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

![Diagram](image1.png)

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 of 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

[Nguyen-Tuong 2008]

Figure 1. Two-variant address partitioning. Figure 2. N-Variant Systems with Data Diversity.
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
Related work [Salamat2008]

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

\[
F_1^1 \preceq F_1^2 \preceq F_1^3 \quad \text{and} \quad F_2^2 \preceq F_2^1
\]
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
consists of $N$ variant modules for required reliability and security, with a record at level $f_{RID1}$; deterministic. Let's consider a functionality user is encrypted using the public/private key pairs; For a registered client:

$$F \text{ extends a sequence of operations in } K \text{ but value added data; For a registered client:}$$

$$I_{o}(0 \text{ data}(0), 1 + 1)$$

The underlying database $D$ serves as a reconfigurable design; For example: consider the piecewise adaptiveness; It has two features to serve its purpose: requirements: but also requirements for reconfiguration and functionality in $L$. Each record $D^i$ stores its value, added operations; We write $O_1$ of operations as $F(\text{new})$ where the piecewise adaptiveness in $D^i$ runs in $L$.
The Monitoring and Reconfiguration Module (MRM) consists of N variant modules for required reliability and security; it allows a registered user to retrieve a record by its record ID. For a registered client, the operation at step 1 involves sending a message containing encrypted data. In the event of a fault, the system's behavior at capability level $T_2$ is defined and interpreted based on application context; it states that: in the event of a fault, the value added data associated with the record is encrypted using the public/private key pairs. For instance: in a transaction, based asynchronous system: a client may also retrieve deterministic data but also requires reconfiguration. It has two features to serve its purpose: adaptiveness; it has two features to serve its purpose: adaptiveness; each layer's design in our framework will include a null model; the details of architecture design in our framework will include a null model. The Monitoring and Reconfiguration Module (MRM) consists of N variant modules for required reliability and security; it allows a registered user to retrieve a record by its record ID. For a registered client, the operation at step 1 involves sending a message containing encrypted data. In the event of a fault, the system's behavior at capability level $T_2$ is defined and interpreted based on application context; it states that: in the event of a fault, the value added data associated with the record is encrypted using the public/private key pairs. For instance: in a transaction, based asynchronous system: a client may also retrieve deterministic data but also requires reconfiguration. It has two features to serve its purpose: adaptiveness; it has two features to serve its purpose: adaptiveness; each layer's design in our framework will include a null model; the details of architecture design in our framework will include a null model.
Layered N-variant Architecture

- Multiple functionalities:
  - System is a collection of functionalities

![Diagram of Layered N-variant Architecture]

- Monitoring and Reconfiguration Module (MRM)
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 the same layer

- Salamat, et. al. 2008
  - Use two variants at the same layer, i.e., layer $L_1$
    $$V_{1}^1 \text{ and } V_{2}^1$$
  - 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.
Reliability and Resilience
Reliability and Resilience

cross-layer monitor

layer control

layer implementation

Abstract

A Hierarchical Formal Model for N-variant Executions in Multi-core Systems

Most new general purpose computers incorporate dual into similar to CSIIRW 6w Model \[\text{Analysis of use R@D of simple kfioffiN compofi]n]t\]o exploit sufficient parallelism to keep cores utilizedfl Most or quadficore processors and higher numbers of cores are on...
Cross-layer monitoring scope

\[ L^{i+1} \quad F^{i+1} \quad L^i \]

\[ L^{i+1} \quad F^i \quad F^{i+1} \setminus F^i \quad L^i \]
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)

\[ F^i (L^i) \neq F^i (L^{i+1}) \]
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 $\langle Q, \Theta, \delta, Q_0, F, P_\delta, P_0 \rangle$, where,

1. $Q$ is a set of states;
2. $\Theta$ is a set of input symbols;
3. $\delta \subseteq P \times \Theta \times P$ 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. In addition, for each $(p, a, p') \in \delta$, we have $\sum_{q \in \{q \mid (p, a, q) \in \delta\}} = 1$. That is, the accumulative probability of all the transitions enabled by a symbol $a$ shall be 1;
7. $P_0 : Q_0 \to (0, 1]$ assigns each transition a probability. In addition, $\sum_{q \in Q_0} P_0(q) = 1$. 

In this paper, we use an extension of probabilistic automata with probas. Tomaton extends transitions in a finite automaton with probas. A probabilistic automaton is a probabilistic automaton that models a system with probabilistic behavior. The first step of performance modeling and analysis is to build a stochastic model for the proposed N-variant architecture. To evaluate the performance of our architecture, we are interested in two metrics: service availability and information security.
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 \textit{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 \textit{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 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;
e, the error flag. \( e = \text{true} \) 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)

\[ \neg d^{(1)} \wedge w^{(1)} \wedge w^{(2)} \wedge \neg e^{(1)} \wedge \neg e^{(2)} \]  
\[ \neg d^{(2)} \wedge e^{(2)} \]  
\[ \neg d^{(2)} \wedge \neg e^{(2)} \]  
\[ [\neg d^{(1)} \wedge w^{(1)} \wedge w^{(2)} \wedge (e^{(1)} \wedge \neg e^{(2)}) \vee (e^{(1)} \wedge \neg e^{(2)})] ] : 1 - Q^{(1,2)} \]  
\[ [\neg d^{(1)} \wedge w^{(1)} \wedge w^{(2)} \wedge \neg e^{(1)} \wedge \neg e^{(2)}] ] : Q^{(1,2)} \]  
\[ [\neg d^{(1)} \wedge w^{(1)} \wedge w^{(2)} \wedge e^{(1)} \wedge e^{(2)}] ] : Q^{(1,2)} \]  
\[ d^{(2)} \]  
\[ d^{(1)} \vee \neg w^{(1)} \vee \neg w^{(2)} \]  
\[ d^{(1)} \wedge w^{(1)} \wedge w^{(2)} \wedge \neg e^{(1)} \wedge \neg e^{(2)} \]  
\[ d^{(1)} \wedge w^{(1)} \wedge w^{(2)} \wedge \neg e^{(1)} \wedge \neg e^{(2)} \]  
\[ d^{(1)} \wedge w^{(1)} \wedge w^{(2)} \wedge e^{(1)} \wedge e^{(2)} \]  
\[ d^{(1)} \wedge w^{(1)} \wedge w^{(2)} \wedge (e^{(1)} \wedge \neg e^{(2)}) \vee (e^{(1)} \wedge \neg e^{(2)})] ] \]
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