Distributed IoT Data Acquisition and Analytics: Animal Behavior Analysis

Sepideh Shamsizadeh
Zurich, Switzerland
sepideh.92sh@gmail.com

Abstract

The paper presents a full-scale system for studying animal behavior using IoT data gathering and analysis. The application is designed to mimic realistic animal movements by converting coordinates generated into spatial descriptions similar to GPS outputs from real-life trackers. Our solution, comprising the Movement Generator and the Data Stream Analyzer, is a reliable toolset. The Movement Generator creates movement data for multiple animals in motion and streams it dynamically, while the Data Stream Analyzer consistently receives this stream and computes metrics, such as average distance moved, and trains models based on historical information about anomaly detection. When deployed on a Kubernetes cluster in containerized form within constrained resources, efficient scalability can be achieved for handling high data loads. By leveraging both Horizontal Pod Autoscaler (HPA) and Vertical Pod Autoscaler (VPA), the system performs well under varying load conditions, ensuring its reliability in large-scale livestock and wildlife management applications.

Keywords

Internet of Things, animal behavior analysis Kubernetes, autoscaling, anomaly detection, geospatial data

1 Introduction

Animal behavior analysis plays a pivotal role in both wildlife management and agricultural practices. By understanding the movement patterns and behaviors of animals, researchers and practitioners can gain valuable insights into their health, welfare, and interactions with their environments. Traditional methods of studying animal behavior, such as manual observation, are labor-intensive and prone to human error, often leading to incomplete or biased data. Recent advancements in technology have introduced automated methods for tracking and analyzing animal behavior. The integration of the Internet of Things (IoT), machine learning, and cloud computing has enabled continuous, real-time monitoring of animals with minimal human intervention.

These technologies facilitate the collection of precise data on animal locations, movement patterns, and physiological parameters, which can be analyzed to detect anomalies indicative of health issues, stress, or environmental changes. This project aims to develop a comprehensive system that tracks animal movements, applies machine learning for anomaly detection, and manages computational resources efficiently using Kubernetes. The system continuously gathers data on animal behavior by leveraging IoT devices such as GPS collars, RFID tags, and various sensors. Machine learning algorithms analyze this data to identify unusual patterns, providing early warnings of potential problems. Kubernetes ensures that the system can scale effectively to handle large volumes of data, maintaining robust performance under varying load conditions. This integrated approach enhances the accuracy and efficiency of animal behavior monitoring, supports wildlife conservation efforts, and improves livestock management practices.

2 Related Work

2.1 Animal Behavior Analysis

Analyzing animal behavior is essential for understanding various biological and ecological phenomena. Traditional behavioral analysis methods often rely on manual observation, which can be time-consuming and subject to observer bias. With technological advancements, automated methods for tracking and analyzing animal behavior have become increasingly popular. Global Positioning Systems (GPS) and Geographic Information Systems (GIS) have significantly enhanced the precision and efficiency of monitoring animal movements. For instance, GPS collars have been effectively used to track the foraging activities of dairy cattle, revealing detailed patterns in grazing, resting, and traveling behaviors [3].

Moreover, machine learning and IoT technologies have further revolutionized animal behavior analysis. IoT-based systems, such as the WildTrack project, utilize passive RFID tags and LoRa communication to monitor the movement and behavior of small animals. This approach allows for continuous, real-time data collection with minimal human intervention, providing valuable insights into animal behavior and environmental interactions [5].

2.2 IoT Data Acquisition

The advent of the Internet of Things (IoT) has revolutionized the field of data acquisition, particularly in wildlife tracking and livestock management. IoT-based systems facilitate data collection on animal movement, behavior, and health, enabling researchers to monitor and analyze these parameters in real time. Technologies like Sigfox and LoRa have been extensively used due to their long-range communication capabilities and low power consumption. Sigfox, for instance, has been successfully deployed to track various animal species, achieving impressive communication distances [6]. Similarly, LoRa has proven effective in projects like WildTrack, which supports continuous and automated wildlife tracking [5].

2.3 Machine Learning for Anomaly Detection

Machine learning has significantly advanced anomaly detection, particularly in animal behavior analysis. Methods such as information granules transform data into semantic multidimensional spaces, capturing nuances in animal movements to identify anomalies [1]. Techniques like Outlier Detection in Animal Multivariate Trajectories (ODAMT) have been applied to monitor cattle reproductive events, demonstrating practical utility in real-world scenarios

[4]. Additionally, machine learning algorithms, including support vector machines, hidden Markov models, and deep learning techniques, provide sophisticated modeling for early anomaly detection and intervention, enhancing wildlife conservation and livestock management [1, 4].

3 Methodology

3.1 Algorithm for Animal Movement

Inspired by [2], the animal movement algorithm simulates realistic cow behavior in a plain field, converting x/y coordinates to WGS84 longitude/latitude pairs to mimic real-world GPS data. Each cow's movement is modeled by an instance of the *CowMovementSimulator* class, initialized with parameters such as field dimensions (*field_width* = 600 meters, *field_height* = 200 meters), time step (*time_step* = 1 second), total simulation time (*total_time* = 18000 seconds), GPS noise level (*gps_noise_level* = 1.0 meter), and reference geographic coordinates (latitude = 47.0 degrees, longitude = 8.0 degrees).

The cow's speed and turning angle are determined probabilistically at each time step, following the methods described in [2]. The speed is sampled from a normal distribution based on the temporal scale of the movement duration, and the turning angle is sampled from a distribution with a mean specific to the temporal scale. Position updates incorporate GPS noise to simulate real-world inaccuracies. Movements are constrained within the field boundaries, and if a cow moves out of bounds, it turns by a random angle between 90 and 180 degrees and continues moving.

The algorithm includes a mechanism to detect stationary periods by analyzing the average distance covered over a specified window of time steps. The cow is considered stationary if the average distance is below a threshold.

The algorithm translates the x/y coordinates into WGS84 longitude/latitude pairs based on reference coordinates to ensure realistic movement. This conversion uses predefined constants for meters per degree latitude (111320 meters) and longitude, adjusted for the reference latitude (111320 meters * cos(47 degrees)).

The *CowMovementSimulator* also includes a function to plot the movement trajectories using Matplotlib, providing a visual representation of the paths taken by each animal over time. This visual representation helps in understanding the movement patterns.

In the main function, an instance of the simulator is created, and the movement is simulated for the total number of time steps. The trajectory is collected, and a few sample outputs are printed. Finally, the movement trajectory is plotted for visualization. Figure shows 1 the simulation of cow movement over 18000 seconds, demonstrating the algorithm's effectiveness in generating realistic movement patterns.

3.2 Tool 1: Movement Generator

The Movement Generator tool is designed to simulate and stream the movement data of multiple animals concurrently. It leverages the *CowMovementSimulator* class to generate realistic movement patterns for each animal. The tool initializes multiple instances of the *CowMovementSimulator*, with the number of animals specified as an input parameter. Each instance simulates the movement of an individual animal within the defined field dimensions.

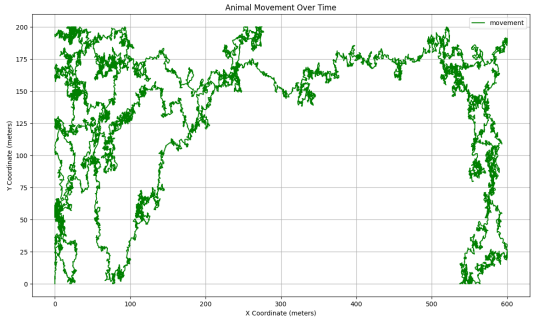

Figure 1: Simulation of cow movement over 18000 seconds.

The core functionality is implemented in the *send_movements* method, which runs an asynchronous loop. During each iteration of this loop, the simulator generates the next position of its respective animal, converts the position data (cow ID, latitude, and longitude) into JSON format, and sends it over a WebSocket connection. This continuous stream of movement data effectively simulates the real-time movement of multiple animals. The tool also includes a WebSocket server to handle multiple clients. New clients can connect and receive the streamed data, allowing for real-time monitoring of the simulated animal movements. The *handler* method manages the registration and unregistration of WebSocket connections.

For visualization, the *plot_trajectories* method collects the generated coordinates and plots them using Matplotlib. This visualization displays the paths each animal takes over time, clearly representing their movement patterns. The method converts latitude and longitude coordinates to meters for plotting, ensuring an accurate spatial representation of the movements.

The Movement Generator tool is designed to run as a long-running asynchronous process. It continuously generates and streams movement data, providing a robust solution for simulating and analyzing real-time animal movement patterns.

3.3 Tool 2: Data Stream Analyzer

The Data Stream Analyzer continuously processes and analyzes real-time movement data streamed from a WebSocket server. This tool is developed using Python and leverages asynchronous programming to handle high-frequency data streams efficiently.

This tool's heart is the DataStreamAnalyzer class, which tracks metrics for each animal. The class maintains a dictionary to store data for each cow, including the total distance traveled, the number of recorded positions, and the last known position. When new data is received, the update_metrics method updates these values. If a previous position exists, it calculates the distance from the last position using the Haversine formula, which accurately computes distances between two points on the Earth's surface.

The average_distance method calculates the average distance traveled per cow by dividing the total distance by the number of positions. This metric helps in understanding each cow's movement patterns and behaviors over time.

The receive_movements function establishes a WebSocket connection to receive and process movement data continuously. It decodes JSON messages containing cow ID, latitude, and longitude and updates the metrics accordingly. Real-time logging is implemented to provide feedback on data reception and processing, helping monitor the system's performance and debug any issues.

The tool uses an LSTM Autoencoder for anomaly detection. The LSTMAutoencoder class defines an LSTM-based model for encoding and decoding the sequences of movement data. The model is pretrained using historical data, and during operation, it continuously processes the incoming data to detect anomalies based on variations from the expected movement patterns.

The tool preprocesses positions by scaling the latitude and longitude data and creating sequences for the LSTM model. The calculate_metrics function periodically computes metrics, including average distance and variations, for anomaly detection. If the variation exceeds a certain threshold, it is flagged as an anomaly.

The tool also supports retraining the LSTM model using the latest data, ensuring the model remains updated and accurate. The retrain_model function retrains the model at specified intervals using the collected movement data, which is essential for adapting to new patterns and behaviors.

3.4 Deployment and Scaling in Kubernetes

The deployment and scaling of the software in Kubernetes involved several key steps, ensuring efficient management and adaptability to varying loads.

The applications were first containerized using Docker. Each tool was packaged into a Docker container to ensure consistent deployment across different environments. The containerization process involved creating Dockerfiles for each tool, specifying the necessary dependencies and runtime configurations.

Once containerized, the applications were deployed to a local Kubernetes cluster using Minikube. Kubernetes manifests, including deployment and service configurations, were created for each tool. These manifests defined the desired state of the applications, specifying the number of replicas, container images, ports, and resource requests and limits. The deployments ensured that the applications ran in the cluster, while the services exposed them to internal and external traffic.

Horizontal Pod Autoscaler (HPA) and Vertical Pod Autoscaler (VPA) mechanisms were implemented to handle increased loads and ensure scalability. The HPA was configured to scale the number of *tool1* replicas based on CPU utilization. This meant that when CPU usage exceeded a specified threshold, additional pods were automatically created to handle the load. The VPA provided recommendations for CPU and memory resources, optimizing the resource allocation for each *tool1* pod.

Load testing was conducted using the *siege* tool to simulate high traffic. The load balancer, set up with a NodePort service, allowed external traffic to reach *tool1*. During the load test, the HPA actively scaled the deployment by increasing the number of replicas in response to high CPU utilization. The VPA adjusted the resource limits and requests based on observed usage, ensuring efficient resource management.

4 Results and Discussion

4.1 Metrics Computation

The results from the Data Stream Analyzer and the model training process provide insights into the system's movement patterns and anomaly detection capabilities.

Each animal's movement was analyzed, and the average distance traveled per position update was calculated. The results indicate that the average distance traveled by each animal was around 1.25 meters. Variations in movement patterns were relatively low, with mean variations ranging from 0.0289 to 0.0990. This indicates that the animals exhibited consistent movement patterns with few deviations from the expected behavior.

The LSTM Autoencoder model was trained over 20 epochs, with training and validation loss decreasing over time. The final training loss stabilized at 0.0014, and the validation loss stabilized at 0.0017, indicating that the model could learn the movement patterns effectively without overfitting. After training, the model detected 71 anomalies in the movement data, suggesting instances where the movement deviated significantly from the learned patterns.

Figure 2 shows the movement trajectories for 10 cows over one hour, demonstrating the tool's effectiveness in tracking and visualizing animal movements.

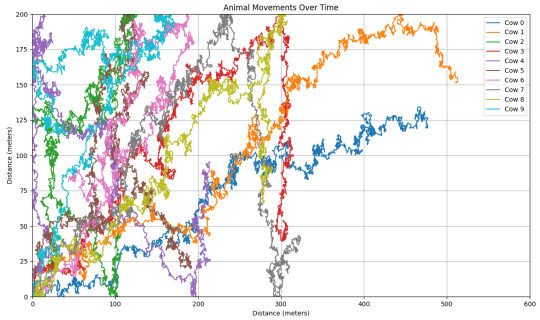

Figure 2: Movement trajectories for 10 cows over one hour.

4.2 Scaling Behavior

The system's scaling behavior was evaluated using both Horizontal Pod Autoscaler (HPA) and Vertical Pod Autoscaler (VPA) mechanisms to ensure efficient resource management and scalability under varying loads. The performance of these autoscalers was monitored during periods of simulated high load.

4.2.1 Horizontal Pod Autoscaler (HPA). The HPA was configured to scale the number of *tool1* replicas based on CPU utilization. During the load tests, which simulated increased traffic to the *tool1* pods, the HPA actively monitored the CPU usage and adjusted the number of replicas accordingly.

Figure 3 shows the CPU usage pattern for *tool1* pods over time, as monitored by Prometheus. As the load increased, the CPU usage spiked, triggering the HPA to scale up the number of replicas to

handle the additional load. Figure 4 from Grafana further corroborates these findings, showing detailed CPU usage over a specified period.

Figure 5 illustrates the number of *tool1* replicas over time, as monitored by Helm. The number of replicas increased from 2 to 10, demonstrating the HPA's ability to respond dynamically to changing resource demands.

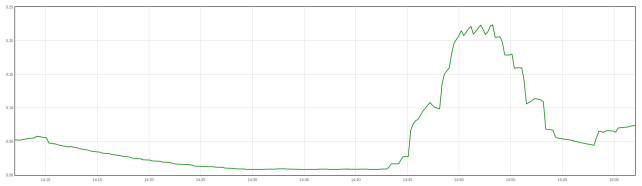

Figure 3: CPU usage of tool1 pods over time monitored by Prometheus.

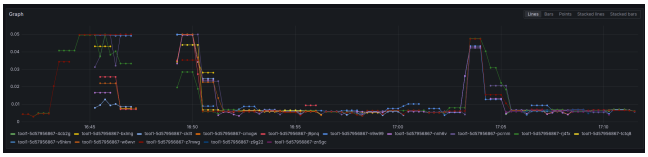

Figure 4: CPU usage of tool1 pods over time monitored by Grafana.

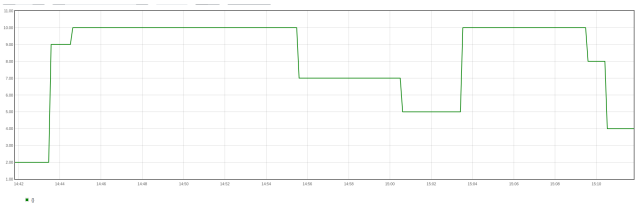

Figure 5: Number of tool1 replicas over time monitored by Prometheus.

The HPA successfully maintained CPU utilization within the target range by increasing the pod count during high load periods and scaling down when the load decreased. This ensured that the system remained responsive and maintained performance without over-provisioning resources.

4.2.2 Vertical Pod Autoscaler (VPA). The VPA was configured to optimize resource allocation for the *tool1* pods by providing recommendations for CPU and memory resources based on observed usage. During the load tests, the VPA adjusted the resource requests and limits to ensure efficient utilization of resources.

The VPA provided dynamic recommendations for CPU and memory, ensuring each pod had adequate resources to handle the load while avoiding resource wastage. This adjustment helped maintain the performance of the *tool1* pods, especially during peak usage times.

5 Conclusion

This paper presented a comprehensive system for analyzing animal behavior using distributed Internet of Things (IoT) data acquisition and advanced analytics, deployed on a Kubernetes cluster. The system leveraged Horizontal Pod Autoscaler (HPA) and Vertical Pod Autoscaler (VPA) to manage computational resources efficiently, ensuring robust performance under varying load conditions.

The HPA dynamically adjusted the number of *tool1* replicas based on CPU utilization, effectively handling increased load during data streaming and processing. The VPA optimized resource allocation by adjusting CPU and memory requests, ensuring efficient utilization of resources.

The load tests demonstrated the system's capability to scale in response to high data loads, maintaining performance and reliability. The HPA scaled the number of replicas up to the maximum limit during peak loads and scaled down during low usage periods, while the VPA continuously optimized resource allocation.

Future work includes further refining the autoscaling policies to handle more complex load patterns and integrating additional metrics for autoscaling decisions. The system can provide a more comprehensive analysis of animal behavior by integrating environmental data such as weather conditions, geographical information, and other relevant data sources. This integration will enable the detection of correlations between environmental factors and animal movements, leading to more accurate predictions and insights.

Additionally, exploring advanced machine learning techniques for anomaly detection and enhancing the system's real-time processing capabilities will contribute to more accurate and efficient animal behavior analysis. This integrated approach will support more informed decision-making in wildlife conservation efforts and livestock management practices.

References

[1] Adam Kiersztyn et al. "The use of information granules to detect anomalies in spatial behavior of animals". In: *Ecological Indicators* 136 (2022), p. 108583.

[2] Patrick Laube and Ross S. Purves. "How fast is a cow? Cross-Scale Analysis of Movement Data". In: *Transactions in GIS* 15.3 (2011), pp. 401–418.

[3] Jairo Mora-Delgado et al. "Application of GPS and GIS to study foraging behavior of dairy cattle". In: *Agronomía Costarricense* 40.1 (2016), pp. 81–88.

[4] Jorge Navarro et al. "Outlier detection in animal multivariate trajectories". In: *Computers and Electronics in Agriculture* 190 (2021), p. 106401.

[5] Robert Ross et al. "WildTrack: An IoT system for tracking passive-RFID microchipped wildlife for ecology research". In: *Automation* 3.3 (2022), pp. 426–438.

[6] Timm A Wild et al. "A multi-species evaluation of digital wildlife monitoring using the Sigfox IoT network". In: *Animal Biotelemetry* 11.1 (2023), p. 13.