Efficient, Expressive, and Effective
Runtime Verification

-Ph.D. Thesis Defense-

Patrick O’Neil Meredith
Thesis Advisor:
University of Illinois at Urbana Champaign

Committee:
Grigore Roşu
Marco Caccamo
Klaus Havelund
Darko Marinov
Runtime Verification (RV)

- With an ever increasing reliance on software, we need more assurances that our software is reliable.
- Safety Critical:
  - Cellphones, Flight Control Systems, EFI
- Runtime Verification uses *runtime information* to check safety properties.
- More scalable than static verification.
Runtime Verification Areas

I worked on the following Runtime Verification areas

- Runtime Monitoring - monitor a system at runtime, attempt to recover in the face of errors
  - *Parametric* software monitoring
  - Hardware and Hardware/software monitoring
- Predictive Analysis - predict when safety properties could be violated
Runtime Monitoring Model

Program Execution
Runtime Monitoring Model

Program Execution

Observation/Abstraction

Event Trace
Runtime Monitoring Model

Program Execution

Observation/Abstraction

Event Trace

Verification

Monitors

M₁

M₂

M₃

...
Runtime Monitoring Model

Program Execution

Observation/Abstraction

Event Trace

Verification

Monitors

Handler

M1

M2

M3

Handler
Predictive Analysis Model

Program Execution

Observation/Abstraction

Event Trace

Thread 1

Thread 2
Predictive Analysis Model

Program Execution

Observation/Abstraction

Event Trace

Interleavings Generation

Causal Model

Thread 1

Thread 2
Predictive Analysis Model

Program Execution

Observation/Abstraction

Event Trace

Interleavings Generation

Verification

Monitors

Causal Model

Thread 1

Thread 2

M_1

M_2

M_3

...
Predictive Analysis Model

Program Execution

Observation/Abstraction

Event Trace

Interleavings Generation

Verification

Monitors

Handler

Thread 1

Thread 2

M_1

M_2

M_3

...
Thesis:

We can make monitoring Efficient Expressive Effective
Thesis:

We can make monitoring

Efficient

Expressive

Effective
Thesis:

We can make monitoring

Efficient

Expressive

Effective
Thesis:

We can make monitoring

Efficient

Expressive

Effective
Goal - Efficiency

- Optimizations for formalism-independent parametric monitoring that vastly improve performance

- Hardware and hardware/software monitoring that has effectively no runtime overhead

- Predictive Analysis that is multiple orders of magnitude faster than previous attempt (jPredictor)
Goal - Expressivity

- Formalism-generic algorithms for increasing parametric monitoring expressivity
- First monitoring algorithm for LTL with future and past
- First efficient monitoring algorithm for context free grammar properties
- First monitoring algorithm for string rewriting systems
Goal - Efficacy

- First formal property monitoring system for hardware and hardware/software systems
- First algorithm for predicting violations of generic parametric properties from a causal model derived from one program trace
Research Overview

Original Program

Static Analysis and Optimization

Logging (JVM)

Generating Event Trace

Trace Slicer

Instrumented Program

Event Trace

Control Dependence Info and Meta-Data

Vector Clocker

Property Checker

Prediction Violations and Location Information

Property

AspectJ

JavaMOP

JavaCFG

JavaSRS

JavaFSM

CFG

SRS

LTL

FSM

ERE

PTLTL

VerilogFSM

VerilogPTLTL

BusMOP

HDL
Building a Causal Model
Property Checking
Research Overview

JavaMOP
JavaSRS
JavaFSM
AspectJ

Original Program

Static Analysis and Optimization

Instrumented Program

Logging (JVM)

Event Trace

Trace Slicer

Control Dependence Info and Meta-Data

Vector Clocker

Property Checker

Prediction Violations and Location Information

Property

Generating Event Trace

Building a Causal Model

Property Checking
What I Have Done

- During my Ph.D. research, I have worked in the following areas:
  - Programming Language Semantics
  - Monitoring Oriented Programming
    - JavaMOP (software)
    - BusMOP (hardware and hardware/software)
  - Predictive Analysis
    - RV-Predict
- Thanks to all of my collaborators!
What I Have Done

- During my Ph.D. research, I have worked in the following areas:
  - Programming Language Semantics
  - Monitoring Oriented Programming
    - JavaMOP (software)
    - BusMOP (hardware and hardware/software)
  - Predictive Analysis
    - RV-Predict
- Thanks to all of my collaborators!
Programming Language Semantics

- Scheme - Scheme Workshop 2007
  - Patrick Meredith, Mark Hills, Grigore Roșu

- Verilog - MEMOCODE 2010
  - Patrick Meredith, Michael Katelman, José Meseguer, Grigore Roșu
JavaMOP
(MOP - Monitoring Oriented Programming)

- ASE 2008 (Sigsoft Distinguished Paper)
  - Patrick Meredith, Dongyun Jin, Feng Chen, Grigore Roşu
- Journal of ASE 2009
  - Patrick Meredith, Dongyun Jin, Feng Chen, Grigore Roşu
- ASE 2009
  - Feng Chen, Patrick Meredith Dongyun Jin, Grigore Roşu
- Journal of STTT 2011
  - Patrick Meredith, Dongyun Jin, Dennis Griffith, Feng Chen, Grigore Roşu
- PLDI 2011
  - Dongyun Jin, Patrick Meredith, Dennis Griffith, Grigore Roşu
- SRS Tech Report 2012
  - Patrick Meredith, Grigore Roşu
BusMOP

- RTSS 2008
  - Rodolfo Pellizzoni, Patrick Meredith, Marco Caccamo, Grigore Roșu

- Emsoft 2009
  - Rodolfo Pellizzoni, Patrick Meredith, Min-Young Nam, Mu Sun, Marco Caccamo, Lui Sha
RV-Predict

- RV 2010
- Patrick Meredith, Grigore Roșu
- RV-Predict Technical Report
- Patrick Meredith, Dennis Griffith, Michael Ilseman, Grigore Roșu

- RV-Predict is a product developed by our startup, Runtime Verification, Inc.
Research Overview

Original Program
- Static Analysis and Optimization
  - Instrumented Program
    - Logging (JVM)
      - Control Dependence Info and Meta-Data

- Generating Event Trace
- Building a Causal Model
- Property Checking
  - Trace Slicer
  - Vector Clocker
  - Property Checker
    - Prediction Violations and Location Information

Property

AspectJ

JavaMOP

JavaCFG
JavaSRS
JavaFSM

CFG
SRS
LTL
FSM
ERE
PTLTL

VerilogFSM
VerilogPTLTL

BusMOP

HDL
Building a Causal Model

Property Checking
Expressivity - Logic Plugins


- Finite State Machines (FSM)
- Extended Regular Expressions (ERE)
- Past Time Linear Temporal Logic (PTLTL)
- Linear Temporal Logic with Past and Future (LTL)
- Context Free Grammars (CFG)
- String Rewriting Systems (SRS)
Expressivity - Logic Plugins


- Finite State Machines (FSM)
- Extended Regular Expressions (ERE)
- Past Time Linear Temporal Logic (PTLTL)
- Linear Temporal Logic with Past and Future (LTL)
- Context Free Grammars (CFG)
- String Rewriting Systems (SRS)
Context-free Grammars (CFG)

Context Free Languages

Regular Languages
Expressivity - CFG


- First algorithm for efficiently monitoring CFGs

- CFGs are good for specifying *structured* properties

S -> P endThread
P -> P acquire P release | #epsilon
Efficiency - CFG


- Monitoring algorithm is based on modified GLR(1) parsing algorithm.

- For pattern languages (like CFG) we wish to match as soon as possible
  
  - A user expects the pattern \( e^+ \) to match every time \( e \) occurs

- This means we must check for matches every event
Efficiency - CFG
ASE 2008, J. ASE 2009

- But... (G)LR(1) uses a stack as state, checking for matches in the normal algorithm demolishes the stack contents

- We need the stack contents for the next event!

- Our algorithm can avoid touching the stack, performs the check in constant time
Efficiency - CFG
ASE 2008, J. ASE 2009

Percent Runtime Overhead

55 program-property pairs tested
String Rewriting Systems (SRS)

- Recursively Enumerable Languages
- Context-free Languages
SRS v. CFG
Expressivity - String SRS

SRS Tech Report

- First algorithm to monitor SRS
- SRS are made up of a series of rules

\[ \wedge \text{release} \rightarrow \#fail \]
\[ \text{acquire endThread} \rightarrow \#fail \]
\[ \text{acquire release} \rightarrow \#\text{epsilon} \]

- After each event arrives the rules are applied until a fixed point is reached
Efficiency - SRS

SRS Tech Report

- Matching extends the Aho-Corasick String searching algorithm (grep) to quickly find rule left-hand-sides (lhs)
- Matching finds all lhs in one pass, except those induced by replacements
- Replacement takes time proportional only to the right-hand-side (rhs) of the matched rule
- Fixed point algorithm optimized to terminate early
Efficiency - SRS

SRS Tech Report

Tested String
Rewriting System

1 0 -> 0 1  2 0 -> 0 2
2 1 -> 1 2  0 1 -> 3
1 3 -> 3 1  3 0 -> 0 3
3 2 -> #epsilon  2 3 -> #epsilon

Test Input

\[0^N1^N2^N\]

Time (ms)

<table>
<thead>
<tr>
<th>N</th>
<th>SRS</th>
<th>Maude</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>33</td>
<td>∞</td>
</tr>
<tr>
<td>1000</td>
<td>42</td>
<td>7112</td>
</tr>
<tr>
<td>5000</td>
<td>236</td>
<td>26132</td>
</tr>
<tr>
<td>10000</td>
<td>∞</td>
<td>∞</td>
</tr>
</tbody>
</table>
Efficiency - SRS

SRS Tech Report

Mean Percent Overhead (on 70 program/property pairs)

JavaMOP w/ ERE
JavaMOP w/ SRS

fop-HasNext

Mean Percent Overhead (on 70 program/property pairs)
Research Overview

Original Program

Static Analysis and Optimization

Logging (JVM)

Generating Event Trace

Trace Slicer

Vector Clocker

Property Checker

Prediction Violations and Location Information

Property Checking

Building a Causal Model

Generating Event Trace

Property

AspectJ

JavaMOP

JavaCFG

JavaSRS

JavaFSM

CFG

SRS

LTL

FSM

ERE

PTLTL

VerilogFSM

VerilogPTLTL

BusMOP

HDL

Control Dependence Info and Meta-Data

Instrumented Program

Event Trace
JavaMOP
JavaMOP

- Provides parametric monitoring for Java programs
  - Parametric algorithm is formalism-independent
- Generates AspectJ code
  - ~500 lines of code per property
- Is the most runtime efficient parametric runtime monitoring system of which we are aware
What is Parametric Monitoring?

- Events can have associated **parameters**
Parametric Monitoring Example

create_iter\langle c, i1 \rangle \quad \text{create_iter}\langle c, i_2 \rangle \quad \text{create_iter}\langle c, i_3 \rangle

\text{use_iter}\langle i_1 \rangle \quad \text{use_iter}\langle i_1 \rangle \quad \text{use_iter}\langle i_1 \rangle
Parametric Monitoring Example

0
create_iter\langle c, i \rangle
update_coll\langle c \rangle
use_iter\langle i \rangle
update_coll\langle c \rangle
create_iter\langle c, i \rangle
create_iter\langle c_1, i_1 \rangle
create_iter\langle c_1, i_2 \rangle
create_iter\langle c_2, i_3 \rangle
use_iter\langle i_1 \rangle
update_coll\langle c_2 \rangle
use_iter\langle i_1 \rangle

1

2
use_iter\langle i \rangle

3
use_iter\langle i \rangle
update_coll\langle c \rangle

\langle c_1, i_1 \rangle

Sunday, July 22, 12
Parametric Monitoring Example

create_iter<\textit{c},i>  
update_coll<\textit{c}>  
use_iter<i>

\begin{array}{c|c}
\text{<c\_1,i\_1>} & 1 \\
\text{create_iter<\textit{c\_1},i\_1>} & \\
\text{create_iter<\textit{c\_1},i\_2>} & \\
\text{create_iter<\textit{c\_2},i\_3>} & \\
\text{use_iter<i\_1>} & \\
\text{update_coll<\textit{c\_2}>} & \\
\text{use_iter<i\_1>} & \\
\end{array}
Parametric Monitoring Example

create_iter\langle c, i \rangle

0

update_coll\langle c \rangle

use_iter\langle i \rangle

1

update_coll\langle c \rangle

use_iter\langle i \rangle

2

create_iter\langle c, i \rangle

use_iter\langle i \rangle

3

update_coll\langle c \rangle

create_iter\langle c_1, i_1 \rangle

\langle c_1, i_1 \rangle 1

\langle c_1, i_2 \rangle 1

use_iter\langle i_1 \rangle

update_coll\langle c_2 \rangle

use_iter\langle i_1 \rangle
Parametric Monitoring Example

```
create_iter<c,i>
create_iter<c1,i1>
create_iter<c1,i2>
create_iter<c2,i3>
use_iter<i1>
use_iter<i>
update_coll<c1>
update_coll<c>
update_coll<c2>
update_coll<c>
```

<table>
<thead>
<tr>
<th>&lt;c1,i1&gt;</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;c1,i2&gt;</td>
<td>1</td>
</tr>
<tr>
<td>&lt;c2,i3&gt;</td>
<td>1</td>
</tr>
</tbody>
</table>

Sunday, July 22, 12
Parametric Monitoring Example

0
  update_coll<c>
  create_iter<c, i>

1
  use_iter<i>
  update_coll<c>
  create_iter<c, i>
  create_iter<c, i>
  create_iter<c, i>

2
  use_iter<i>
  update_coll<c>
  update_coll<c>

3
  update_coll<c>
  use_iter<i>

create_iter<c, i>
create_iter<c, i>
create_iter<c, i>
create_iter<c, i>
create_iter<c, i>
create_iter<c, i>
create_iter<c, i>
create_iter<c, i>
use_iter<i>
use_iter<i>
use_iter<i>
use_iter<i>
use_iter<i>
use_iter<i>
update_coll<c>
update_coll<c>
update_coll<c>
update_coll<c>
update_coll<c>
update_coll<c>
update_coll<c>
update_coll<c>

| <c1, i1> | 1 |
| <c1, i2> | 1 |
| <c2, i3> | 1 |
| <i1>     |   |
Parametric Monitoring Example

create_iter\(<c, i>\)

update_coll\(<c>\)

use_iter\(<i>\)

create_iter\(<c_1, i_1>\)
create_iter\(<c_1, i_2>\)
create_iter\(<c_2, i_3>\)
use_iter\(<i_1>\)
update_coll\(<c_2>\)
use_iter\(<i_1>\)

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;c_1, i_1&gt;</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>&lt;c_1, i_2&gt;</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>&lt;c_2, i_3&gt;</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>&lt;i_1&gt;</td>
<td>X</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Parametric Monitoring Example

create_iter\langle c, i \rangle

update_coll\langle c \rangle

use_iter\langle i \rangle

create_iter\langle c_1, i_1 \rangle
create_iter\langle c_1, i_2 \rangle
create_iter\langle c_2, i_3 \rangle
use_iter\langle i_1 \rangle
update_coll\langle c_2 \rangle
use_iter\langle i_1 \rangle

\langle c_1, i_1 \rangle \quad 1
\langle c_1, i_2 \rangle \quad 1
\langle c_2, i_3 \rangle \quad 1
\langle i_1 \rangle \quad X

Sunday, July 22, 12
Parametric Monitoring Example

0 -> 1
update_coll\(<c>\) -> use_iter\(<i>\) -> create_iter\(<c, i>\)

1 -> 2
use_iter\(<i>\) -> update_coll\(<c>\)

2 -> 3
update_coll\(<c>\) -> use_iter\(<i>\)

create_iter\(<c_1, i_1>\)
create_iter\(<c_1, i_2>\)
create_iter\(<c_2, i_3>\)
use_iter\(<i_1>\)
update_coll\(<c_2>\)
use_iter\(<i_1>\)

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>X</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;c_1, i_1&gt;</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>&lt;c_1, i_2&gt;</td>
<td></td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>&lt;c_2, i_3&gt;</td>
<td></td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>&lt;i_1&gt;</td>
<td></td>
<td></td>
<td>X</td>
</tr>
</tbody>
</table>
Parametric Monitoring Example

create_iter\langle c, i \rangle

update_coll\langle c \rangle

use_iter\langle i \rangle

create_iter\langle c_1, i_1 \rangle

create_iter\langle c_1, i_2 \rangle

create_iter\langle c_2, i_3 \rangle

use_iter\langle i_1 \rangle

update_coll\langle c_2 \rangle

use_iter\langle i_1 \rangle

 update_coll\langle c \rangle

 use_iter\langle i \rangle

 update_coll\langle c \rangle

<\langle c_1, i_1 \rangle, 1>

<\langle c_1, i_2 \rangle, 1>

<\langle c_2, i_3 \rangle, 2>

<\langle i_1 \rangle, X>

<\langle c_2 \rangle, 0>
Parametric Monitoring Example

create_iter\(<c, i>\)

update_coll\(<c>\)

use_iter\(<i>\)

create_iter\(<c_1, i_1>\)
create_iter\(<c_1, i_2>\)
create_iter\(<c_2, i_3>\)
use_iter\(<i_1>\)
update_coll\(<c_2>\)
use_iter\(<i_1>\)

| <c_1, i_1> | 1 |
| <c_1, i_2> | 1 |
| <c_2, i_3> | 2 |
| <i_1>      | X |
| <c_2>      | 0 |
| <c_2, i_1> | X |

Sunday, July 22, 12
Parametric Monitoring Example

create_iter\langle c, i \rangle
update_coll\langle c \rangle
use_iter\langle i \rangle

0

update_coll\langle c \rangle
use_iter\langle i \rangle

1

update_coll\langle c \rangle

2

use_iter\langle i \rangle

3

create_iter\langle c_1, i_1 \rangle
create_iter\langle c_1, i_2 \rangle
create_iter\langle c_2, i_3 \rangle
use_iter\langle i_1 \rangle
update_coll\langle c_2 \rangle
use_iter\langle i_1 \rangle

\begin{align*}
\langle c_1, i_1 \rangle & : 1 \\
\langle c_1, i_2 \rangle & : 1 \\
\langle c_2, i_3 \rangle & : 2 \\
\langle i_1 \rangle & : X \\
\langle c_2 \rangle & : 0 \\
\langle c_2, i_1 \rangle & : X
\end{align*}
Efficiency
ASE 2009

- We can leverage semantic knowledge of the property to determine which instances are unneeded.

- We have an algorithm that uses an abstraction of the property called *enable sets*.
  - We can generate this abstraction for FSM and CFG.

- Using the algorithm we only create the four instances we need!
Efficiency
ASE 2009

Percent Runtime Overhead

- bloat-UnsafeMapIterator: 935, OOM
- bloat-SafeSyncMap: 2267, 858, 660
- bloat-SafeIterator: 11258
- pmd-UnsafeMapIterator: 769, 749, 196

66 program-property pairs tested
Efficiency
PLDI 2011

- Enable sets are great, we create fewer monitors
- But what if a monitor becomes unnecessary?

- Previously, for \(<c_1, i_1>\) both \(c_1\) and \(i_1\) had to be garbage collected for \(<c_1, i_1>\) to be collected

- This is terrible! Iterators usually have far shorter lifespans than Collections
We use another abstraction, coenable sets, to determine when we can remove an instance.

Again, uses semantic knowledge of the property.

\(<c_1, i_1>\) can be collected once \(i_1\) dies!
Efficiency
PLDI 2011

Percent Runtime Overhead

75 program-property pairs tested
Expressivity
ASE 2008 and J. STTT 2011

- Suffix Matching
  - Attempt to match each suffix of the current trace
  - Useful for properties that may start monitoring at any point

- Parameter Binding Modes
  - Filters on which monitor instances may trigger handlers

- Connectedness
  - Filters out instances where parameters are not connected by events
Research Overview

- Original Program
- Static Analysis and Optimization
- Logging (JVM)
- Generating Event Trace
- Building a Causal Model
- Property Checking
- Trace Slicer
- Vector Clocker
- Property Checker
- Prediction Violations and Location Information

- AspectJ
- JavaMOP
- JavaCFG
- JavaSRS
- JavaFSM
- VerilogFSM
- VerilogPTLTL
- BusMOP
- HDL
- CFG
- SRS
- LTL
- FSM
- ERE
- PTLTL

Control Dependence Info and Meta-Data
BusMOP
BusMOP
RTSS 2008 and Emsoft 2009

- Provides formal property monitoring for hardware and software/hardware systems
  - Monitors reads and writes to system buses
- Generates HDL code
  - ~450 lines of code per property
- Is the only bus monitoring system for formal properties of which we are aware
Efficacy
RTSS 2008 and Emsoft 2009

- BusMOP used in two contexts
  - PCI Bus monitoring (RTSS 2008)
  - System on Chip monitoring (Emsoft 2009)
- BusMOP monitors are the same for each, only interface changes
Efficacy - PCI Bus
RTSS 2008
Efficacy - PCI Bus

RTSS 2008
Efficacy - PCI Bus

RTSS 2008

- Events are reads and writes on PCI Bus to specific memory locations or PCI interrupts.

- Monitors synthesized to FPGA plugged into bus.
  - Passively *sniffs* reads, writes and interrupts on bus.
Efficacy - PCI Bus
RTSS 2008

- Cannot stop bad writes, use roll-back mechanisms to recover when properties are violated

- Test case: ADC/DAC
  - Successfully formalized all driver requirements
  - e.g., Do not allow the clock divider to be set too low during conversion
Efficiency and Expressivity

RTSS 2008

- 0% overhead when there are no handlers
- Even with handler contention performance overhead is unmeasurably small
  - We tested with a property that issued a handler on every event
- Handler capabilities were enough to recover from every misbehavior in the specification
- Supports FSM, ERE, and PTLTL
Efficacy - SoC Design
Emsoft 2009
Efficacy - SoC Design
Emsoft 2009

- Architecture Analysis & Design Language (AADL) specification used to generate HDL/Software for SoC

- BusMOP properties added to AADL
  - Creates hardware monitors for each processor/hardware module
Efficacy - SoC Design
Emsoft 2009

- Events may refer to AADL memory entities
  - Still correspond to reads and writes

- Test case: pacemaker
  - Properties ensure no interference between critical and non-critical functions
  - e.g., do not allow logging process to write to memory used for pacing. If there are more than 1000 bad writes reset logging process
Efficiency and Expressivity - SoC Design

Emsoft 2009

- Only a small amount of latency is added, throughput is unaffected
- Handlers can actually stop bad behavior before it happens, unlike in the PCI case
- Handler capabilities were enough to recover in the face of misbehaving system components
- (Still) Supports FSM, ERE, and PTLTL
Research Overview
Original Program

Static Analysis and Optimization

Instrumented Program

Logging (JVM)

Event Trace

Control Dependence Info and Meta-Data

Trace Slicer

Vector Clocker

Property Checker

Prediction Violations and Location Information

Property Checking

Building a Causal Model

Generating Event Trace

Generating Event Trace
RV-Predict
RV 2010, RV-Predict Tech Report 2012

- We wish to predict properties even when they do not occur in an observed trace.
- jPredictor introduced concepts - provided prediction for races and atomicity violations
  - jPredictor is too inefficient to work with real world programs
  - jPredictor has no algorithm for predicting generic parametric properties
RV-Predict Pipeline

Original Program

Static Analysis and Optimization

Instrumented Program

Logging (JVM)

Event Trace

Control Dependence Info and Meta-Data

Trace Slicer

Vector Clocker

Property Checker

Prediction Violations and Location Information

Property

Building a Causal Model Generating Event Trace

Property Checking
Efficiency
RV 2010, RV-Predict Tech Report 2012

- Loop peeling to reduce number of events
- Trace Slicer must operate on reverse traces, log events in reverse, never reverse trace on disk
- Compress log with fast compression
- Pipelined stages so each event need be read only once from disk
Efficiency
RV-Predict Tech Report 2012

- jPredictor is 30x slower than RV-Predict on average without loop peeling in RV-Predict
- ~2000x slower in some cases
- RV-Predict uses 85x less disk space (consistently)
Efficiency and Efficacy

RV-Predict Tech Report 2012

- First algorithm for parametric property prediction
- Prunes much of the search space
- Specialized version for race detection
Conclusion - Efficient, Expressive, and Effective

- JavaMOP shows that formalism independent parametric monitoring can be efficient.
- BusMOP shows that Runtime Verification can be efficient and effective in a new domain.
- The various logic plugins show that Runtime Verification can be expressive without sacrificing efficiency.
- RV-Predict shows that Predictive Analysis can be efficient and effective. Its use of all logic plugins also makes it expressive.