Revisiting Benchmarking for Concurrent Runtime Verification

Duncan Paul Attard · Monday, June 19th 2023
University of Glasgow
A litmus test for tools

1. Benchmarking
   - Quantify overhead
2. Stress test
3. Reproduce results
4. Compare tools

Duncan Paul Attard · University of Glasgow
A litmus test for tools

1. Benchmarking
   - Quantify overhead
   - Stress test
2. Gather meaningful metrics
3. Reproduce results
4. Compare tools

Duncan Paul Attard · University of Glasgow
A litmus test for tools

1. Quantify overhead
2. Stress test
3. Reproduce results
4. Compare tools

- Benchmarking
- Gather meaningful metrics
- Generate high loads
- Cover general scenarios

Duncan Paul Attard · University of Glasgow
A litmus test for tools

1. Quantify overhead
2. Stress test
3. Reproduce results
4. Compare tools

- Benchmarking
- Gather meaningful metrics
- Generate high loads
- Be configurable

Duncan Paul Attard · University of Glasgow
A litmus test for tools

1. Benchmarking
   - Quantify overhead
   - Generate high loads

2. Stress test
   - Gather meaningful metrics
   - Be configurable

3. Reproduce results
   - Generate high loads
   - Cover general scenarios

4. Compare tools
   - Be configurable
   - Cover general scenarios

Duncan Paul Attard · University of Glasgow
Reactive systems

Reactive system
Reactive systems

1. Responsive

Reactive system
Reactive systems

1. Responsive

2. Resilient

Duncan Paul Attard · University of Glasgow
Reactive systems

1. Responsive
2. Resilient
3. Message driven

Duncan Paul Attard · University of Glasgow
Reactive systems

1. Responsive
2. Resilient
3. Message driven
4. Elastic
Requirements for reactive systems

Gather meaningful metrics

Generate high loads

Be configurable

Cover general scenarios

Reactive system aspects

Responsive
Resilient
Message-driven
Elastic
**Requirements for reactive systems**

- Gather meaningful metrics
- Generate high loads
- Be configurable
- Cover general scenarios

Responsive

1. Response time/latency
   - Memory consumption
   - Scheduler/CPU usage

Duncan Paul Attard · University of Glasgow
**Requirements for reactive systems**

1. **Responsive**
   - Response time/latency
   - Memory consumption
   - Scheduler/CPU usage

2. **Resilient**
   - Scalable
   - Uniform load distribution

- Gather meaningful metrics
- Generate high loads
- Be configurable
- Cover general scenarios

Duncan Paul Attard - University of Glasgow
Requirements for reactive systems

- Responsive
  - Gather meaningful metrics
  - Generate high loads
- Resilient
  - Be configurable
  - Cover general scenarios
- Message-driven
  - Control reaction to messages
  - Reproduce load conditions

1. Reactive system aspects
   - Response time/latency
   - Memory consumption
   - Scheduler/CPU usage
2. Scalable
   - Uniform load distribution
3. Grow + shrink dynamically
   - Use typical load profiles

Duncan Paul Attard · University of Glasgow
Requirements for reactive systems

1. **Responsive**
   - Response time/latency
   - Memory consumption
   - Scheduler/CPU usage

2. **Resilient**
   - Scalable
   - Uniform load distribution

3. **Message-driven**
   - Control reaction to messages
   - Reproduce load conditions

4. **Elastic**
   - Grow + shrink dynamically
   - Use typical load profiles

- Gather meaningful metrics
- Generate high loads
- Be configurable
- Cover general scenarios
Current BM tools cater for limited to no concurrency
The state of the art

“Current BM tools cater for limited to no concurrency”

× Wrong tool
The state of the art

"Current BM tools cater for limited to no concurrency”

× Wrong tool

? Right tool

× Wrong job
The state of the art

“Current BM tools cater for limited to no concurrency”

× Wrong tool       ? Right tool       ? Right tool
× Wrong job        ? Right job
× Not enough
The typical recipe
The typical recipe

Client

Server

Harness

VM

- Web/REST
- SMTP
- Broker

Load gen

- TCP traffic

Scripts

CSV

Gather meaningful metrics

1. Generate high loads
2. Cover general scenarios
3. Be configurable
The typical recipe

- Web/REST
- SMTP
- Broker

(master-worker)

Client

Server

Harness

VM

Duncan Paul Attard · University of Glasgow
The typical recipe

- JMeter
- TSUNG
- Gatling

Load gen → Client

Harness → VM → Server

- Web/REST
- SMTP
- Broker

(master-worker)

1. Generate high loads
2. Be configurable
3. Cover general scenarios
4. Gather meaningful metrics

Duncan Paul Attard · University of Glasgow
The typical recipe

- Web/REST
- SMTP
- Broker

Load gen
(synthetic load)

Client

Harness

VM

Server

Duncan Paul Attard · University of Glasgow
The typical recipe

- JMeter
- TSUNG
- Gatling

Load gen
(synthetic load)

Client

Harness

VM

Server

- Web/REST
- SMTP
- Broker

(master-worker)

Gather meaningful metrics

1.
Generate high loads

2.
Be configurable

3.
Cover general scenarios

4.

Duncan Paul Attard · University of Glasgow
The typical recipe

- JMeter
- TSUNG
- Gatling

Client

Load gen
(synthetic load)

VM

Harness
(master-worker)

Server

TCP traffic
(network assumptions)

TCP traffic
(network assumptions)

• Web/REST
• SMTP
• Broker

Gather meaningful metrics
1 and
Generate high loads
2
Be configurable
3
Cover general scenarios
4

Duncan Paul Attard · University of Glasgow
The typical recipe

Load gen -> Client -> Server

- Web/REST
- SMTP
- Broker

(response time)

(memory + CPU)

(TCP traffic)

(synthetic load)

(network assumptions)

Duncan Paul Attard · University of Glasgow
The typical recipe

- **JMeter**
- **TSUNG**
- **Gatling**

(synthetic load)

**Web/REST**

**SMTP**

**Broker**

(server)

Client

Server

- Scripts
- Load gen

- Harness
- VM

TCP traffic

Gather meaningful metrics

1. Generate high loads
2. Be configurable
3. Cover general scenarios

Duncan Paul Attard · University of Glasgow
The typical recipe

- Gather meaningful metrics
- Generate high loads
- Be configurable
- Cover general scenarios

- Load gen
- Scripts
- CSV
- response time
- memory + CPU
- TCP traffic

- VM
- Harness

- Client
- Server

- Web/REST
- SMTP
- Broker

(synthetic load)

(network assumptions)
The typical recipe

Gather meaningful metrics

- Scripts
- Load gen
- Harness

1. Generate high loads
2. Be configurable
3. Cover general scenarios

- Web/REST
- SMTP
- Broker

TCP traffic

(response time)

(memory + CPU)

(Duncan Paul Attard · University of Glasgow)
The typical recipe

1. Gather meaningful metrics
   - Web/REST
   - SMTP
   - Broker

2. Generate high loads
   - Apache JMeter™
   - TSUNG™
   - Gatling

(1) and X

- Scripts
- Load gen
- CSV
- response time
- memory + CPU
- TCP traffic
- (network assumptions)

Client

Server

VM

Harness

Duncan Paul Attard · University of Glasgow
The typical recipe

1. Gather meaningful metrics
   - Web/REST
   - SMTP
   - Broker
   - (master-worker)

2. Generate high loads
   - Apache JMeter
   - TSUNG
   - Gatling
   - (synthetic load)

(response time)

(memory + CPU)

(TCP traffic)

(network assumptions)

Duncan Paul Attard · University of Glasgow
The typical recipe

1. Gather meaningful metrics
   - Web/REST
   - SMTP
   - Broker
   (master-worker)

2. Generate high loads
   - Apache JMeter™
   - TSUNG
   - Gatling
   (synthetic load)

3. Be configurable
   - 1. Generate high loads
   - 2. Gather meaningful metrics
   - 3. Be configurable

TCP traffic
(network assumptions)

Gather meaningful metrics
(response time)

Load gen

Scripts

CSV

Client

Server

VM

Harness

Be configurable

Generate high loads

Duncan Paul Attard · University of Glasgow
**The typical recipe**

1. Gather meaningful metrics
   - Web/REST
   - SMTP
   - Broker
   - (master-worker)
   - TCP traffic (network assumptions)

2. Generate high loads
   - JMeter
   - TSUNG
   - Gatling
   - (synthetic load)

3. Be configurable
   - Scripts
   - Load gen
   - CSV
   - response time
   - memory + CPU

Duncan Paul Attard · University of Glasgow
The typical recipe

1. Gather meaningful metrics
   - Web/REST
   - SMTP
   - Broker
   (master-worker)
   - TCP traffic
   - (network assumptions)

2. Generate high loads
   - Apache JMeter
   - TSUNG
   - Gatling
   (synthetic load)

3. Be configurable
   - Scripts
   - CSV
   - memory + CPU
   - response time
   - Load gen

Duncan Paul Attard · University of Glasgow
The typical recipe

1. Gather meaningful metrics
   - Web/REST
   - SMTP
   - Broker
   - (master-worker)

2. Generate high loads
   - Apache JMeter™
   - TSUNG
   - Gatling
   - (synthetic load)

3. Be configurable

Be configurable

Gather meaningful metrics

Generate high loads

(synthetic load)

TCP traffic

(network assumptions)

Client

Server

VM

Harness

CSV

Scripts

Load gen

response time

memory + CPU

Duncan Paul Attard · University of Glasgow
The typical recipe

1. Gather meaningful metrics
   - response time
   - memory + CPU
   - TCP traffic
   - (network assumptions)

2. Generate high loads
   - Scripts
   - Load gen
   - (synthetic load)
   - CSV
   - (master-worker)

3. Be configurable
   - Web/REST
   - SMTP
   - Broker

4. Cover general scenarios
The typical recipe

1. Gather meaningful metrics
   - phusion's TSUNG
   - Apache JMeter
   - Gatling
   - Scripts (synthetic load)
   - Load gen
   - CSV
   - response time
   - memory + CPU
   - TCP traffic
   - (network assumptions)

2. Generate high loads
   - Load gen
   - (synthetic load)

3. Be configurable
   - Web/REST
   - SMTP
   - Broker
   - (master-worker)

4. Cover general scenarios

Duncan Paul Attard · University of Glasgow
Wouldn’t it be nice if…

1. Gather meaningful metrics
2. Generate high loads
3. Be configurable
4. Cover general scenarios
Wouldn’t it be nice if...

1. Gather meaningful metrics
2. Generate high loads
3. Be configurable
4. Cover general scenarios

- Ease of use
- \( \approx \) real scenarios
- Meets requirements 1 to 4
Wouldn’t it be nice if…

On Benchmarking for Concurrent Runtime Verification*

Luca Aceto²,³ ‡, Duncan Paul Attard¹,² ‡, Adrian Francalanza¹ ‡, and Anna Ingólfsdóttir² ‡
Our approach

1. Gather meaningful metrics
2. Generate high loads
3. Be configurable
4. Cover general scenarios
Our approach

1. Response time/latency
   - Memory consumption
   - Scheduler usage

2. Generate high loads

3. Be configurable

4. Cover general scenarios
Our approach

1. Response time/latency
   - Memory consumption
   - Scheduler usage

2. Scalability using the right implementation language

3. Be configurable

4. Cover general scenarios
Our approach

1. Response time/latency
   - Memory consumption
   - Scheduler usage

2. Scalability using the right implementation language

3. Control model reactivity
   - Short convergence time
   - Reproduce initial conditions

4. Cover general scenarios
Our approach

1. Response time/latency
   - Memory consumption
   - Scheduler usage

2. Scalability using the right implementation language

3. Control model reactivity
   - Short convergence time
   - Reproduce initial conditions

4. Master-worker architecture
   - Load modelled on PDFs:
     - Steady

Duncan Paul Attard  ·  University of Glasgow
Our approach

1. Response time/latency
   - Memory consumption
   - Scheduler usage

2. Scalability using the right implementation language

3. Control model reactivity
   - Short convergence time
   - Reproduce initial conditions

4. Master-worker architecture
   - Load modelled on PDFs:
     - Steady
     - Pulse

Duncan Paul Attard · University of Glasgow
Our approach

1. Response time/latency
   • Memory consumption
   • Scheduler usage

2. Scalability using the right implementation language

3. Control model reactiveness
   • Short convergence time
   • Reproduce initial conditions

4. Master-worker architecture
   • Load modelled on PDFs:
     - Steady
     - Pulse
     - Burst

Duncan Paul Attard · University of Glasgow
Non-negotiable implementation constraints

”Observing software influences its runtime behaviour”

✓ Measurement precision
Non-negotiable implementation constraints

“Observing software influences its runtime behaviour”

- ✔ Measurement precision
- ✗ Variability
- ✗ Perturbations
- ✗ Runtime overhead
Non-negotiable implementation constraints

"Observing software influences its runtime behaviour"

- Measurement precision
- Variability
- Perturbations
- Runtime overhead

Ease of use \(\approx\) real scenarios

Duncan Paul Attard · University of Glasgow
Meeting the implementation constraints

- Measurement precision
- Runtime overhead
- Perturbations
- Variability

1. Global runtime snapshots
   • Stratified sampling
2. Sampling up to error $\epsilon$
3. Incremental computation
   • Asynchronous IO
   • Uniform load scheduling
4. Configurable seeds
   • Erlang
Meeting the implementation constraints

- Measurement precision
- Runtime overhead
- Perturbations
- Variability

1. Global runtime snapshots
   - Stratified sampling

2. Sampling up to error $\epsilon$

3. Incremental computation
   - Asynchronous IO
   - Uniform load scheduling

4. Configurable seeds
   - **Erlang**

Duncan Paul Attard · University of Glasgow
Meeting the implementation constraints

- Measurement precision
- Runtime overhead
- Perturbations
- Variability

1. Global runtime snapshots
   - Stratified sampling
2. Sampling up to error $\epsilon$
3. Incremental computation
   - Asynchronous IO
   - Uniform load scheduling
4. Configurable seeds
   - Erlang
Meeting the implementation constraints

1. Measurement precision
   - Global runtime snapshots
   - Stratified sampling

2. Runtime overhead
   - Sampling up to error $\epsilon$

3. Perturbations
   - Incremental computation
   - Asynchronous IO
   - Uniform load scheduling

4. Variability
   - Configurable seeds
   - Erlang

Duncan Paul Attard · University of Glasgow
The impact on RV benchmarking

Synthetic experiment set-up

- **Portable** and **controllable** experiments
- **Different** load models: Steady, Pulse, Burst
- Approximates **real** web-server traffic

Uncover real reactive system issues

- **Bottlenecks:** memory consumption +
- **Performance degradation:** \( \Rightarrow \) latency

Non-scalable RV tools:

- \( \Rightarrow \) processors \( \Rightarrow \) no \( \Rightarrow \) latency

Duncan Paul Attard · University of Glasgow
The impact on RV benchmarking

Synthetic experiment set-up
- **Portable** and **controllable** experiments
- **Different** load models: Steady, Pulse, Burst
- Approximates **real** web-server traffic

Uncover real reactive system issues
- Bottlenecks: ↑ memory consumption + ↑ scheduler usage
- Performance degradation: → load ⇒ → latency
- Non-scalable RV tools: → processors ⇒ no ↓ latency
Where do we stand?

Synthetic benchmarking
Where do we stand?

How can we increase adoption?

- Synthetic ≠ bogus

Synthetic benchmarking
Where do we stand?

How can we increase adoption?

- Synthetic ≠ bogus

Synthetic benchmarking

- Tool evaluation

- Is it acceptable for papers?
Where do we stand?

How can we increase adoption?

- Synthetic ≠ bogus

Synthetic benchmarking

Tool evaluation

Is it acceptable for papers?

- Reproducible?
  - Controllable?
  - Deployable?

Distribution

Simulated

- Easy to package (e.g., single VM image)
- Controllable (e.g., Python scripts)
- Reproducible (e.g., artifact evaluation)

Real

Duncan Paul Attard · University of Glasgow
Thanks