App-Learner Project – Final Report:

Sally Turutov, Matan Beigel and Ido Magner

Introduction:

In this project, we aim to analyze and predict resource consumption in a Kubernetes (K8) environment. The data we have consists of two different metrics: CPU and Memory, which are sampled every minute. Each application has multiple runtime windows within the same pod. The memory values are measured in bits, while the CPU values are obtained by multiplying the number of cores in use by 5. The dataset separated to time windows, allowing us to identify trends and patterns in the data.

Our mission is to thoroughly research the problem and preprocess the data. This involves aggregating the data points and apply normalization to enhance the performance of our model. By analyzing the data, we aim to provide valuable insights both on the data itself and on the model, we will be working with. We will leverage a deep learning (DL) model to handle this bivariate problem and predict the resource consumption for the next 24 points, corresponding to the next 12 hours. We will visualize the results using graphs, enabling a clear representation of the resource consumption patterns.

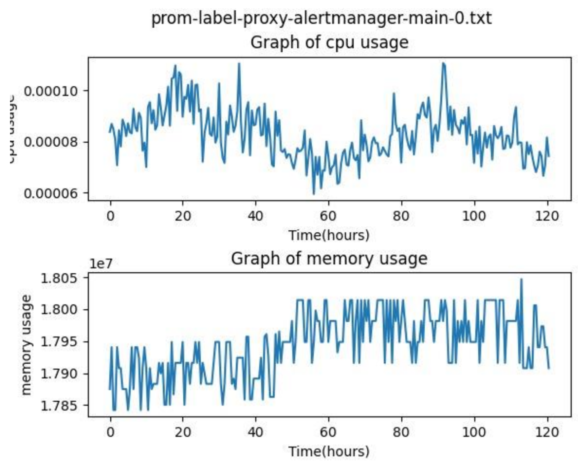
Ultimately, this project aims to provide valuable insights into resource consumption in a Kubernetes environment and develop a predictive model that can forecast the resource consumption for the next hour and day. By understanding and predicting resource utilization, organizations can optimize their infrastructure, ensure efficient allocation of resources, and make informed decisions to enhance system performance in a dynamic and scalable environment like Kubernetes (K8).

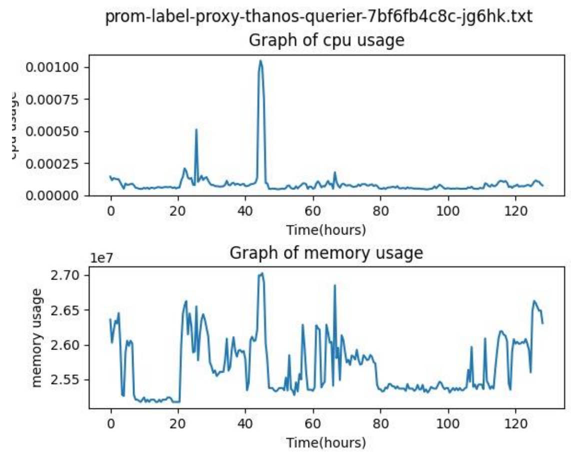
Initial Data Observation and Pre-processing:

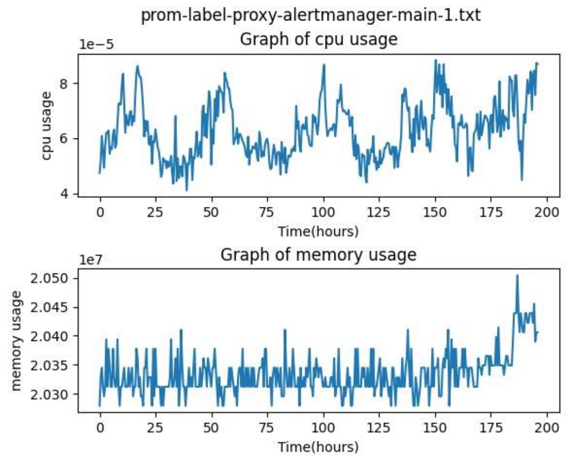
In this project, our objective is to analyze and predict resource usage in a Kubernetes (K8) environment. The available data includes two main metrics: CPU and Memory, recorded every minute.

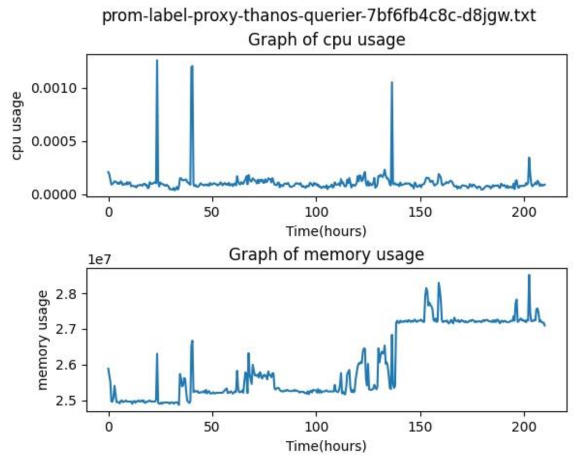
Our initial approach involved examining the JSON data files and finding matching data points for CPU and memory based on time. However, we noticed overlaps of time windows for the same application between pods, which needed to be considered during data processing for the model. To simplify the process, we decided to work with separate JSONs for each application, containing data only for that specific application. As a starting point, we focused on one application, "prom-label-proxy" (randomly selected), and visualized the data for each pod separately, with graphs depicting CPU and memory usage.

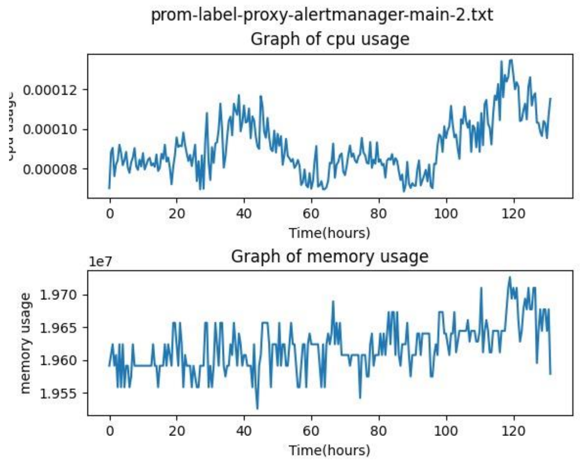
For the analysis, we aggregated the data using the median aggregation method (which can be easily modified if needed), converting the raw data into more manageable 30-minute intervals. It's essential to note that the graphs do not display specific timestamps but instead show counts of time from the beginning of data collection (0, 1, 2, and so on). These counts enable us to observe resource consumption trends over time without directly referring to real-time information. We created two graphs for each pod: one illustrating memory usage and the other displaying CPU utilization. These visualizations offer an overview of resource consumption patterns and fluctuations in memory and CPU usage across different pods associated with the "prom-label-proxy" application.


prom-label-proxy-alertmanager-main-0.txt
Graph of cpu usage
cpu usage
Time(hours)
Graph of memory usage
1e7
memory usage
Time(hours)


prom-label-proxy-thanos-querier-7bf6fb4c8c-jg6hk.txt
Graph of cpu usage
cpu usage
Time(hours)
Graph of memory usage
1e7
memory usage
Time(hours)


prom-label-proxy-alertmanager-main-1.txt
Graph of cpu usage
1e−5
cpu usage
Time(hours)
Graph of memory usage
1e7
memory usage
Time(hours)


prom-label-proxy-thanos-querier-7bf6fb4c8c-d8jgw.txt
Graph of cpu usage
cpu usage
Time(hours)
Graph of memory usage
1e7
memory usage
Time(hours)


prom-label-proxy-alertmanager-main-2.txt
Graph of cpu usage
cpu usage
Time(hours)
Graph of memory usage
1e7
memory usage
Time(hours)

Further Data Pre-processing:

After collecting data for a specific application, we matched the CPU and memory data based on their timestamps. During this process, we observed two important points: there are gaps between some of the time windows for each pod, and the time windows for CPU and memory data are not identical.

To address these observations, we took a preliminary step by considering only the overlapping data and inserting zeroes between time windows. This approach was adopted as an initial solution to test the model's behavior. It's important to note that this solution is temporary, and our intention is to find a better and more refined approach in the future. However, for the initial testing of the model, we proceeded with this method.

The Model:

In this project, we are tasked with using the multivariate model of LSTNET for time series forecasting. Here's a brief explanation of the LSTNet model:

LSTNet is a framework designed to address multivariate time series forecasting challenges by combining CNN and RNN components. The CNN component focuses on extracting short-term local dependencies among input variables, while the RNN component is responsible for capturing more complex long-term dependencies. To further enhance the model's effectiveness, LSTNet introduces a unique recurrent structure known as Recurrent-skip, which efficiently captures very long-term dependence patterns while aiding optimization.

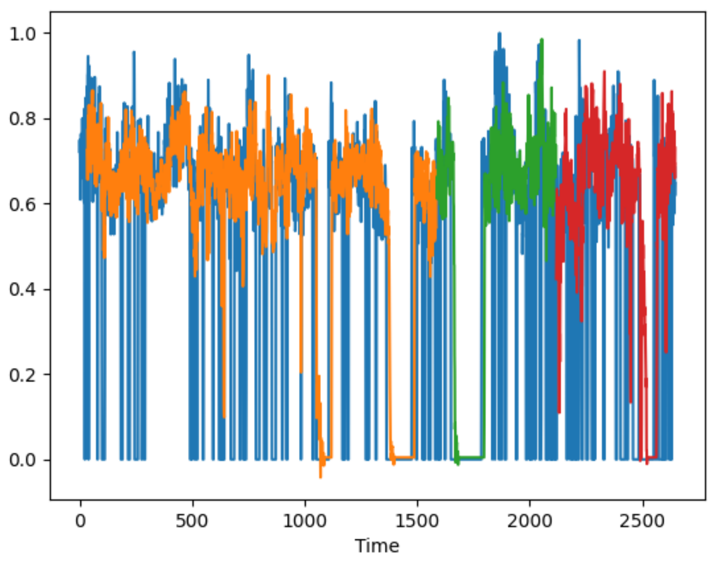
Moreover, LSTNet incorporates a traditional autoregressive linear model running in parallel with the non-linear neural network part. This addition enhances the model's robustness for time series data with varying scales, making it adaptable to different types of multivariate time series datasets.

The model receives its input in the form of a text file containing multivariate time series data. Each row in the text file corresponds to a timestamp, while each column represents a specific variable or feature of interest. For instance, if we are predicting CPU and memory usage for an application running on Kubernetes pods, the text file will have rows representing different timestamps and columns representing the CPU and memory usage data for each pod.

Starting with the model:

Following the initial data pre-processing and model training, we proceeded to experiment with the extracted data from the "prom-label-proxy" application. During these experiments, we tested different hyperparameters, and the outcomes are illustrated in the graph below.

We begin by presenting the results for one of the pods' CPU data for the application. In the graph, the original data (with missing values filled as zeros) is represented by the blue line. The training data is shown in orange, the validation data in green, and the testing data in red.


1.0
0.8
0.6
0.4
0.2
0.0
0
500
1000
1500
2000
2500
Time