Survivability Applications

This sequence is based on the paper:


Other material is from the references of that publication

The focus here is on system architectures for survivability and formal analysis tools.
Multi-core Systems

- They are here and they will grow!

- Assumptions about the future of multi-core
  - number of cores is increasing
  - most applications still have limited means of using multi-threading
  - degree of parallelism is bound by the largest anti-chain of the execution graph
  - implications on speedup
Reliability and Redundancy

- Redundancy has greatly benefitted reliability
- In the past: homogeneous redundancy
- New focus on heterogeneous redundancy
  - avoidance of common mode faults
Common Mode Faults

- If a SW/HW component fails under a certain input, then it does not matter how many identical components one uses for redundancy => they all fail

- Dissimilarity as an approach toward independence of faults

- Two main approaches
  - N-version software
  - N-variant software
N-version Software

- N-version programming (late 70s)
  - software is derived by multiple teams from the same specification in isolation
  - expectation: common mode fault is reduced or eliminated
  - different results by different versions indicate fault
- limitations
  - how dissimilar are implementations?
  - is there true independence of development?
  - how does one measure the “degree of dissimilarity”?
N-variant Software

- Inspired by N-version software

- Different variants are generated in a more “automated” fashion

- Expectation is that a fault affecting on variant will not affect another in an identical way

- Again, differences detected by different variants indicate fault
Resilient Multi-core systems

- Utilize idle resources to increase resilience
- Specifically

Utilize idle cores for resilience mechanisms
Related work


- Focus on transient faults

Figure 1. Many-core model with replica partition.
Related work [Cox2006]

  - A set of automatically diversified variants execute on same inputs
  - Difference in referencing memory is observed
  - Identifies execution of injected code
  - Check out section 3. Model of their paper
Related work [Cox2006]

Example of two variants using disjoint memory space. Any absolute memory access will be invalid in one the variants.

Figure 1. N-Variant System Framework.
Security through redundant data diversity

Anh Nguyen-Tuong, David Evans, John C. Knight, Benjamin Cox, Jack W. Davidson

attack classes, regardless of the vulnerability
expressed. The N-variant architecture enables high-
keys used to generate the variants can be openly pu-
positions used to generate variants can be simple and t-
assumptions about keeping secrets. The transforma-
redundancy, using an architecture that combines
contain security vulnerabilities.

In general, diversity techniques attempt to thwart
services in the service software. Despite much eff
thwarts attacks that corrupt UID values.

provide high-assurance arguments against a class of
for given attack classes). In this work, we develop
ble to compromise all the variants with the same in
they use redundancy (to require an attacker to
Unlike other diversity-based approaches, N-variant

As shown in Figure 1, each variant has a different
address space partitioning.

Figure 1. Two-variant address partitioning.

In the 38
th
stems and Networks

Figure 2. N-Variant Systems with Data Diversity.

Our earlier work introduced N-variant systems and
preparation function (\(P_{i}\)). The detection property states that if one variant i

consider a data variation for a target type

The detection property states that if one variant i

All trusted data of type

App Interpreter

App Interpreter

Malicious Data

Normal Data

Malicious Data

Normal Data

Target Interpreter

Target Interpreter

\(R_i(D)\)

\(R_i(D)\)

\(P_0\)

\(P_1\)

\(P_0^{-1}\)

\(P_1^{-1}\)

\(P_0\)

\(P_1\)

\(P_0\)

\(P_1\)

\(R_i(D)\)

\(R_i(D)\)

\(P_0\)

\(P_1\)

\(P_0\)

\(P_1\)
Related work [Salamat2008]

B. Salamat, et. al. 2008

- Multi-Variant Program Execution: Using Multi-Core Systems to Defuse Buffer-Overflow Vulnerabilities
- International Conference on Complex, Intelligent and Software Intensive Systems
- Variants use different direction in memory allocation
- Buffer overflow “crashes” into different neighboring memory
Figure 1. System calls that change the global state are executed by the monitor and the results are communicated to all instances.
General Scheme

- Execution of multiple versions masks or detects faults

- Overhead
  - N-folding amount of work
  - Redundancy management

- What can be absorbed?
Two Step Approach

- Specification Model
- Layered adaptive architecture
Specification Model

- Adaptive Functional Capability Model (AFCM)
  - System comprised of functionalities $F_1 \cdots F_m$
  - Core operations that are mission critical
  - Non-critical, but value-added operations

\[
F_1^n \preceq F_1^m \preceq F_1^3
\]
Example: Multi-level Secured Record Keeping
Example

- **Secured database system** $D$
  - each record in $D$ contains two sets of data, i.e., $d = \{d_1, d_2\}$
  - $d_1$ contains mission critical data
  - $d_2$ non-mission critical, but value-added data
The Monitoring and Reconfiguration Module (MRM) is responsible for accessing the record that is associated with a particular user's information. This record contains a value-added data. For a registered client:

- New operations in deterministic simplicity: we assume that the encryption algorithm in use is user-encrypted using the public/private key pairs.

Furthermore, states that each sequence of operations in a system can be defined in terms of inclusion relation if and only if.

In such a system, we can define that each layer contains two sets of data: set $F_0$ and a non-mission-critical set $F_1$. Each record $K_0$ is a decreasing function of $F_0$ and each registered user's public key $O_0$. We write $d_i$ for $O_0$. The functional capabilities among a user: MRM:

- Recovery to a lower capability levels defined in AFCM: e.g., recovery to a lower capability levels, which specify reconfiguration requirements.

First, the model associates each functionality with capabilities. The purpose of AFCM is to specify not only functional reconfigurability but also to ensure that the system can operate correctly.

Each record $K_0$ is a set of non-mission-critical data. In such a system, we can define that each layer extends a sequence of operations in the next layer down.

For example, consider the piece-wise implementation in the next layer down:

![UML Sequence Diagram](image-url)
A registered user keeps his/her private key encrypted using the public/private key pairs. For instance, in a transaction, based on application context:

1. The underlying database contains two sets of data: Set \( D_0 \) and Set \( D_1 \).
2. The system is defined by the set of successful interactions.
3. Each layer is implemented by two layers: decreasing function and increasing function.
4. Sequence functionality of the system is defined and interpreted based on application context;
5. The adaptive secured database system contains actual records and it can only be accessed by the interface between a user and the database.

Figure: Sequence diagram for an adaptive secured database system.
Layered N-variant Architecture

- Multiple functionalities:
  - System is a collection of functionalities

Diagram showing layered architecture with various functionalities and N-variants implemented in different layers. The Monitoring and Reconfiguration Module (MRM) is highlighted as a key component for system reconfiguration and real-time fault tolerance.
Adaptability and Reconfiguration

Layers have two purposes
- lower layer monitors higher layer
- layers are basis for reconfiguration
- disagreement results in
  - scaling back to lower layer
  - graceful degradation
Special Cases

- Limitation of current research
  - all functionalities are defined on same layer

- Salamat, et. al. 2008
  - use two variants at the same layer, i.e., layer $L_1$
    
    $V_1^1$ and $V_2^1$
  
  - the two variants focus on memory referencing
Special Cases

- Cox, et. al. 2006
  - use variants at the same layer, i.e., layer $L_1$
  - the variants focus on memory referencing
Matching expectations

- Specify a suitable system
  - get an idea with GSPN model (Gen. Stochastic Petri Nets)
    - see if/how goal can be met
    - see if the overhead realistic

- Implementation
  - probabilistic automaton-based model
    - closer to real behavior
    - starting point towards implementation
Petri Nets

From Markov Chains to Petri Nets

- discussion on Markov Chains
- discussion on Petri Nets

you will not be an expert based on this discussion, but you should understand the general ideas, the strength and mathematical/computational limitations.
extra page for notes
Reliability and Resilience
Reliability and Resilience

Cross-layer monitor

Layer control

Layer implementation

Abstract

Introduction

Common applications still allow little parallelism and it is likely that cores may be underutilized or running idle. If variants can execute on idle cores this overhead has to potential to be reasonably absorbed. If variants can execute on the same inputs, any difference in output will automatically shut down layer i and used to detect the execution of injected code. Simultaneously, in Figure 4, capability of security and is achieved by comparing "w Kringsu et al." and "Vizienis u" and is typically discussed in the context of N-version programming. It is assumed that several software variants execute in order to allow detection or masking of errors, as was shown in [8].

The availability of unused or underutilized cores has been used to increase tolerance. Idle resources have also been used to increase security. For example, in [5] a set of automatically diversified variants execute on the same inputs. Any difference in output is detected.

Abstract

1 Introduction

Most new general purpose computers incorporate dual redundancy to benefit reliability. Redundant executions have been explored in the context of N-version programming. In N-version programming it is assumed that several software variants execute in order to allow detection or masking of errors. Any difference in output is detected. For example, in [5] a set of automatically diversified variants execute on the same inputs. Any difference in output is detected. In [6] it is shown that monitoring higher layers can effectively cross-monitor lower layers, which has positive effects when protected between layers the shut-down of the higher layer is not necessary.

The availability of unused or underutilized cores has been used to increase tolerance. Idle resources have also been used to increase security. For example, in [5] a set of automatically diversified variants execute on the same inputs. Any difference in output is detected.
Cross-layer monitoring scope

\[ L^{i+1} \]

\[ F^{i+1} \]

\[ L^i \]

\[ F^i \]

\[ L^{i+1} \]

\[ F^i \]

\[ F^{i+1} \setminus F^i \]

\[ L^i \]

\[ F^i \]
Stochastic Activity Networks

Example: Möbius

check out www.mobius.illinois.edu
SAN for cross-layer monitoring

Note the difference between GSPN and SAN (Stochastic Activity Network)

\[
L_{up}^{i+1} \quad F^i(L^i) \neq F^i(L^{i+1})
\]

\[
L_{up}^i \quad L_{down}^{i+1}
\]
Stochastic Models

- Evaluation of performance of architecture
  - model stochastic behavior using probabilistic models
  - use probabilistic model checking

- Metrics of interest
  - service availability
  - information security
Probabilistic Automata

N-tuple $\langle Q, \Theta, \delta, Q_0, F, P_\delta, P_0 \rangle$

1. $Q$ is a set of states,
2. $\Theta$ is a set of input symbols,
3. $\delta \subseteq Q \times \Theta \times Q$ is a set of transitions,
4. $Q_0 \subseteq Q$ is a set of start states,
5. $F \subseteq Q$ is a set of accepting states,
6. $P_\delta : \delta \to (0, 1]$ assigns each transition a probability, and
7. $P_0 : Q_0 \to (0, 1]$ assigns each start state a probability.
Probabilistic automaton: Example 1

$L^1$ of $F_1$
\( P_{k|n} \) is the probability that,

1. The maximal number of \( n \) variants producing the same result is \( k \), and;
2. The result is correct.

\( Q_{k|n} \) is the probability that,

1. The maximal number of \( n \) variants producing the same result is \( k \), and;
2. The result is incorrect.
\( v \), the number of working variants. The built-in voting mechanism decides the status of variants by simple majority. For example, if at the start of a clock cycle all 3 variants are working and during the cycle only 2 of 3 variants produce the same result, then the voting mechanism will mark these 2 variants as working, and the other one as \textit{not} working:
\( w \), the status of a layer. Initially all layers are working. If at one point the voting mechanism cannot decide which variant it can trust, for instance, in case that all 2 working variants report different value, it simply marks the layer as *not* working:

\[
(v^{(1)} = 3) \land w^{(1)} \land \neg e^{(1)} \quad \text{or} \quad (v^{(1)} = 2) \land w^{(1)} \land e^{(1)}
\]

\[
(v^{(1)} = 2) \land w^{(1)} \land \neg e^{(1)} \quad \text{or} \quad (v^{(1)} = 2) \land w^{(1)} \land e^{(1)}
\]

\[
(v^{(1)} = 2) \land \neg w^{(1)} \quad \text{or} \quad (v^{(1)} = 2) \land w^{(1)} \land e^{(1)}
\]

\[
(v^{(1)} = 3) \land w^{(1)} \land e^{(1)} \quad \text{or} \quad (v^{(1)} = 3) \land w^{(1)} \land \neg e^{(1)}
\]
$e$, the error flag. $e = true$ indicates that an erroneous output is produced by the layer. This could happen when, for example, all the working variants produce the exactly same erroneous output, although this is a very unlikely scenario especially when we apply N-variant technique. We will discuss this in more details later.
Monitoring and Reconfiguration Sub-Module in layer 2 (MRSM2)
Computational Experiments

Analysis used:

- Symbolic Hierarchical Automated Reliability/Performance Evaluator (SHARPE) to analyze GSPNs
- Probabilistic model checker PRISM to analyze the probabilistic automaton-based model
Figure 8. Probability of services being disabled for the GSPN model.
Figure 9. Probability of services being disabled for the probabilistic automaton-based model.
GSPN model

Probability Automaton-based model
Conclusions

- Hierarchical Formal Model was introduced
  - Adaptive Functional Capability Model (AFCM)
  - Multi-layer architecture
  - Adaptation capabilities
  - Reconfiguration capabilities
- Use Petri Net to deal with design specification experimentation
- Use model checking to go from design to implementation