Simulating Animal Movement

Arsim Dzambazoski

Table of contents

Intro

References i

- asyncio.
- P. Blackwell.
  Random diffusion models for animal movement.
  Ecological Modelling, 100(1):87–102, 1997.
- E. A. Codling, M. J. Plank, and S. Benhamou.
  Random walk models in biology.
  Journal of The Royal Society Interface, 5(25):813–834, 2008.
- vertical-pod-autoscaler.
- E. Gurarie and O. Ovaskainen.
  Characteristic spatial and temporal scales unify models of animal movement.
  The American Naturalist, 178(1):113–123, 2011.
  PMID: 21670582.

- R. A. Maller, G. Müller, and A. Szimayer.
  Ornstein–Uhlenbeck Processes and Extensions, pages 421–437.
  Springer Berlin Heidelberg, Berlin, Heidelberg, 2009.
- Minikube.
- E. Renshaw and R. Henderson.
  The correlated random walk.
  Journal of Applied Probability, 18, 06 1981.
- E. D. Ungar, Z. Henkin, M. Gutman, A. Dolev, A. Genizi, and D. Ganskopp.
  Inference of animal activity from gps collar data on free-ranging cattle.
  Rangeland Ecology & Management, 58(3):256–266, 2005.

Introduction

- Using GPS data to analyse animal movement has become an indespensible technique.
- Motion sensor data significantly improved the predictive ability for different animal activities [9]
- Common models to describe animal behavior are the Ornstein-Uhlenbeck process and random walks
- This project used simple random walks

Models

Ornstein-Uhlenbeck Process

$$dX = \gamma X dt + \sigma dB$$

[6], [2] $\gamma, \sigma \geq 0$ are real constants B describes Brownian motion

- First proposed in 1930
- Wide range of applications in Finance,Physics,Biology and Ecology
- Markov process
- Stationary
- Gaussian

Random Walk

- Correlated Random Walk [8]
    - successive direction steps are correlated
- Simple Random Walk [3],[5]
    - succesive direction steps are uncorrelated
    - isotropic
- Model Setup
    - Simple Random Walk
    - every path is iid

Simple Random Walk

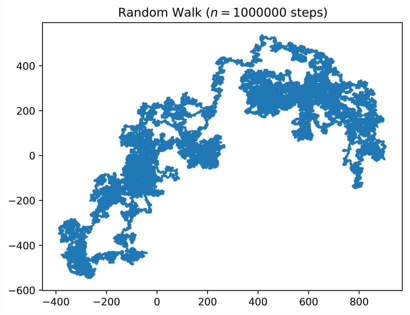


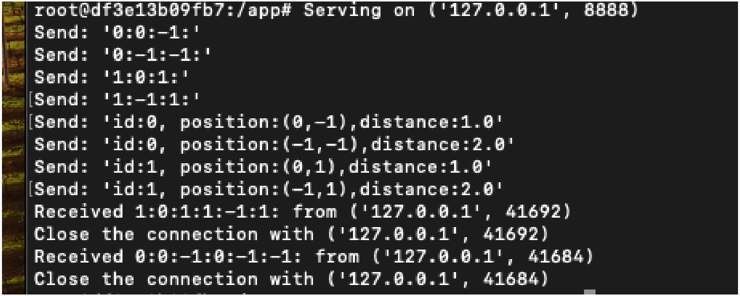
Figure 1: Random walk for one animal

Generator and Reader

Generator and Reader

- asyncio is a library to write concurrent code using the async/await syntax.[1]
- generator connects to the reader at 127.0.0.1 port 8888
- To make the reader accessible to the network it should we should bind it to 0.0.0.0.
- generator sends $id : x : y$
- reader receives message and sends $id : x : y : d$ where d is the total distance to the origin

Generator/Reader Output



```
root@df3e13b09fb7:/app# Serving on ('127.0.0.1', 8888)
Send: '0:0:-1:'
Send: '0:-1:-1:'
Send: '1:0:1:'
Send: '1:-1:1:'
Send: 'id:0, position:(0,-1),distance:1.0'
Send: 'id:0, position:(-1,-1),distance:2.0'
Send: 'id:1, position:(0,1),distance:1.0'
Send: 'id:1, position:(-1,1),distance:2.0'
Received 1:0:1:1:-1:1: from ('127.0.0.1', 41692)
Close the connection with ('127.0.0.1', 41692)
Received 0:0:-1:0:-1:-1: from ('127.0.0.1', 41684)
Close the connection with ('127.0.0.1', 41684)
```

Figure 2: Terminal Output of the script

Deployment

Deployment

- To deploy the software to Kubernetes we need to build a Docker image first.
- How does Docker handle dependencies?
- For local deployment we used Minikube [7]
- Kubernetes Deployment

Docker

- The generator depends on the reader
- With Docker-Compose we can run both python scripts
- Use the depends_on attribute inside the compose.yaml file

Kubernetes

- To make the Docker images accessible to Minikube: Minikube image load image-name
- imagePullPolicy needs to be set to Never
- The reader is setup as a service and the generator is a job
- To make the service accessible to the network we need minikube tunnel

Scaling of the Cluster

Motivation

- When the load is high the application might need more resources when the load is low it needs fewer resources. Allocating the maximum resources would be costly that is where automatic up and down scaling comes in.

HPA

- The first scaling method is horizontal pod autoscaling (HPA)
- HPA scales the number of pods
- For autoscaling to run we need first to enable the metrics-server
  - minikube addons enable metrics-server
  - minikube addons list
- We need CPU and memory requsts and limits
- requests are what the service requests
- Limits are how much is availabe

Experiment with HPA

```
NAME                 REFERENCE                        TARGETS        MINPODS   MAXPO
DS   REPLICAS   AGE
reader-deployment   Deployment/reader-deployment   cpu: 0%/50%   1         10
     1          3h35m
reader-deployment   Deployment/reader-deployment   cpu: 9%/50%   1         10
     1          3h38m
```

Figure 3: HPA with 100 millicores and 125Mb

The HPA tries to keep CPU utilization below 50%. If it goes above 50% more pods are created

VPA

- The second scaling method is vertical pod autoscaling (VPA) [4]
- VPA scales CPU and memory
- VPA needs to be downloaded from Github
- we need a VPA configuration inside the service yaml file
- targetRef which is the deployment we wish to autoscale
- initialise an updatePolicy

Experiment with VPA

```
    Type:                    RecommendationProvided
  Recommendation:
    Container Recommendations:
      Container Name:  reader
      Lower Bound:
        Cpu:     25m
        Memory:  262144k
      Target:
        Cpu:     25m
        Memory:  262144k
      Uncapped Target:
        Cpu:     25m
        Memory:  262144k
      Upper Bound:
        Cpu:     1403m
        Memory:  1438074878
Events:          <none>
(base) arsimdzambazoski@Mini-von-Arsim iot %
```

Figure 4: VPA recommendations

Conclusion

Summary

- Created a simple random walk to simulate animal movement.
- Created the generator and reader
- Deployed the software to a local Kubernetes cluster
- Looked into HPA and VPA scaling behavior
- Next steps
    - Make model more sophisticated with correlated random walk and Ornstein-Uhlenbeck process.

Demo

Questions?