Simulating Animal Movement

Arsim Dzambazoski
arsim.d@icloud.com

Abstract

I created a simple algorithm to simulate animal movement. With that algorithm I created a data stream generator that sends the generated data over the network and a reader that reads the data stream. The generator and the reader were both deployed on a local Kubernetes cluster.

ACM Reference Format:
Arsim Dzambazoski. 2024. Simulating Animal Movement. In . ACM, New York, NY, USA, 3 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 Introduction

Using GPS data to analyse animal movement has become an indespensible technique. [1] focuses on characterizing the spatial and temporal activities of free-ranging cows using GPS data over a 2-year period. [11] revealed that motion sensor data significantly improved the predictive ability for different animal activities, with specific models developed for grazing, resting, and traveling behaviors. [7] investigated how varying the temporal scale affects three key movement parameters: speed, sinuosity, and turning angle, while also exploring the relationship between temporal scale and uncertainty in trajectory data points. My focus will be on using a simple random walk algorithm to simulate the animal movement and then create a data generator and data stream reader to simulate the network communication.The software is then deployed to Kubernetes.

2 Subtask 1

A common model to describe animal behavior is the Ornstein-Uhlenbeck model which is explained in [3] and [8]. The Ornstein-Uhlenbeck model is a continous time model there are also discrete time models like the correlated random walk,which is described in [6] and [10].The difference between a correlated random walk and a simple random walk is that in the simple random walk the next step is independent of previous steps while in the correlated random walk they are dependent. I decided to use a simple random walk since it is simpler to implement. The steps are independent of each other. The movement of the animals are iid. Which means all animals have the same distribution and the movements are independent of each other. [4] gives an overview of random walks used in biology. Figure 1 shows the trajectory of the random random walk for $n = 1000000$ steps.

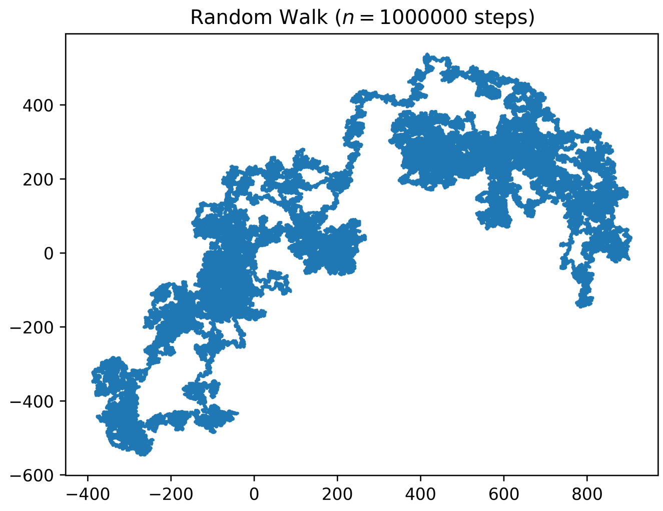
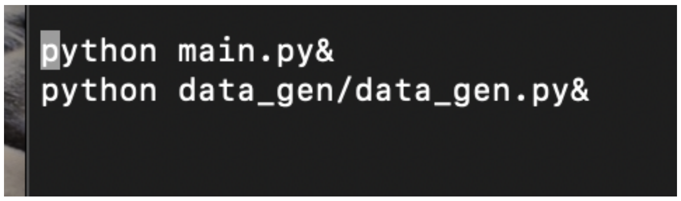


Figure 1: Random walk for one animal



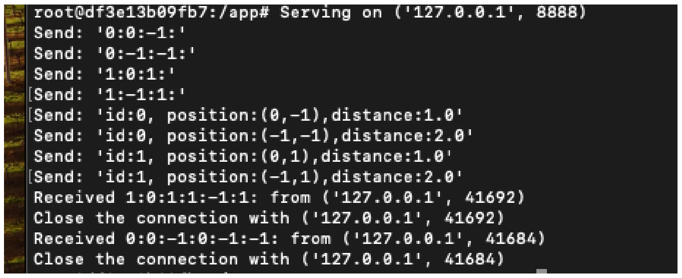
```
python main.py&
python data_gen/data_gen.py&
```

Figure 2: The shell script

3 Subtask 2

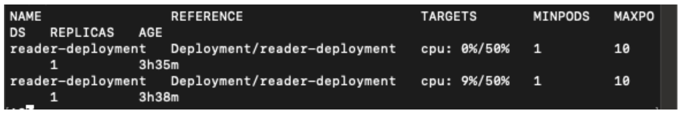
For the asynchronous production of data I used the asyncio [2] Python library. The data generator connects to 127.0.0.1 at port 8888. To make the reader accessible to the network it should we should bind it to 0.0.0.0. In every time step the generator sends data to the network in the format $id : x : y$. For every animal one generator is put into a list and with the gather method we can run all generators concurrently. The reader sets up a server that listens to all incoming connections. While the stream is read the current total distance is calculated. The reader sends $id : x : y : d$, where d is the distance to the origin. To make it easier to run both scripts they were put together in a shell script. The shell script can be seen in Figure 2. We run both scripts with the python command and let them run simultaneously with &.

The terminal output is shown in Figure 3. Here we have two animals and two steps. The first four sends are from the data generator and the next batch is from the reader. Here the reader returns the position and the total distance. The last messages are the reader confirming it received the data and closing the connection.



```
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 3: Terminal Output of the script



```
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 4: HPA with 100 millicores and 125Mb

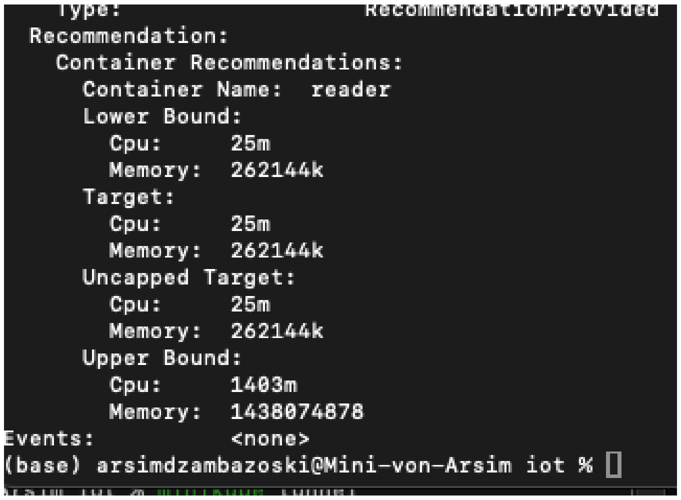
4 Subtask 3

The first step in deploying the software to Kubernetes is to create a docker image. The data generator connects to the reader so the reader need to run first. Creating two images and deploying seperately would make it difficult. The data generator pod wouldn't start since it can't find the reader because the reader has a different port and IP address inside the pod. In Docker it is possible to define the dependencies with the depends_on attribute inside a compose.yaml file. With Docker-Compose we can build the image and then run in it inside Docker. Kubernetes doesn't have something equivalent. My solution is to run the reader as a service and the data generator is run as a job. For running Kubernetes locally on my machine I used Minikube [9].

4.1 HPA vs VPA

Now that we have cluster setup we come to the scaling of clusters. When the load is high the application might need more resources when the load is low it needs fewer resources. Alocating the maximum resources would be costly that is where automatic up and down scaling comes in. There are two methods horizontal pod autoscaling (HPA) and vertical pod autoscaling (VPA). HPA scales the number of pods while VPA scales the CPU and memory directly. For HPA to run we need first to enable the metrics-server and also give CPU and memory requsts and limits.The requests are how much CPU and memory the pods requests and limits are how much is avalaible. The requests were 100 millicores and 512Kb and the limits were 250 millicores and 125Mb. Here millicores mean one thousandth of a CPU core so 1000 millicores are a whole CPU core Mb stands for Megabyte and Kb is Kilobyte. In HPA the controller will increase the number of replicas to maintain an average CPU utilization of 50%. The data generator ran 200 animals and 10000000 number of timesteps each. The result of the autoscaler are shown in Figure 4. It shows that the CPU utilization is just 9% so one pod is enough. This also shows that the 100 millicores are more than enough and it could run with a fraction of that.

The VPA is a separate project that needs to be downloaded from Github. To run it we need a VPA configuration inside our yaml file



```
    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 5: VPA recommendations

where we define our deployment. In the configuration we need to define a targetRef which is the deployment we wish to autoscale and also set a updatePolicy. More information about the the VPA can be found on the Github page [5]. The result can be seen in Figure 5. There we can see that the VPA recommends a lower bound and target of 25 millicores and an upper bound of 1403 millicores for the CPU. the target corresponds to the request and the upper bound is the limit. From the HPA we saw that with 100 millicores we had an CPU utilization of only 9% so we were underutilizing the CPU. The 25 millicores offer a better utilization of the CPU.

5 Conclusion

I managed to create a simple random walk algorithm to simulate animal movement. With that algorithm I created a data stream generator that sends the generated data over the network and a reader that reads the data stream. I managed then to deploy the software to Kubernetes. Then finally I looked at the scaling behavior of HPA and VPA. In an next I would extend the simple random walk to a correlated random walk or using the Ornstein-Uhlenbeck process for the data generation. This would give a more realistic simulation.

References

[1] Estell Rick E. Fredrickson Ed L. Doniec Marek Detweiler Carrick Rus Daniela James Darren Nolen Barbara Anderson Dean M., Winters Craig. 2012. Characterising the spatial and temporal activities of free-ranging cows from GPS data. *The Rangeland Journal 34, 149-161* (2012).

[2] asyncio [n. d.]. *asyncio.* Retrieved July 8, 2024 from https://docs.python.org/3/library/asyncio.html

[3] P.G. Blackwell. 1997. Random diffusion models for animal movement. *Ecological Modelling* 100, 1 (1997), 87–102. https://doi.org/10.1016/S0304-3800(97)00153-1

[4] Edward A Codling, Michael J Plank, and Simon Benhamou. 2008. Random walk models in biology. *Journal of The Royal Society Interface* 5, 25 (2008), 813–834. https://doi.org/10.1098/rsif.2008.0014 arXiv:https://royalsocietypublishing.org/doi/pdf/10.1098/rsif.2008.0014

[5] github:vpa [n. d.]. *vertical-pod-autoscaler.* Retrieved July 8, 2024 from https://github.com/kubernetes/autoscaler/tree/9f87b78df0f1d6e142234bb32e8acbd71295585a/vertical-pod-autoscaler#quick-start

[6] Eliezer Gurarie and Otso Ovaskainen. 2011. Characteristic Spatial and Temporal Scales Unify Models of Animal Movement. *The American Naturalist* 178, 1 (2011),

113–123. https://doi.org/10.1086/660285 arXiv:https://doi.org/10.1086/660285 PMID: 21670582.

[7] Patrick Laube and Ross S Purves. 2011. How fast is a cow? Cross-scale analysis of movement data. *Transactions in GIS* 15, 3 (2011), 401–418.

[8] Ross A. Maller, Gernot Müller, and Alex Szimayer. 2009. *Ornstein–Uhlenbeck Processes and Extensions.* Springer Berlin Heidelberg, Berlin, Heidelberg, 421–437. https://doi.org/10.1007/978-3-540-71297-8_18

[9] minikube [n. d.]. *Minikube.* Retrieved July 8, 2024 from https://minikube.sigs.k8s.io/docs/

[10] Eric Renshaw and Robin Henderson. 1981. The Correlated Random Walk. *Journal of Applied Probability* 18 (06 1981). https://doi.org/10.2307/3213286

[11] Eugene D. Ungar, Zalmen Henkin, Mario Gutman, Amit Dolev, Avraham Genizi, and David Ganskopp. 2005. Inference of Animal Activity From GPS Collar Data on Free-Ranging Cattle. *Rangeland Ecology Management* 58, 3 (2005), 256–266. https://doi.org/10.2111/1551-5028(2005)58[256:IOAAFG]2.0.CO;2