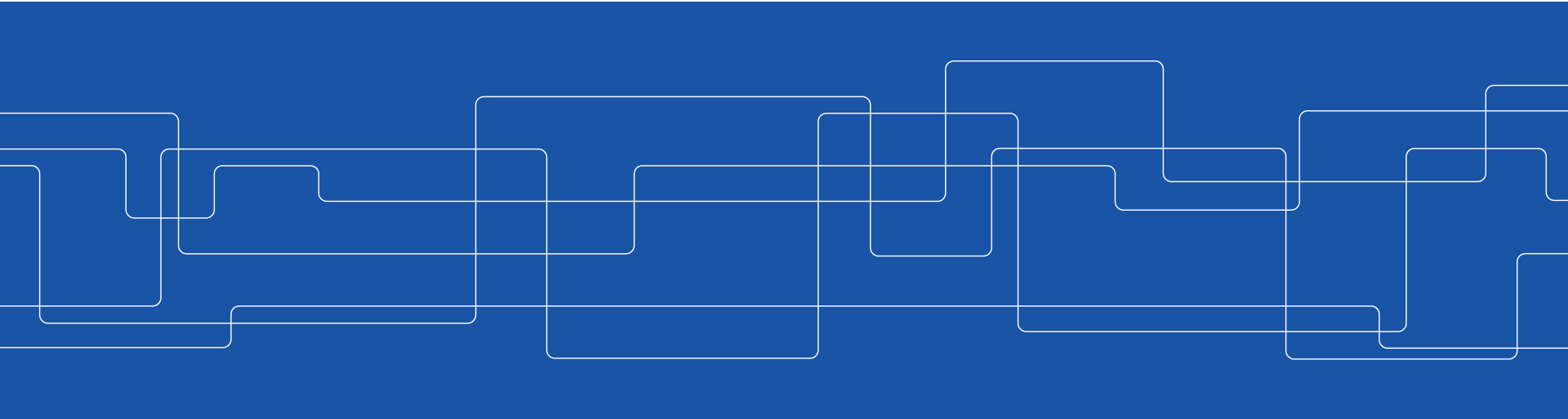
KTH ROYAL INSTITUTE
OF TECHNOLOGY

A Chaos Engineering System for Live Analysis and Falsification of Exception-handling in the JVM

Long Zhang

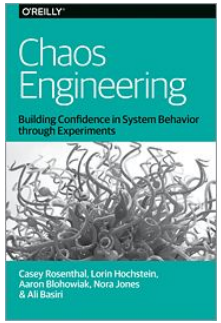
longz@kth.se



Contents

- Background Research Work About Chaos Machine
- The Architecture And Workflow of Chaos Machine
- Evaluation Work by 3 Real Projects
- Future Work

A Famous Book



- Examples of inputs for chaos experiments (Chapter 1, page 4):
  - Simulating the failure of an entire region or datacenter.
  - Partially deleting Kafka topics over a variety of instances to recreate an issue that occurred in production.
  - Injecting latency between services for a select percentage of traffic over a predetermined period of time.
  - **Function-based chaos (runtime injection): randomly causing functions to throw exceptions.**
  - Code insertion: Adding instructions to the target program and allowing fault injection to occur prior to certain instructions.
  - Time travel: forcing system clocks out of sync with each other.
  - Executing a routine in driver code emulating I/O errors.
  - Maxing out CPU cores on an Elasticsearch cluster.
- The opportunities for chaos experiments are boundless and may vary based on the architecture of your distributed system and your organization's core business value.

https://www.oreilly.com/ideas/chaos-engineering

An Excellent Talk

Link to The Video

Level 0: Chaos Monkey
Level 1: Infrastructure Failures
Level 1.5: Network Failures
Level 2: Application Failures

We Are Focusing on…:

Building confidence in system behavior through EXPERIMENTS in RUNTIME

Javaagent, Java byte-code, ASM

Java Virtual Machine

Error-handling behaviors
(Try-catch blocks' resilience)

A Chaos Engineering System for Live Analysis and Falsification of Exception-handling in the JVM

Background Research Work

Exception Handling Analysis and Transformation Using Fault Injection - Study of Resilience Against Unanticipated Exceptions

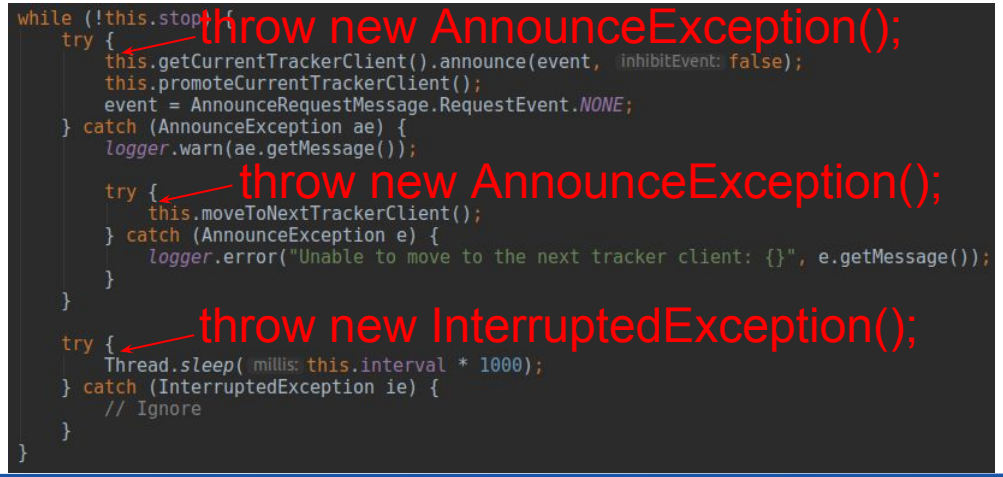
Benoit Cornu, Lionel Seinturier, Martin Monperrus

- Resilience analysis by injecting exceptions during test suite execution
- Definition of two contracts for exception handling
- An empirical evaluation on 9 open sources applications

https://hal.inria.fr/hal-01062969/file/exception-analysis-resilience-ist.pdf

Background Research Work

- Try-catch block short-circuit testing
  - A corresponding exception at the beginning
  - Make the whole try block invalid




```
while (!this.stop) {
    try {
        this.getCurrentTrackerClient().announce(event, inhibitEvent: false);
        this.promoteCurrentTrackerClient();
        event = AnnounceRequestMessage.RequestEvent.NONE;
    } catch (AnnounceException ae) {
        logger.warn(ae.getMessage());

        try {
            this.moveToNextTrackerClient();
        } catch (AnnounceException e) {
            logger.error("Unable to move to the next tracker client: {}", e.getMessage());
        }
    }

    try {
        Thread.sleep( millis: this.interval * 1000);
    } catch (InterruptedException ie) {
        // Ignore
    }
}
```

The Overview of ChaosMachine

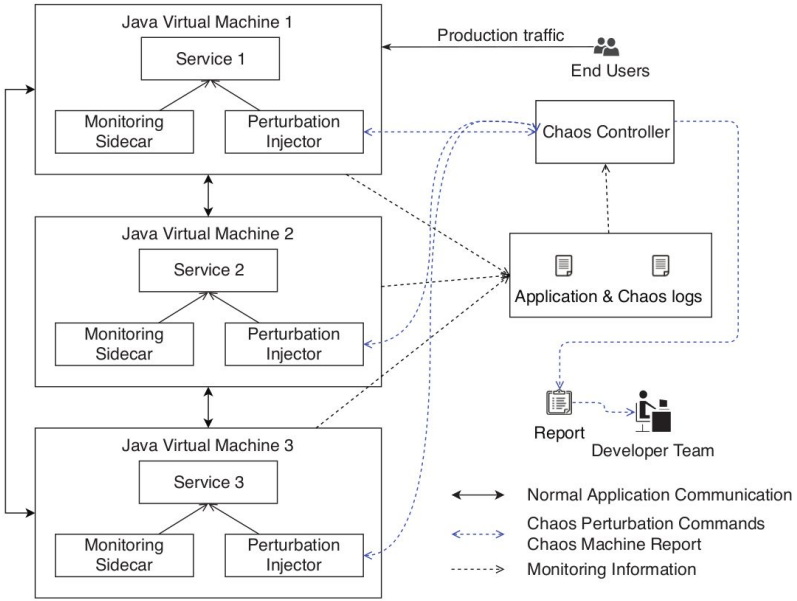


Fig. 1. The components of CHAOSMACHINE

- Input
  - Arbitrary software in Java
  - Hypotheses
- Architecture
  - Monitoring sidecars
  - Perturbation Injectors
  - Chaos Controller
- Output
  - A report and monitoring logs

Hypotheses

- Resilience hypothesis
  - The observable behavior of the catch block, executed upon exception, is equivalent to the observable behavior of the try-block when no exception happens.
- Observability hypothesis
  - An exception caught in the catch block results in user-visible effects.
- Debug hypothesis
  - An exception caught in the catch block results in an explicit message in logs.
- Silence hypothesis
  - It fails to provide the expected behavior upon exception while providing no troubleshooting information whatsoever, i.e., it is neither observable nor debuggable.

Experiments And Modes

ChaosMachine performs two kinds of experiments:

- **Falsification experiments**
- **Exploration experiments**

When ChaosMachine does not introduce chaos, it is in: observation mode.

Monitoring sidecar

Classification	Details
Try-catch blocks information	• Position: which class, which method • Exception type: exception class full name • Executed times in both normal mode and injection mode
Application logs	• When exception is injected, whether it will print error messages
Exit status	• UNIX code (mainly for client applications) • HTTP response code
Generic metrics	• CPU usage • Latency • ...
Domain-specific metrics	• Client applications, e.g. TTorrent: whether the files can be downloaded • Web applications, e.g. XWiki: HTTP response code, response body

Perturbation Injector

- Using JavaAgent to transform bytecodes

- In the beginning of each try-catch block
  - “ChaosMonkey.doChaos(...)”

Chaos Controller

- Activates or deactivates perturbation injectors (the blast radius)
- Analyzes the information recorded by the monitoring sidecars
- Generates a report

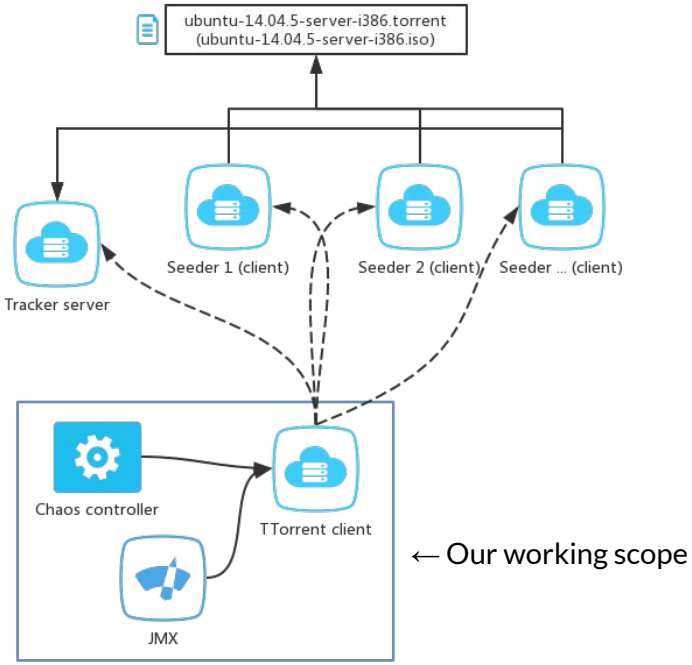
TABLE I
INTERPLAY BETWEEN THE 3 COMPONENTS AND THE 3 MODES OF CHAOSMACHINE

	Observation Mode	**Exploration Mode**	**Falsification Mode**
Monitoring Sidecar	Monitors all the relevant execution information	Monitors how the system reacts according to a perturbation	Monitors whether an hypothesis is falsified
Perturbation Injector	Not active	Injects a specific perturbation	Injects a specific perturbation
Chaos Controller	Deactivate all the perturbation injectors to keep the system running as usual	Controls perturbation injectors to conduct a sequence of chaos experiments so as to infer new hypotheses	Controls perturbation injectors according to a specific hypothesis

About The Report: What Can Be Learned

- Try-catch classification
  - Fragile ones
  - Possible resilient ones
- Logs handling mechanisms

Experiments On TTorrent



← Our working scope

Highlights:

- A pretty good project with 1000+ stars
- A real-world production usage
- Real production environment and traffic

Experiments On TTorrent

TABLE II
THE RESULTS OF CHAOS EXPERIMENTATION WITH EXCEPTION INJECTION ON 27 TRY-CATCH BLOCKS IN THE TTORRENT BITTORRENT CLIENT

Try-catch block information	Execution Anal./Expl.	Logged	Downl.	Exit status	System metrics	RH	OH	DH	SH
BEValue/getBytes,ClassCastException,0	41 / 1	yes	no	crashed	-		x	x	
BEValue/getNumber,ClassCastException,0	15 / 1	yes	no	crashed	-		x	x	
BEValue/getString,ClassCastException,0	37 / 1	yes	no	crashed	-		x	x	
BEValue/getString,UnsupportedEncodingException,1	37 / 1	yes	no	crashed	-		x	x	
ClientMain/main,CmdLineParser$OptionException,0	1 / 1	yes	no	crashed	-		x		
ClientMain/main,Exception,1	1 / 1	yes	no	crashed	-		x	x	
Announce/run,AnnounceException,0	1 / 60	yes	no	stalled	-		x	x	
Announce/run,InterruptedException,2	1 / 760	no	yes	normally	more threads			x	
Announce/run,InterruptedException,3	1 / 1	no	yes	normally	no diff	x			
Announce/run,AnnounceException,4	1 / 1	yes	yes	normally	no diff	x		x	
Announce/stop,InterruptedException,0	1 / 1	no	yes	normally	no diff	x			
ConnectionHandler/run,SocketTimeoutException,0	1290 / 1030	no	yes	normally	no diff	x			
ConnectionHandler/run,IOException,1	1290 / 1	yes	yes	stalled	higher cpu			x	
ConnectionHandler/run,InterruptedException,2	1290 / 2	yes	no	stalled	no diff			x	
ConnectionHandler/stop,InterruptedException,0	1 / 1	no	yes	normally	no diff	x			
ConnectionHandler$ConnectorTask/run,Exception,0	50 / 50	yes	no	stalled	no diff			x	
Handshake/craft,UnsupportedEncodingException,0	50 / 48	yes	no	stalled	no diff			x	
PeerExchange/send,InterruptedException,0	90763 / 210	no	no	stalled	no diff				x
PeerExchange/stop,InterruptedException,0	46 / 44	no	yes	normally	no diff	x			
PeerExchange$OutgoingThread/run,InterruptedException,0	90805 / 32984841	no	no	stalled	higher cpu			x	x
PeerExchange$OutgoingThread/run,InterruptedException,1	90763 / 288	no	no	stalled	no diff				x
PeerExchange$OutgoingThread/run,IOException,2	90805 / 43	yes	no	stalled	no diff			x	
PeerExchange$OutgoingThread/run,IOException,3	90763 / 46	yes	no	stalled	no diff			x	
Piece/validate,NoSuchAlgorithmException,0	2564 / 5427	yes	no	stalled	higher cpu			x	
HTTPAnnounceRespMessage/parse,InvalidBEncodingException,0	3 / 30	yes	no	stalled	no diff			x	
HTTPAnnounceRespMessage/parse,InvalidBEncodingException,1	3 / 30	yes	no	stalled	no diff			x	
HTTPAnnounceResponseMessage/parse,UnknownHostException,2	3 / 30	yes	no	stalled	no diff			x	
total: 27/52	460626 / 32992950	18/27	8/27	7/27	4/27	6/27	7/27	20/27	3/27

- Total try-catch blocks: 52
- Covered by workload: 27
- Possible resilient ones: 6
- Silent ones: 3

Experiments On TTorrent

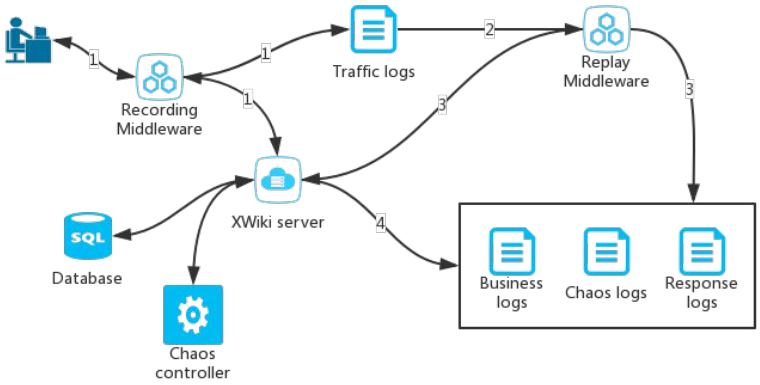
- A worst case for developers (But a best case for us :D)

```java
/**
 * Send a message to the connected peer.
 *
 * <p>
 * The message is queued in the outgoing message queue and will be
 * processed as soon as possible.
 * </p>
 *
 * @param message The message object to send.
 */
public void send(PeerMessage message) {
    try {
        this.sendQueue.put(message);
    } catch (InterruptedException ie) {
        // Ignore, our send queue will only block if it contains
        // MAX_INTEGER messages, in which case we're already in big
        // trouble, and we'd have to be interrupted, too.
    }
}
```

- Successfully injected the exception
- Didn’t capture error info in application logs
- Failed to download the files
- The application is stalled

No one knows what's happening here!

Experiments On XWiki (And Broadleaf Commerce)



Recording/replaying traffic: goreplay.org

- Step 1: production traffic recording

TABLE II
XWIKI'S USER TRAFFIC RECORDS FOR REPLAYING AND TESTING

Request functionalities	Type	Quantity
Directly downloading files, like .css, .js, .png ...	GET	80
Rendering pages or fetching content parts	GET	148
	POST	18
Logging in, adding comments, updating user data	POST	4
Logging out	GET	1
total	GET	229
	POST	22

- Step 2+3: user traffic replaying
- Step 4: analyzing information

Experiments On XWiki (And Broadleaf Commerce)

TABLE III
RESULTS ON CHAOS EXPERIMENTATION ON 268 TRY-CATCH BLOCKS IN XWIKI COVERED BY THE CONSIDERED WORKLOAD

Packages	Covered	Executions in Anal. / Expl.	RH	OH	DH	SH
org/xwiki/a*	1	273 / 273	0	0	1	0
org/xwiki/c*	20	112968 / 119544	0	6	20	0
org/xwiki/d*	2	855 / 1398	0	0	2	0
org/xwiki/e*	11	20882 / 99204	0	1	11	0
org/xwiki/f*	23	44813 / 222	0	0	23	0
org/xwiki/i*	8	1142 / 280	0	0	8	0
org/xwiki/l*	12	295530 / 73048	0	1	12	0
org/xwiki/m*	9	38360 / 37739	0	1	9	0
org/xwiki/n*	10	62 / 190837	0	0	8	2
org/xwiki/o*	2	43753 / 68154	0	0	2	0
org/xwiki/p*	4	5403 / 3075	0	0	4	0
org/xwiki/q*	3	262 / 142	0	0	3	0
org/xwiki/r*	93	1137420 / 272944	5	7	70	14
org/xwiki/s*	15	20522 / 31826	2	5	15	0
org/xwiki/t*	2	83 / 81	0	0	2	0
org/xwiki/u*	20	13795 / 6229	0	8	16	1
org/xwiki/v*	5	3201 / 831	0	2	5	0
org/xwiki/w*	21	2526 / 3140	0	2	16	5
org/xwiki/x*	7	890 / 580	0	0	6	1
Total	268/1567	1742740 / 909547	7	33	233	23

- Total try-catch blocks: 1567
- Covered by workload: 268
- Possible resilient ones: 7
- Silent ones: 23

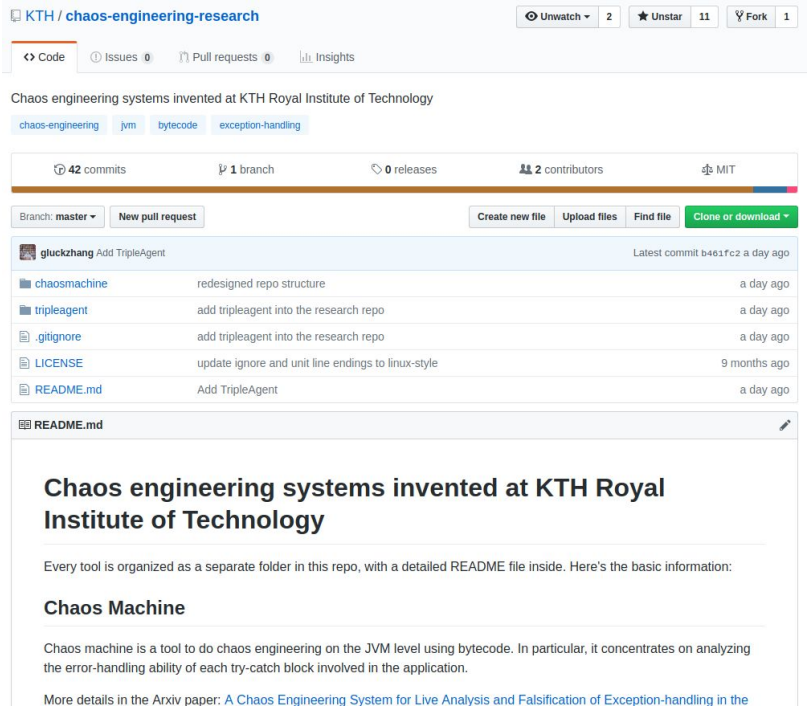
Our Contributions

- The conceptual foundations of chaos engineering in the context of exception-handling in Java
- A novel system, called CHAOSMACHINE, that assesses exception-handling capabilities in production
- An empirical evaluation of CHAOSMACHINE on 3 real-world Java systems totaling 630k line of codes, containing 339 try-catch blocks executed by the considered production traffic

Future Work

- More advanced perturbation strategies
- Perturbation injection + failure-obliviousness

Our Contributions



- Paper available at arXiv
- Source-code available at GitHub

Thanks for listening!

Long Zhang

longz@kth.se