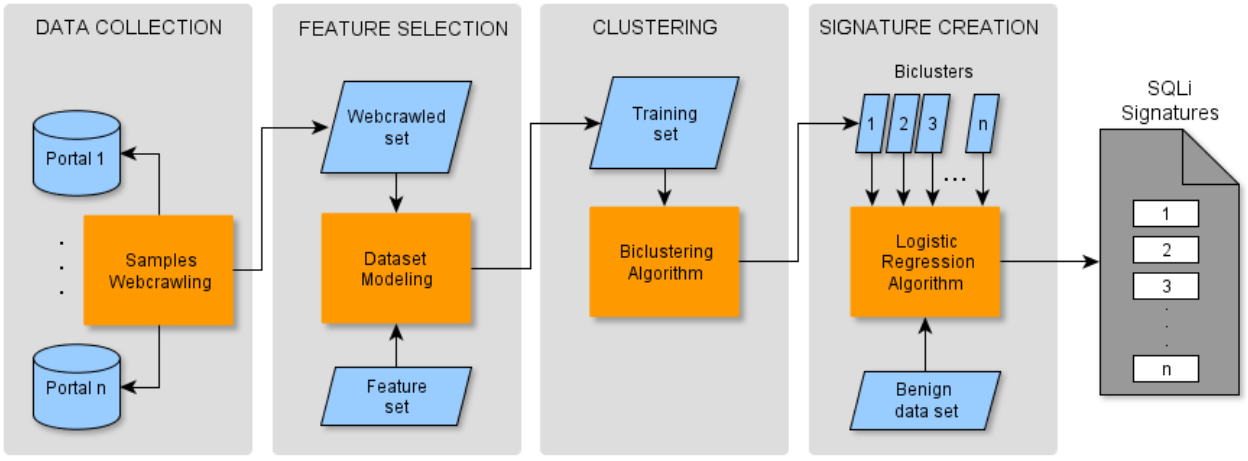

Figure 7. High-level solution approach to creating intrusion detection signatures by mining structured or unstructured attack and legitimate data. Here we show the resulting signatures are for SQL injection attacks (SQLi). The solution approach is general and we are applying this to a variety of attack types.

3.3 What's Coming Up

In continuing work, we are developing more sophisticated machine learning techniques for detecting and predicting subtle software bugs that fall in various categories, including resource leak, resource exhaustion, and race conditions. We are developing partially automated techniques for feature selection for feeding into our models, since the number of possible features is large. We are developing methods to perform the failure diagnosis and approximation in a workload-aware, or content-aware, manner because different workloads to an application can generate different behavior patterns, for correctly functioning applications. The challenge here is to capture the workload patterns or the input data patterns quickly and at the right level of abstraction that is needed for the model to generate high fidelity output.

In the security work, we are building proactive techniques to prevent novel attacks (zero-day attacks) from breaching the security of multiple connected components in a distributed system. Our solution will involve learning from prior attacks such that variants of these attacks can be thwarted. A complimentary solution strategy that we are developing involves randomizing the locations and configurations of key services and assets through judicious use of deception. Deception has been used for many millennia, perhaps for as long as life existed on planet earth. Plants, animals, including humans, and insects have been using deceptive techniques as a means for defense and survival. Our work will show how to plan and integrate deception in computer security defenses, e.g., by creating a deceptive file system where accessing certain key files will provide false information. For enterprise security, we are using moving target defense (MTD) implemented using a Software Defined Network (SDN) layer to thwart adversaries, even when the vulnerability being exploited is not known *a priori*. SDN gives us a novel tool to quickly reconfigure the network in anticipation of attack paths or to thwart a currently spreading attack.

4 Research Thrust 3: Dependability of Embedded Wireless Networks

4.1 Problem Statement

Embedded wireless networks are plagued by the possibility of bugs manifesting only at deployment. However, debugging deployed embedded wireless networks is challenging for several reasons—the remote location of deployed nodes, the non-determinism of execution that can make it difficult to replicate a buggy run, and the limited hardware resources available on a node. One promising method to debug distributed systems is record and replay. In short, record and replay logs a trace of predefined events while a deployed application is executing, enabling replaying of events later using debugging tools. Existing recording methods fail on embedded wireless networks due to the many sources of non-determinism, failing to capture the complete code execution, thus negating the possibility of a faithful replay and causing a large class of bugs to go unnoticed. Further, they overflow the available storage resources on the node and violate real-time requirements of many applications.

The visibility afforded by record and replay can be useful for post-deployment testing [23], replay-based debugging [24,25], and for performance and energy profiling of various software components [26,27]. Prior software-based solutions to address this problem have incurred high execution overhead and intrusiveness [24,26]. The intrusiveness changes the intrinsic timing behavior of the application, thereby reducing the fidelity of the collected profile. Prior hardware-based solutions [28,29] have involved the use of dedicated ASICs or other tightly coupled changes to the embedded node's processor, which significantly limits their applicability. Therefore, our goal is to design novel hardware-software approaches that can be deployed at scale (and therefore must be low cost and low energy hogs) that can collect traces of different kinds of events without perturbing the timing of the application.

In recent years, advances in hardware and software tools have led to many real-world deployments of multi-hop wireless networks, i.e., wireless networks which require little fixed infrastructure. Management of already deployed multi-hop networks is an important issue. One of the crucial management tasks is that of software reconfiguration. During the lifetime of a multi-hop network, software running on the nodes may need to be changed for various reasons like correcting software bugs, modifying the application to meet the changing environmental conditions in which the network is deployed, adapting to evolving user requirements, etc. Since a multi-hop network may consist of hundreds or even thousands of nodes which may be situated at places which are difficult or, sometimes, impossible to access physically, remote reprogramming of multi-hop networks is essential. The two most critical metrics for such reprogramming are energy efficiency and speed. Since the performance of the network may be degraded, or even reduced to zero, during software update process, the technique must minimize reprogramming time.

We are also motivated by the trend of having more resource rich, mobile devices that we carry on our bodies, such as, smartphones. As more critical applications are put in these smartphones, it is important to analyze the failure characteristics of these platforms – how do the different components fail, how are these different from traditional software failures, considering that the software has an event-driven flavor and need to be able to handle multiple input devices, sometime providing inputs concurrently. This investigation should lead to uncovering vulnerabilities that are exposed either due to careless programming or due to maliciously crafted inputs being sent in from the outside, such as, in the form of text messages or inputs fed to a sensor on the smartphone. Further, there is a need being expressed by data center owners to monitor and control data center assets from smart phones. This has to be done

while preserving security guarantees, restricting the energy and bandwidth usage on the smart phones, and providing actionable information to the system administrator despite the small form factors of the client devices.

On the security side of this project, we ask ourselves the question – can the primitive state of secure programming on these low-cost embedded devices be improved, through transformations inserted into the compilation workflow (so that embedded application developers do not have to become security experts too) and through low-cost monitoring of behavior at runtime. This is a challenge due to several reasons—the software stack on these devices is often monolithic with no separation of privileged and unprivileged code, the amount of memory and stable storage resource is limited, and the applications often have real-time requirements.

4.2 Solution Approach

In-situ reprogramming and reconfiguration

We have developed a suite of reprogramming techniques for embedded wireless networks, which can reprogram the nodes while they are deployed *in situ* in the environment [45, 46]. We currently have a US patent for the fastest multi-hop reprogramming protocol [47], which has been licensed by several companies from Purdue's Office of Technology Commercialization. One of our solutions, called Zephyr [30], is an incremental reprogramming protocol that exploits the fact that in real world scenario, the software running on the sensor nodes evolves with incremental changes to the functionality. Zephyr significantly reduces reprogramming time and energy by wirelessly transferring only the difference between the old and new versions of the software, rather than the entire new software. The wireless nodes build the new image using the difference and the old image.

High fidelity record and replay for debugging

We have designed and prototyped AVEKSHA [26], a hardware-software approach for tracing applications running in an embedded wireless node in a non-intrusive manner. Our approach is based on the key insight that most embedded processors have an on-chip debug module (which has traditionally been used for interactive debugging) that provides significant visibility into the internal state of the processor. We designed a debug board (shown in Figure 8) that interfaces with the on-chip debug module of an embedded node's processor through the JTAG port and provides three modes of event logging and tracing of varying granularities. Using expressive triggers that the on-chip debug module supports, AVEKSHA can watch for, and record, a variety of programmable events of interest, such as, a read from a peripheral I/O device. A key feature of AVEKSHA is that the target processor does not have to be stopped during event logging (in two of the three modes), subject to a limit on the rate at which logged events occur. AVEKSHA also performs power monitoring of the embedded wireless node and, importantly, enables power consumption data to be correlated to events of interest.

AVEKSHA is an operating system-agnostic solution. We demonstrate its functionality and performance using applications in TinyOS and in Contiki. We show that AVEKSHA can trace tasks and other generic events at the function and task-level granularity. We have also used AVEKSHA to find a subtle bug in the TinyOS low power listening protocol.

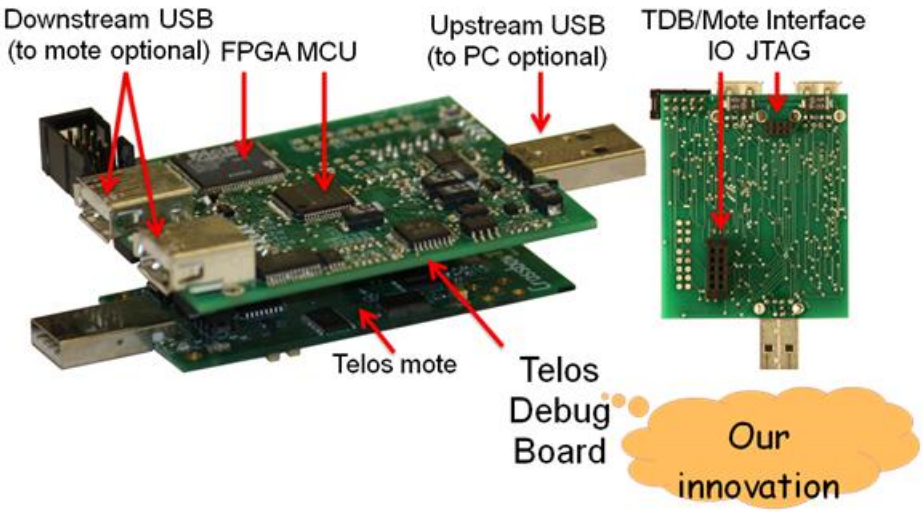

Figure 8. AVEKSHA's debug board interfaced with a TI microcontroller

In follow-on work, we have developed a complete software-only system-level record and replay technique on embedded wireless device, called TARDIS [48]. It handles *all* of the sources of non-determinism and compresses each one in a resource efficient manner using respective domain-specific knowledge. The compression scheme for each source of non-determinism is informed by a careful observation of the kinds of events that typically occur in WSN applications, for example, the use of register masking which reduces the number of bits which must be recorded—instead of the full length of the register, only the bits that are left unmasked need be recorded. The compression schemes are also chosen to be lightweight in their use of compute resources. Furthermore, the compression is done in an opportunistic manner, whenever there is "slack time" on the embedded microcontroller so that the application's timing requirement is not violated.

Countering strategic adversaries in inter-dependent CPS

We are also working on security of Cyber Physical Systems (CPS), taking a macro, economics-oriented viewpoint of the problem as it applies to interacting CPS's belonging to multiple organizations, such as the smart electric grid [63, 64] and industrial control systems [65]. Interdependent CPS's contain competitive environments in which strategic adversaries can launch cyber-attacks to extract profits from the system. When multiple actors are competing, the disruption of key assets can create large swings in the profitability of each actor by changing the supply and demand dynamics of the underlying physical system. These swings in profits can be leveraged by an attacker to extract profits from the system. While countering these attacks through traditional means is one possibility, that is challenging because not all aspects of the CPS can be secured due to budgetary and legacy reasons as well as competitive pressures among the stakeholders. We explore the implications of architectural changes on system resilience when faced with a profit-seeking adversary. Changes in the physical system may mitigate or exacerbate the likelihood of cyber-attacks on system assets, and we present a strategy for optimizing potential changes to minimize attacks. We also explore the impact of information sharing on system behavior and the potential for deception to improve system resilience. We exercise these design principles on real-world inter-connected CPS systems, in collaboration with domain experts at USC/ISI (DETER simulation testbed) and UIUC (smart electric grid). One example where we have successfully applied our method is an interconnected natural gas and electric power infrastructure to show the potential security

improvements provided by architectural changes [63]. We have also developed a DoS injection tool that can simulate the DoS attack on such a system and play what-if scenarios to decide on security investments [66].

Enabling robust protections for bare-metal systems

Embedded Systems are found everywhere. The Internet of Things is increasing the number and connectivity of these systems. Increased connectivity makes security vitally important. Many of these systems are and will be small bare-metal systems. Bare-metal systems execute a single application directly on the hardware without multiple layers of abstractions. This software must manage the hardware and implement the application logic. Fundamental in bare-metal system design is a tension between security and software design. Security requires that access to some hardware (e.g. changing memory protections) be restricted, but as the only software running on the system, it must be able to manage *all* hardware. We solve this tension by use of our technique called *privilege overlays* [67]. Privilege overlays use static analysis to identify those instructions of the program that must execute with privileges, and enables elevating only these instructions to execute with privileges. This provides the foundation on which code integrity, diversity, and strong stack protections are built. Our compiler, EPOXY (Figure 9), based on an LLVM pass, enables these protections to be applied without modifying the application logic. We show that these protections are both effective from a security perspective and on average have less than a 2% impact on execution time and energy consumption.

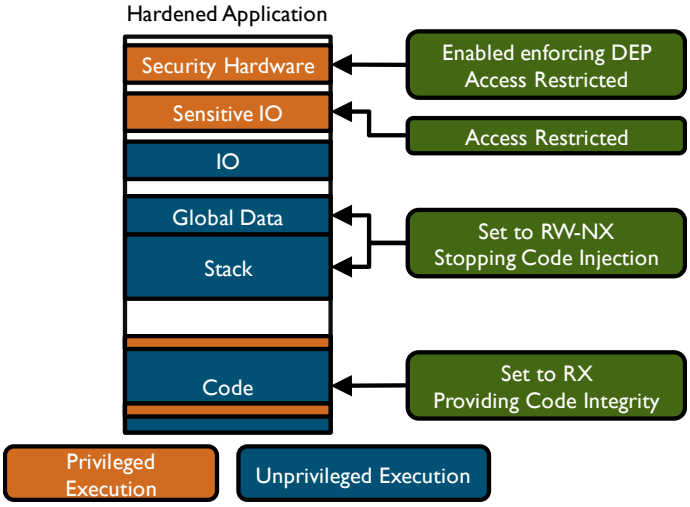

Figure 9. Schematic showing our security technique for executing code on bare-metal systems with the minimum privilege level required. The system called EPOXY, introduces privilege overlays inserted through a compiler pass, to elevate the privileges of only those few pieces of code that need to perform security-sensitive operations, such as, changing the locations of the interrupt vector tables.

Data analysis for failures in mobile devices

For the smartphone study, we have analyzed the bug reports of the two open source OSes – Android and Symbian OS and come up with a failure characterization in the different modules [32]. Our study indicates that Development tools, Web browsers, and Multimedia applications are most error-prone in both these systems. We further categorized the different types of code modifications required for the fixes. The analysis shows that 78% of errors required minor code changes, with the largest share of

these coming from modifications to attribute values and conditions. Our final analysis focuses on the relation between customizability, code complexity, and reliability in Android and Symbian. We find that despite high cyclomatic complexity, the bug densities in Android and Symbian are surprisingly low. However, the support for customizability does impact the reliability of mobile OSes and there are cautionary tales for their further development.

In further work [33], we have developed a software fault injector to test how robust the Inter Process Communication (IPC) mechanism is in Android (the Android term for this is "Intent"). We have used the injector to discover vulnerabilities exploitable through random (or crafted) Intents. We then provide recommendations for hardening of Android IPC. During our experiments we sent more than 6 million Intents to 800+ application components across 3 versions of Android and discovered a significant number of input validation errors. In general less than 10% of the components tested crashed; all crashes are caused by unhandled exceptions. Our results suggest that Android has a sizable number of components with unhandled NullPointerExceptions across all versions. The most striking finding that we have is the ability to run privileged processes from user level applications without requiring the user-level application to be granted any special permission at install time. In more recent work [68], we have characterized the vulnerability of applications for physical fitness monitoring, developed in the Android Wear OS, to malformed inputs.

4.3 What's Coming Up

Our record-and-replay technique needs to take into account interactions among the nodes, rather than looking at a single node at a time. Such interaction can lead to propagation of failures. Further, is it necessary to bring the trace back to a central node for the purposes of recreating the events, or can replay be done opportunistically at nearby nodes. We believe this is possible, and as such, will relieve the requirement to bring the bulky traces across the wireless network to a single central point.

While there has been a great deal of work that analyzes security of interdependent CPS, it predominantly relies on classical models of perfectly rational and optimal behavior to represent the human decision-makers. In contrast, there is a substantial body of work in behavioral economics and psychology showing that humans are only partially rational and thus, consistently deviate from the above-mentioned classical models. For example, human perceptions of risks, rewards, and losses can differ substantially from their true values, and these perceptions can have a significant impact on the investments made to protect the systems that the individuals are managing. In ongoing research, we are comprehensively characterizing the decisions made by humans to protect their systems using more realistic models of behavioral decision-making. The research encompasses both formal theory to rigorously analyze and predict the outcomes that should be expected under alternative models of behavioral decision-making, and laboratory experiments with human subjects to evaluate the predications made by the theory and to identify new behavioral models for analysis.

In the area of security of embedded systems, we are pursuing three inter-locking areas of research: (i) **Static analyses** to identify security and functionality characteristics of parts of the application, (ii) **Runtime execution techniques** to minimize the performance impact of the privilege overlay, and (iii) **Benchmarking** of security and functionality achieved through targeted injections of exploits.

With emerging mobile devices, we are doing reliability analysis of devices in the personal healthcare area (think, Google's Moto360 watch and Fitbit wristband) to see what parts of the software stack are vulnerable and what systematic changes can be made to the development environment.

5 Take-Aways

There are exciting research and deployment problems in the area of dependable systems. Constraints and requirements from specific application domains add to the richness of our problem space, coming currently from scientific computing applications, internet-scale distributed services, computational genomics, and embedded wireless networks. At the Dependable Computing Systems Lab, we are forging ahead, with a diverse group of collaborators within and outside Purdue, within academia and in industrial organizations, to address some of the most important challenges in dependable system design and implementation. Our work is being recognized through publications at top venues, adoption within our collaborating industrial organizations, and our leadership role at our professional societies (IEEE and ACM). We encourage you to contact us if you have interests in this direction.

6 References

1. M. Fahey, J. Larkin, and J. Adams, "I/O performance on a massively parallel Cray XT3/XT4," IEEE International Symposium on Parallel and Distributed Processing (IPDPS), pp.1-12, 2008.
2. Kazuki Ohta, Dries Kimpe, Jason Cope, Kamil Iskra, Robert Ross, and Yutaka Ishikawa, Optimization Techniques at the I/O Forwarding Layer, IEEE International Conference on Cluster Computing (Cluster 2010), pp. 312-321, September 2010, Heraklion, Greece.
3. Greg Bronevetsky, Ignacio Laguna, Saurabh Bagchi, Bronis R. de Supinski, Dong H. Ahn, and Martin Schulz, "AutomaDeD: Automata-Based Debugging for Dissimilar Parallel Tasks," At the 40th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN), pp. 231-240, June 28-July 1, 2010, Chicago, IL.
4. Gregory L. Lee, Dong H. Ahn, Dorian C. Arnold, Bronis R. de Supinski, Matthew Legendre, Barton P. Miller, Martin Schulz, and Ben Liblit, "Lessons learned at 208K: towards debugging millions of cores," In Proceedings of the ACM/IEEE conference on Supercomputing (SC), pp. 1-9, 2008, Austin, TX.
5. Ignacio Laguna, Todd Gamblin, Bronis R. de Supinski, Saurabh Bagchi, Greg Bronevetsky, Dong H. Anh, Martin Schulz, and Barry Rountree, "Large scale debugging of parallel tasks with AutomaDeD," In Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis (SC), pp. 1-10, 2011, Seattle, WA.
6. Ignacio Laguna, Dong H. Anh, Bronis R. de Supinski, Saurabh Bagchi, and Todd Gamblin, "Probabilistic Diagnosis of Performance Faults in Large Scale Parallel Applications," At the 21st International Conference on Parallel Architectures and Compilation Techniques (PACT), pp. 213-222, September 19-23, 2012, Minneapolis, MN.
7. F. H. Streitz, J. N. Glosli, M. V. Patel, B. Chan, R. K. Yates, B. R. de Supinski, J. Sexton, and J. A. Gunnels, "Simulating solidification in metals at high pressure: The drive to petascale computing," Journal of Physics: Conference Series, 46(1):254, 2006.
8. Bowen Zhou, Milind Kulkarni, and Saurabh Bagchi, "Vrisha: Using Scaling Properties of Parallel Programs for Bug Detection and Localization," At the 20th ACM International Symposium on High-Performance Parallel and Distributed Computing (HPDC), pp. 85-96, June 8-11, 2011, San Jose, CA.
9. George Candea, Shinichi Kawamoto, Yuichi Fujiki, Greg Friedman, Armando Fox, "Microreboot - A Technique for Cheap Recovery," At the 6th Symposium on Operating Systems Design and Implementation (OSDI), pp. 1-14, San Francisco, CA, December 2004.
10. IBM, "Infosphere BigInsights," At: http://www.ibm.com/software/data/infosphere/biginsights/. Last accessed: June, 2017.
11. L. Cherkasova, K. Ozonat, N. Mi, J. Symons, and E. Smirni, "Anomaly? Application Change? Or Workload Change? Towards Automated Detection of Application Performance Anomaly and Change," At the IEEE International Conference on Dependable Systems & Networks (DSN), pp. 452-461, Anchorage, Alaska, June 24-27, 2008.

12. K. Ozonat, "An information-theoretic approach to detecting performance anomalies and changes for large-scale distributed web services," At the IEEE International Conference on Dependable Systems and Networks (DSN), pp.522-531, Anchorage, Alaska, June 24-27, 2008.
13. Mike Y. Chen, Anthony Accardi, Emre K c man, Jim Lloyd, Dave Patterson, Armando Fox, Eric Brewer, "Path-Based Failure and Evolution Management," At the 1st Symposium on Networked Systems Design and Implementation (NSDI), pp. 1-14, San Francisco, CA, March 29–31, 2004.
14. Z. Guo, G. Jiang, H. Chen, and K. Yoshihira, "Tracking Probabilistic Correlation of Monitoring Data for Fault Detection in Complex Systems," At the IEEE/IFIP International Conference on Dependable Systems and Networks (DSN), pp. 259-268, Philadelphia, PA, 25-28 June, 2006.
15. J. Alonso, J. Torres, J. L. Berral, and R. Gavalda, "Adaptive on-line software aging prediction based on machine learning," At the IEEE/IFIP International Conference on Dependable Systems and Networks (DSN), pp. 507-516, Chicago, IL, June 28-July 1, 2010.
16. Felix Salfner and Steffen Tschirpke, "Error Log Processing for Accurate Failure Prediction," At the 1st USENIX Workshop on the Analysis of System Logs (WASL), pp. 1-8, San Diego, CA, December 7, 2008.
17. A. W. Williams, S. M. Pertet, and P. Narasimhan,"Tiresias: Black-Box Failure Prediction in Distributed Systems," At the IEEE International Parallel and Distributed Processing Symposium (IPDPS), pp. 1-8, Long Beach, CA, 26-30 March 2007.
18. Jan Rellermeyer and Saurabh Bagchi, "Dependability as a Cloud Service - A Modular Approach," In: Proceedings of the 2nd International Workshop on Dependability of Clouds, Data Centers, and Virtual Machine Technology (DCDV 2012, in conjunction with DSN 2012), pp. 1-6, Boston, MA, June 2012.
19. Gaspar Modelo-Howard, Saurabh Bagchi, and Guy Lebanon, "'Determining Placement of Intrusion Detectors for a Distributed Application through Bayesian Network Modeling," At the 11th International Symposium on Recent Advances in Intrusion Detection (RAID), pp. 271-290, Boston, MA, September 15-17, 2008.
20. Gaspar Modelo-Howard, Jevin Sweval, and Saurabh Bagchi, "Secure Configuration of Intrusion Detection Sensors for Changing Enterprise Systems," At the 7th ICST International Conference on Security and Privacy for Communication Networks (Securecomm), pp. 1-20, London, United Kingdom, September 7-9, 2011.
21. Timothy Tsai, Nawanol Theera-Ampornpunt, and Saurabh Bagchi, "A Study of Soft Error Consequences in Hard Disk Drives," Accepted to appear at the 42nd Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN), pp. 1-8, Boston, MA, June 25-28, 2012.
22. Purdue University, "NEES – Cybersecurity," At: http://nees.org/explore/security. Last accessed: June, 2017.
23. M. Wang, Z. Li, F. Li, X. Feng, S. Bagchi, and Y.-H. Lu, "Dependence-based multi-level tracing and replay for wireless sensor networks debugging," in Proceedings of the 2011 SIGPLAN/SIGBED conference on Languages, compilers and tools for embedded systems (LCTES), pp. 91–100, New York, NY, 2011.

24. V. Sundaram, P. Eugster, and X. Zhang, “Efficient diagnostic tracing for wireless sensor networks,” in Proceedings of the 8th ACM Conference on Embedded Networked Sensor Systems (SenSys), pp. 169–182, New York, NY, 2010.
25. S. Choudhuri and T. Givargis, “Flashbox: a system for logging non-deterministic events in deployed embedded systems,” in Proceedings of the 2009 ACM symposium on Applied Computing, pp. 1676–1682, 2009.
26. Matthew Tancreti, Mohammad Sajjad Hossain, Saurabh Bagchi, and Vijay Raghunathan, “AVEKSHA: a hardware-software approach for non-intrusive tracing and profiling of wireless embedded systems,” In Proceedings of the 9th ACM Conference on Embedded Networked Sensor Systems (SenSys), pp. 288-301, New York, NY, 2011.
27. Rodrigo Fonseca, Prabal Dutta, Philip Levis, and Ion Stoica, “Quanto: Tracking Energy in Networked Embedded Systems,” In Proceedings of the 8th USENIX Symposium on Operating Systems Design and Implementation (OSDI), pp. 1-14, San Diego, CA, December 8-10, 2008.
28. T. Stathopoulos, D. McIntire, and W. J. Kaiser, “The energy endoscope: Realtime detailed energy accounting for wireless sensor nodes,” in Proceedings of the 7th international conference on Information processing in sensor networks (IPSN), pp. 383–394, Washington, DC, 2008.
29. Green Hills Software Inc., “Processor Probes,” At: http://www.ghs.com/products/debugdevices.html. Last accessed: June, 2017.
30. Rajesh Krishna Panta, Saurabh Bagchi, and Samuel P. Midkiff, “Zephyr: Efficient Incremental Reprogramming of Sensor Nodes using Function Call Indirections and Difference Computation,” At the USENIX Annual Technical Conference (USENIX '09), June 14-19, 2009, pp. 411-424, San Diego, CA.
31. Rajesh Krishna Panta, Madalina Vintila, and Saurabh Bagchi, “Fixed Cost Maintenance for Information Dissemination in Wireless Sensor Networks,” At the 29th IEEE Symposium on Reliable Distributed Systems (SRDS), pp. 54-63, October 31-November 3, 2010, New Delhi, India.
32. Amiya Kumar Maji, Kangli Hao, Salmin Sultana, and Saurabh Bagchi, “Characterizing Failures in Mobile OSes: A Case Study with Android and Symbian,” At the 21st Annual International Symposium on Software Reliability Engineering (ISSRE), pp. 249-258, Nov 1-4, 2010, San Jose, California.
33. Amiya K. Maji, Fahad A. Arshad, Saurabh Bagchi, and Jan S. Rellermeyer, “An Empirical Study of the Robustness of Inter-component Communication in Android,” In Proceedings of the 42nd IEEE/IFIP International Conference on Dependable Systems and Networks (DSN), pp. 1-12, Boston, MA, June 25-28, 2012.
34. Subrata Mitra, Ignacio Laguna, Dong H. Ahn, Todd Gamblin, Martin Schulz, and Saurabh Bagchi, “Scalable Parallel Debugging via Loop-Aware Progress Dependence Analysis,” Accepted to appear as a poster at the 25th IEEE/ACM International Conference for High Performance Computing, Networking, Storage and Analysis (Supercomputing), pp. 1-2, November 17-22, 2013.
35. Fahad Arshad, Rebecca Krause and Saurabh Bagchi, “Characterizing Configuration Problems in Java EE Application Servers: An Empirical Study with GlassFish and JBoss,” Accepted to appear at the 24th

IEEE International Symposium on Software Reliability Engineering (ISSRE), pp. 1-10, Pasadena, CA, November 4-7, 2013.

36. Ignacio Laguna, Subrata Mitra, Fahad A. Arshad, Nawanol Theera-Ampornpunt, Zongyang Zhu, Saurabh Bagchi, Samuel P. Midkiff, Mike Kistler (IBM Research), and Ahmed Gheith (IBM Research), "Automatic Problem Localization in Distributed Applications via Multi-dimensional Metric Profiling," Accepted to appear at the 32nd International Symposium on Reliable Distributed Systems (SRDS), pp. 1-10, Braga, Portugal, September 30-October 3, 2013.
37. Bowen Zhou, Jonathan Too, Milind Kulkarni, and Saurabh Bagchi, "WuKong: Automatically Detecting and Localizing Bugs that Manifest at Large System Scales," At the 22nd International ACM Symposium on High Performance Parallel and Distributed Computing (HPDC), pp. 131-142, New York City, New York, June 17-21, 2013.
38. Lawrence Livermore National Lab and Purdue University, "AutomaDeD: Debugging Tool based on Statistical Analysis," Open source release at: https://github.com/scalability-llnl/AutomaDeD, Release date: September 19, 2014.
39. Ahn, Dong H., Bronis R. de Supinski, Ignacio Laguna, Gregory L. Lee, Ben Liblit, Barton P. Miller, and Martin Schulz. "Scalable temporal order analysis for large scale debugging." In Proceedings of the Conference on High Performance Computing Networking, Storage and Analysis (Supercomputing), pp. 1-11, 2009.
40. Subrata Mitra, Ignacio Laguna, Dong H. Ahn (LLNL), Saurabh Bagchi, Martin Schulz (LLNL), and Todd Gamblin (LLNL), "Accurate Application Progress Analysis for Large-Scale Parallel Debugging," At the ACM International Symposium on Programming Language Design and Implementation (PLDI), pp. 193-203, Edinburgh, UK, June 9-11, 2014.
41. Koller, Ricardo, Akshat Verma, and Anindya Neogi. "WattApp: an application aware power meter for shared data centers." In Proceedings of the 7th international conference on Autonomic computing, pp. 31-40. ACM, 2010.
42. Varadarajan, Venkatanathan, Thawan Kooburat, Benjamin Farley, Thomas Ristenpart, and Michael M. Swift. "Resource-freeing attacks: improve your cloud performance (at your neighbor's expense)." In Proceedings of the 2012 ACM conference on Computer and communications security (CCS), pp. 281-292, 2012.
43. R. Di Pietro and L. V. Mancini. Intrusion Detection Systems. Springer Publishing Company, First Edition, 2008.
44. Gaspar Modelo-Howard, Christopher Gutierrez, Fahad Ali Arshad, Saurabh Bagchi, and Yuan Qi, "pSigene: Webcrawling to Generalize SQL Injection Signatures," At the 44th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN), pp. 45-56, June 23-26, 2014, Atlanta, GA.
45. Rajesh Krishna Panta, Issa Khalil, and Saurabh Bagchi, "Stream: Low Overhead Wireless Reprogramming for Sensor Networks," At the 26th Annual IEEE Conference on Computer Communications (INFOCOM), pp. 928-936, May 6-12 2007, Anchorage, Alaska, USA.

46. Mark D. Krasniewski, Rajesh K. Panta, Saurabh Bagchi, Chin-Lung Yang, and William J. Chappell, “Energy-efficient, On-demand Reprogramming of Large-scale Sensor Networks,” ACM Transactions on Sensor Networks (TOSN), Volume 4, Issue 1, pp. 1-38, January 2008.
47. Saurabh Bagchi, Ness B. Shroff, Issa Khalil, Rajesh K. Panta, Mark D. Krasniewski, James V. Krogmeier, “Protocol for Secure and Energy-Efficient Reprogramming of Wireless Multi-hop Sensor Networks,” US Patent 8,811,188, granted: August 2014, filed: January 2012.
48. Matthew Tancreti, Vinaitheerthan Sundaram, Saurabh Bagchi, and Patrick Eugster, “TARDIS: Software-Only System-Level Record and Replay in Wireless Sensor Networks,” At the 14th ACM/IEEE Conference on Information Processing in Sensor Networks (IPSN), pp. 286-297, April 13-17, 2015, Seattle, WA.
49. Ayush Patwari, Ignacio Laguna (LLNL), Martin Schulz (LLNL), and Saurabh Bagchi, “Understanding the Spatial Characteristics of DRAM Errors in HPC Clusters,” Accepted to appear at the 7th Fault Tolerance for HPC at eXtreme Scales (FTXS) Workshop (co-located with HPDC), pp. 1-6, Jun 26, 2017, Washington DC.
50. Subrata Mitra, Suhas Raveesh Javagal, Amiya K. Maji (ITaP), Todd Gamblin (LLNL), Adam Moody (LLNL), Stephen Harrell (ITaP), and Saurabh Bagchi, “A Study of Failures in Community Clusters: The Case of Conte,” At the 7th IEEE International Workshop on Program Debugging (IWPD), co-located with ISSRE, pp. 1-8, Oct 23-27, 2016, Ottawa, Canada.
51. Subrata Mitra, Greg Bronevetsky (Google), Suhas Javagal, and Saurabh Bagchi. "Dealing with the unknown: Resilience to prediction errors." At the International Conference on Parallel Architecture and Compilation (PACT), pp. 331-342, October 18-21, 2015, San Francisco, CA.
52. Greg Bronevetsky (Google), Ignacio Laguna (LLNL), Saurabh Bagchi, and Bronis R. de Supinski (LLNL), “Automatic Fault Characterization via Abnormality-Enhanced Classification,” At the 42nd Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN), pp. 1-12, June 25-28, 2012, Boston, MA.
53. Tara Thomas, Anmol Bhattad, Subrata Mitra, and Saurabh Bagchi, “Probabilistic data assertions to detect silent data corruptions in parallel programs,” In Proceedings of the IEEE 35th Symposium on Reliable Distributed Systems (SRDS), pp. 41-50, September 26-29, 2016, Budapest, Hungary.
54. Juan D. Rodriguez, Aritz Perez, and Jose A. Lozano. "Sensitivity analysis of k-fold cross validation in prediction error estimation." IEEE transactions on pattern analysis and machine intelligence 32, no. 3 (2010): 569-575.
55. Andrew Wilson, Elad Gilboa, John P. Cunningham, and Arye Nehorai. "Fast kernel learning for multidimensional pattern extrapolation." In Advances in Neural Information Processing Systems, pp. 3626-3634. 2014.
56. Shanchieh Jay Yang, Jared Holsopple, and Moises Sudit. "Evaluating threat assessment for multi-stage cyber attacks." In Military Communications Conference, 2006. MILCOM 2006. IEEE, pp. 1-7. IEEE, 2006.
57. Subrata Mitra, Manish Gupta, Sasa Misailovic (U of Illinois at Urbana-Champaign), Saurabh Bagchi, “Phase-Aware Optimization in Approximate Computing,” In Proceedings of the 2017 IEEE/ACM

International Symposium on Code Generation and Optimization (CGO), pp. 185-196, Feb 4-8, 2017, Austin, TX.

58. Stelios Sidiroglou-Douskos, Sasa Misailovic, Henry Hoffmann, and Martin Rinard. "Managing performance vs. accuracy trade-offs with loop perforation." In Proceedings of the 19th ACM SIGSOFT symposium and the 13th European conference on Foundations of software engineering, pp. 124-134. ACM, 2011.
59. Sui, Xin, Andrew Lenharth, Donald S. Fussell, and Keshav Pingali. "Proactive control of approximate programs." ACM SIGOPS Operating Systems Review (ASPLOS) 50, no. 2, pp. 607-621, 2016.
60. Kanak Mahadik, Christopher Wright, Jinyi Zhang, Milind Kulkarni, Saurabh Bagchi, and Somali Chaterji. "SARVAVID: A Domain Specific Language for Developing Scalable Computational Genomics Applications." In Proceedings of the 2016 International Conference on Supercomputing (ICS), pp. 1-12, June 1-3, 2016, Istanbul, Turkey.
61. Wei Tang, Jared Bischof, Narayan Desai, Kanak Mahadik, Wolfgang Gerlach, Travis Harrison, Andreas Wilke, and Folker Meyer. "Workload characterization for MG-RAST metagenomic data analytics service in the cloud." In Big Data (Big Data), 2014 IEEE International Conference on, pp. 56-63. IEEE, 2014.
62. Andreas Wilke, Jared Bischof, Wolfgang Gerlach, Elizabeth Glass, Travis Harrison, Kevin P. Keegan, Tobias Paczian, William L. Trimble, Saurabh Bagchi, Ananth Grama, Somali Chaterji, and Folker Meyer, “The MG-RAST metagenomics database and portal in 2015,” Nucleic Acids Research, volume 44 (Database Issue), pp. 590-594, December 2015.
63. Paul Wood, Saurabh Bagchi, and Alefiya Hussain (USC/ISI), “Profiting from Attacks on Real-Time Price Communications in Smart Grids,” At the 9th IEEE International Conference on Communication Systems and Networks (COMSNETS), pp. 1-8, Jan 4-8, 2017, Bangalore, India.
64. Paul Wood, Saurabh Bagchi, and Alefiya Hussain (USC/ISI), “Defending Against Strategic Adversaries in Dynamic Pricing Markets for Smart Grids,” At the 8th International Conference on Communication Systems and Networks (COMSNETS), pp. 1-8, January 5-9, 2016, Bangalore, India.
65. Ashish R. Hota, Abraham A. Clements, Shreyas Sundaram, and Saurabh Bagchi, “Optimal and Game-Theoretic Deployment of Security Investments in Interdependent Assets,” At the 7th Conference on Decision and Game Theory for Security (GameSec), pp. 1-13, Nov 2-4, 2016, New York City, New York.
66. Tawfeeq Shawly, Jun Liu, Nathan Burow, Saurabh Bagchi, Robin Berthier (University of Illinois at Urbana-Champaign), and Rakesh B. Bobba (University of Illinois at Urbana-Champaign), “A Risk Assessment Tool for Advanced Metering Infrastructures,” At the 5th IEEE International Conference on Smart Grid Communications (SmartGridComm), pp. 989-994, November 3-6, 2014.
67. Abraham A Clements, Naif Saleh Almakhdhub, Khaled Saab, Prashast Srivastava, Jinkyu Koo, Saurabh Bagchi, and Mathias Payer, “Protecting Bare-metal Embedded Systems with Privilege Overlays,” At the IEEE International Symposium on Security and Privacy (Oakland), pp. 1-15, May 22-24, 2017, San Jose, California.

68. Naixing Wang, Edgardo Barsallo Yi, and Saurabh Bagchi, “On Reliability of Android Wearable Health Devices,” pp. 1-2, https://arxiv.org, 2017.