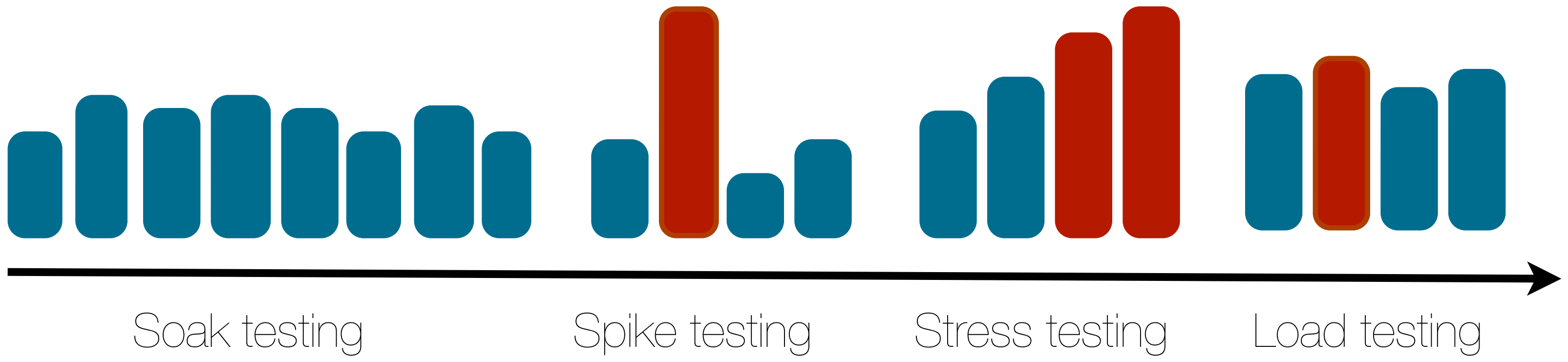
Tutorial

Scalable load testing using properties

Diana Corbacho
Erlang Solutions

Jun 10, 2015
EUC

What is ‘load testing’?



Responsiveness and stability under a particular workload

How are web services load tested?

- Generate a test description from the API description using a textual or graphical representation
- Execute the test description
- Analyse results
- Maybe start again

How do we load test web services?

- Generate a test description from the API description using a textual representation
- Use PBT to generate random user and load profiles based in the test description
- Let QuickCheck automate the execution and analyse each individual result
    - Yes! QuickCheck will run as many tests as required until it finds the boundaries of the service
- After hours, days or weeks of testing (maybe we are soak testing?) it will provide the final results

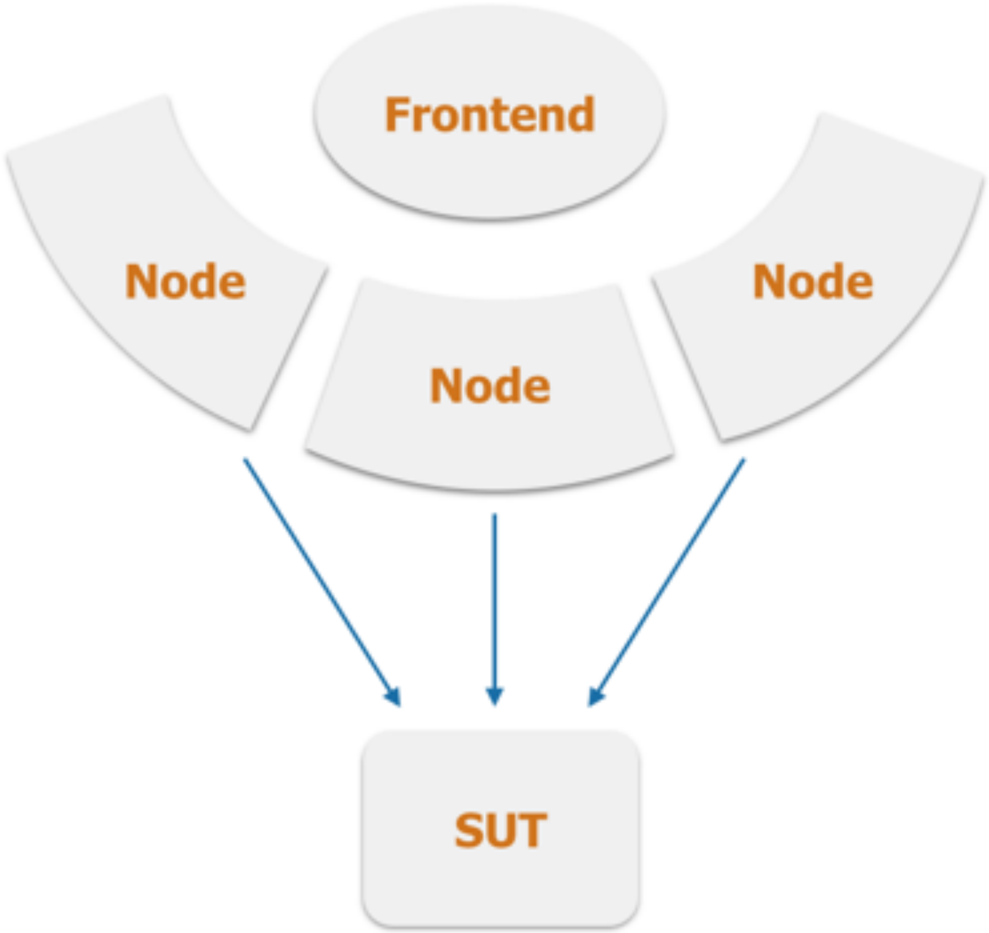
What that means?

- Less resources
- 24/7 hardware utilisation, no downtime
- More complete results
  - Better understanding of the service
  - Informed decisions

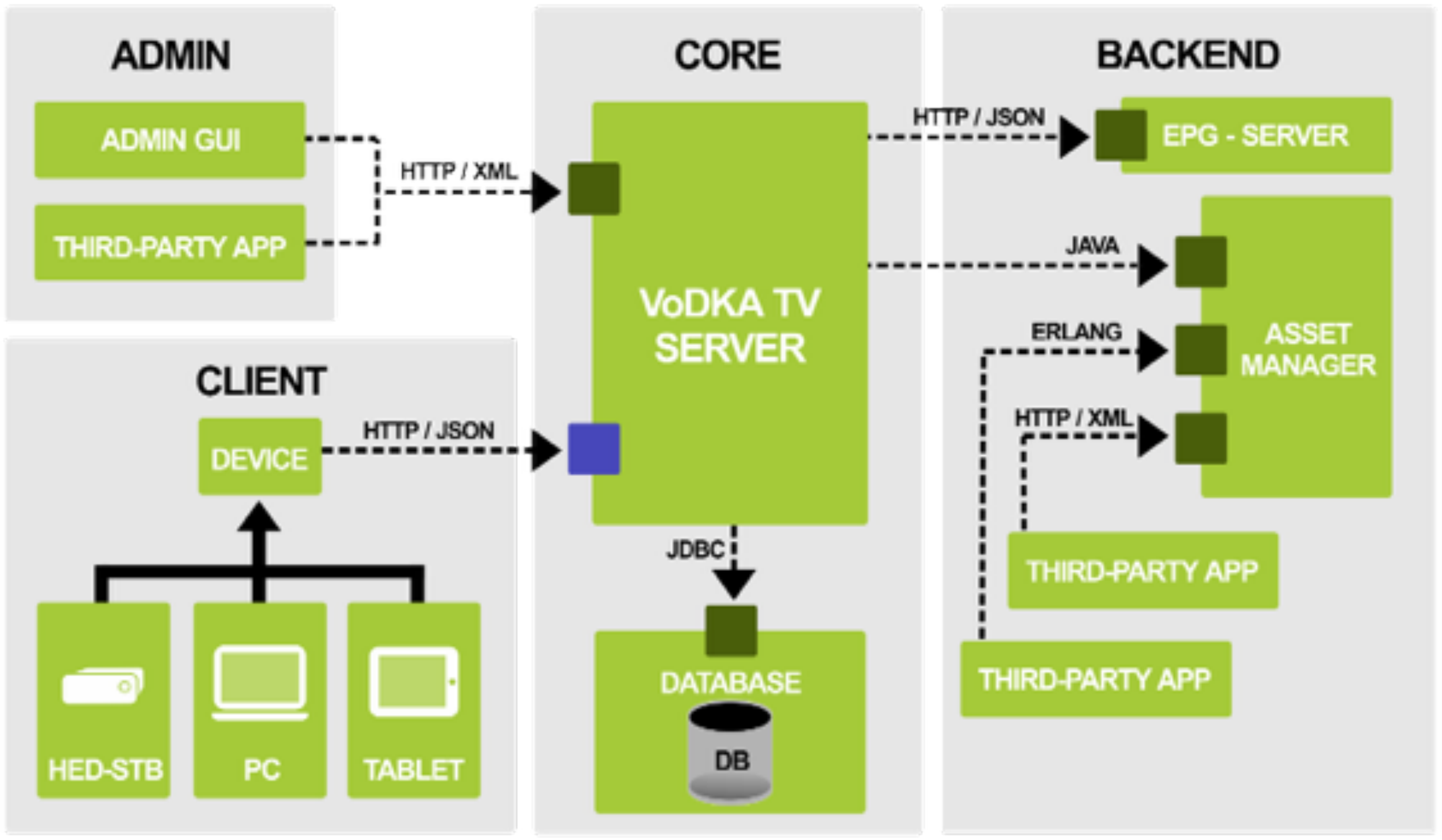
PBT & Megaload

- Property-based testing applied to load testing
- Scalable platform to run load tests: cloud & physical hardware
- Web interface
- Real-time statistics and graphs
- DSL to ease test description
- Multiprotocol

Megaload scalability



Industrial example: VoDKA TV

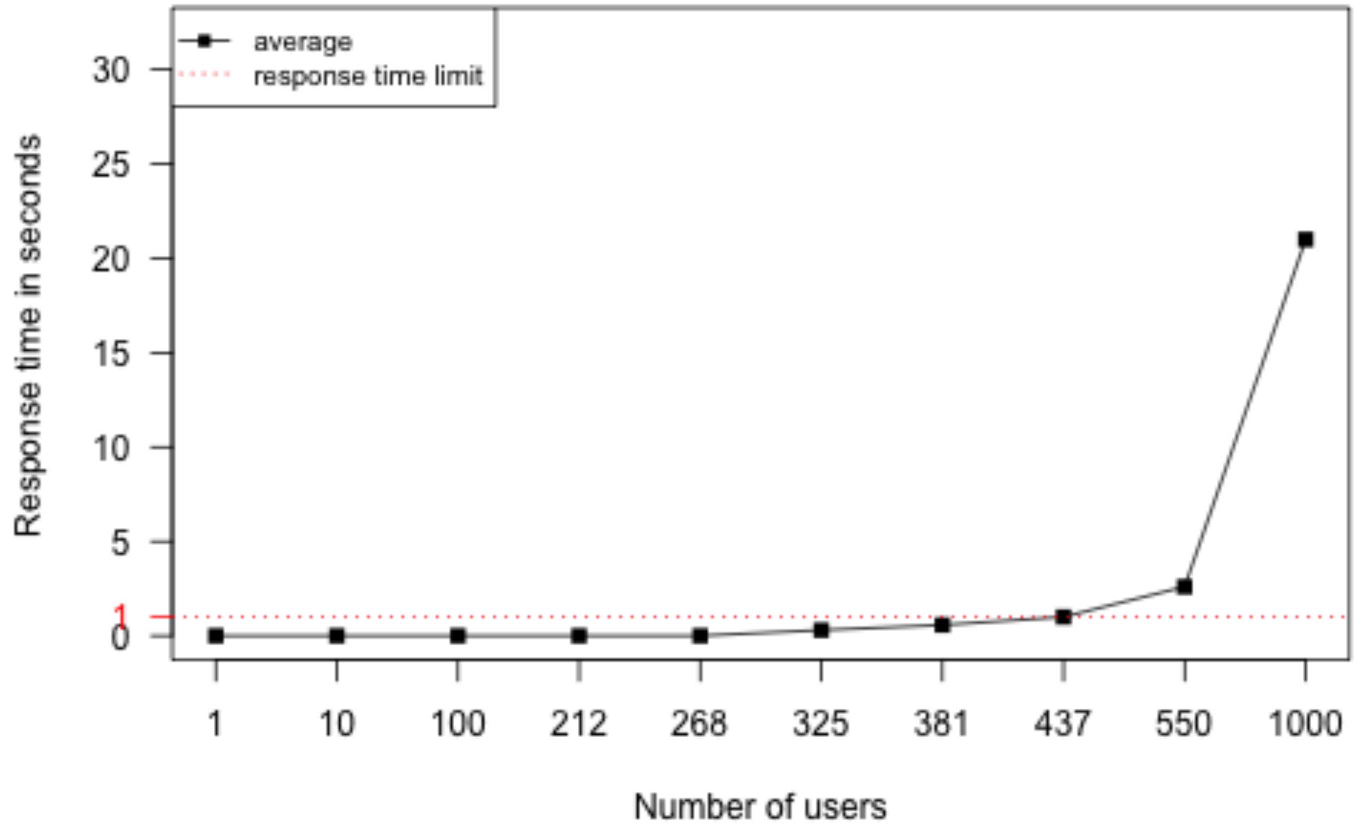


Properties (I)

- At what number of users the system breaks its service-level agreement?
  - SLA metrics
    - Average response time
    - Number of connection refused
    - Number of timeout requests

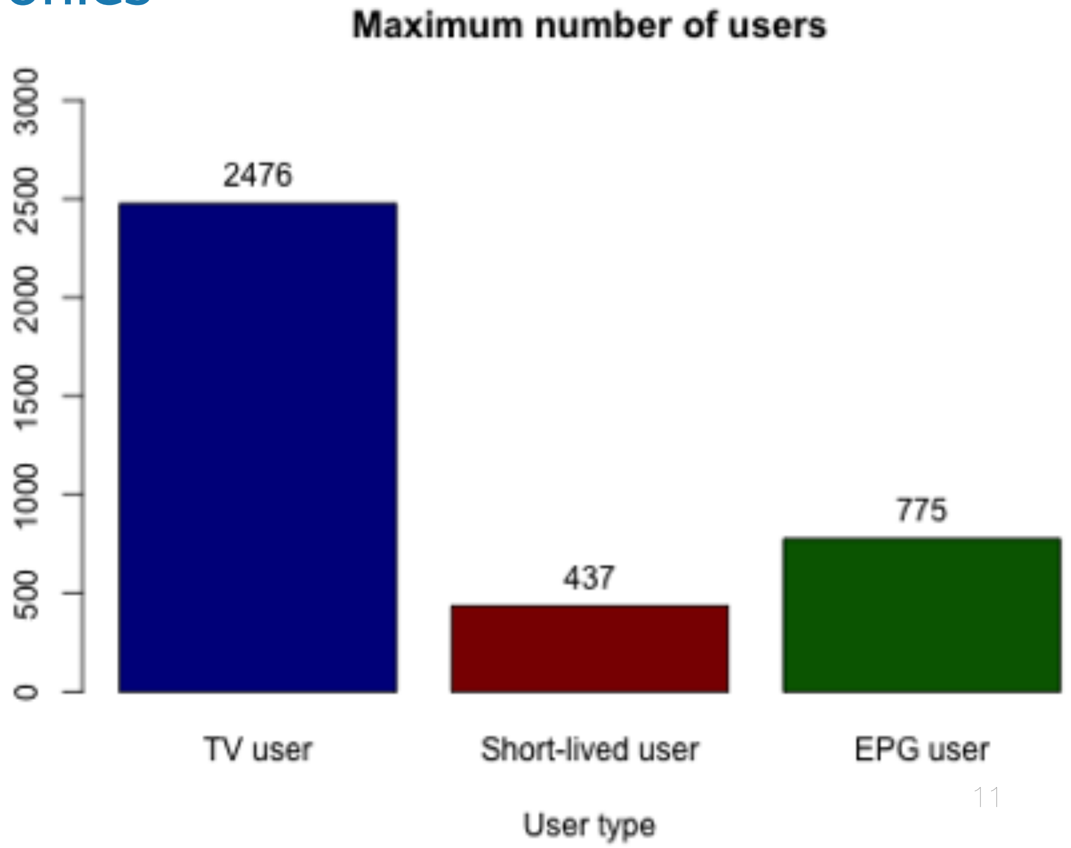
Properties (I)

- At what number of users the system breaks its service-level agreement?



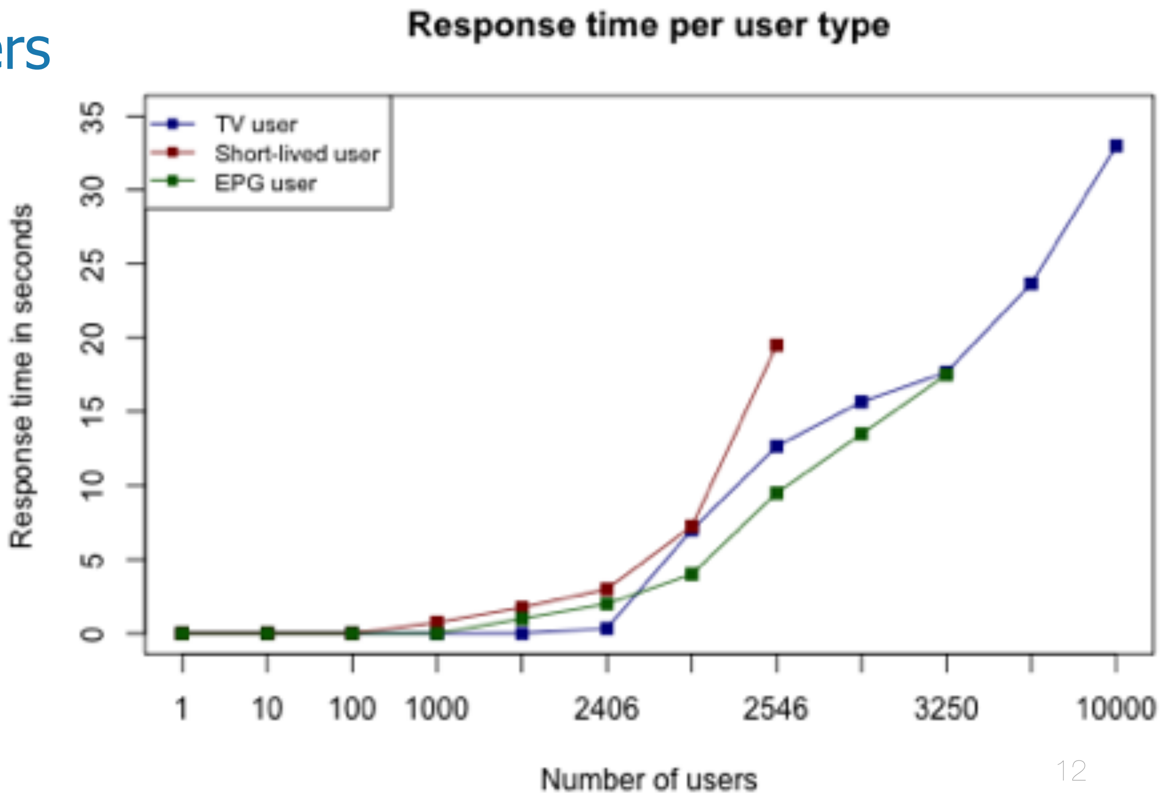
Properties (II)

- What number of users with each profile is the system able to support within the SLA boundaries?
  - Semi-static users profiles
    - Short-lived users
    - TV users
    - EPG users



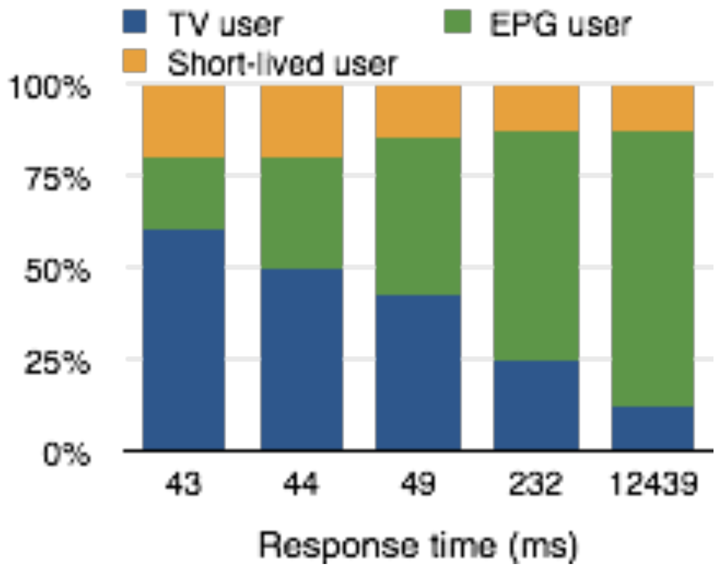
Properties (II)

- What number of users with each profile is the system able to support within the SLA boundaries?
  - Semi-static users profiles
    - Short-lived users
    - TV users
    - EPG users



Properties (III)

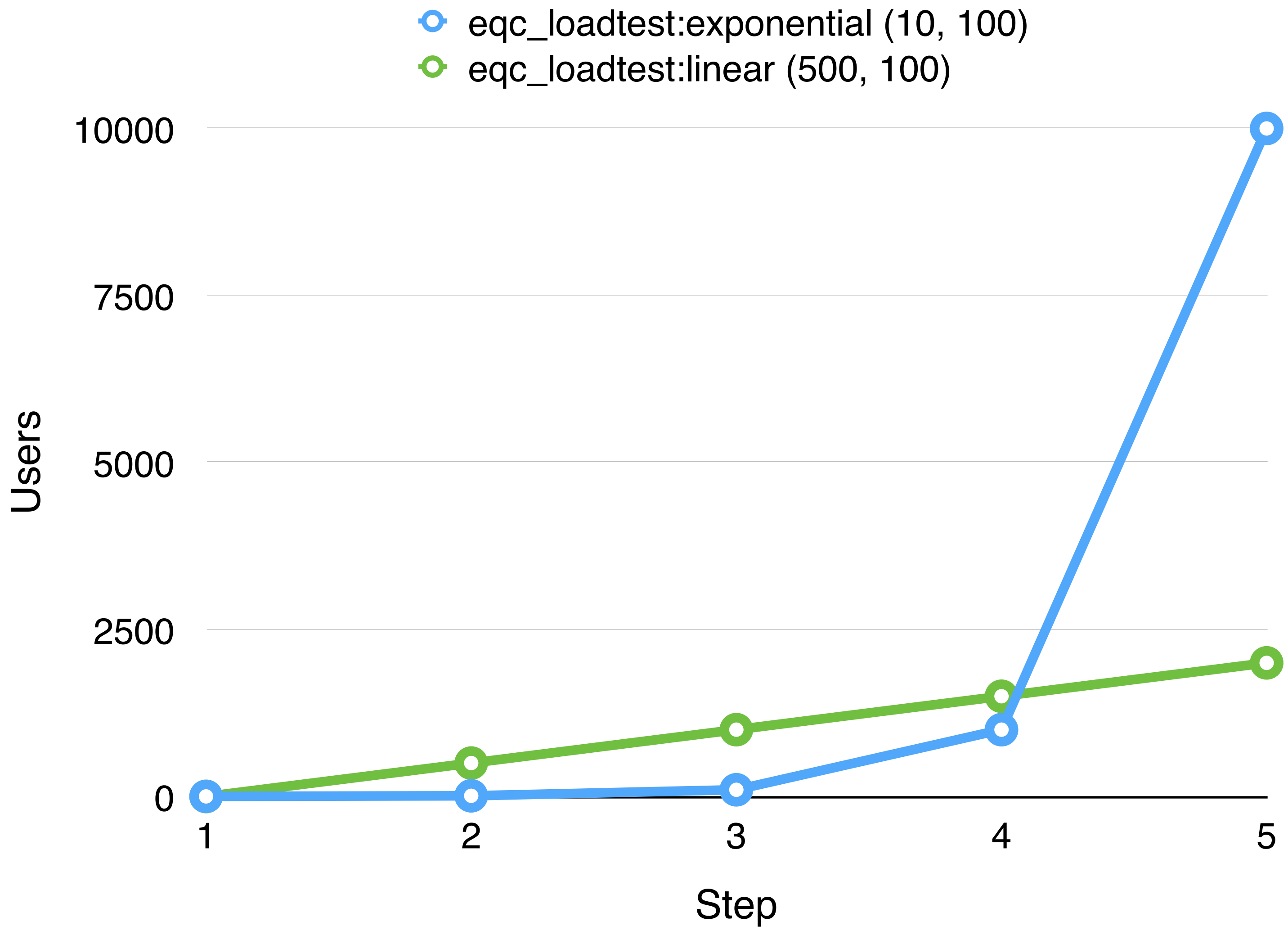
- What are the combinations of users the system is able to support within the SLA boundaries?
  - Fix some parameters - as number of users - within limits discovered in previous properties
  - Output example: 60% TV users - 40% EPG users



Properties (IV)

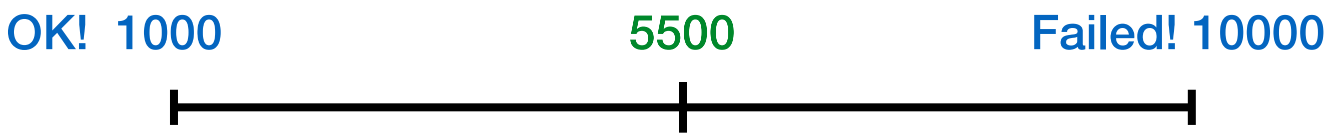
- What are the URLs that cause the most performance problems?
  - Measure response time and other metrics defined in the SLA for each individual request

Generators


eqc_loadtest:exponential (10, 100)
eqc_loadtest:linear (500, 100)
10000
7500
5000
2500
0
Users
1
2
3
4
5
Step

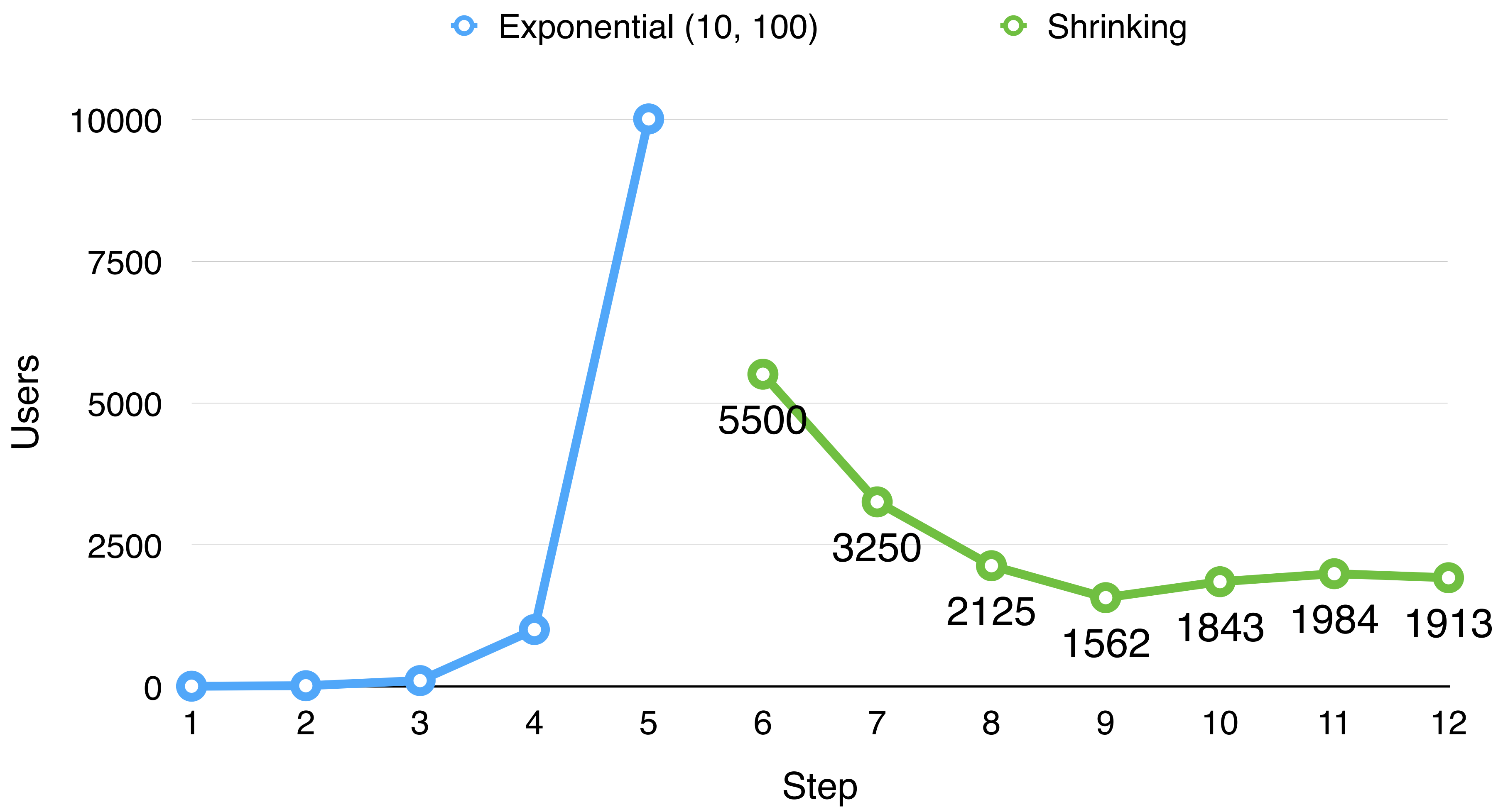
Shrinking strategy

Binary search between largest success and smaller failure



Faster convergence for load testing than normal QuickCheck heuristics

Shrinking

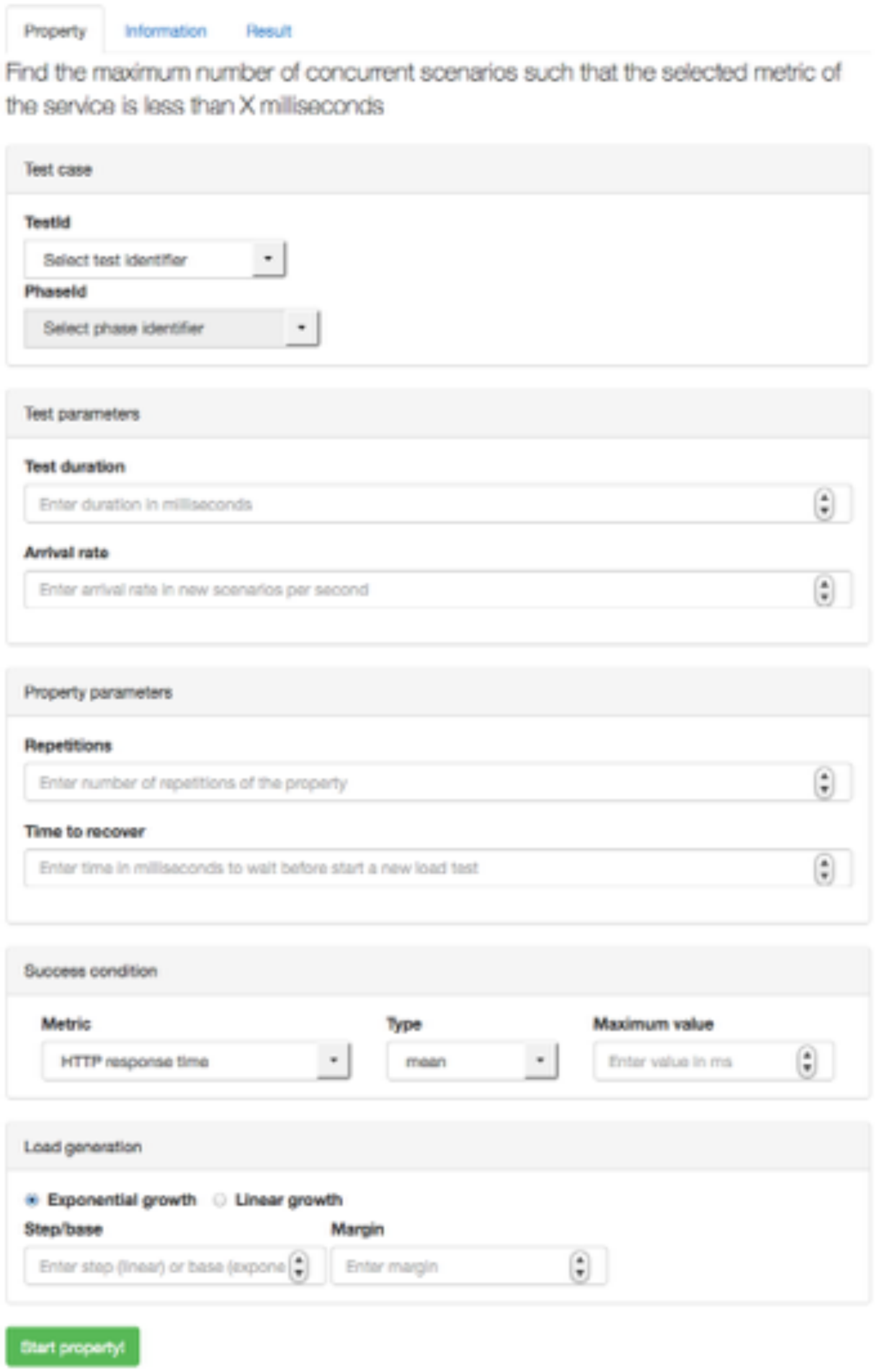


Fails from 1900 users - Shrinking to 1913

How many users the system is able to support within the SLA boundaries?

```
prop_users() ->
  eqc_loadtest:loadcheck(
  ?FORALL(
      Users, eqc_loadtest:exponential(10, 100),
       begin
           {ok, ok} =loader:update_phase(<<phase1>>,
                                                        [{<<"concurrent_scenarios">>, Users},
                                                         {<<"arrival_rate">>, 15},
                                                         {<<"duration">>, Users * 150 + Duration}]),
             loader:start_load(“onlinetv_eqc"),
             wait_until_terminated(),
             {AvgResponseTime, _ConnRefused, _Timeouts} = get_stats(),
             (AvgResponseTime < 200000)
     end)).
```

GUI integration



Find the maximum number of concurrent scenarios such that the selected metric of the service is less than X milliseconds

Test case

TestId: Select test identifier

PhaseId: Select phase identifier

Test parameters

Test duration: Enter duration in milliseconds

Arrival rate: Enter arrival rate in new scenarios per second

Property parameters

Repetitions: Enter number of repetitions of the property

Time to recover: Enter time in milliseconds to wait before start a new load test

Success condition

Metric	Type	Maximum value
HTTP response time	mean	Enter value in ms

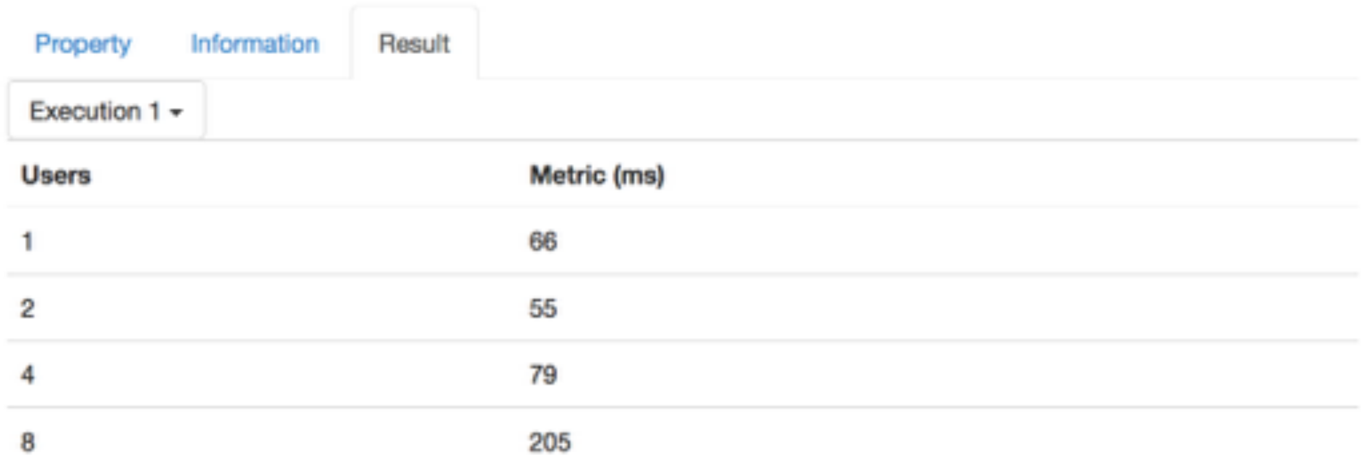
Load generation

[✓ Exponential growth] Linear growth

Step/base: Enter step (linear) or base (expone

Margin: Enter margin

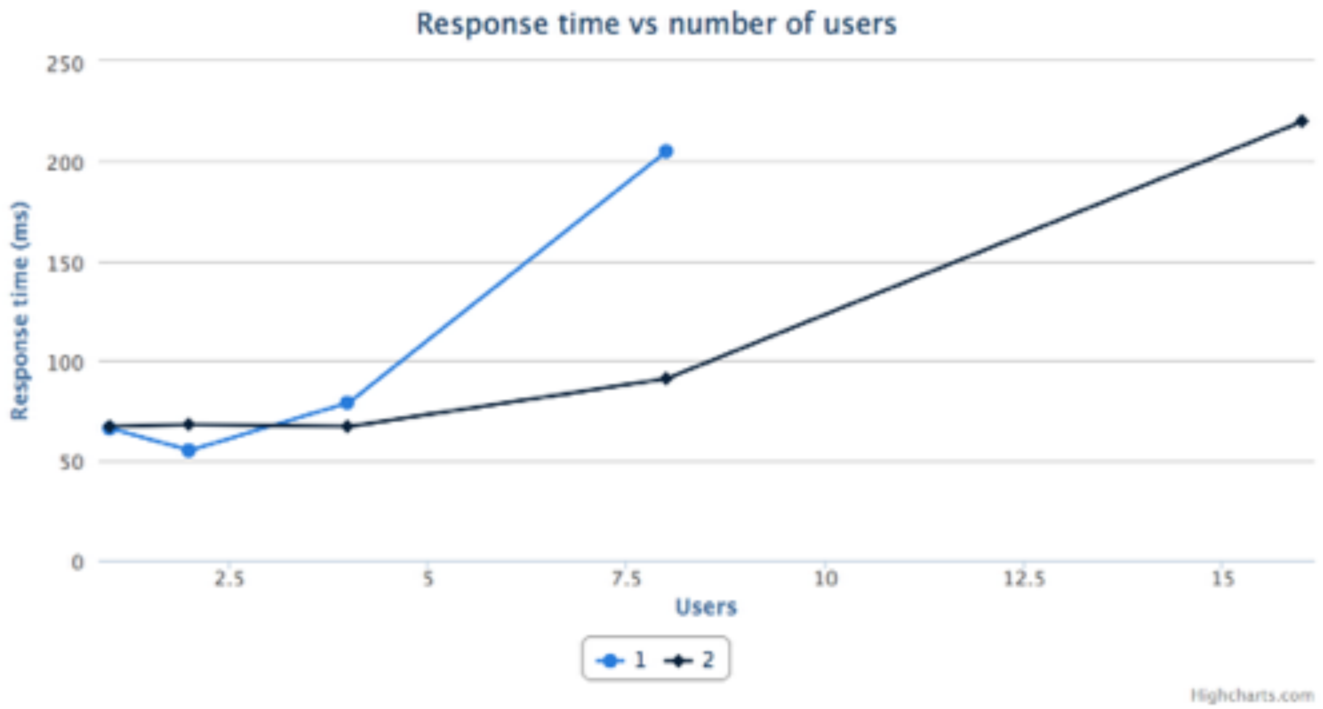
Results



Property | Information | Result

Execution 1 ▾

Users	Metric (ms)
1	66
2	55
4	79
8	205



What combinations of user behaviour the system is able to support within the SLA boundaries?

```
list_gen(Scenarios) ->
    ?SUCHTHAT(List, non_empty(list(elements(Scenarios))),
              lists:all(fun(S) -> lists:member(S, List) end, Scenarios)).

prop_scenario_combinations() ->
  ?FORALL(
      Component, list_gen([<<"watchtv">>, <<"tv">>, <<"epg">>]),
      begin
          {ok, ok} = loader:update_scenario(<<"general_user">>,
                                            [{<<"components">>, Component}]),
          loader:start_load("onlinetv_eqc"),
          wait_until_terminated(),
          {AvgResponseTime, ConnRefused, Timeouts} = get_stats(),
          Prop = (AvgResponseTime < 185000) and (ConnRefused == 0)
                     and (Timeouts == 0),
          update_stats(Component, AvgResponseTime, Prop),
          Prop
     end).
```

```
.
Failed! After 2 tests.
[<<"epg">>,<<"watchtv">>,<<"tv">>,<<"epg">>,<<"epg">>]
Shrinking
.
.
.
(1 times)
[<<"epg">>,<<"watchtv">>,<<"tv">>,<<"epg">>]
|              Components                              |  Runs   | ResTime  | Failed |
|------------------------------------------------------------------------------------|
| <<"watchtv">> 20% <<"tv">> 20% <<"epg">> 60% |    1     |  210ms   |   *    |
| <<"watchtv">> 25% <<"tv">> 25% <<"epg">> 50% |    1     |  200ms   |   *    |
| <<"watchtv">> 25% <<"tv">> 50% <<"epg">> 25% |    3     |  181ms   |        |
| <<"watchtv">> 33% <<"tv">> 33% <<"epg">> 33% |    2     |  168ms   |        |
| <<"watchtv">> 50% <<"tv">> 25% <<"epg">> 25% |    2     |  145ms   |        |
```

Demo

Let’s run some tests!

Questions?

Feel free to contact us
for any queries at
megaload@erlang-solutions.com

http://prowessproject.eu/megaload