An Architecture for Resource Analysis, Prediction and Visualization in Microservice Deployments

Xavier Geerinck

Supervisor(s): Wim Van Den Breen, Pieter Leys (Cisco), Stefaan Vander Rasieren (Cisco)

Abstract—**This article tries to improve the reliability of High Availability systems in large microservice deployments by introducing a multi-layered architecture that is able to analyze, visualize and predict the resource usage of microservices.**

Keywords— **Architecture, Machine Learning, Prediction, Regression, Analysis, Visualization**

I. Introduction

FOREKNOWLEDGE is one of the critical deciding factors for any enterprise to outdistance the competition in a timely manner. When a cloud enterprise is not able to detect this within a timely manner, this can lead to decreasing revenues and the occurrence of potential bottleneck issues. Resulting in a decreased customer satisfaction and Service Level Agreement violations due to the decreased uptime of these services.

This dissertation tackles this by introducing a multi-layered architecture that is able to analyze, visualize and predict the resource usage of microservices in a large scale deployment. Through the use of this analysis, prediction models can be created that are able to issue warnings before the occurrence of these bottlenecks. The enterprise can then use these warnings to devise a new strategy for the microservices deployments to comply the the Service Level Agreements and increase the customer satisfaction.

From the results obtained in this dissertation, it was made clear that the introduced architecture is able to grow, iteratively learn by itself and return the models needed. While still being modular enough to meet the complex demands of different enterprise structures.

II. Platform Architecture

A. Introduction

Data Collection is a task in the world today that is becoming more and more important. Big data warehouses are not unusual anymore and are more and more becoming data lakes, increasing the need to store and process this data within given time constraints. This thesis is no different from that, the main part being the collection of data and analyzing this data to predict the future and what should happen in it. In this chapter a platform architecture is detailed that is able to process, analyze and visualize data that is being produced by different microservices with the goal to predict their resource usage and to issue alerts once a microservice should be scaled up to be able to handle the load on the service.

B. Orchestrator

The created architecture starts by the creation of an orchestrator. This orchestrator manages the different microservices running on each host within the infrastructure. By using an orchestrator, we can utilize the global replication feature that will replicate a microserviceon each host within the orchestration infrastructure. This allows us to create a microservice that can collect resources from the different hosts within our infrastructure, and send these to a central processing hub. In this article Apache Kafka was used as this central processing hub.

C. Layers

Once this data is being collected in a central processing hub, we are able to process it. To be able to create accurate predictions for the resource usage, three layers are introduced.

C.1 Real time Layer

Starting with the Real Time Layer, we need to process the data coming in on the central processing hub, one item at a time. This is done by creating a consumer that is able to create Direct Streams to our processing hub. This consumer then also applies different prediction algorithms that will create our results. Once the consumer is done processing the data, we can forward the found facts into a different topic on the processing hub.

C.2 Batch Layer

For our Batch Layer, data has to be collected through a consumer. This consumer will forward the data into a database that is chosen by us. In the case of this article the choice was made to use InfluxDB as this database system.

This forwarding is done by creating a sink where all our processed data will be sent to. Seeing that this is a custom sink which is not available as a library, custom code had to be written to do this.

The eventual goal for this layer is to create a base model that is able to predict patterns that occurred in the history of the different microservices and that can then be updated through the use of the real time layer. This can be seen as creating a preconfiguration model that will be utilized when starting a new microservice. This preconfiguration model will allow the real time layer to create more accurate predictions from the start. It is important to note that this layer should not be a continuous running process but rather a process that should be run after a specific period.

C.3 Visualization Layer

Finally a visualization layer is created through the use of cutting edge technologies such as react.js and redux, that introduces an easy to use interface, detailing how the orchestration infrastructure is built up. This visualization layer is able to process incoming data in realtime and visualize warnings when a microservice is about to hit its capacity.

III. Model Training

The most critical part of this architecture are the machine learning algorithms used within the different layers. These machine learning algorithms will result in a model that will decide if a cost efficient solution can be found.

For both these algorithms, the choice was made to use Linear regression to prove that both layers work as intended. Linear Regression will try to find the correlation between different datapoints and plot a trend line through these points. As the name indicates, it is a linear algorithm providing a result in the form of $y = a * x + b$.

Finding predictions through this algorithm can then easily be done by extracting the x component for a specific y value.

A. Batch Training

To train the batch layer, a linear regression algorithm that utilizes Gradient Descent. Gradient Descent will try to minimize the cost function illustrated in Algorithm 1, by using the mathematical gradient to quickly iterate and converge to the minimum of this function.

Algorithm 1 Gradient Descent for Linear Regression

```
1: repeat
2:     $\theta_0 = \theta_0 - \alpha \frac{1}{m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)})$
3:     $\theta_1 = \theta_1 - \alpha \frac{1}{m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)}) * x^{(i)}$
4: until convergence
```

This algorithm can easily be implemented using Apache Flink. Apache Flink is a big data processing framework which is able to process big data as batches or as realtime streams [2]. In our batch layer we are able to process this data as batches and apply gradient descent on it. Resulting in the lowest cost for this batch. This result is then saved towards a local file that can be used as a starting point for the real time training algorithm.

B. Real time Training

Real time training is performed by creating a consumer on the processing hub that allows us to consume incoming data in real time. This consumer will then run the same gradient descent algorithm as described in the batch training section. But with as a difference that a batch size of one datapoint is used. This is also called 'Stochastic Gradient Descent' [3].

The function of this layer is to adapt towards to incoming data in a real time fashion. This allows enterprises to quickly react to new trends happening within the different microservices.

IV. Results

After creation of the different layers, several results can be extracted. First a discussion is made about the architecture as a whole, whereafter conslusions are drawn. Thereafter a more in depth review of the different layers is made with their created models.

A. Platform Architecture

Looking at the architecture created, it can be seen that the architecture is built up out of different modules that are interchangeable. By following this design, the architecture is able to scale and adapt to the complex needs of the different enterprise use cases.

B. Machine Learning Models

Looking at the results of the different Machine Learning models as illustrated in the figures below, several conclusions can be made.

For the real time model, illustrated by Figure 1. It can be seen that when no preconfiguration is created by the use of a batch model that a specific convergence time is needed for the model to become accurate. Due to the short lived nature of microservices, this is a time that should be minimalised as much as possible.

This preconfiguration can be made by the batch layer. Illustrated by Figure 2. Here it can be seen that the algorithm tries to find a graph through the specific datapoints, resulting in a preconfigured model that can be re-used and trained upon. This algorithm also takes the historical features into account such as trend lines, anomalies, spikes and others.

When the preconfiguration and the real time model are finally combined as illustrated in Figure 3, it can be seen that the algorithm behaves as expected. The initial convergence time of the algorithm is way shorter and allows for better predictions to be made from the start.

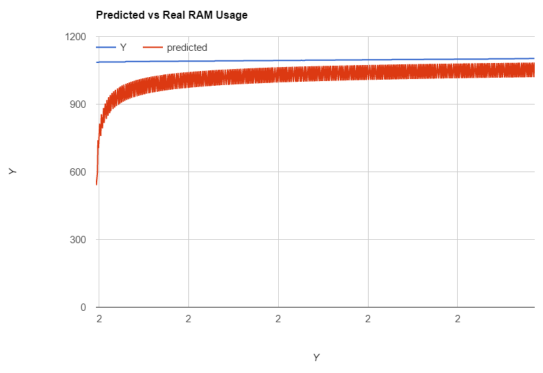

Fig. 1. Realtime Training Model

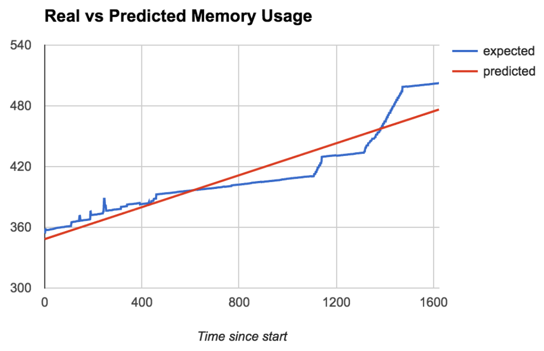

Fig. 2. Batch Training Model

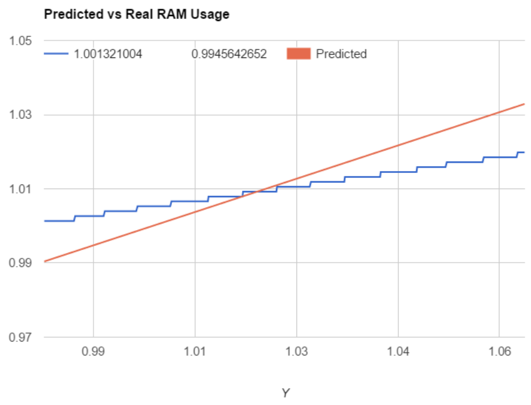

Fig. 3. Preconfigured Real Time Training Model

V. Conclusion

This thesis introduced a platform architecture that is able to predict how microservices their resources are going to change over a period of time. By creating a modular design, we introduced a platform that can achieve higher accuracies by adapting specific components within the architecture. Through this platform, enterprises are now able to scale their microservice deployments more efficiently. This results in a cost decrease by freeing up resources when they are not needed.

References

[1] Microservices, http://microservices.io/patterns/microservices.html [Online; accessed 13-December-2016]

[2] Apache Flink, https://flink.apache.org/, [Online; accessed 17-October-2016]

[3] Zhang, Tong. ”Solving large scale linear prediction problems using stochastic gradient descent algorithms.” Proceedings of the twenty-first international conference on Machine learning. ACM, 2004.