Rx: Treating Bugs As Allergies: A Safe Method to Survive Software Failures

Feng Qin, Joseph Tuccek, Jagadeesan Sundaresan and Yuanyuan Zhou

University of Illinois at Urbana Champaign

Presented at SOSP 05

Outline

- Introduction
- Main idea
- Architecture
- Design and Implementation Issues
- Evaluation and Results
- Related work
- Conclusion

Introduction

Motivation

- **An hour of downtime for a financial company costs company costs $6mil**
- Software failures reduce system availability
- Software defects - 40%
- Memory-related+concurrency bugs - 60%
- Cannot get rid of bugs
- Need highly available applications

Previous Solutions

- Four categories:
  - Rebooting
  - checkpointing, rollback, re-execute
  - Application-specific recovery
  - Speculate on programmer intentions

Allergies are an inspiration

- When a person suffers from an allergy, the most common treatment is to remove the allergens from their **living environment**
- In software, many bugs resemble allergies: their manifestation can be avoided by changing the **execution environment**
- The idea
  - Rollback the program to a recent checkpoint when a bug is detected
  - Dynamically change the execution environment based on the failure symptoms
  - Re-execute the buggy code region in the new environment

Examples of allergen bugs

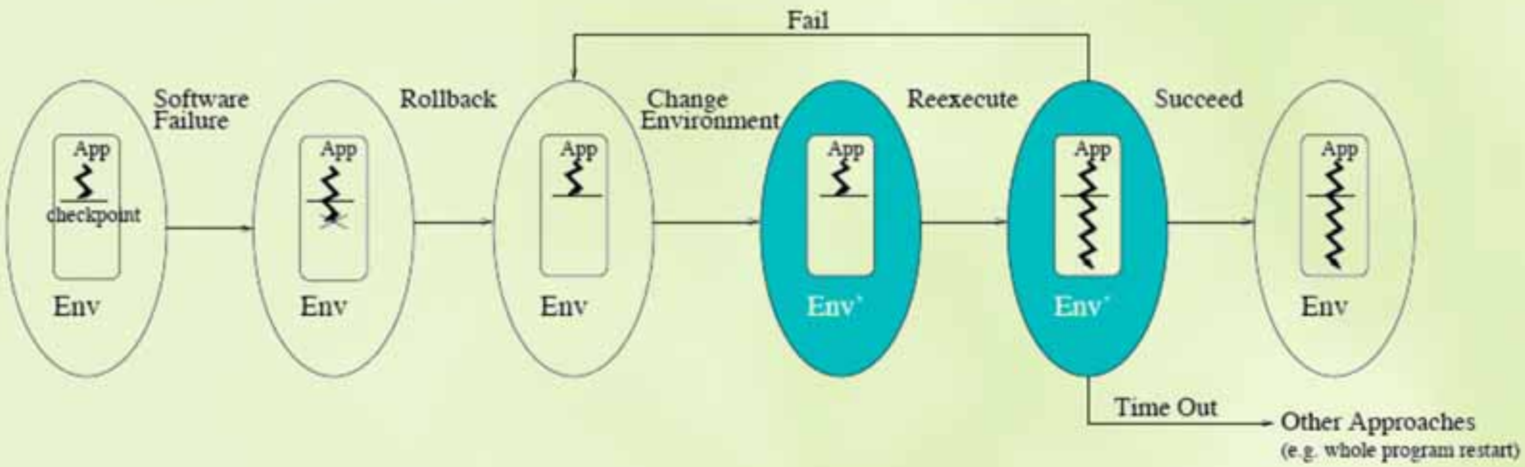
- Memory corruption
- Buffer overrun
- Un-initialized reads
- Data races
- malicious request

Rx does it better

- Comprehensive
- Safe
- Noninvasive
- Efficient
- Informative

Main idea

Main Idea



- Checkpoint
- Sense bug
- Analyze symptoms and determine cure
- Re-execute from checkpoint
  - New environment
- Repeat until it goes away
  - Or time out

The execution environment

- Definition: Almost everything that is external to application:
  - Low level: hardware devices, processor architecture..
  - mid level: OS kernel scheduling, virtual memory manager, drivers, file system, network
  - High level: standard libraries, third party libraries
- Requirement for environmental change
  - Correctness-preserving: execute according to the APIs
  - Useful: potentially avoid software bugs

Categorizing useful changes

Category	Environmental Changes	Potentially-Avoided Bugs	Deterministic?
Memory Management	delayed recycling of freed buffer	double free, dangling pointer	YES
	padding allocated memory blocks	dynamic buffer overflow	YES
	allocating memory in an alternate location	memory corruption	YES
	zero-filling newly allocated memory buffers	uninitialized read	YES
Asynchronous	scheduling	data race	NO
	signal delivery	data race	NO
	message reordering	data race	NO
User-Related	dropping user requests	bugs related to the dropped request	Depends

Table 1: Possible environmental changes and their potentially-avoided bugs

Working with the changes

- Successful change - record
- Failure - see if it occurred before
- Else
  - Try low overhead changes first
- If failure doesn't go away with useful change
  - keep rollback to previous checkpoint OR
  - Make another change

Architecture

Rx Design

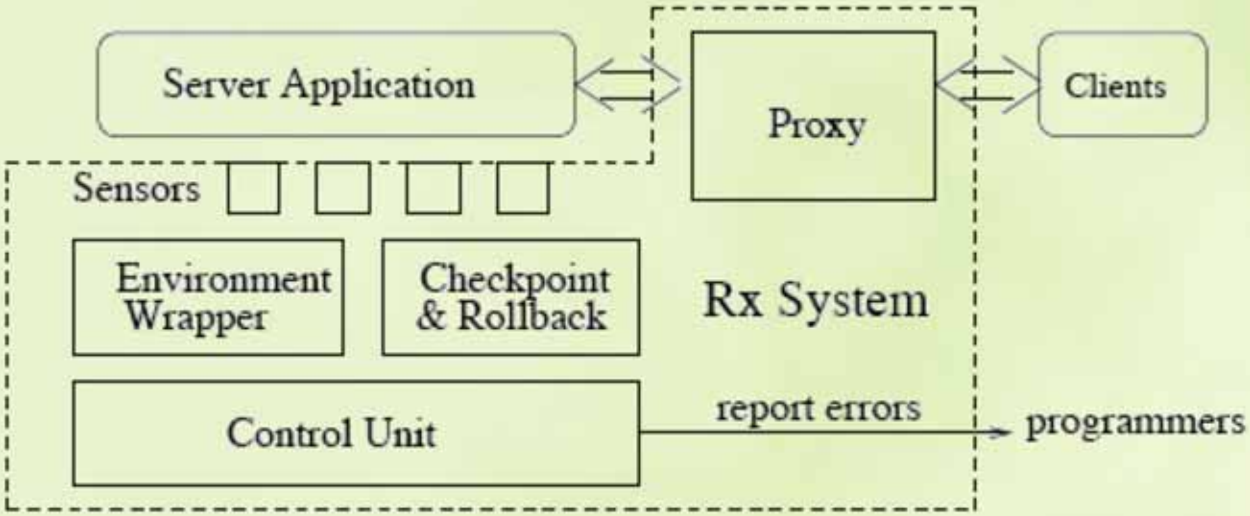


Figure 2: Rx architecture

Sensors

- dynamically monitoring application execution
  - Exception sensors
  - Bug-specific sensor
- Dynamic bug detection tools
- send failure signature to Control Unit

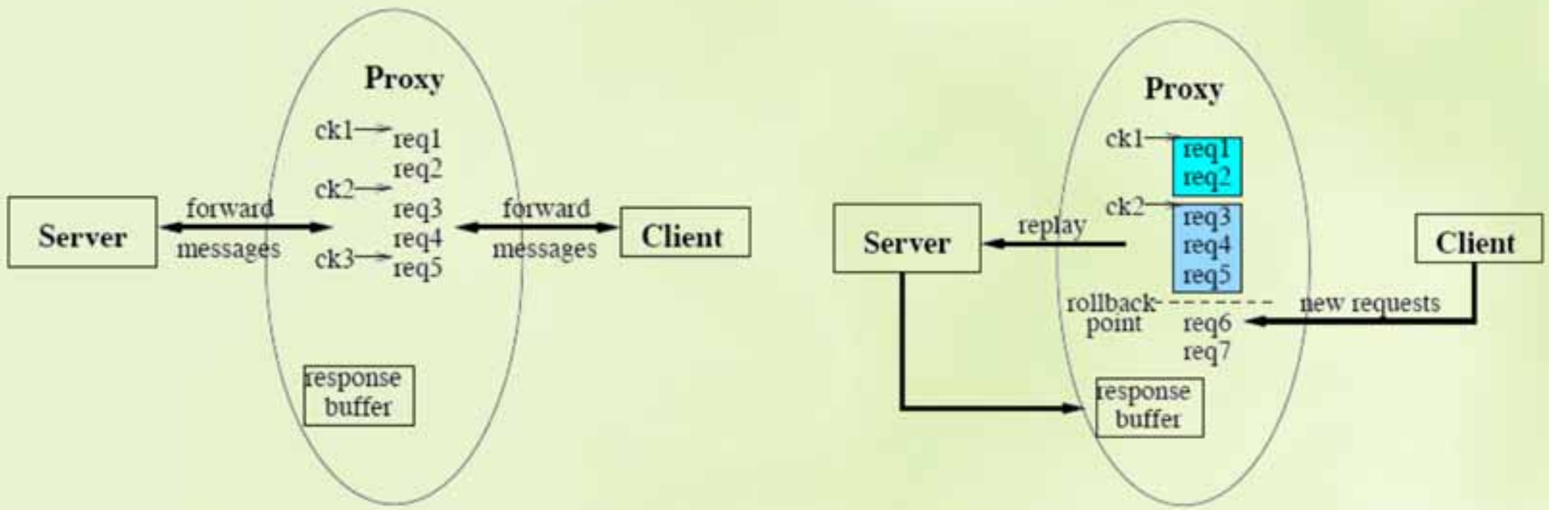
Checkpoint & Rollback

- Memory snapshots
- File versioning
- Less checkpoint maintenance

Environment Wrappers

- Perform changes in the execution environment (re-execution)
  - Memory wrapper
  - Message wrapper
  - Process scheduling
  - Signal delivery
  - Dropping user requests

Proxy



(a) Proxy behavior in normal mode

(b) Proxy behavior in recovery mode

Control Unit

- Coordinates all of the components in the Rx
- Three functions
  - Directs the checkpointing and rollback process
  - Diagnose failure based on symptoms and experiences
  - Provides feedback to programmer

Design and Implementation Issues

Design and Implementation Issues

- Inter-Server Communication
- Multi-threaded Process Checkpointing
- Unavoidable Bug/Failure of Rx

Evaluation and Results

Evaluation

App	Ver	Bug	#LOC	App Description
MySQL	4.1.1.a	data race	588K	a database server
Squid	2.3.s5	buffer overflow	93K	a Web proxy cache server
Squid-ui	2.3.s5	uninitialized read		
Squid-dp	2.3.s5	dangling pointer		
Apache	2.0.47	stack overflow	283K	a Web server
CVS	1.11.4	double free	114K	a version control server

Table 2: Applications and Bugs (App means Application. Ver means Version. LOC means lines of code).

Effectiveness

apps	Bugs	Symptoms	Changes	Restart recovery	Rx recovery
Squid	buffer overflow	SEGV	Padding	5.113s	0.095s
[Squid-ui]	uninit. read	SEGV	Zero All	5.000s	0.126s
[Squid-dp]	dangling ptr	SEGV	Delay Free	5.006s	0.113s
Apache	stack overflow	Assert	Drop Req.	1.115s	0.026s
CVS	double free	SEGV	Delay Free	0.010s	0.017s
MySQL	data race	SEGV	Sched. Change	3.500s	0.161s

Performance

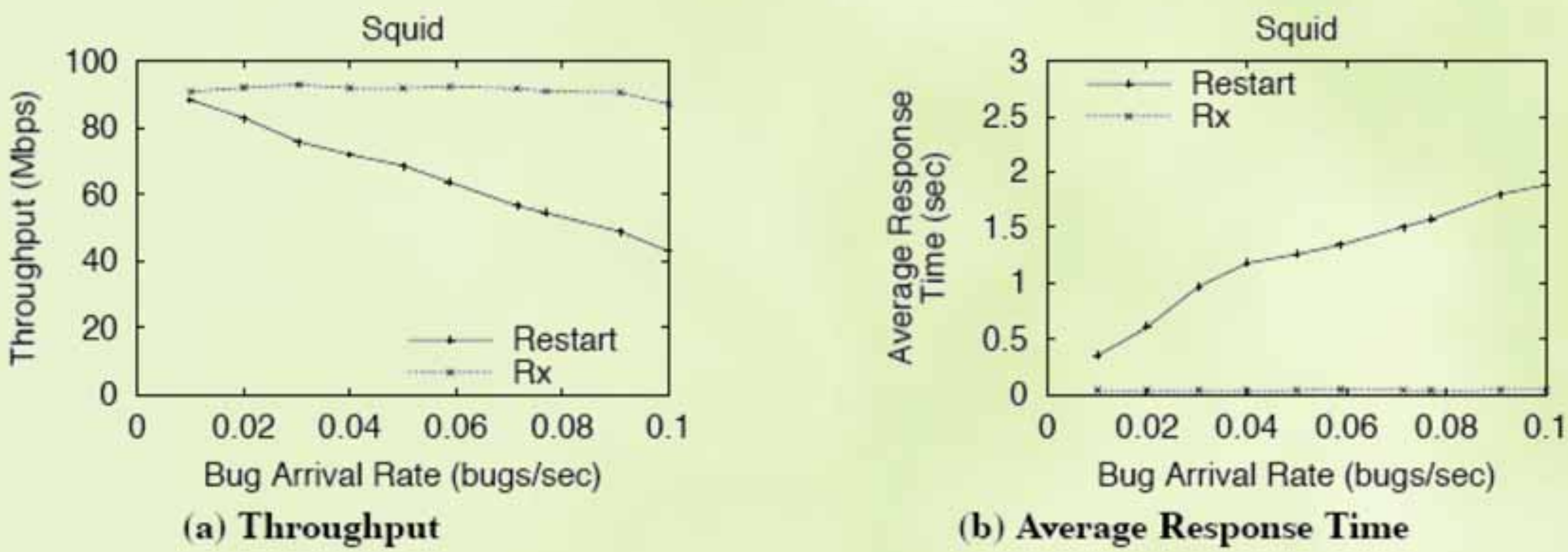
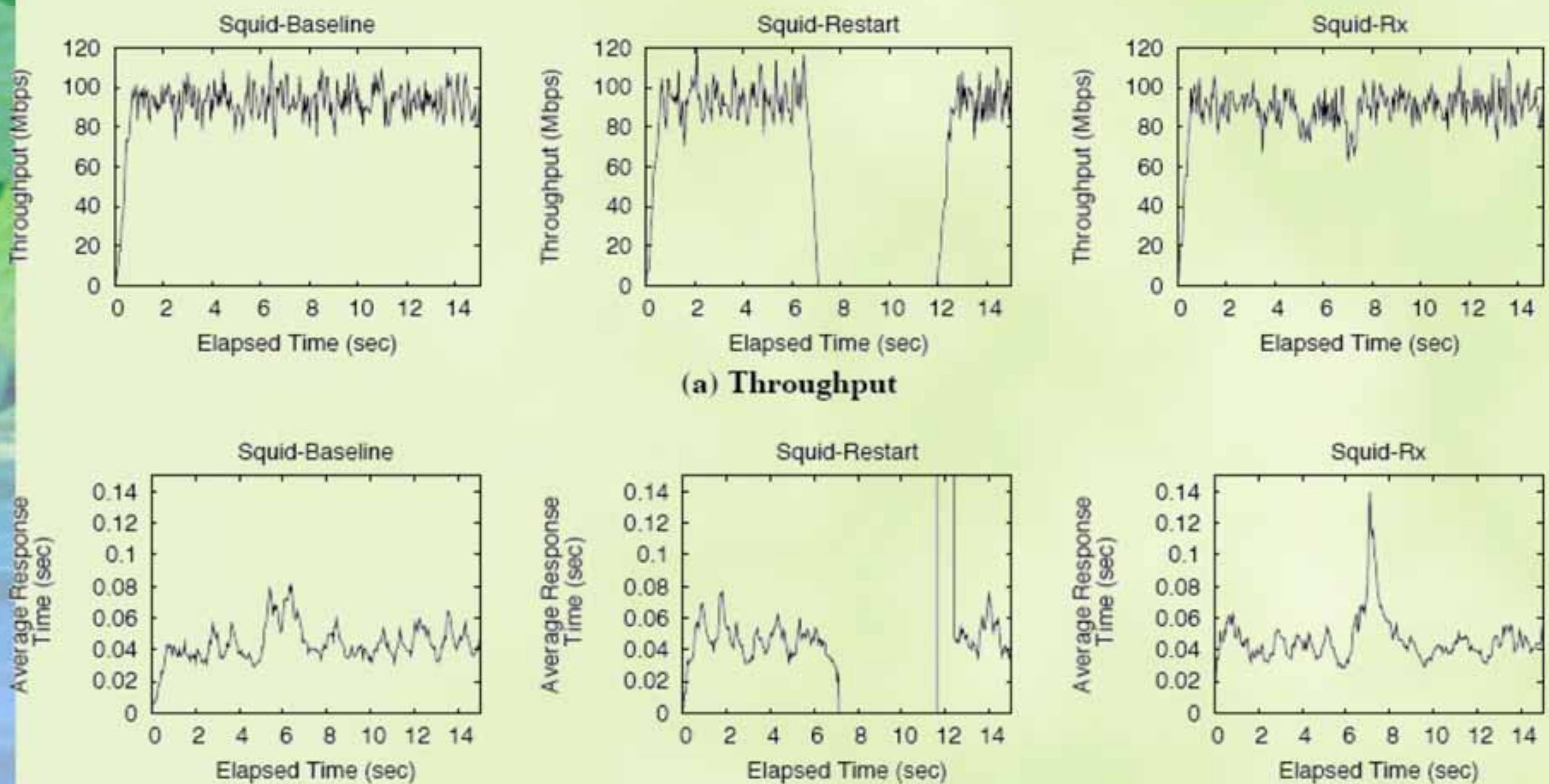


Figure 5: Throughput and average response time with different bug arrival rates

Performance



(a) Throughput

(b) Average Response Time

Related Work

Related Work

- Recovery-Oriented Computing
- Shadow drivers
- Noisemakers

Conclusion

- safe, non-invasive and informative method for quickly surviving software failures
  - Caused by common software defects
- Like all approaches it has its limitations
- It can effectively and efficiently recover from many software failures, but not all