

The Actor Model as a Load Testing Framework

Nelson Vides
Senior Erlang Consultant and Core MongooseIM developer

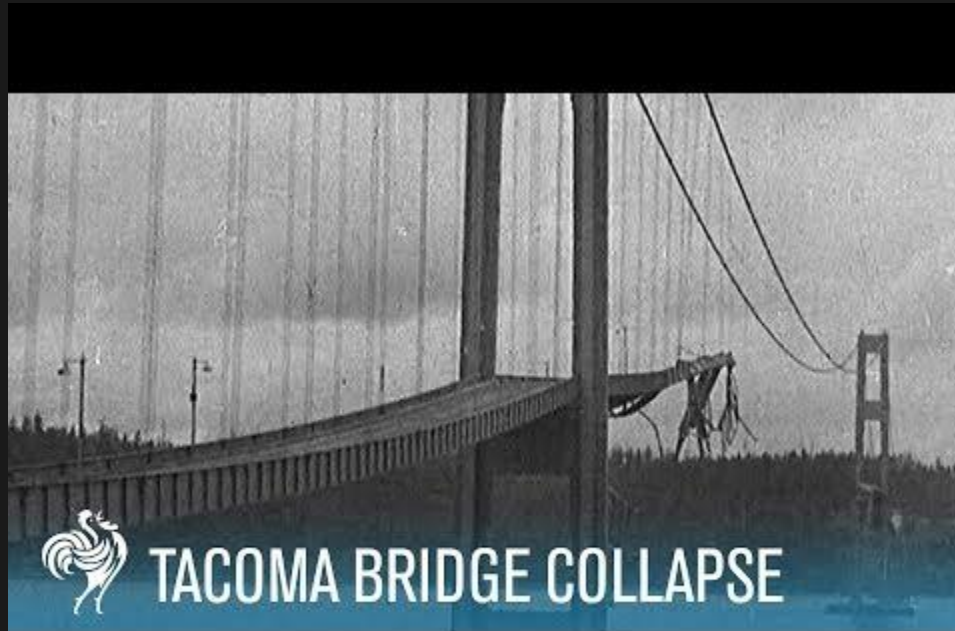
nelson.vides@erlang-solutions.com





An analogy

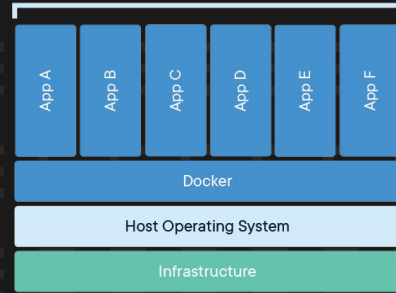
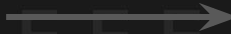
The Tacoma Bridge



The Tacoma Bridge



The (virtual) Tacoma Bridges



The (virtual) Tacoma Bridges

- Interactions?
- Traffic capacity?
- Amplifying factor?
- Old forces that only now start to matter?



Terminology



Framework

- A framework is a generic term commonly referring to an essential supporting structure which other things are built on top of. [\[Wikipedia\]](#)
- A supporting structure around which something can be built [\[Cambridge dictionary\]](#)
- A system of rules, ideas, or beliefs that is used to plan or decide something [\[Cambridge dictionary\]](#)



Framework



A system of rules, ideas, or beliefs that is used to plan or decide something

[\[Cambridge dictionary\]](#)

Model



A system of postulates, data, and inferences presented as a mathematical description of an entity

[\[Merriam-Webster\]](#)

Testing

The process of using or trying something to see if it works, is suitable, obeys the rules, etc.

[\[Cambridge dictionary\]](#)

Load

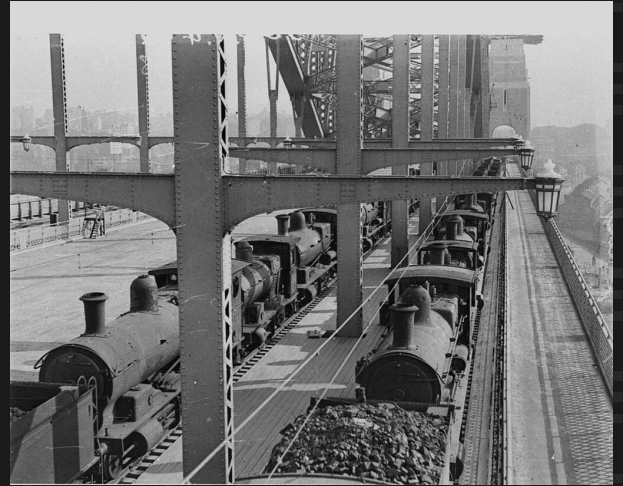
A mass or quantity of something taken up and carried, conveyed, or transported [\[Thesaurus\]](#)



Load Testing

The process of trying something can carry a mass or quantity of work, and verify how it behaves under varying such quantities

- Performance — at minimum load
- Load — at very high load
- Stress — at failure



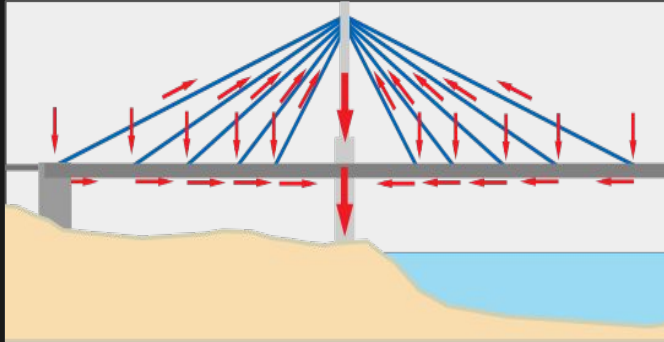
Load Testing Framework

A system of tools and ideas to apply loads in all the possible ways the system *allows*:

- Unit of measurement
- Interactions



The loads



The forces



The users

Actors

An Actor can, in response to a message it receives:

- send a finite number of messages to other Actors
- create a finite number of new Actors
- designate the behavior to be used for the next message it receives



The universal primitive

Everything is a...



An aerial photograph of a forest, showing a dense network of tree roots and branches. The lighting is dramatic, with bright highlights on the roots and deep shadows in the canopy. A crow is perched on a branch in the upper right quadrant.

**Light thickens, and the crow/
Makes wing to the rooky wood.**

Shakespeare, Macbeth, Act 3, Scene 2

A Murder of Crows

- Interactions?
- Traffic capacity?
- Amplifying factor?
- Old new forces?



Run all the actors

- *init* a scenario, once
 - Metrics, conditions, databases
- *Start* all the actors, every single time
 - Each actor executes its force
- Run the scenario
 - Locally or distributed



Throttle

Set a Rate per Interval for an action

- Progressive rate control
- Actors wait for throttler's approval
- Actors ask other actors to wait for throttler's approval



Coordinate

- Pick a number of users for an action
 - all
 - a list
 - a set of distinct pairs
- Users are given to a callback as they join the coordination plan



An aerial photograph of a wooden bridge crossing a river with rapids. The bridge is made of many parallel wooden planks and runs diagonally from the top-left towards the bottom-right. The river water is white and turbulent as it flows over rocks. The surrounding area is a dense forest of dark green trees.

EXAMPLE

TIME

Match'em!



Match'em!

FAINT HAT: 11



CURVED APPLE PIE: 11



ROUND STUFFED ANIMAL: 10



QUIET TOWEL: 11



COOLING HAT: 9



MOANING STUFFED ANIMAL: 11



```
%% amoc_scenario
-spec init() -> ok.
init() ->
    CoordinatorPlan = coordinator_plan(),
    amoc_coordinator:start(join_users_in_room, CoordinatorPlan),
    amoc_throttle:start(connect_user, 60),
    amoc_throttle:change_rate_gradually(connect_user, 6000, 60000, ?MINUTES(1), ?SECONDS(5), 90),
    amoc_throttle:start(create_and_start_game, 60),
    amoc_throttle:change_rate_gradually(create_and_start_game, 600, 60000, ?MINUTES(1), ?SECONDS(5), 90),
    start_metrics(),
    ok.
```

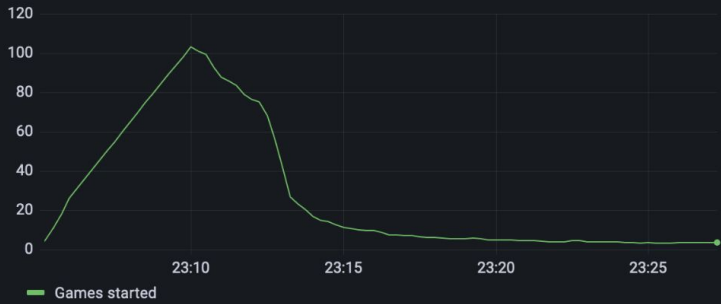


```
coordinator_plan() ->
  DeckSize = amoc_config:get(set_deck_size),
  Rounds = amoc_config:get(rounds_per_game),
  UsersPerGame = amoc_config:get(users_per_game),
  [
    {UsersPerGame, fun(_Event, Users) ->
      [FirstPid | OtherPids] = [ Pid || {Pid, _} <- Users],
      Msg = {create_and_start_game, <<"freeForAll">>, OtherPids, DeckSize, Rounds},
      amoc_throttle:send(create_and_start_game, FirstPid, Msg)
    }
  ].
```

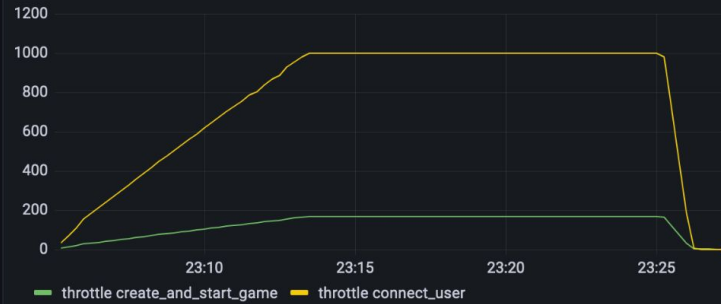
```
-spec start(amoc_scenario:user_id()) -> any().
start(MyId) ->
    amoc_throttle:send_and_wait(connect_user, waiting_until_allowed_to_connect),
    Host = amoc_config:get(host),
    UserName = integer_to_binary(MyId),
    {Delay, Jitter} = get_difficulty(),
    User = #user{host = Host, user_name = UserName,
                delay = Delay, jitter = Jitter},
    amoc_coordinator:add(join_users_in_room, UserName),
    gen_statem:enter_loop(mm_gamer, [], initial_state, User, [{next_event, internal, init_ws}]).
```

Games played

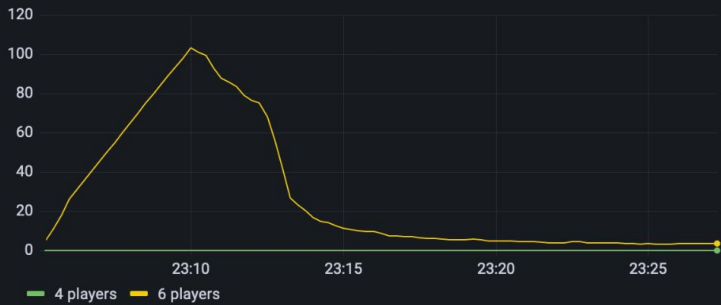
mm games started per minute



amoc games launched per minute



Games by number of players



amoc coordinator

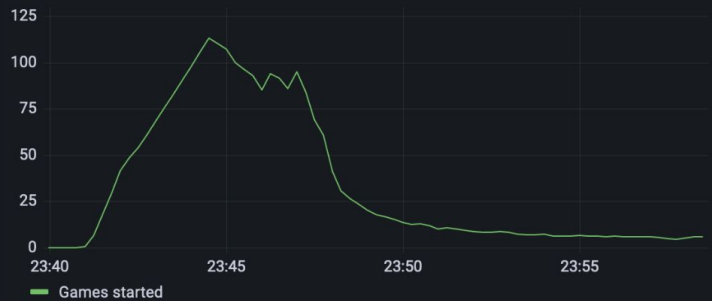


```
coordinator_plan() ->
  DeckSize = amoc_config:get(set_deck_size),
  Rounds = amoc_config:get(rounds_per_game),
  UsersPerGame = amoc_config:get(users_per_game),
  [
    {UsersPerGame + 1, fun(_Event, Users) ->
      UserPids = [ Pid || {Pid, _} <- Users],
      {Group1, Group2} = lists:split(rand:uniform(length(Users)), UserPids),
      create_and_start_game(Group1, DeckSize, Rounds),
      create_and_start_game(Group2, DeckSize, Rounds)
    end}
  ].
```

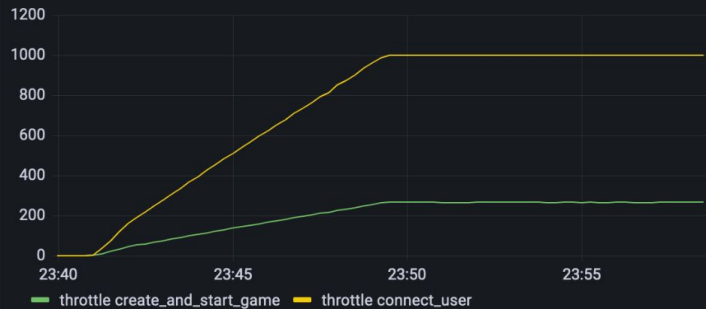
```
create_and_start_game([FirstPid | OtherPids], DeckSize, Rounds) ->
  Msg = {create_and_start_game, <<"freeForAll">>, OtherPids, DeckSize, Rounds},
  amoc_throttle:send(create_and_start_game, FirstPid, Msg).
```

Games played

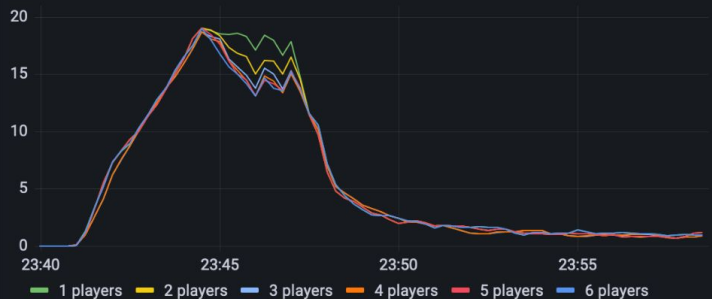
mm games started per minute



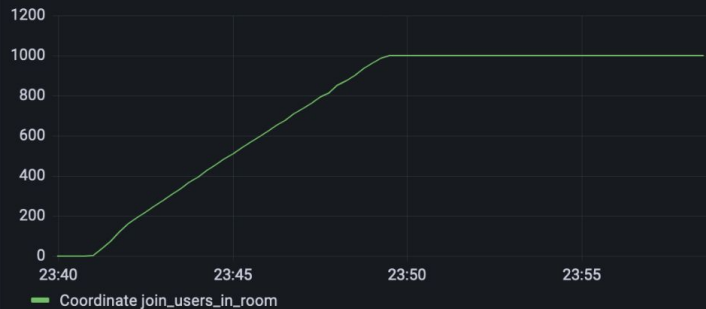
amoc games launched per minute




Games by number of players




amoc coordinator



☰ README.md 

A Murder of Crows



A Murder of Crows, aka amoc, is a simple framework for running massively parallel tests in a distributed environment.

It can be used as a rebar3 dependency:


```
{deps, [
  {amoc, {git, "https://github.com/esl/amoc", {tag, "2.0.1"}}}
]}.
```

[MongooseIM](#) is continuously being load tested with Amoc. All the XMPP scenarios can be found [here](#).

In order to implement and run locally your scenarios, follow the chapters about [developing](#) and [running](#) a scenario locally. Before [setting up the distributed environment](#), please read through the configuration overview. If you wish to run load tests via http api, take a look at the [REST API](#) chapter.

<https://github.com/esl/amoc>

<https://github.com/esl/amoc/pull/145>

README.md 

amoc_arsenal_xmpp

This is a collection of XMPP-related scenarios for [amoc](#), that [Erlang Solutions](#) uses to load test [MongooseIM](#) - our robust, scalable and efficient XMPP server.

<https://github.com/esl/amoc-arsenal-xmpp>



[@NelsonVides](#)



[esl/MongooseIM](#)



[esl/amoc](#)

Nelson Vides
Senior Erlang Consultant and Core MongooseIM developer

nelson.vides@erlang-solutions.com





Contact us

London | Stockholm | Krakow | Budapest | US Remote



www.erlang-solutions.com

general@erlang-solutions.com