Boosting Simulation Performance with Python

Eran Friedman
How to use Discrete-Event Simulation to run your system faster than real-time?
About Me

- Eran Friedman
- Team lead @ Fabric
- Nowadays developing the Ground Robot
Outline

- Simulations - why?
- How to use DES?
- How to use SimPy?
- What are the challenges?
- How to distribute?
Simulation

“An approximate imitation of the operation of a process or system ...”

- Wikipedia
Simulation

Backend

Orders  Stock  Motion

...
Simulation

Backend

Orders  Stock  Motion  ...

..
Importance of Simulations

COVID-19
Importance of Simulations
Automated regression tests

Regression:
“when you fix one bug, you introduce several newer bugs.”
Importance of Simulations
Analyze performance & compare algorithms
Importance of Simulations
Run in the cloud
Importance of Simulations

Verify warehouse layout
Importance of Simulations

Inject failures & improve robustness
Importance of Simulations
Simulate a large facility
Discrete-Event Simulation (DES)

- Operations are modeled as sequence of events
- Simulation jumps to the next event
- Simulation maintains its own clock
- Example: 2 m/s, 10 time-ticks/second

\[
\begin{align*}
  t=0 & \quad x=0\text{cm} \\
  t=0.1 & \quad x=20\text{cm} \\
  t=0.2 & \quad x=40\text{cm}
\end{align*}
\]
**Discrete-Event Simulation (DES)**

- Operations are modeled as sequence of events
- Simulation jumps to the next event
- Simulation maintains its own clock
- Example: 2 m/s, 10 time-ticks/second

![Diagram with time and distance intervals]

- $t=0$: $x=0\text{cm}$
- $t=0.1$: $x=20\text{cm}$
- $t=0.2$: $x=40\text{cm}$
Discrete-Event Simulation (DES)

- Operations are modeled as sequence of events
- Simulation jumps to the next event
- Simulation maintains its own clock
- Example: 2 m/s, 10 time-ticks/second

$t=0$
$x=0\text{cm}$

$t=0.1$
$x=20\text{cm}$

$t=0.2$
$x=40\text{cm}$
SimPy Library

- Discrete-event simulation (DES) framework
- Created in 2002
- MIT license
- Pure Python
- No dependencies
SimPy Overview

Environment

Event queue

Processes:

r0  r1

$t = 0$
SimPy Overview

Environment

Event queue

Processes:

r0

r1

r0

t=0

r1

t=0

 setTimeout

 t = 0
SimPy Overview

Environment

Executing:
- r0
  - t=0

Event queue

Processes:
- r0
- r1
SimPy Overview

Environment

Executing -

Event queue

Processes:

- r0
  - t = 0
- rl
  - t = 0
SimPy Overview

Environment

Event queue

Executing -

Processes:

r0

r1

t = 0

t = 0.1
SimPy Overview

Environment

Executing -

<table>
<thead>
<tr>
<th>r1</th>
<th>t=0</th>
</tr>
</thead>
</table>

| r0 | t=0.1 |

Event queue

Processes:

| r0 | rl |
SimPy Overview

Environment

Executing -

Event queue

Processes:

r0

r1

t = 0

t = 0.1
SimPy Overview

Environment

Executing -

Event queue

Processes:

- r0
- rl

- rl
  t=0.1

- r0
  t=0.1

t = 0
SimPy Overview

Environment

Executing -

Event queue

Processes:

r0

r1

r0
t=0.1

r1
t=0.1

Clock

t = 0.1
SimPy Example - Robot Race

- A robot’s speed is about 2-4 meters/second
from random import randint
import simpy

num_robots = 3
sim_time = 30  # seconds
time_tick = 0.5

class Robot:
    def move(self, env, robot_id):
        pos = 0
        while True:
            pos += randint(1, 2)
            print(f"{env.now} r_{robot_id} moved to {pos}")
            yield env.timeout(time_tick)

env = simpy.Environment()

for i in range(num_robots):
    r = Robot()
    env.process(r.move(env, robot_id=i))

env.run(until=sim_time)
SimPy Example - Robot Race

- All SimPy processes run in a single thread
- Parameters that affect performance:
  - Number of simulated components
  - Time tick granularity
- Can run in ‘real-time’ mode
Benefits of DES

- Accelerates development time and faster CI
Benefits of DES

- Accelerates development time and faster CI
- Realistic and deterministic simulation
Benefits of DES

- Accelerates development time and faster CI
- Realistic and deterministic simulation
- Simulate any date and time of the day
Multi-Threaded System

Backend

Orders  Stock  Motion  . . .
Time Leak - Event-Driven Component

- Not naturally tied to time
- SimPy supports event-driven processes
- Not suitable for multi-threaded systems
Time Leak - Event-Driven Component

- Not naturally tied to time
- SimPy supports event-driven processes
- Not suitable for multi-threaded systems

Solution:
Inherit from *Queue* and create a SimPy process that *joins* on itself in each time tick
from threading import Thread
from queue import Queue
import simpy

time_tick = 1
sim = True

class EventDrivenQueue(Queue):
    def __init__(self, env, *args, **kwargs):
        super().__init__(*args, **kwargs)
        if sim:
            env.process(self._sim_join(env))

    def _sim_join(self, self, env):
        while True:
            self.join()
            yield env.timeout(time_tick)

class EventDrivenComponent:
    def run(self):
        while True:
            msg = q.get()
            print(f"Got {msg}"
            q.task_done()

class SimRobot:
    def work(self, self, env):
        i = 1
        while True:
            q.put(f"msg {i}" Attorney)
            i += 1
            yield env.timeout(time_tick)
Implementation

- SimPy code runs in simulation only
- Can’t use the usual time-related functions. Wrapping time-related functionality in our own module
  - `time.time()`
  - `time.sleep()`
  - ...
- Debugging - simulation timestamp in log
Distributed Simulation

- SERVICE
  - Local simpy

- SERVICE
  - Local simpy

- SERVICE
  - Local simpy

- SERVICE
  - Local simpy

BARRIER
SERVER
Distributed Simulation

create simpy process
start local simpy

loop
do some work

progress local simpy

SERVICE

ready
approve

BARRIER SERVER

once all clients are ready
Distributed Simulation

create simpy process
start local simpy

loop
do some work

sim time freezes
progress local simpy

ready
approve
once all clients are ready

SERVICE

BARRIER SERVER
Summary

- Simulation is a powerful tool
- DES makes it more powerful
- SimPy is SimPLe
- Time leak - synchronize all components time
- Easy to extend to a distributed simulation
Thank You!