FINJ: A Fault Injection Tool for HPC Systems

A. Netti, Z. Kiziltan, O. Babaoglu, A. Sirbu, A. Bartolini and A. Borghesi

Department of Computer Science and Engineering (DISI), University of Bologna

11° Resilience Workshop, 28/08/2018

Introduction

- **Exascale** systems are the current goal of the HPC community
- Will be attained through sheer **parallelism**
  - Increased **power** consumption
  - Excessively-high **failure** rates
- More intelligent and effective **resiliency** techniques must be developed

Introduction

- **Fault injection**: the deliberate triggering of faults in a system to research resiliency techniques
- Most available frameworks are very **specific**
- Our contribution, the **FINJ** framework
  - **Integration** of different fault types
  - **Workload** support for complex experiments
  - **Ease of use** and customizability

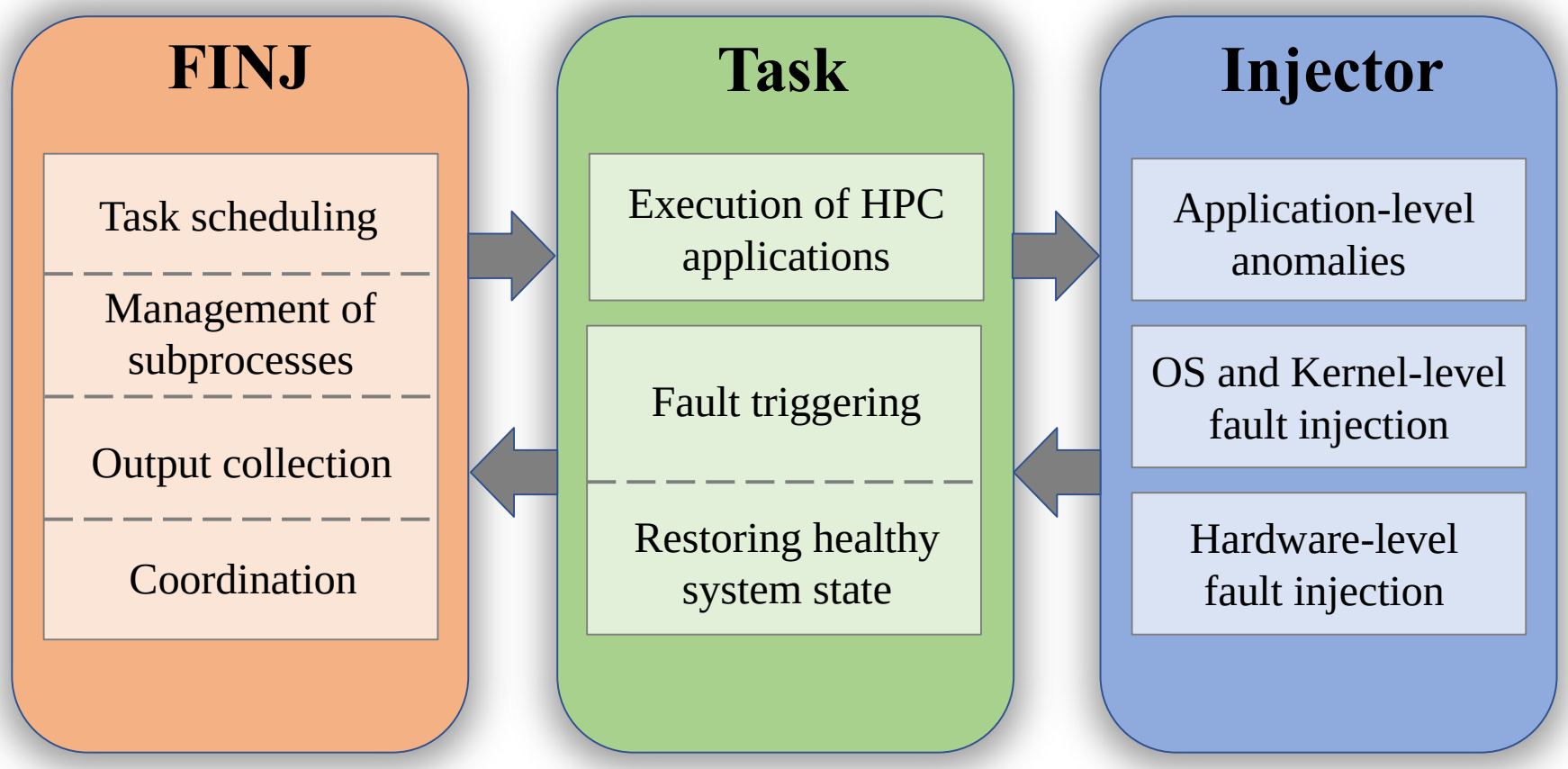
Outline

- Overview of FINJ
- Implementation
- Usage Example
- Live Demo
- Conclusions

Overview of FINJ

- FINJ is based on **tasks**
  - Fault-triggering programs or HPC applications
  - Can be any executable file
  - Can be pinned to specific cores
  - Tied to a specific duration and starting time
- A set of task specifications makes a **workload**
  - Workloads are in **CSV** format
  - A specific run of a workload is an **injection session**

Overview of FINJ



Overview of FINJ

- FINJ **Engine**
  - Starts and terminates tasks on target hosts
  - Communicates all output to controllers
- FINJ **Controller**
  - Orchestrates injection sessions with task commands
  - Can control multiple engines
  - Stores all output collected by engines
- TCP-based **message** protocol for communication

Implementation

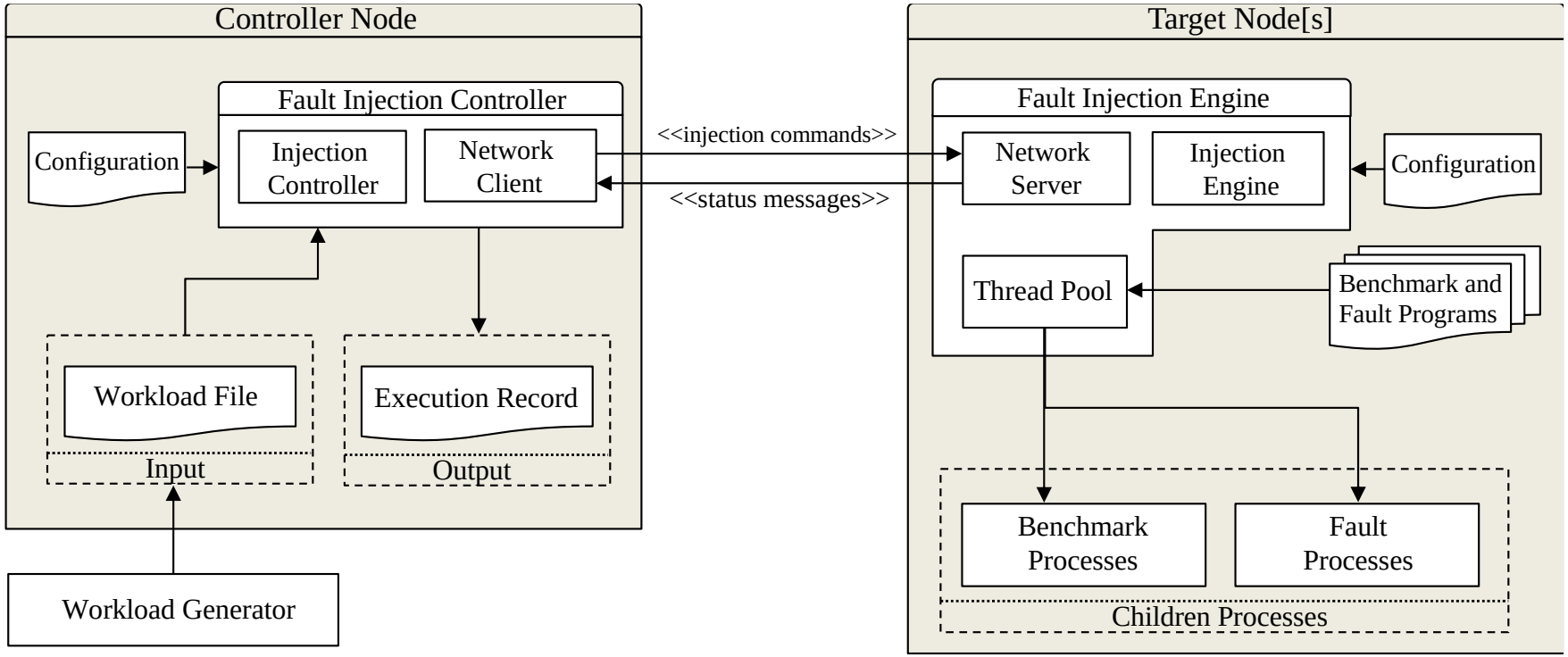
- FINJ is written in **Python**
- **Modular** architecture, easy to customize
- Extremely low **overhead**
- Main **components**
  - Thread pool
  - TCP-based client and server
  - I/O readers and writers

Implementation

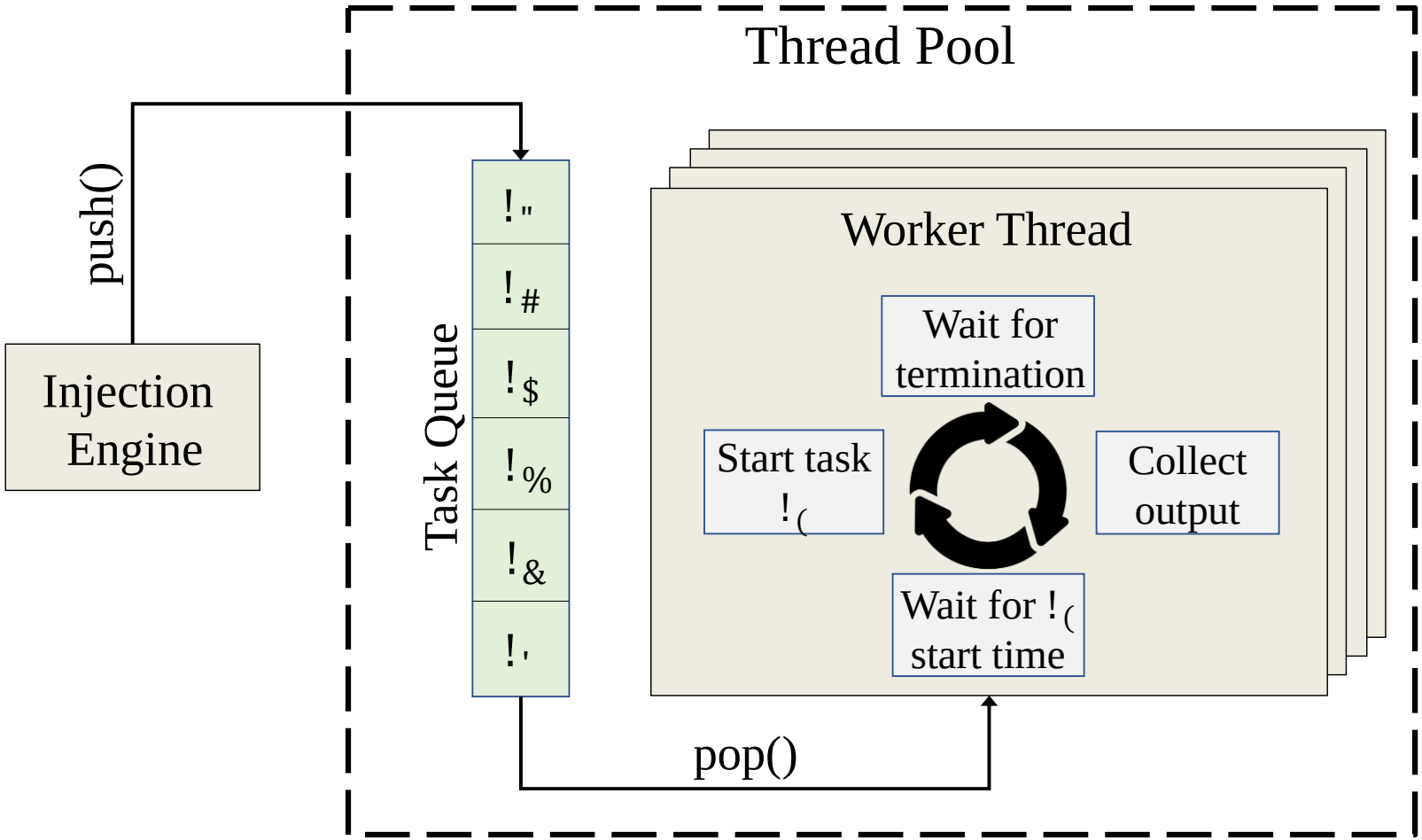
- **Additional** components are supplied with FINJ
  - A **workload generator**, to create large workloads with specific statistical features
  - A collection of anomalies and **fault-triggering programs** ready to be used
- Currently available on **GitHub**
- Open-source under the **MIT** license

https://github.com/AlessioNetti/fault_injector

Implementation



Implementation



Usage Example

- Usage **example** on a real HPC node
  - 2x Intel E5-2630v3 CPUs, 128GB RAM, CentOS 7.3
- Intel **HPL** benchmark used to load the system
- Two **faults** in the workload
  - **leak**: causes a memory leak
  - **cpufreq:** reduces maximum allowed CPU frequency
- **LDMS** used to collect performance metrics

Usage Example

- Format of the **workload**

```
timestamp;duration;seqNum;isFault;cores;args
0;1723;1;False;0-7;./hpl lininput
355;244;2;True;6;sudo ./cpufreq 258
914;291;3;True;4;./leak 316
```

- Commands to **instantiate** FINJ locally

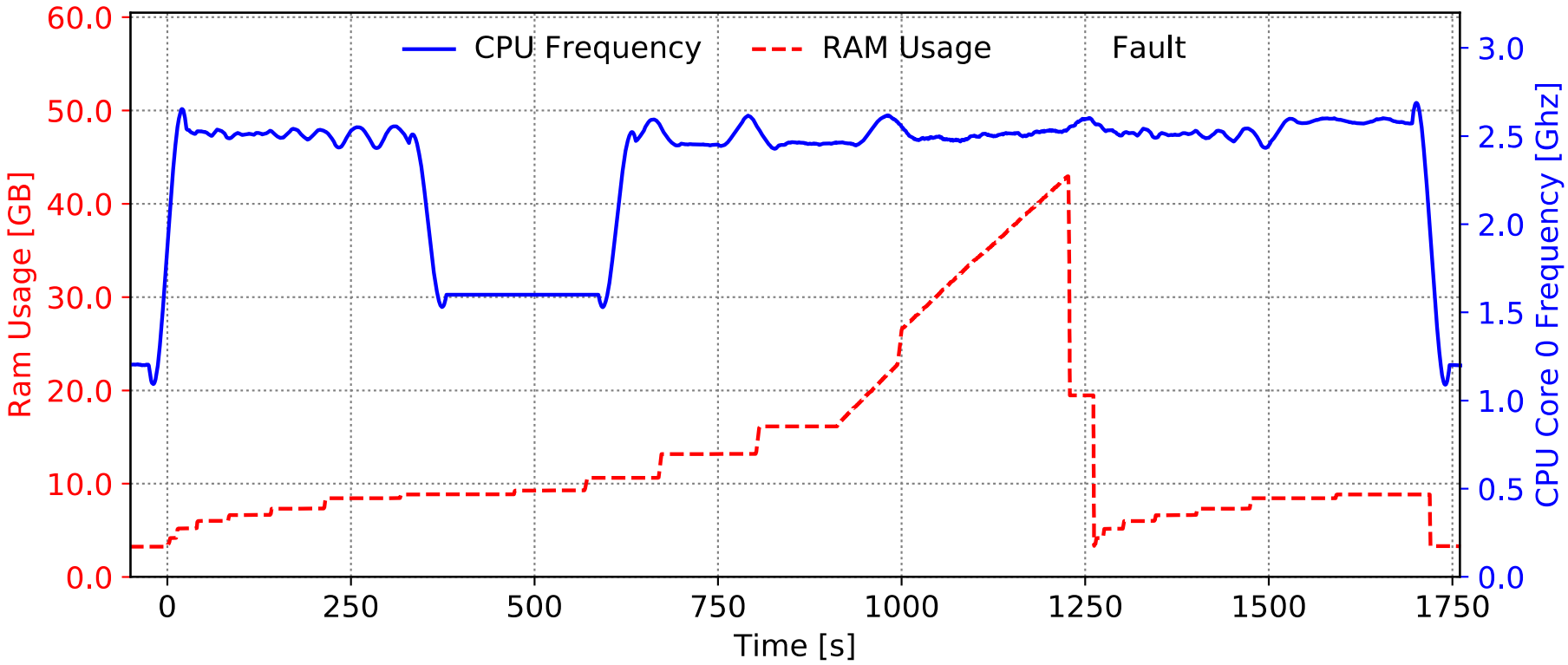
```
python finj_engine -p 30000 &
python finj_controller -w sample.csv -a localhost:30000
```

Usage Example

- **Output** of the injection session

```
timestamp;type;args;seqNum;duration;isFault;cores;error
1529172604;command_session_s;None;None;None;None;None;None
1529172624;status_start;./hpl lininput;1;1723;False;0-7;None
1529172979;status_start;sudo ./cpufreq 258;2;258;True;6;None
1529173237;status_end;sudo ./cpufreq 258;2;258;True;6;None
1529173538;status_start;./leak 316;3;316;True;4;None
1529173855;status_end;./leak 316;3;316;True;4;None
1529174347;status_end;./hpl lininput;1;1723;False;0-7;None
1529174348;command_session_e;None;None;None;None;None;None
```

Usage Example



Usage Example

Presentation continues with a live demo of the FINJ tool

Conclusions

- FINJ is a **flexible** fault injection framework
  - Can be **integrated** with any other fault injector
  - Can control **complex** experiments
- Future work
  - Testing at **scale**
  - Integration with other **transport** methods
  - Implementation of alternative task **triggers**