EDFI: A Dependable Fault Injection Tool for Dependability Benchmarking Experiments

Cristiano Giuffrida  Anton Kuijsten  Andrew S. Tanenbaum

Vrije Universiteit Amsterdam

19th IEEE Pacific Rim International Symposium on Dependable Computing

Vancouver, BC, Canada
December 2-4, 2013
Fault Injection

- Well-established dependability benchmarking technique.
- Inexpensively emulates realistic faults in a synthetic setting.
- Previously applied to several different categories of systems.

Common application scenarios:

- Characterize system behavior under errors.
- Evaluate effectiveness of fault-tolerance mechanisms.
- High-coverage testing of error-handling code.
Existing Fault Injection Techniques

**Preruntime, location-based**
- Weak faultload coverage guarantees (poor *precision*).
- Cannot prevent spurious fault activation (poor *controllability*).

**Run-time, location-based**
- Similar to preruntime, but injections are triggered by HW/SW traps.
- Seeks to address the controllability issues of preruntime strategies.

**Run-time, time-based**
- Periodically interrupts execution to perform fault injection.
- Significantly constraints the nature of the injected faults.
- Poor reproducibility and comparability of the results.
WWW: What We Want

- A “dependable” fault injection tool.
- Provides strong faultload coverage guarantees.
- Structurally prevents spurious fault activation.
- Guarantees full observability of the experiments.
- Delivers reproducible and comparable fault injection experiments.
- Adapts to different systems and (nondeterministic) workloads.
EDFI Overview

- Statically injects multiple simultaneous faults over the entire code.

- Relies on instrumentation to dynamically activate faults on demand.

- Guarantees a controlled and predetermined faultload at runtime.

- Delivers *precise*, *controllable*, and *observable* experiments.

- Supports several possible realistic software fault types.

- Allows users to specify complex static and dynamic fault models.
EDFI: A Dependable Fault Injection Tool for Dependability Benchmarking Experiments

Anton Kuijsten
Execution-driven Fault Injection

Overview

- Compiles the code into multiple (faulty and nonfaulty) versions.
- Interleaves them in a user-controlled way during the experiment.
- Implemented using an efficient *basic block cloning* strategy.

Basic block cloning

- **Pristine BB**: original unmodified BB, used during normal execution.
- **Fault-free BB**: original unmodified BB, used during the experiment.
- **Faulty BB**: transformed BB with faults, used during the experiment.
- **FDP BB**: fault decision point, used during the experiment.
Basic Block Cloning Example

(a) Original CFG  
(b) CFG after function entry instrumentation
Static Fault Model

- Shapes the faultload distribution at preruntime.
- Specifies fault types, locations, conditions, and intensity.
- User inputs fault types and their probability of occurrence.
- Static fault handlers (SFHs) specify conditions of occurrence.
- Static fault impact factors (SFIFs) amplify fault probabilities.
- SFHs and SFIFs are evaluated on a per-fault location basis.
class StaticFaultHandler {
    virtual void init(Module &M, string &params) {}
    virtual bool canInject(Value *faultLocation, double faultProb) = 0;
    virtual void inject(Value *faultLocation) = 0;
};

class StaticFaultAnalyzer {
    static double getMaxSFIF(void);
    virtual void init(Module &M, string &params) {}
    virtual double getSFIF(Value *faultLocation) = 0;
};
Dynamic Fault Model

- Controls and alters the faultload distribution at runtime.

- Dynamic fault triggers (DFTs):
  - Determine when to switch to faulty execution.
  - Invoked by instrumentation at fault decision points.
  - Support for transient, intermittent, and temporal faults.

- Dynamic fault loggers (DFLs):
  - Determine what to do when faults are activated.
  - Invoked by instrumentation at fault activation points.
  - Support for event logging and fault injection statistics.
void edfi_onstart(edfi_context_t *context);

int edfi_onfdp(edfi_context_t *context, const char *file, int line);

void edfi_onfault(edfi_context_t *context, const char *file, int line, int num_fault_types, ...);

void edfi_onstop(edfi_context_t *context);
ImplementEDFI with support for generic UNIX programs.

Evaluated on 2 popular server programs (Apache httpd and MySQL).

Modest virtual memory overhead observed at runtime (1.5-44.5%).

Low performance overhead during normal execution (0.7-5.4%).

DFTs and DFLs impact during the experiment: 1.8-644.8%.
EDFI: A Dependable Fault Injection Tool for Dependability Benchmarking Experiments

Anton Kuijsten

Precision

Fault load degradation (%) vs. Fault probability (%)

- EDFI
- Location-based
- Interface-level

Graph showing the relationship between fault load degradation and fault probability for different fault injection methods.
Controllability

EDFI: A Dependable Fault Injection Tool for Dependability Benchmarking Experiments
Anton Kuijsten
Precision-controllability Tradeoff

EDFI: A Dependable Fault Injection Tool for Dependability Benchmarking Experiments

Anton Kuijsten
Summary


- Guarantees a predetermined and controlled faultload at runtime.

- Delivers *precise, controllable*, and *observable* experiments.

- Supports sophisticated static and dynamic fault models.

- Provides fine-grained control over the fault injection experiment.

- Minimizes perturbations to the system during normal execution.
EDFI: A Dependable Fault Injection Tool for Dependability Benchmarking Experiments

Thank you!
Any questions?

Cristiano Giuffrida, Anton Kuijsten, Andy Tanenbaum
{giuffrida,kuijsten,ast}@cs.vu.nl

Vrije Universiteit Amsterdam