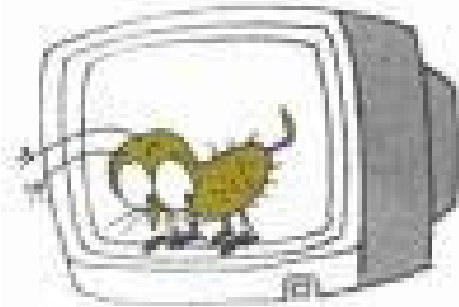


Rx: Treating Bugs As Allergies

Feng Qin, Joseph Tucek, Jagadeesan Sundaresan and Yuanyuan Zhou
University of Illinois at Urbana-Champaign
SOSP 2005



Presented By Weihang Jiang

CS 523



Motivation

- High availability is required for many applications
 - Average cost for an hour downtime of financial applications exceeds \$6M [Gartner'98]

- Software failures greatly reduce system availability
 - Account for 40% of system failures [Marcus'00]



Existing Solutions to Survive Software Failures

- Rebooting Techniques
 - Whole-system rebooting, micro-rebooting
- General checkpointing and recovery
 - Fail-over system, progressive retry, recovery block
- Application-specific recovery
 - Multi-process model, transaction programming model, n-version programming
- Recently proposed non-conventional mechanisms
 - Failure-oblivious computing, reactive immune systems



Limitations of State-of-the-Art Techniques

- Existing solutions suffer from one or more of the following limitations:
 - Cannot recover from deterministic bugs [Chandra-DSN00]
 - Relative long recovery and warm-up time (several seconds)
 - Require application restructuring to be failure-aware
 - Speculatively "fix" bugs---unsafe



Our Idea: Treat Bugs As Allergies

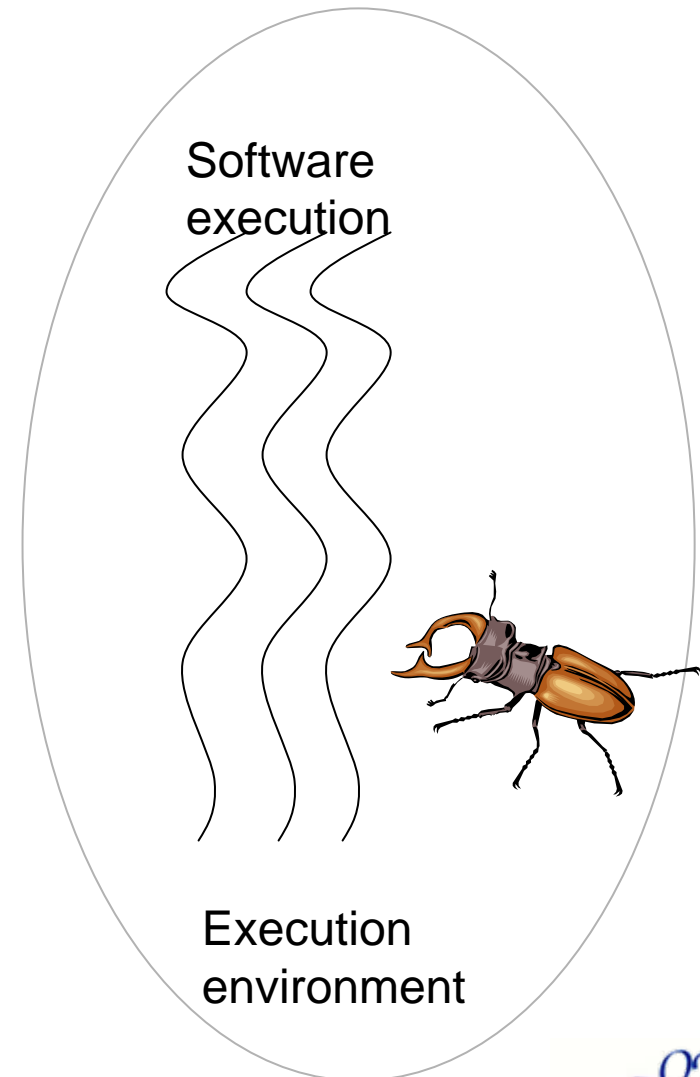
- Allergies:
 - **Hard to cure**
 - But you can **avoid** them by changing the living environment!
- ↓
- Avoidance Therapy for Allergies



Rx: Avoidance Therapy for Software Bugs

- Software Bugs:
 - **Hard to fix** on the fly
 - Many depend on execution environments
 - Possible to **avoid by changing the execution environments!**

- Goals:
 - Deterministic bugs → Non-deterministic bugs
 - Non-deterministic bugs → lower probability to occur



Execution Environments

- What can be included in execution environments?
 - Higher level - Libraries, e.g. glibc, 3rd party libraries
 - Middle level - OS kernel, e.g. system calls, network protocols
 - Lower level - Hardware, e.g. devices

- What changes are good?
 - Need to be safe
 - Cannot violate API
 - Possible to influence applications' execution



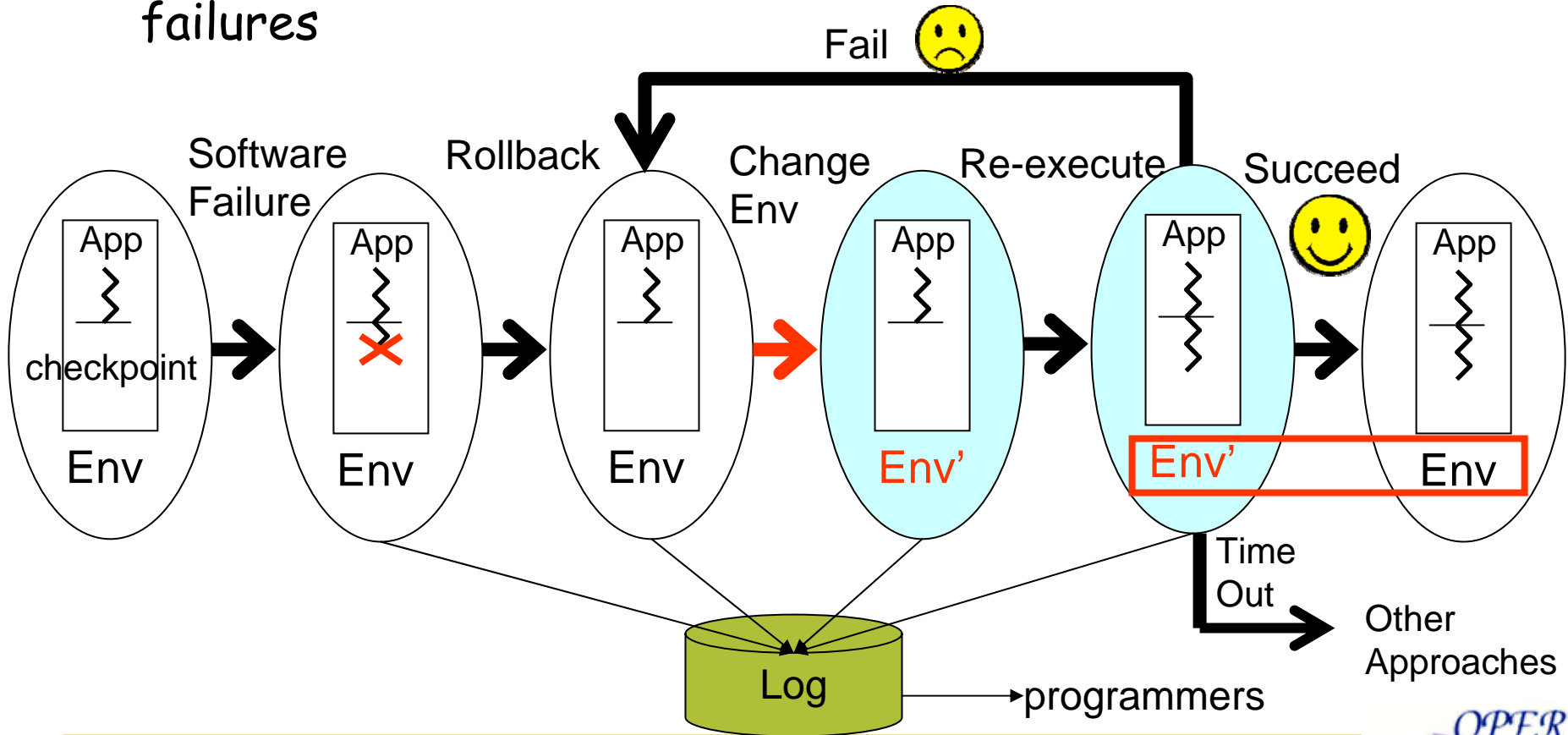
Examples of Environmental Changes

Category	Potentially-Avoided Bugs	Environmental Changes
Memory Management	Double free, dangling pointer	Delaying recycling of freed buffer
	Dynamic buffer overflow	Padding allocated memory blocks
	Memory corruption	Allocating memory in an isolated location
	Un-initialized reads	Zero-filling newly allocated memory buffers
Timing-Related		Scheduling
	Concurrency Bugs	Signal delivery
		Message Reordering
User-Related	Bugs related to user requests	Selectively Dropping user requests



The Rx Process

- Change execution environment **on-demand** upon software failures





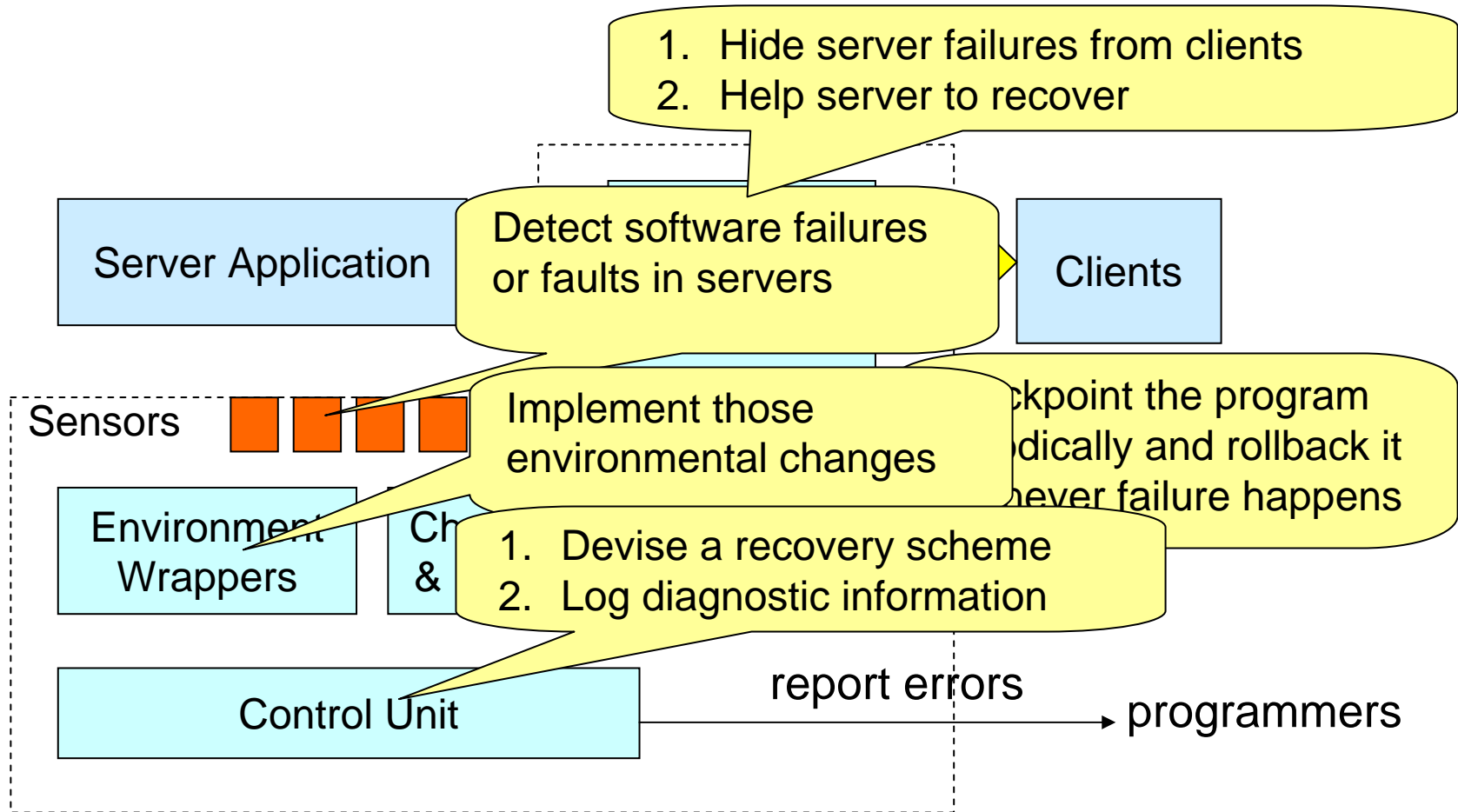
Outline

- Motivation
- ➔ ■ Rx Main Idea
- Rx Design and Issues
- Evaluation
- Conclusion

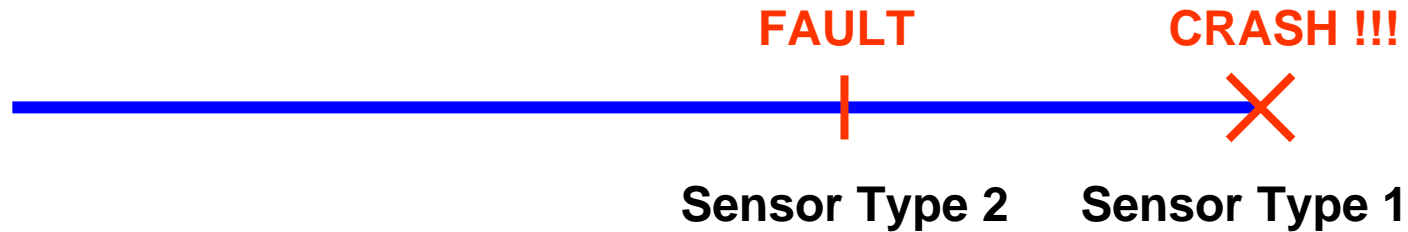




Rx Architecture



Sensors



- Type1. Exception-based sensors
 - Detect crashes, exceptions, etc

- Type2. Low-overhead bug detectors
 - Others: CCured [PLDI04], etc.
 - Ours: *SafeMem* [HPCA05], iWatcher[ISCA04], AccMon[Micro04]

Checkpoint and Rollback (CR)

- Light-weight scheme
 - Not for hardware failures/OS crashes
 - In-memory checkpoint with Copy-On-Write
 - Support multiple checkpoints and rollback
 - More details in our previous work: Flashback [USENIX04]
- Extensions from Flashback
 - non-determinism → simpler implementation
 - recovery time bound → discard old checkpoints



Outline

- Motivation
- Rx Main Idea
- ➔ ■ Rx Design and Issues
- Evaluation
- Conclusion



Evaluation Methodology

- Rx's properties to be evaluated:
 - Functionality
 - Recovery performance
 - Time overhead
 - Benefits of learning from history

- **Two alternatives** for comparison:
 - Whole program restart
 - Simple rollback and re-execute **without environmental changes**



Experiments Setup

- Four **representative** server applications
 - MySQL: database server
 - Squid: web proxy and cache server
 - Apache: web server
 - CVS: source code control server
- Six **representative** bugs
 - Four real bugs: buffer overflow, data race, stack overflow, and double free
 - Two injected bugs: un-initialized read and dangling pointer



Overall Results

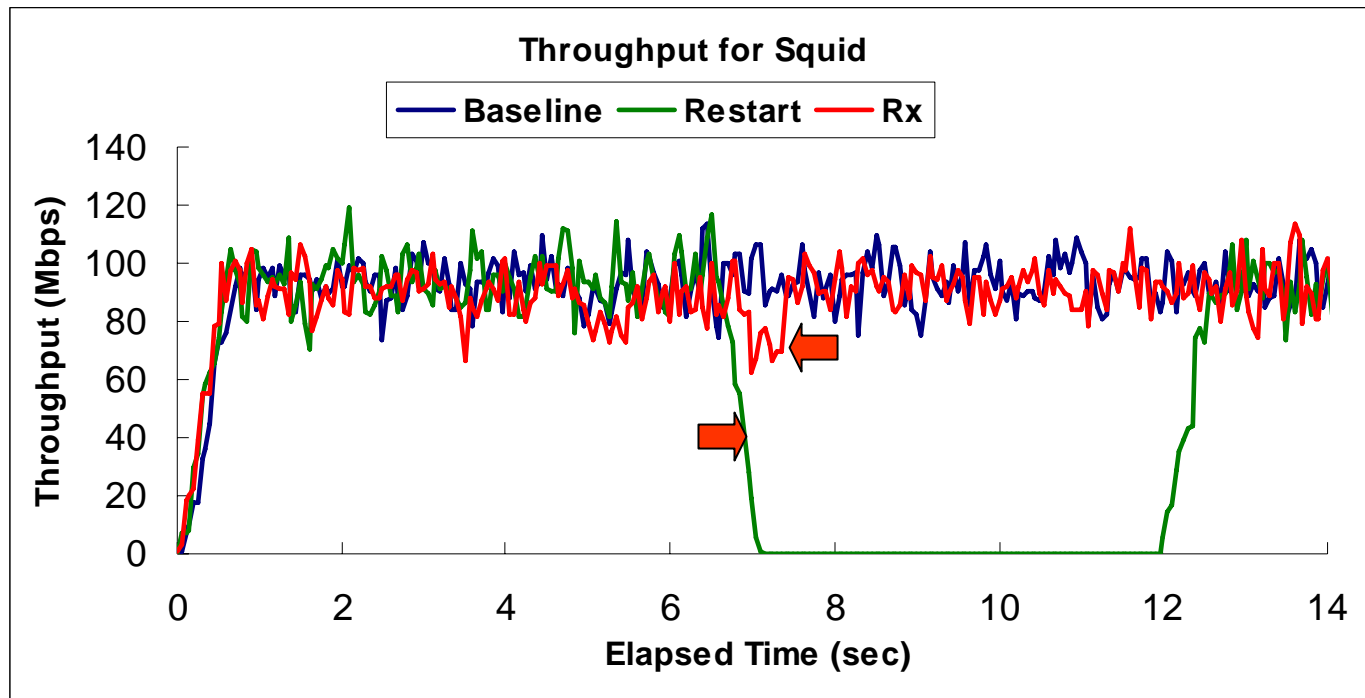
- Survive 6 bugs in 4 server applications
- Recovery time of .017 to .161 seconds

<i>App</i>	<i>Bug</i>	<i>Recover Failure?</i>		<i>Recovery Time (s)</i>	
		<i>Alternatives</i>	<i>Rx</i>	<i>Restart</i>	<i>Rx</i>
<i>Squid</i>	<i>Buffer Overflow</i>	<i>No</i>	<i>Yes</i>	<i>5.113</i>	<i>0.095</i>
<i>MySQL</i>	<i>Data Race</i>	<i>0.4</i>	<i>Yes</i>	<i>3.500</i>	<i>0.161</i>
<i>Apache</i>	<i>Stack Overflow</i>	<i>No</i>	<i>Yes</i>	<i>1.115</i>	<i>0.026</i>
<i>CVS</i>	<i>Double Free</i>	<i>No</i>	<i>Yes</i>	<i>0.010</i>	<i>0.017</i>
<i>Squid-ui</i>	<i>Uninit Read</i>	<i>No</i>	<i>Yes</i>	<i>5.000</i>	<i>0.126</i>
<i>Squid-dp</i>	<i>Dangling Pointer</i>	<i>No</i>	<i>Yes</i>	<i>5.006</i>	<i>0.113</i>



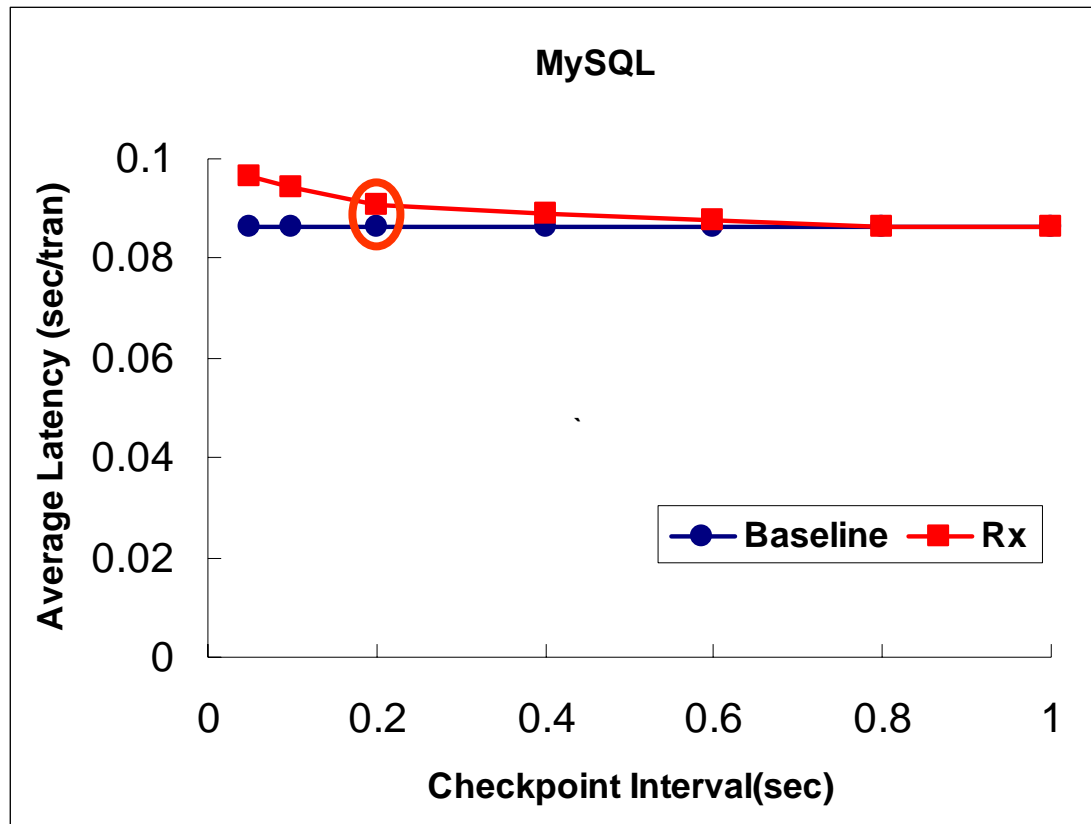
Recovery Performance

- Rx hides server failures, while restart cannot



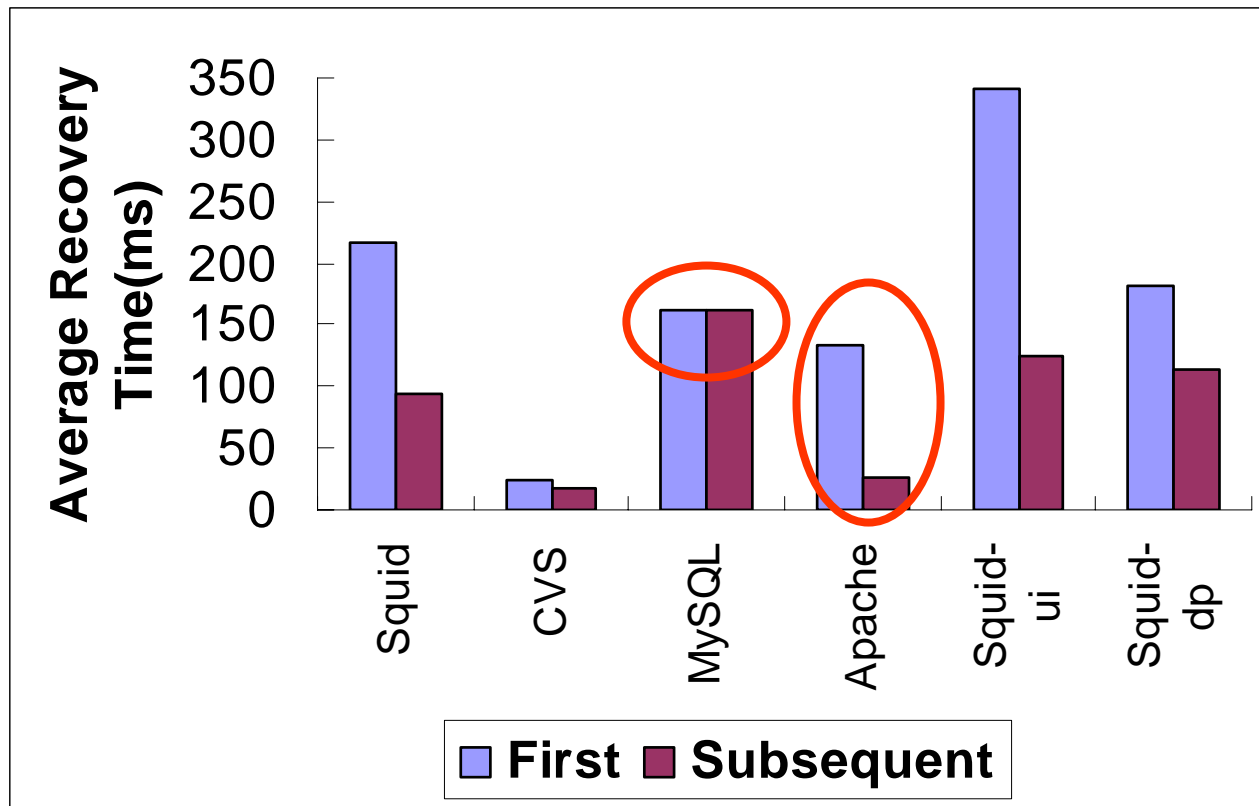
Rx Time Overhead

- Rx incurs a small time overhead for small checkpoint intervals



Benefits of Learning from History

- Learning from history is effective



Conclusions and Limitations

- **Avoidance Therapy** for common software failures
 - Bugs, you can't beat them, but you can avoid them

- **Limitations**
 - Cannot avoid all types of bugs (e.g. semantic bugs)
 - Bugs manifest silently---Rx claims success incorrectly
 - Need more powerful sensors (on-going work)



Future Work

- Future Work
 - Extend Rx to support multi-tier server applications
 - Borrow ideas from machine learning to help failure diagnose



Thanks!



Advantages of Rx

- **Comprehensive**: survives common software bugs, both non-deterministic and deterministic
- **Safe**: does not speculatively “fix” bugs
- **Noninvasive**: requires few to no modification to applications' source code
- **Efficient**: requires no rebooting or warming up
- **Informative**: provides additional diagnostic information for postmortem analysis

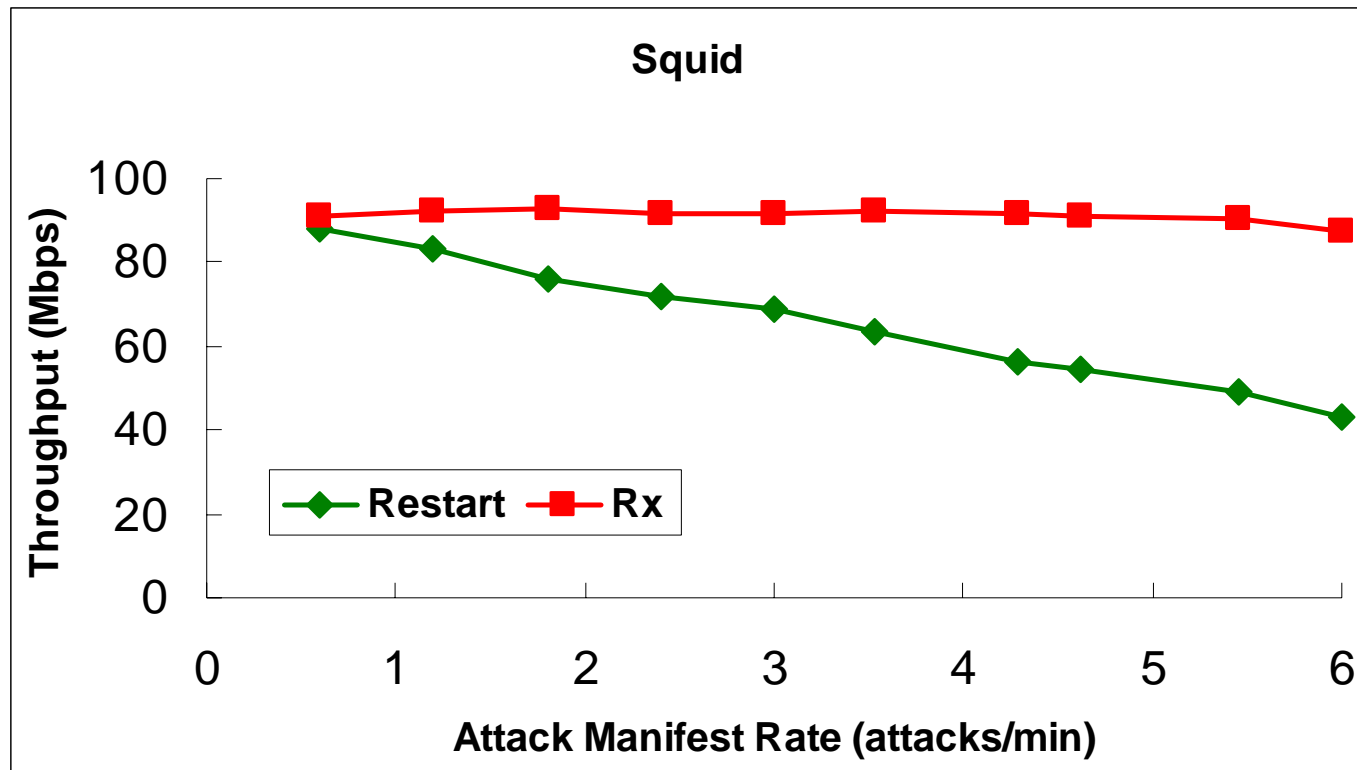


Challenging Issues

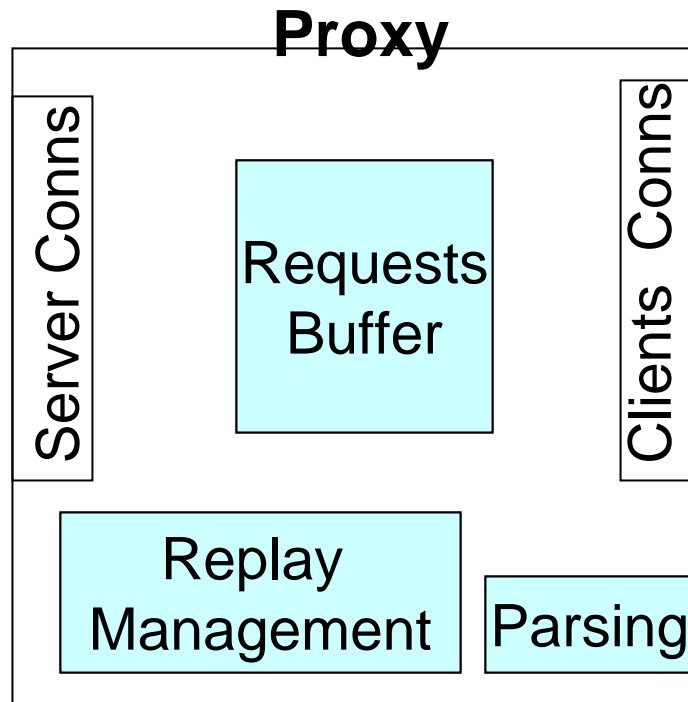
- **Determine the order of environmental changes**
 - First time: Follow a certain order, cheaper one first
 - Subsequent times: Learn from history
 - Based on information from sensors
- **Output commit problem**
 - Request-response matching
 - Make sure only one response is sent to the user
- **Guarantee session semantics**
 - Signature matching for committed responses
- **Others (refer to the paper)**

Performance Under Continuous Attacks

- Rx better tolerates continuous attacks



Proxy



- Hides server failures from clients
- Helps server to recover from failures
- Introduces non-determinism into server recovery process
- Understand protocols to parse message boundary