Non Intrusive Detection of Errors in High Throughput Distributed Applications

Saurabh Bagchi
Dependable Computing Systems Lab (DCSL)
School of Electrical and Computer Engineering
Purdue University
Work done with: Ignacio Laguna, Fahad Arshad, Gunjan Khanna

Work supported by:
IBM, National Science Foundation (NSF), Purdue Research Foundation (PRF)

PURDUE UNIVERSITY

A Million Dollars an Hour:
The Cost of Downtime

Managing the Costs of System Downtime

By Robert Manning

September 7, 2004: Research indicates that IT system downtime costs American businesses $1 million an hour. *CIO Update* guest columnist Bob Manning identifies three core challenges of business continuity.

A million dollars an hour. That's what IT system downtime costs

Source: http://www.cioupdate.com

- Average ***system downtime cost*** is $1 million / hour for American businesses
  - Source: Meta Group, 2002

- 39% outages, 61% degradations (service is slower than usual; almost unavailable)
  - Source: Infonetics Research, 2007

PURDUE UNIVERSITY

Service Degradation Example: *The "It works for me" Syndrome*

Bob
(*User*)

1. Logs in his *books.com*
2. Cannot add book to shopping cart
3. Complains to Customer Support

Alice
(*System Administrator*)

4. She is perfectly able to order from her machine
5. Not able to *detect* and *diagnose* the problem

Requirements of a Failure Detection System

- Detect failures and anomalies while the application is executing, i.e., *online detection*
  - *Why?* May trigger diagnosis and response, which limit the effect of the failure
- *Non-intrusive* to the application
  - Does not require significant changes to the application
  - Does not degrade application's performance
  - Why?
  1. Application source code may not be available or be completely understood
  2. Performance requirement is obvious
- Generalizable detection system
  - Can detect a broad spectrum of failures
  - Easy to translate to new applications

Roadmap

- Monitor Detection System
- Dealing with High Rate of Messages through Sampling
- Intelligent Sampling Approach
- Hidden Markov Model-based State Estimation
- Experiments and Results
- Optimization of Rules to Match
- Conclusion

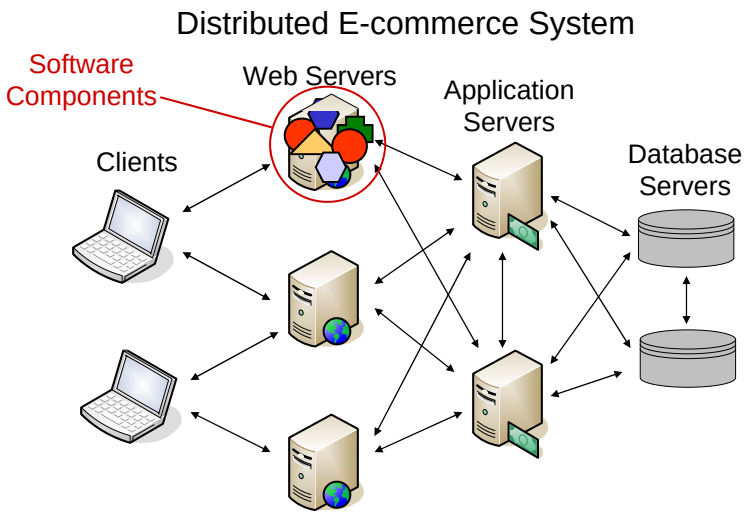
PURDUE UNIVERSITY

Application Model

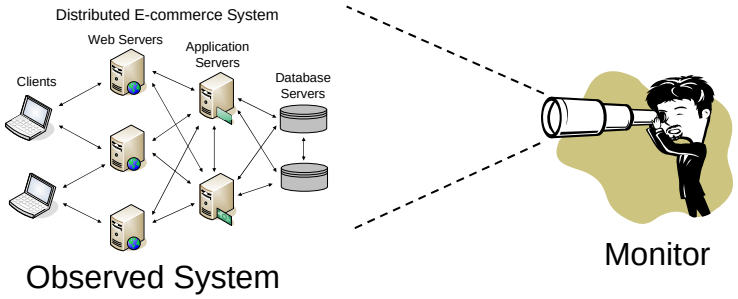
- Distributed application with multiple services
- Each service has multiple components
- Each component interacts using well-defined interfaces
  - J2EE calls
  - RPC
  - Network calls
- The interfaces can be tapped into for accessing the interactions
- The components can be stateful

PURDUE UNIVERSITY

Sample Application

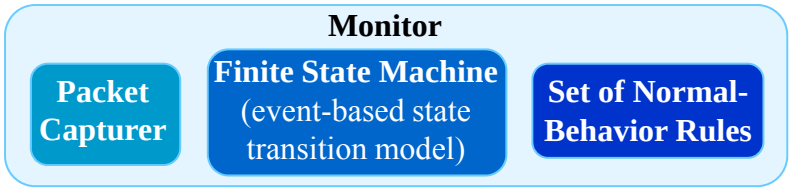


The *Monitor* Detection System



- Using an external Monitor to detect network protocol errors
- Monitor could be co-located or in vicinity of a component
- Observe messages exchanged
- Match against a rule base (Anomaly based)

Stateful Rule-based Detection



Sequence of Events in Detection

A *message* is captured → (1) Deduce current application *state* → (2) Match rules based on the deduced *state* → (3) If a rule does not satisfy, signal *alarm* (4)

Structure of Rule Base

- Rule matching engine invoked by State Maintainer
- Rules defined based on protocol specifications *and* QoS requirements.
- Rules are anomaly based
- Rules can be
  - Combinatorial: Valid for entire duration except for transients
  - Consists of expressions of state variables arranged as an expression tree yielding Boolean result
  - Temporal: Associated time component for precondition and postcondition

Temporal Rules

Type I:

$S_p = \text{true for } T \in (t_N, t_N + k) \Rightarrow S_q = \text{true for } T \in (t_I, t_I + b)$

Type II:

S_t is the state of an object at time t : $S_{t+\Delta} \neq S_t$, if event E_i takes place at t

Type III:

$L \leq |V_t| \leq U \ \forall t \in (t_i, t_i + k)$

Type IV:

$\forall t \in (t_i, t_i + k)\ L \leq |V_t| \leq U \Rightarrow L' \leq |B_q| \leq U',\ \forall q \in (t_n, t_n + b)$

Normal-Behavior Rules in Monitor

- Sample rules from two applications:
  - TRAM, reliable multicast protocol (*TDSC '06*)
  - PetStore, three-tier e-commerce system (*SRDS '07*)

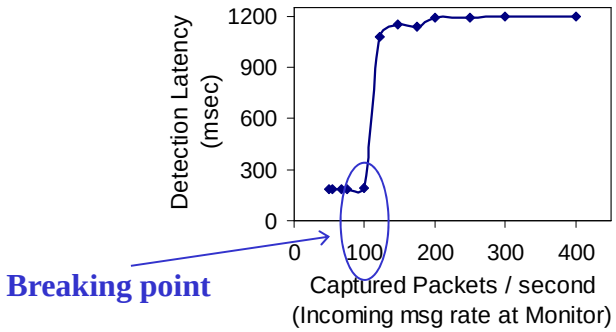
TRAM	Three-Tier System
A receiver must send an ACK *after receiving 20* DATA *If a sender receives a* HELLO*, it should send a* REPLY	*If function* `CalculateBalance` *is called in* `ShoppingCard` *component, it should reply in less than 50 msec*

Roadmap

- Monitor Detection System
- Dealing with High Rate of Messages through Sampling
- Intelligent Sampling Approach
- Hidden Markov Model-based State Estimation
- Experiments and Results
- Optimization of Rules to Match
- Conclusion

PURDUE UNIVERSITY

Breaking Point with High Rate of Messages



- After breaking point, latency increases sharply
- True alarms decrease because packets are dropped
- Breaking point expected in any stateful detection systems
- Goal: Graceful degradation of detection latency and accuracy

PURDUE UNIVERSITY

Avoiding the Breaking Point: *Random Sampling*

- *Processing Load* in Monitor = $R_{in} \times C$,
  R_{in}: Incoming message rate, C: Processing cost per message
- We reduce *Processing Load* by reducing R_{in}
  - Only a fraction of incoming messages is processed
  - *n* out of *m* messages are randomly sampled
- Detection performed only with sampled messages
- Sampling is activated if $R_{in} \geq$ *breaking point* (or close to *breaking point*)

PURDUE UNIVERSITY

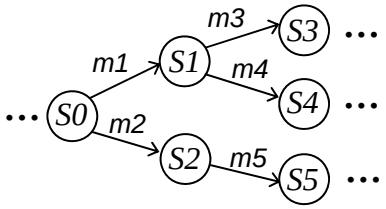
How Do We Sample? First Shot …

- We choose uniform random sampling: rate of sampling is dependent on the rate of incoming messages
  - Uniform random method is oblivious to the incoming message type
  - Any sampling approach based on information in the incoming message will require processing before sampling
- Drop message at the rate of 1 in every $R_{in} /(R_{in} - R_{th})$ messages
  - Incoming rate is recalculated after a window of 30 seconds
- Scale the constants in the rules by a factor of R_{th} / R_{in}
  - Original rule — "Receive 10 Acks in 100 sec"
  - Rule modified — "Receive 10.(R_{th} / R_{in}) Acks in 100 sec"

PURDUE UNIVERSITY

Non-Deterministm Caused by Sampling

A portion of a Finite State Machine



Events in Monitor

(1) SV = { *S0* }
(2) A message is dropped
(3) SV = { *S1*, *S2* }
(4) A message is sampled
(5) The message is *m5*
(6) SV = { *S5* }

Definitions:

State Vector (SV) — The state(s) of the application from Monitor's point of view (deduced state(s))

Non-Determinism — Monitor is no longer aware of the exact state the application is in (because of dropped messages)

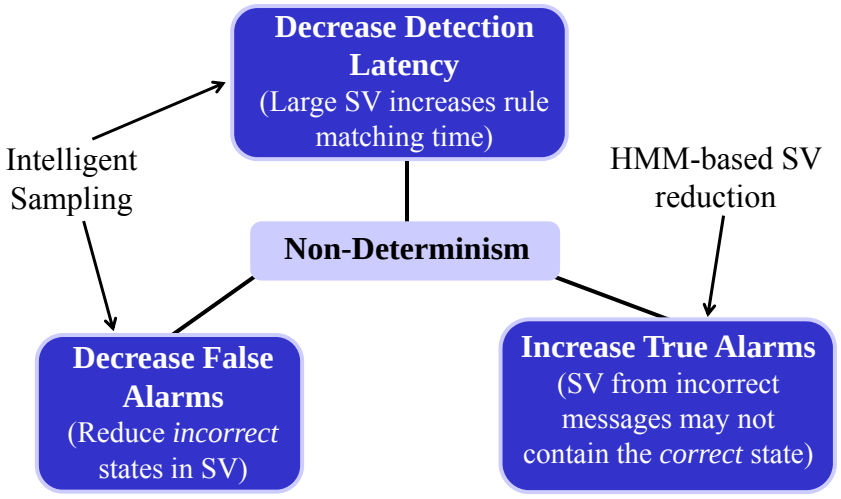
PURDUE UNIVERSITY

Roadmap

- Monitor Detection System
- Dealing with High Rate of Messages through Sampling
- Intelligent Sampling Approach
- Hidden Markov Model-based State Estimation
- Experiments and Results
- Optimization of Rules to Match
- Conclusion

PURDUE UNIVERSITY

Challenges with Non-Determinism



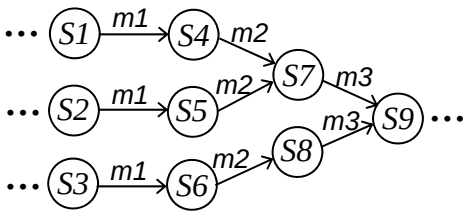
Intelligent Sampling

- We want to reduce SV when sampling
  - Reduce detection latency and false alarms
- All incoming messages are *observed* at runtime
  - Monitor determines their type in the application's FSM
  - Example: Is the message an ACK packet?
    Is the message a *CalculateBalance* function call?
- Only messages with a *desirable* property are sampled

What is a Desirable Property of a Message?

Suppose, SV={ *S1*, *S2*, *S3* }, *Sampling Rate* = 1/3

A portion of a Finite State Machine



Sampled Message is: ⇒	**SV**
m1	{ *S4*, *S5*, *S6* }
m2	{ *S7*, *S8* }
m3	{ *S9* }

Discriminative Size (DS) — Number of times a message appears in transitions to different states in the FSM

- Desirable property is a small *discriminative size*

PURDUE UNIVERSITY

Details of Intelligent Sampling

Sample selectively

Variables:
numMsgs, *numSampled*

m-numMsgs = n-numSampled

Window of *m* messages
messsages that need to be sampled = *n*

Sample all messages here

- Sample selectively ⇒ Messages with discriminative size below a threshold
- Message type is given by the combination (component, method, "call or return")
- The windowing parameters (*m*, *n*) are recalculated when the message rate changes

PURDUE UNIVERSITY

Benefits of Intelligent Sampling

- Non-determinism is kept bounded
  - Less computational cost to update SV
- Reduction in number of rules to match
  - Results in lower detection latency
- Less number of incorrect states in SV
  - Results in reduction of false alarms

PURDUE UNIVERSITY

Roadmap

- Monitor Detection System
- Dealing with High Rate of Messages through Sampling
- Intelligent Sampling Approach
- Hidden Markov Model-based State Estimation
- Experiments and Results
- Optimization of Rules to Match
- Conclusion

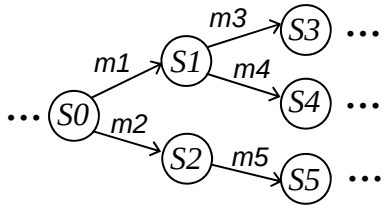
PURDUE UNIVERSITY

Problem of Sampling an *Incorrect* Message

- SV is updated once a message is sampled
- What if an *incorrect* message is sampled?

  Incorrect message — message is incorrect in current states

  e.g., message from buggy component



- Suppose SV = { *S1*, *S2* } and *m3* is changed to *m5*
- SV = { *S5* } (incorrect SV!, it should be { *S3* })

Probabilistic State Vector Reduction: *A Hidden Markov Model Approach*

- Hidden Markov Model (HMM) is used to determine which states in the SV are likely (i.e., has higher probability)
- We can ask to an HMM:
  - *What is the probability of the application being in any state, given a sequence of observations (or messages)?*
- The HMM is trained (offline) with application traces
- Good fit for an HMM since the correct state of the application is hidden from Monitor while the messages are observable
- In a subsequent stage, Monitor prunes states from the SV that have low probability values

Hidden Markov Model Basics

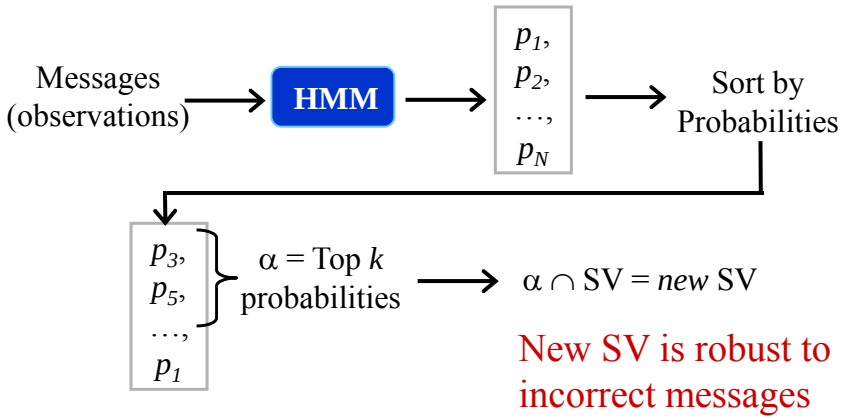
- In a Markov model, the probability of transitioning to the next state only depends on the current state
- In a Hidden Markov Model (HMM), states in the model are not observable, but outcomes are
- An outcome is generated based on the state according to an associated probability distribution
- An HMM is characterized by
  - A set of states
  - A set of observation symbols
  - The state transition probability distribution A
  - The observation probability distribution B (given a state i, what is the probability of observation j)
  - Initial state probability distribution π
- Baum-Welch algorithm is used to learn A, B, and π from application traces

PURDUE UNIVERSITY

State Vector Reduction with the HMM

- Monitor asks to the HMM: $\{p_1, p_2, \ldots, p_N\}$, where

  $p_i = P(S_i \mid O)$, S_i: application state i, O: observation's sequence



New SV is robust to incorrect messages

PURDUE UNIVERSITY

Roadmap

- Monitor Detection System
- Dealing with High Rate of Messages through Sampling
- Intelligent Sampling Approach
- Hidden Markov Model-based State Estimation
- Experiments and Results
  - Application
  - Error Injection
  - Metrics
  - Results
- Optimization of Rules to Match
- Conclusion

PURDUE UNIVERSITY

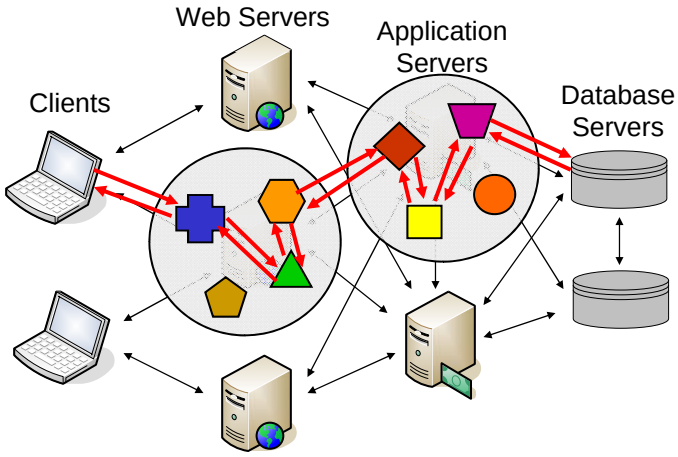
Experimental Testbed: Java Duke's Bank Application

- Emulates a distributed three-tier online banking system
- Users can perform different transactions:
  - Access account information, transfer money, etc.
- Application is stressed with a load of concurrent users
  - Incoming message rate at Monitor varies with user load

PURDUE UNIVERSITY

Web Interaction: A Sequence of *calls* and *returns*

Distributed E-commerce System



Comparison of Monitor with other Detection Techniques

- Convolution algorithm (*SOSP '03*):
  - Finds *delay relationships* between component calls
  - Training phase required to learn normal behavior
  - Unusual delays can be detected from learned models
- Pinpoint (*NSDI '04*):
  - A Probabilistic Context-Free Grammar (PCFG) is used to model normal patterns of web interactions from traces
  - Web interaction structures that does not fit the PCFG are detected as abnormal

Error Injection for Experiments

- Four types of errors are injected:
  - *Response Delays*: a response delay in a method call
  - *Null calls*: a call to a component that is never executed
  - *Unhandled Exceptions*
  - *Incorrect Message Sequences:* change the web interaction structure
- Errors are injected in components touched by web interactions
  - A web interaction is faulty if one (or more) of its components is faulty

PURDUE UNIVERSITY

Performance Metrics Used in Experiments

- *Accuracy* — (True Alarms) % of *true* detections out of the web interactions where errors were injected
- *Precision* — (False Alarms) % of *true* detections out of the total number of detections
- *Detection Latency* — time elapsed between the error injection and its detection

PURDUE UNIVERSITY

Example of Accuracy and Precision

	Web Interactions				
	1	2	3	4	5
Error Injected		X	X		X
Detection (An alarm is signaled)	X	X		X	X

Accuracy = 2/3 = 0.67

Precision = 2/4 = 0.5

Adjusting Sampling Rate

- We want to prevent Monitor from being overwhelmed beyond breaking point
- Sampling rate is adjusted in Monitor by:

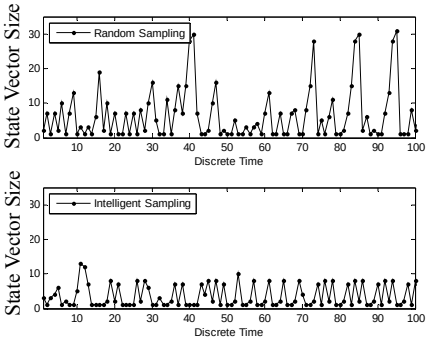
$$R_S = 1 - \frac{(R - R_{th})}{R}, \qquad R > R_{th},$$

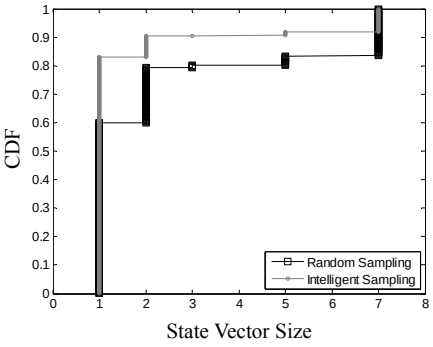
R_S: sampling rate, R: current rate,

R_{th}: rate of breaking point (3 concurrent users)

	Concurrent Users					
	4	8	12	16	20	24
Average Messages / sec	70.00	118.97	194.28	236.86	314.38	362.74
Sampling rate	6/8	4/8	2/8	2/8	1/8	1/8

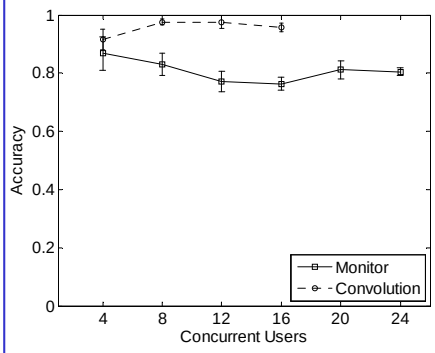
Results: State Vector Reduction

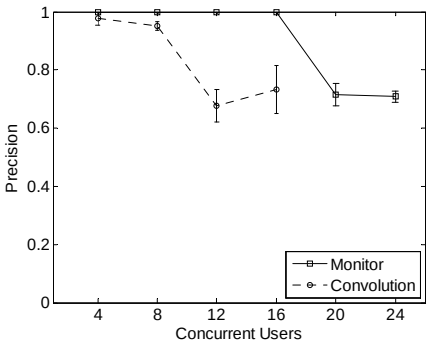




- Peaks are not observed in intelligent sampling (IS)
  - IS capability of selecting messages with small discriminative size
- SV of size 1 is more frequent in IS

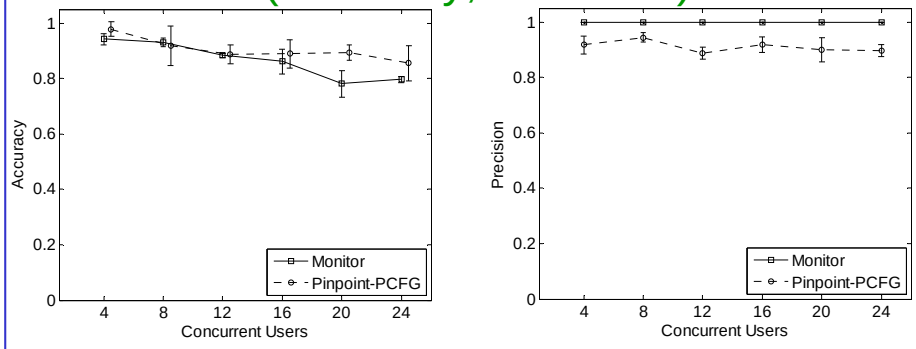
Results: Monitor vs. Convolution (Accuracy, Precision)





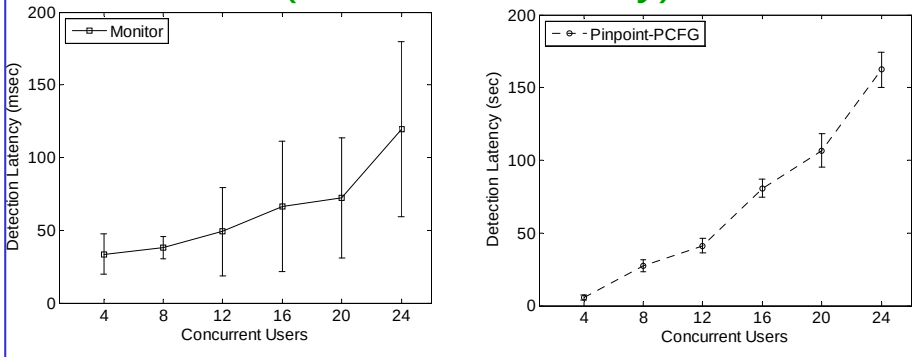
- Only response delays are injected in components
- Monitor's accuracy is about 0.8 even in high rates
- Precision decreases due to its difficulties in differentiating injected delays from delays due to high loads

Results: Monitor vs. Pinpoint (Accuracy, Precision)



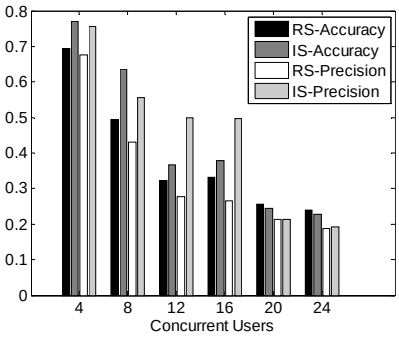
- Anomalies in web interactions are injected
- Monitor and Pinpoint expose similar levels of accuracy
- Precision in Monitor (1.0) is higher than in Pinpoint (0.9)

Results: Monitor vs. Pinpoint (Detection Latency)



- Detection latency in Monitor is in the order of **milliseconds**, while in Pinpoint is in **seconds**
- The PCFG has a high space and time complexity

Results: Random Sampling vs. Intelligent Sampling



- Only response delays are injected
- Definitions of accuracy and precision are changed to the level of components
- RS outperforms IS on average for almost all loads

Results: Memory Consumption

	Average Memory Usage (MB)	
	Virtual Memory	Memory in RAM
Monitor	282.27	25.53
Convolution	—	—
PCFG	933.56	696.06

- Monitor does not rely on large data structures
- PCFG in Pinpoint has high space and time complexity:
  - $O(RL^2)$ and $O(L^3)$

  R: number of rules in the grammar, *L*: size of a web interaction
- Pinpoint thrashes due to high memory requirements

Roadmap

- Monitor Detection System
- Dealing with High Rate of Messages through Sampling
- Intelligent Sampling Approach
- Hidden Markov Model-based State Estimation
- Experiments and Results
- Optimization of Rules to Match
- Conclusion

Which Rules to Match

- Sampling and HMM reduce the number of states to consider for rule matching
- What about reducing the number of rules in a state that need to be matched?
- Many rules are computationally expensive – convolution of two signals, matching strings using context free grammar, re-training a classifier based on user input
- *Intuition*: Computationally expensive rules only need to be matched when there is evidence of system instability
- Previous work has shown that errors are more likely when instability in the system is observed
  - Example: Increasing average response time in a web server may indicate an imminent failure due to resource exhaustion

Reducing Rule Matching Load

- Problem Statement: To classify the system state to two classes (stable, unstable) based on a bounded number of system measurements
  - System measurements can mean CPU usage, response time, memory utilization
- Goal: Come up with classifier F

$F: O \rightarrow C$

$O: o_{t-N}, \ldots, o_{t-1}$

$C: c^+, c^-$

- The problem in its full generality is very broad
- We have addressed a specific instance of this

PURDUE UNIVERSITY

Detection of Memory Leak Error

- Apache Tomcat web server is instrumented to inject a memory leak dynamically
- Testbed: E-commerce site that simulates the operation of an online store as per the TPC-W benchmark
- Rule for detecting error:

1. Create ARMA models for memory usage under normal and memory leak, say λ and λ' (offline)
2. Take online readings of memory usage
3. Using n previous measurements, predict the n future measurements individually using λ and λ'
4. If the future measurements match better with the prediction using λ', flag an error

PURDUE UNIVERSITY

Selective Trigger for Expensive Rule

- ARMA model based prediction and matching can be computationally expensive
- Trigger: System instability
  - Measure standard deviation of *m* previous observations
  - If it is greater than a threshold, then trigger rule matching

Rule Matching Criteria	Memory Leak Detected	Average Matching Latency (msec.)
Always matched	yes	19.283
$\sigma \geq 0.5$	yes	7.115
$\sigma \geq 1.0$	no	1.25

Roadmap

- Monitor Detection System
- Dealing with High Rate of Messages through Sampling
- Intelligent Sampling Approach
- Hidden Markov Model-based State Estimation
- Experiments and Results
- Optimization of Rules to Match
- Conclusion

Summary

- Monitor system for non-intrusive detection of failures
  - Decomposition into Observer (application) and Observed (Monitor) system
  - Stateful detection
  - Anomaly based rules, both combinatorial and temporal
- High rate of messages overwhelms any detection system
- Solution: Reduce the fraction of messages that is processed
  - Sampling, which introduces non-determinism
- Challenges of non-determinism addressed by intelligent sampling and HMM-based probabilistic state estimation
  - These reduce the number of states that need to be considered
- Optimize matching expensive rules
  - Use fast estimation of system instability as trigger

Ongoing Work

- How to measure system instability? How much instability is an indication of imminent failure?
- How to broaden the scope of failures that can be handled? Failures with short-lived prior symptoms?
- How to make the HMM-based processing more efficient?
- How to make the rules adaptive to observed load in the system?