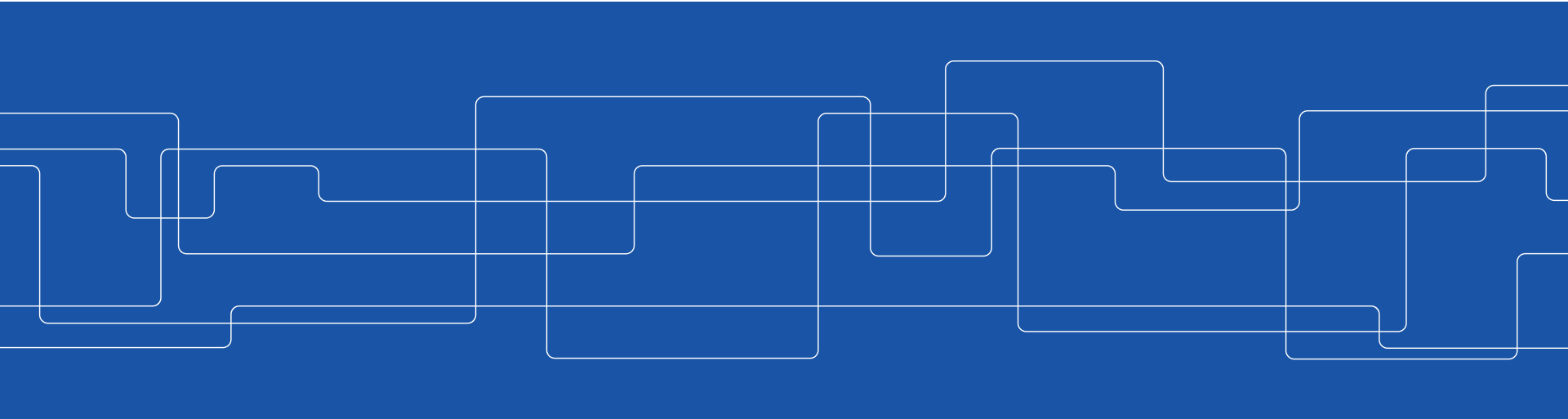
KTH ROYAL INSTITUTE
OF TECHNOLOGY

TripleAgent

Monitoring, Perturbation and Failure-obliviousness for Automated Resilience Improvement in Java Applications

Long Zhang longz@kth.se



Contents

- Motivation
- Background
- Design
- Evaluation
- Discussion

Motivation

```
public class ExampleClass {
    public void foo() {
        Stub stub = new Stub();
        stub.doSth();
    }
}
```

```
public class Stub {
    public void doSth() throws IOException {
        ...
    }
}
```

```
public class ExampleClass {
    public void foo() {
        Stub stub = new Stub();
        try {
            stub.doSth();
        } catch (IOException e) {
            …
        }
    }
}
```

Error: java: unreported exception java.io.IOException; must be caught or declared to be thrown.

Motivation

Technique of Ambidexterity, Zhou Botong, The Legend of the Condor Heroes

https://en.wikipedia.org/wiki/Zhou_Botong

TripleAgent

Monitoring, Perturbation and Failure-obliviousness for Automated Resilience Improvement in Java Applications

Background - Monitoring

- Improve the observability of applications
  - What
  - When
  - Why
  - Necessary information for our experiments

Chaos Engineering Observability by Russ Miles

Publisher: O'Reilly Media, Inc.

Release Date: April 2019

ISBN: 9781492051046

Background - Perturbation

- Software Fault Injection Techniques
  - During unit testing: mutation testing
  - During integration: grey box testing
  - In production: chaos engineering

Roberto Natella, Domenico Cotroneo, and Henrique S. Madeira. 2016. Assessing Dependability with Software Fault Injection: A Survey. ACM Comput. Surv. 48, 3, Article 44 (February 2016), 55 pages. DOI: https://doi.org/10.1145/2841425

Background - Failure-obliviousness

- To prevent the application from crashing when an error occurs at runtime: to discard certain failures in a principled way
- Example:
  - write data into an invalid memory address
  - invalid index numbers of an array

M. Rinard, C. Cadar, D. Dumitran, D.M. Roy, T. Leu, and W.S. Beebee Jr. Enhancing server availability and security through failure-oblivious computing. In Proceedings of the 6th conference on Symposium on Operating Systems Design & Implementation, pages 21–21. USENIX Association, 2004.

Design - Concepts

- An invocation chain: m2 → m1 → m0

```
Class0 {
    public void m0() throws EA, EB {
        // a statement
        ...
    }
}
```

```
Class1 {
    public void m1() throws EA, EB {
        new Class0().m0();
    }
}
```

```
Class2 {
    public void m2() {
        try {
            new Class1().m1();
        } catch (EA a) {
            ...
        } catch (EB b) {
            ...
        }
    }
}
```

Design - Monitoring Agent

- Static information
- Dynamic information
- Application specific metrics

Design - Perturbation Agent

- Bytecode instrumentation
  - Done at runtime
  - No need to access source code
- Failure scenarios
  - Only inject one exception
  - Allways inject exceptions

Design - Concepts

- An invocation chain: m2 → m1 → m0

```
Class0 {
    public void m0() throws EA, EB {
        // code injected with code transformation
        PAgent.throwExceptionPerturbation(key1);
        PAgent.throwExceptionPerturbation(key2);
        // a statement
        PAgent.throwExceptionPerturbation(key3);
        PAgent.throwExceptionPerturbation(key4);
        ...
    }
}
```

```
Class1 {
    public void m1() throws EA, EB {
        new Class0().m0();
    }
}
```

```
Class2 {
    public void m2() {
        try {
            new Class1().m1();
        } catch (EA a) {
            ...
        } catch (EB b) {
            ...
        }
    }
}
```

Design - Failure-oblivious Agent

- Instrument a method with a try-catch wrapper
  - Off: throw the caught exception again
  - On: silence the caught exception and prevents the propagation

Design - Failure-oblivious Agent

- When an exception is silenced, the possible outcomes:
  - The application runs normally
  - The application runs in a gracefully degraded mode
  - The application crashes

Design - Concepts

- An invocation chain: m2 → m1 → m0

```
Class0 {
    public void m0() throws EA, EB {
        // code injected with code transformation
        PAgent.throwExceptionPerturbation(key1);
        PAgent.throwExceptionPerturbation(key2);
        // a statement
        PAgent.throwExceptionPerturbation(key3);
        PAgent.throwExceptionPerturbation(key4);
        ...
    }
}
```

```
Class1 {
    public void m1() throws EA, EB {
        new Class0().m0();
    }
}
```

```
Class1 {
    public void foo() throws EA, EB {
        try {
            new Class0().m0();
        } catch (Exception e) {
            if (FOAgent.modeIsOn(key)) {
                // the exception is silenced
            } else { throw e; }
        }
    }
}
```

```
Class2 {
    public void m2() {
        try {
            new Class1().m1();
        } catch (EA a) {
            ...
        } catch (EB b) {
            ...
        }
    }
}
```

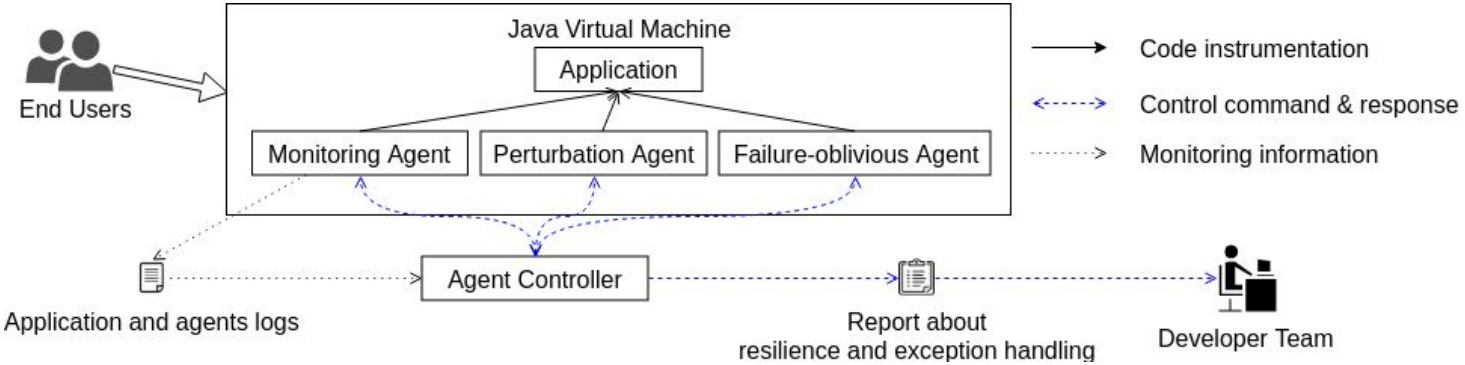
Agent Controller

- Activate or deactivate a specific perturbation point
- Analyze the recorded information
- Calculate the classification of perturbations
- Evaluate candidate failure-oblivious methods

Agent Controller

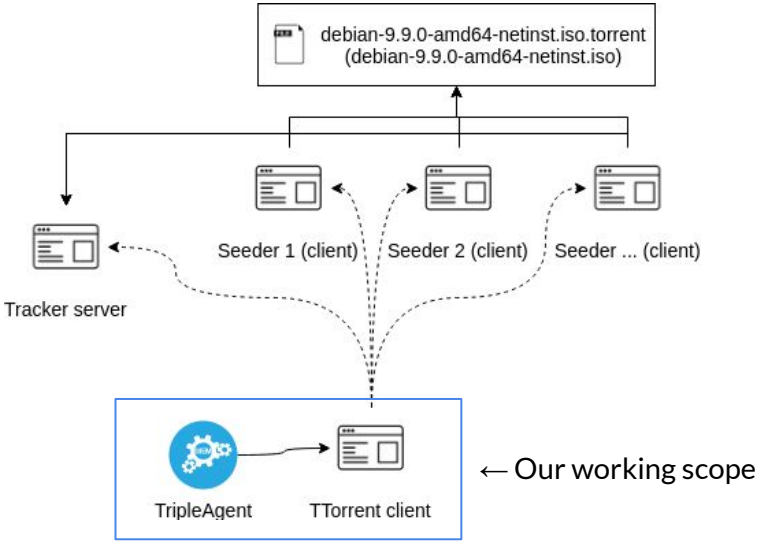
- The classification of perturbations
  - Fragile points: one exception -> crash or freeze
  - Sensitive points: continuously injecting exceptions -> crash or freeze
  - Immunized points: no matter how many exceptions -> still acceptable

Design - Architecture



- Input
  - Arbitrary software in Java
  - Workload
  - Acceptability oracles
- Output
  - Classification of perturbation points
  - Failure-oblivious methods
  - Logs of monitored information

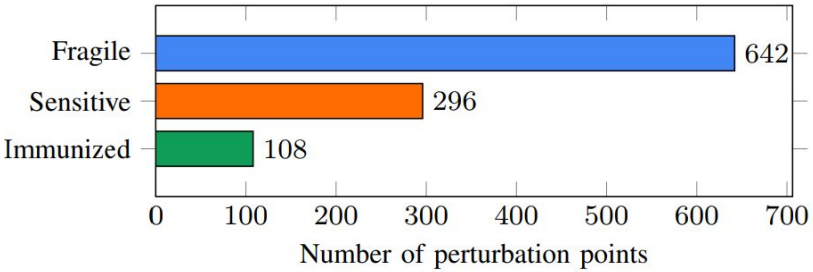
Evaluation - TTorrent



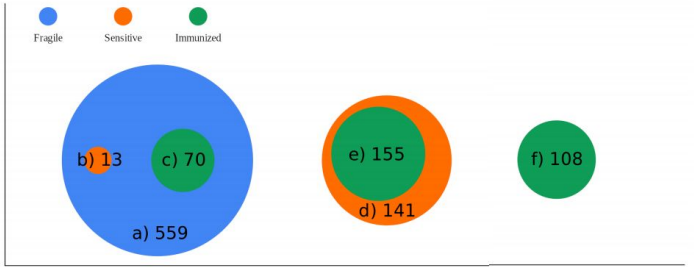
Why we use TTorrent:

- 1000+ stars on GitHub
- Run it as a common user
- Real production environment and traffic

Evaluation - TTorrent



Category of perturbation points



a) Fragile stays fragile, b) Fragile to sensitive, c) Fragile to immunized
d) Sensitive stays sensitive, e) Sensitive to immunized, f) Immunized stays immunized

Resilience improvement

Discussion - Overhead

- Application level: the execution time
- Operating system level: CPU usage etc.
- Binary code level: the code bloat

THE OVERHEAD OF AN EXPERIMENT ON TTORRENT

Evaluation Aspects	Original Version	Instrumented Version	Variation
Downloading time	20.4s	21.1s	3.5%
CPU time	15.0s	18.3	22.2%
Memory usage	47M	49M	4.3%
Peak thread count	30	32	6.7%
Relevant class files size	16.7KB	16.8KB	0.6%

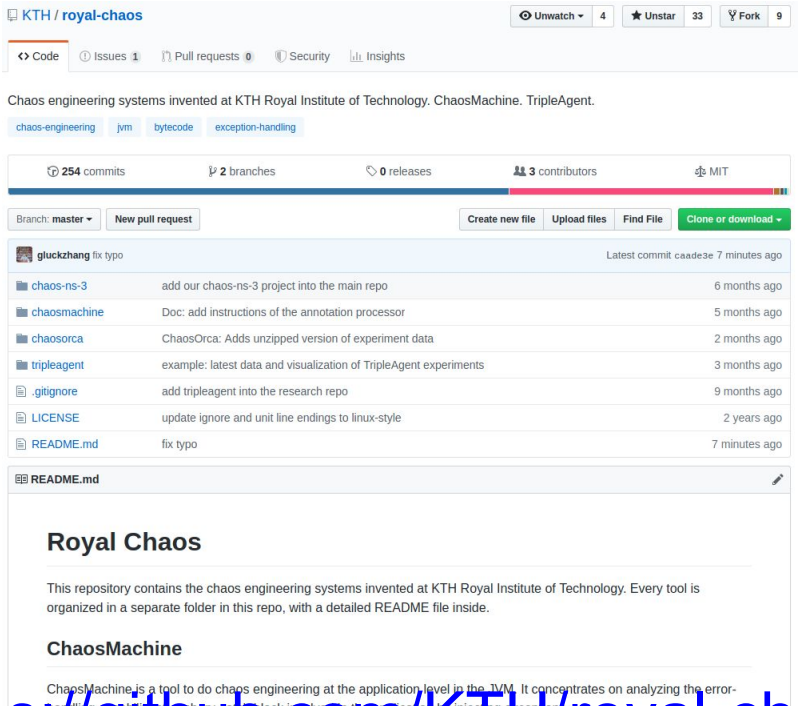
Our Contributions

- The concept of joint usage of fault injection and failure-oblivious code instrumentation to evaluate and improve resilience against uncaught or mishandled exceptions.
- A system called TripleAgent that combines monitoring, perturbation injection and failure-oblivious computing in Java, implemented with agents for the JVM: https://github.com/KTH/royal-chaos
- An empirical evaluation of TripleAgent on two medium-sized real-world applications.

Future Work on TripleAgent

- Devising and implementing fault models that stimulate common mode failures or data errors.
- Exploring the design space of perturbation and failure-obliviousness strategies.
- Applying TripleAgent in a real production environment.

Royal-Chaos @ GitHub



https://github.com/KTH/royal-chaos

Thanks for listening!

Long Zhang longz@kth.se