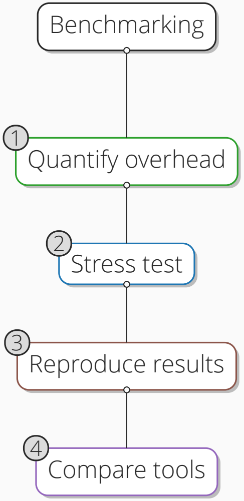
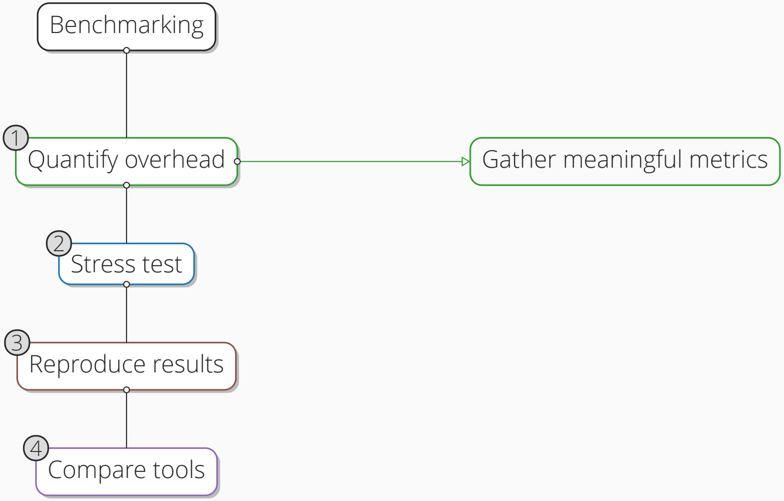
Revisiting Benchmarking for Concurrent Runtime Verification

Duncan Paul Attard · Monday, June 19th 2023
University of Glasgow

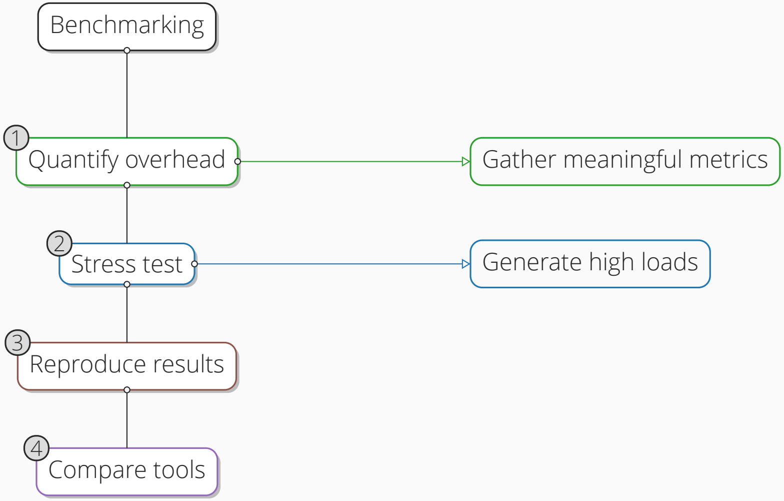
A litmus test for tools



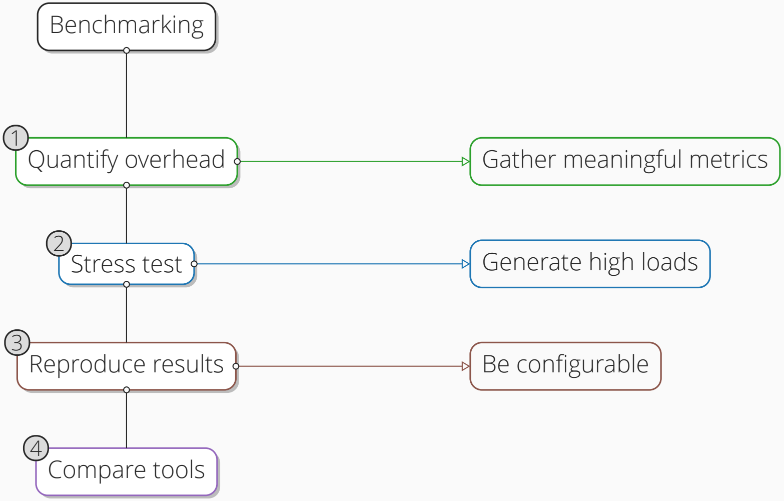
A litmus test for tools



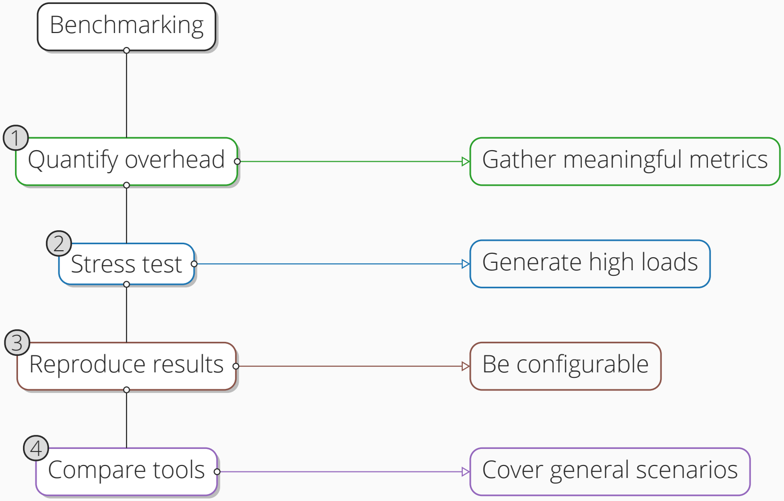
A litmus test for tools



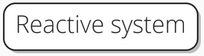
A litmus test for tools



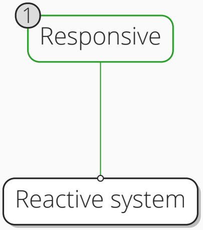
A litmus test for tools



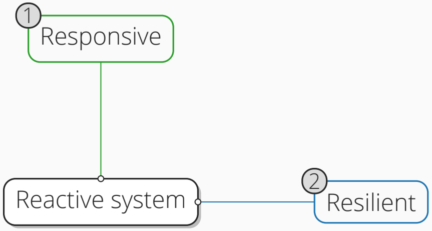
Reactive systems



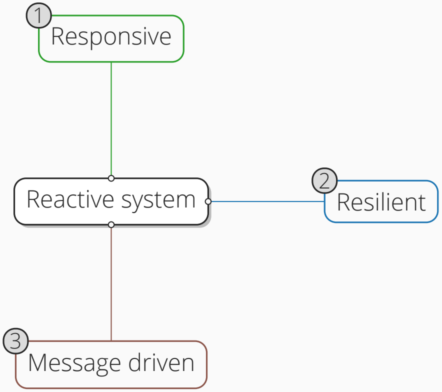
Reactive systems



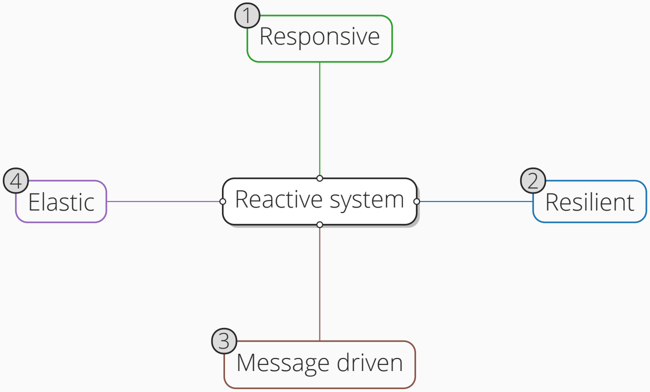
Reactive systems



Reactive systems



Reactive systems



Requirements for reactive systems

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

+ Reactive system aspects

Requirements for reactive systems

Gather meaningful metrics → Responsive

1

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

Generate high loads

Be configurable

Cover general scenarios

Requirements for reactive systems

Gather meaningful metrics —Responsive→ ①

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

Generate high loads —Resilient→ ②

- Scalable
- Uniform load distribution

Be configurable

Cover general scenarios

Requirements for reactive systems

Gather meaningful metrics → Responsive

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

Generate high loads → Resilient

2
- Scalable
- Uniform load distribution

Be configurable → Message-driven

3
- Control reaction to messages
- Reproduce load conditions

Cover general scenarios

Requirements for reactive systems

Gather meaningful metrics → Responsive → ①
- Response time/latency
- Memory consumption
- Scheduler/CPU usage

Generate high loads → Resilient → ②
- Scalable
- Uniform load distribution

Be configurable → Message-driven → ③
- Control reaction to messages
- Reproduce load conditions

Cover general scenarios → Elastic → ④
- Grow + shrink dynamically
- Use typical load profiles

The state of the art

“ *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

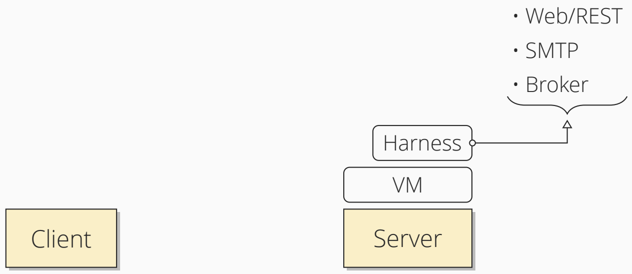
❝ *Current BM tools cater for limited to **no** concurrency* ❞

✗ Wrong tool	? Right tool	? Right tool
	✗ Wrong job	? Right job
		✗ Not enough

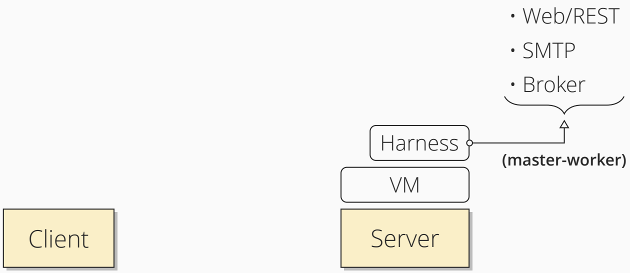
The typical recipe

Client
Server

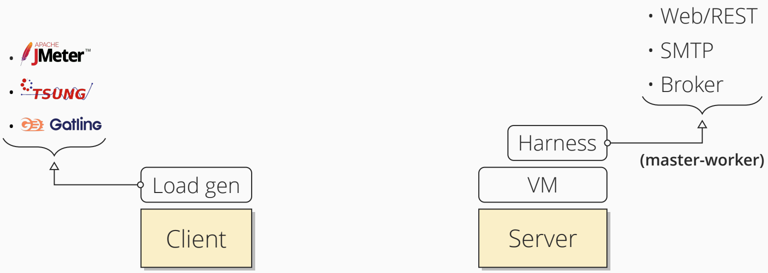
The typical recipe



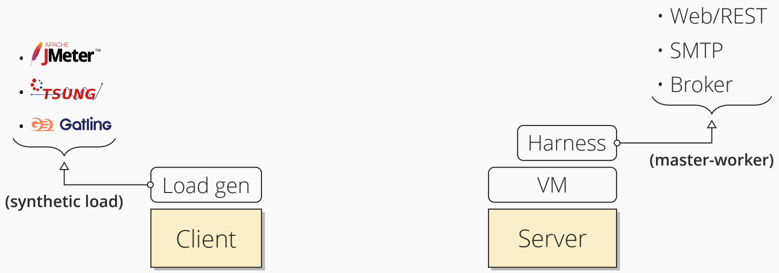
The typical recipe



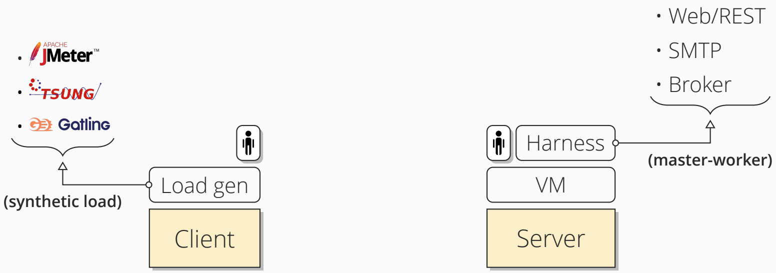
The typical recipe



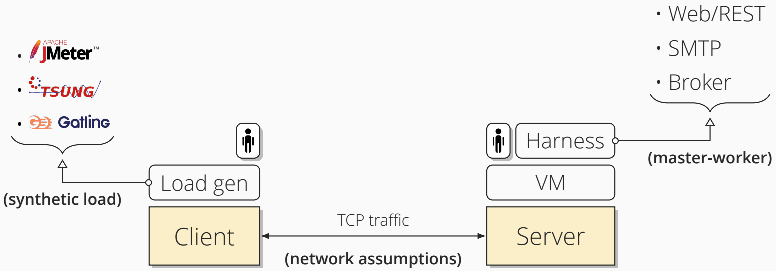
The typical recipe



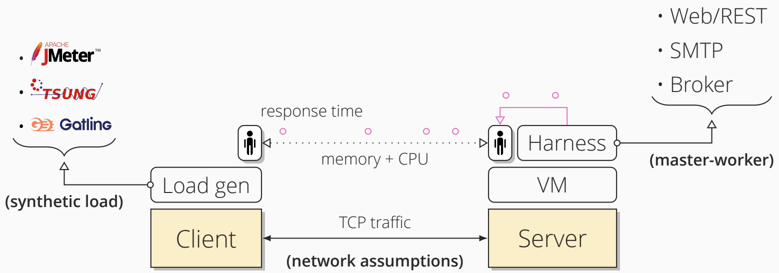
The typical recipe



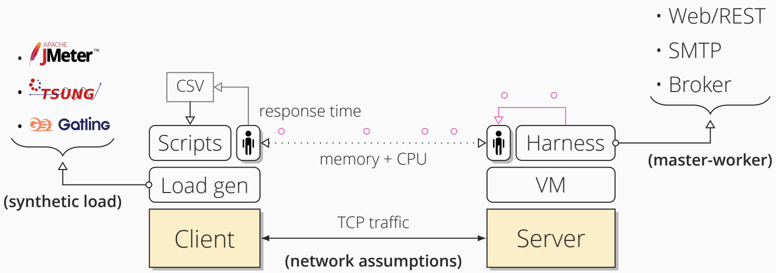
The typical recipe



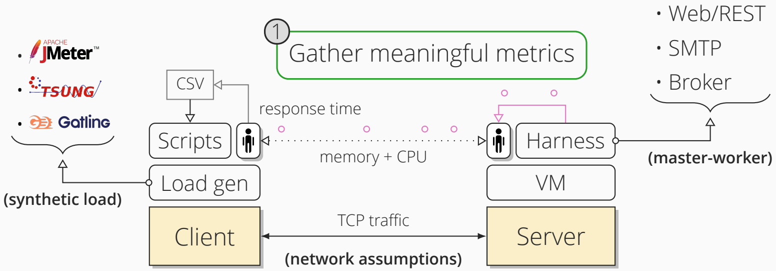
The typical recipe



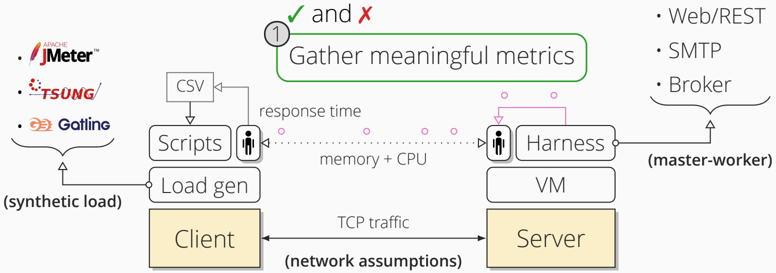
The typical recipe



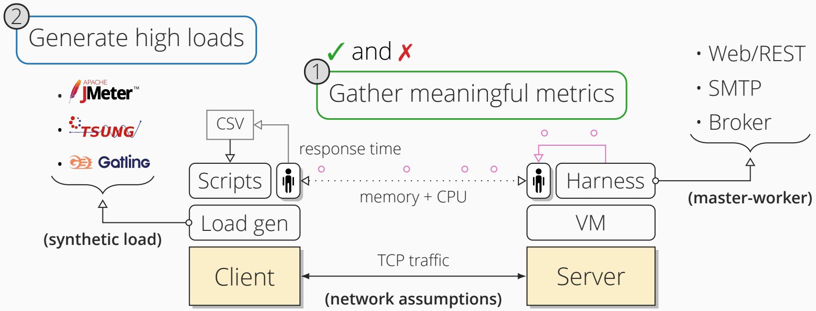
The typical recipe



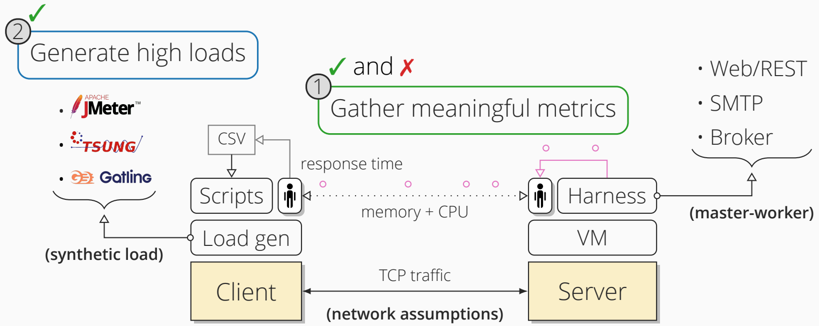
The typical recipe



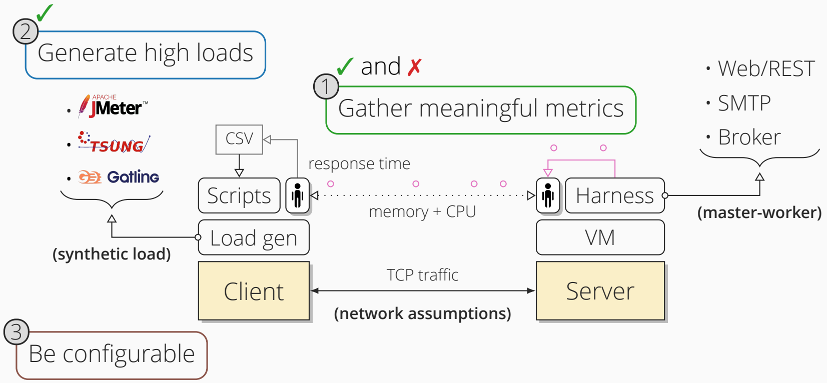
The typical recipe



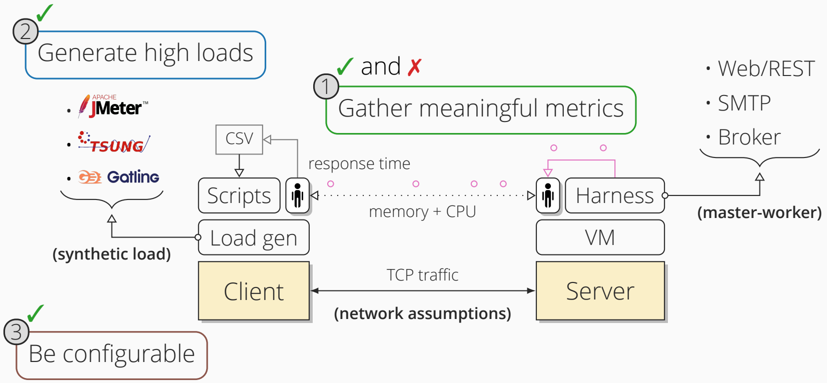
The typical recipe



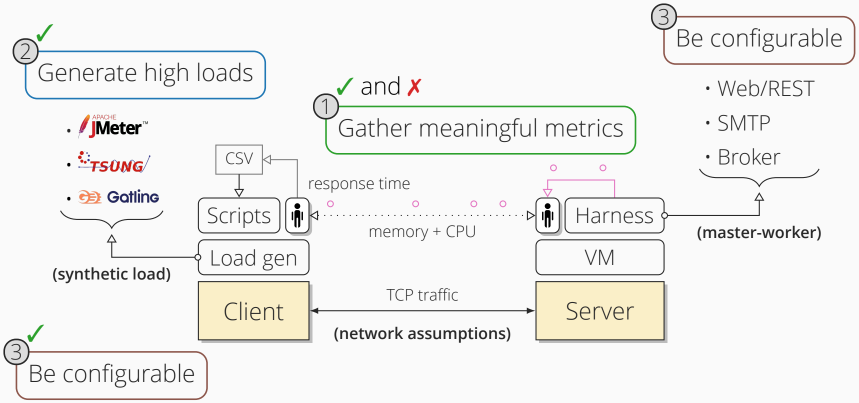
The typical recipe



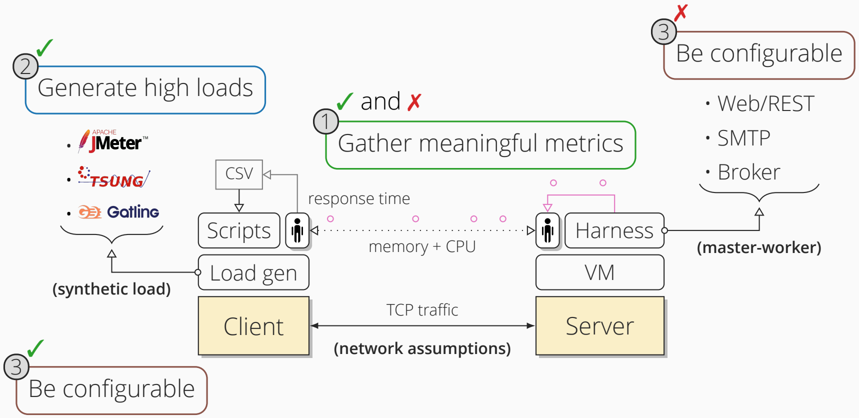
The typical recipe



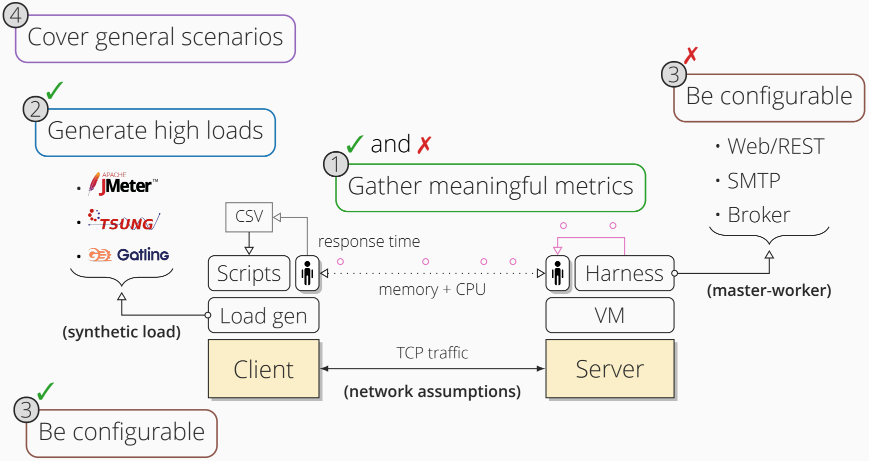
The typical recipe



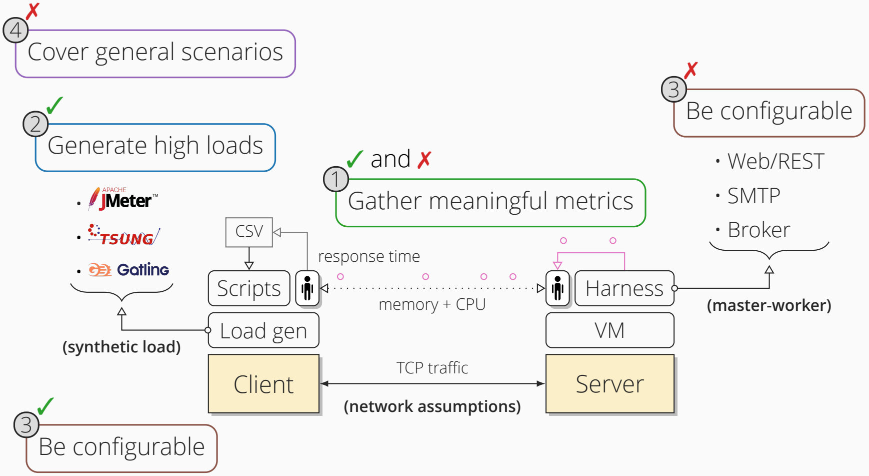
The typical recipe



The typical recipe



The typical recipe



Wouldn't it be nice if...

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

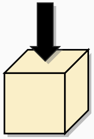
Wouldn't it be nice if...

① Gather meaningful metrics

② Generate high loads

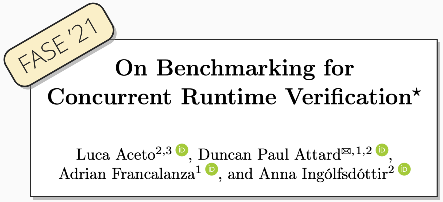
③ Be configurable

④ Cover general scenarios



- Ease of use
- ≈ real scenarios
- Meets requirements ① to ④

Wouldn't it be nice if...



On Benchmarking for Concurrent Runtime Verification*

Luca Aceto[2,3], Duncan Paul Attard✉,[1,2], Adrian Francalanza[1], and Anna Ingólfsdóttir[2]

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 reactiveness
- Short convergence time
- Reproduce initial conditions

4

Cover general scenarios

Our approach

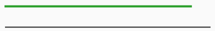
1
- Response time/latency
- Memory consumption
- Scheduler usage

2
- Scalability using the right implementation language

3
- Control model reactiveness
- Short convergence time
- Reproduce initial conditions

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

Steady



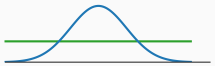
Our approach

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:



Our approach

1

- Response time/latency
- Memory consumption
- Scheduler usage

2

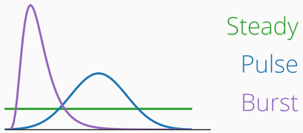
- Scalability using the right implementation language

3

- Control model reactiveness
- Short convergence time
- Reproduce initial conditions

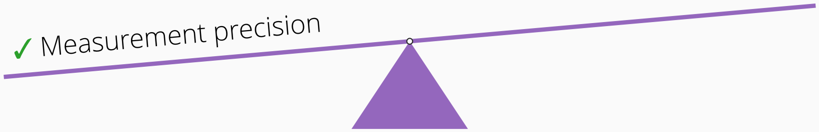
4

- Master-worker architecture
- Load modelled on PDFs:



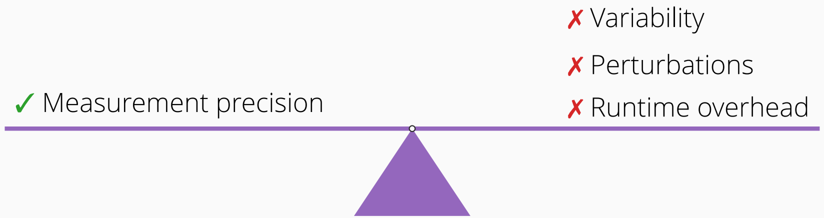
Non-negotiable implementation constraints

❝ *Observing software influences its runtime behaviour* ❞



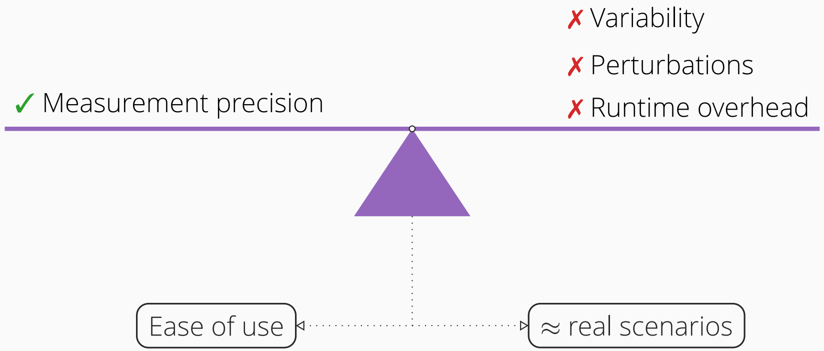
Non-negotiable implementation constraints

❝ *Observing software influences its runtime behaviour* ❞



Non-negotiable implementation constraints

❝ *Observing software influences its runtime behaviour* ❞



Meeting the implementation constraints

↗ Measurement precision

1
- Global runtime snapshots
- Stratified sampling

↘ Runtime overhead

2
- Sampling up to error ϵ

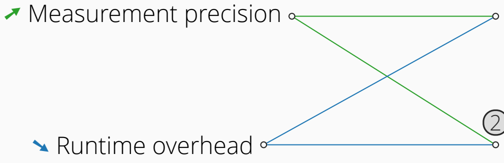
↘ Perturbations

3
- Incremental computation
- Asynchronous IO
- Uniform load scheduling

↘ Variability

4
- Configurable seeds
- Erlang

Meeting the implementation constraints



↗ Measurement precision

↘ Runtime overhead

↘ Perturbations

↘ Variability

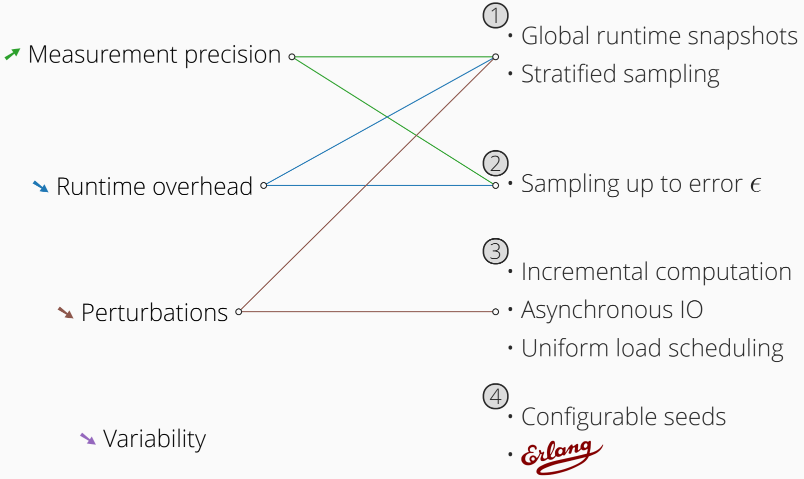
1
- Global runtime snapshots
- Stratified sampling

2
- Sampling up to error ϵ

3
- Incremental computation
- Asynchronous IO
- Uniform load scheduling

4
- Configurable seeds
- Erlang

Meeting the implementation constraints



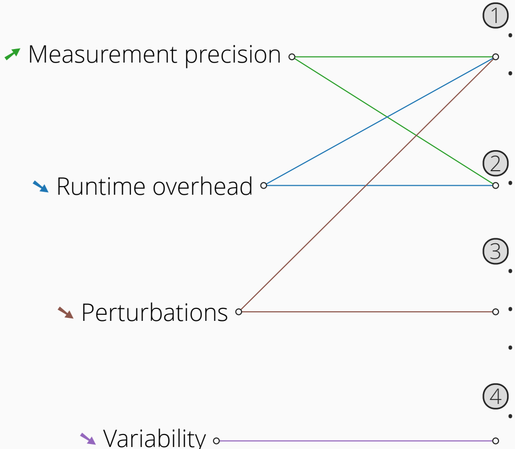
1
- Global runtime snapshots
- Stratified sampling

2
- Sampling up to error ϵ

3
- Incremental computation
- Asynchronous IO
- Uniform load scheduling

4
- Configurable seeds
- Erlang

Meeting the implementation constraints



1.
   - Global runtime snapshots
   - Stratified sampling
2.
   - Sampling up to error ϵ
3.
   - Incremental computation
   - Asynchronous IO
   - Uniform load scheduling
4.
   - Configurable seeds
   - Erlang

The impact on RV benchmarking

Synthetic experiment set-up

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

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

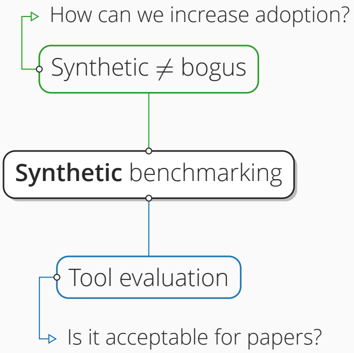
Where do we stand?

How can we increase adoption?

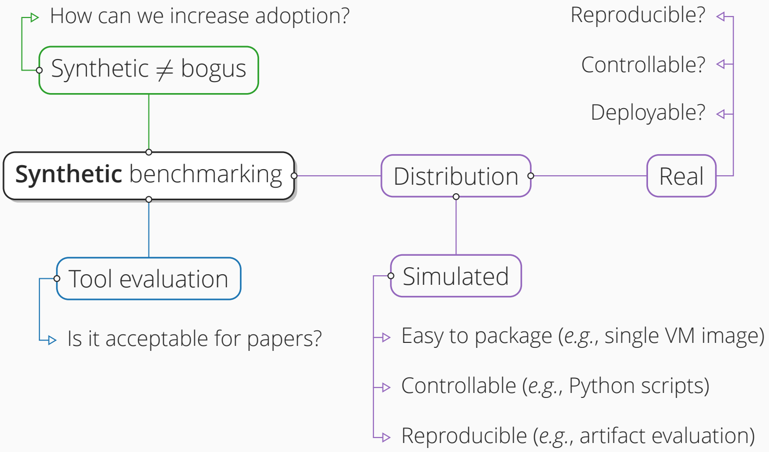
Synthetic ≠ bogus

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

Thanks