UNIVERSITÄT PADERBORN
Die Universität der Informationsgesellschaft
Faculty for Computer Science, Electrical Engineering and Mathematics
Department of Computer Science

PG-AICON: Artificial Intelligence for Computer Networks

Dynamic Storage Allocation of Virtual Network Functions

by
Indranil Ghosh, Lameya Afroze, Nayela Tasnim Labonno, Shraddha Pawar

Prof. Dr. Holger Karl
Advisor : Marvin Illian

Paderborn, March 19, 2021

1 Problem Statement

Resource usage of Virtualized Network Functions (VNF) requires study and optimization due to its significance in terms of time and monetary cost. CPU cores, memory, and storage availability play a key role in the performance of VNFs. While CPU allocation in NFV is being researched and is more and more well understood, storage as another type of resource has not been analyzed much. The significance of storage is yet increasing, particularly in orchestration frameworks, like Cloud, Kubernetes etc. first, to comprehend the storage estimate and then to assign the required storage or scale it up or down.

Although Network Function Virtualization (NFV) provides the facility of increasing or reducing the resources according to the requirement of the system, they can only be allocated statically by the client. This static allocation results in over-allocation sometimes which increases cost of the client.

To overcome this problem, the storage behavior of VNFs has been assessed in this task to determine their long-term storage demand in terms of other relevant system metrics. According to the analysis results, a dynamic storage allocation system is intended to be devised which will allocate storage to different VNFs based on their requirement. The VNFs we have considered to conduct our experiments on are as below:

- A message broker (*Apache Kafka*)
- A proxy server (*Squid*)
- A Database (*InfluxDB*)

2 Our Approach

The storage optimization task of VNFs is divided into two phases.

In the first phase, the VNFs have been analyzed with a view to drawing some correlations between the storage usage of a VNF to other environment metrics, for instance, ingestion rate, response time etc. Multiple experiments have been run to observe how the storage consumption changes over time with respect to different ingestion rates. Then, with the help of different supervised machine learning models, it is found out which set of metrics are playing an important role to determine the storage demand of the VNFs.

In the second phase, a storage allocator is devised to assign storage to the VNFs from phase 1 in a way which is optimized in terms of cost and downtime.

3 Implementation

3.1 Kubernetes and GlusterFS

The VNFs are running as pods on a distributed Kubernetes cluster of 3 Virtual Machines. The cluster comprises one master node and two worker nodes. As

containers are ephemeral, they may crash and the files will be lost. To solve this problem, we have connected a GlusterFS storage to Kubernetes to abstract the volume away from Kubernetes services. Another reason of using GlusterFS as volume provisioner is because it comes with the convenience of online volume expansion. In the second phase of our work, we intend to allocate storage to VNFs in a cost saving way, where we need to exploit the *dynamic provisioning* feature of GlusterFS. To install and manage Gluster volume, *Heketi* is used in this setup.

3.2 Persistent Volume and Persistent Volume Claim

We are also using Persistent Volume and Persistent Volume Claim features of Kubernetes to manage durable storage in our cluster. *PersistentVolume (PV)* is a piece of storage in the cluster that is dynamically provisioned using *StorageClass* for individual VNFs.

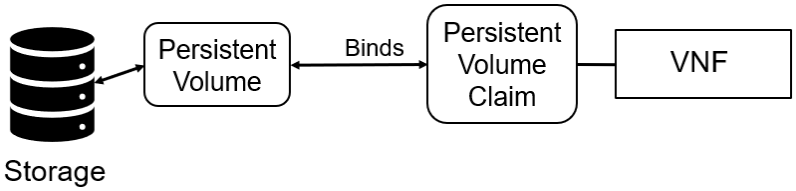

Figure 1: How Persistant Volume works

PersistentVolumeClaim is the request for and claim to some *PersistentVolume* specifying the storage amount and access mode. VNFs or pods use claims as their volumes. The cluster inspects the claim to find the bound volume and mounts that volume for the VNF data.

The gist of using PV and PVC is we want to collect storage usage metrics from the VNFs environment by sending data generated by the `Digital Twin`. A PV makes sure that the data will remain even after the pod or container is crashed or removed somehow.

Figure 1 depicts how a volume is bound to a certain pod/VNF by PVC. The PV is backed by a persistent disk on the host machine. When a pod comes up and requests a PVC, it is then bound to the PV. And on the occurence of the pod going down at some later point in time, the PVC will stay there and hold the VNF's data. When Kubernetes uploads a new pod again, it will then be able to pick up the data from the PVC.

3.3 Apache Kafka(2.12-2.2.0)

Our first VNF is a message broker named **Apache Kafka**; a distributed, data streaming system. Kafka is widely used in many large companies such as Spotify,

Uber, PayPal and Netflix, to name a few because of its durability, speed and fault tolerance. Some basic building blocks of Kafka are:

- Topic: In Kafka, topics are where records are stored and published. Producer applications write data to topics and consumer applications read from topics.
- Partitions: Kafka topics can be further divided into several partitions. If an environment contains multiple consumers, then it is possible for those consumers to read from the topic in parallel if there are multiple partitions.
- Producer: Creates a record and publishes it to some topic.
- Consumer: After a record is published to Kafka topic, consumer can request and fetch it.
- Kafka Broker: A Broker is a Kafka server that runs in a Kafka Cluster. It handles all requests from clients.

Kafka Architecture Figure 2 shows the implementation of Kafka VNF. We have used **Digital Twin** as the data generator. It is a virtual representation of a real life machine. It creates mimicked data about different states of a machine.

Another component in the Kafka architecture is the OPCUA connector. The OPCUA client fetches messages from digital twin and publishes them to Kafka topic using *kafka_producer* object.

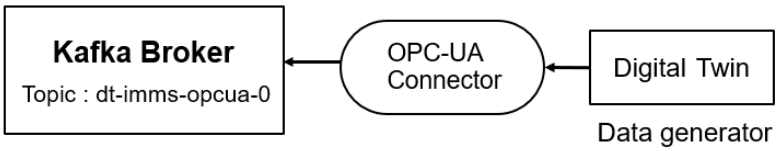

Figure 2: Apache Kafka VNF architecture

Our Kafka instance contains one topic called *dt-imms-opcua-0* and a single partition called *dt-imms-opcua-0-0* with the following configurations:

- cleanup.policy: *(delete)* This is default cleanup policy of Kafka. It will discard old segments when their retention time or size limit is reached.
- retention.bytes: *(62914560 Bytes)* This configuration defines the maximum size upto which a partition can grow before the old log segments are discarded to free up space if the `delete` retention policy is used.
- segment.bytes: *(20971520 Bytes)* Kafka topics are further divided into chunks called segments. This configuration controls the segment file size for the Kafka log.

Metrics Selection Following metrics are extracted from Kafka container and OPCUA connector to characterize storage demand of Kafka. Two separate set of metrics are merged using timestamps.

- VNF Runtime /s
- Message Ingestion Rate MB/s
- Message Size /Bytes
- Producing Time /ms
- Consuming Time /ms
- Response Time /ms
- CPU Utilization %
- Memory Consumption /MB
- Disk Free Space /GB
- Brick Usage /MB
- Total Kafka Disk Usage /MB
- Topic Usage /MB

3.4 Squid (3.5.27)

Our second VNF is **Squid Proxy Server** which is one of the most popular proxy server. It redirects user requests from clients to the server. Although squid supports several protocols e.g., FTP, HTTP, ICP etc., the experiments are run only for HTTP requests.

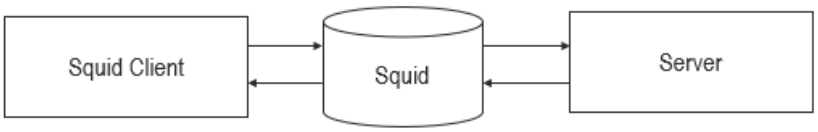

Figure 3: Squid VNF architecture

Squid Architecture The implementation of Squid VNF architecture is depicted in figure 3. We have used `squidclient` to send requests to Squid which

is a command-line tool similar as `curl` or `wget`. When a request arrives to the proxy server for the first time, it checks whether the file is available in Squid cache or not. If it is already present in the cache directory, then it will respond HIT otherwise it will save a copy of the requested file in its cache directory and return MISS as a response.

Squid Configuration All the settings of Squid proxy server are configured in the /etc/squid/squid.conf file and one can modify them according his/her system requirements. For example, the TCP port can be changed by changing `http_port` which is 3128 by default.

- Uses default TCP port and allows proxies for all client by setting `http_access allow all`.
- Uses /var/spool/squid as cache_dir and ufs as squid storage format.
- Cache directory is assigned 256 MB as disk space with 16 main directories and 256 sub-directories for each of the main directories.
- Cache memory is assigned 16 MB space.

Metrics Selection Following metrics are considered to analyze the storage demand of Squid application. These metrics are collected using two bash scripts `squid_request.sh` and `squid_metrics.sh` and merged into a .csv file based on the timestamp value.

- VNF RunTime /s
- Squid Memory Usage /KB
- CPU Utilization %
- Hit/Miss
- Ingestion Rate MB/s
- Total Squid Disk Usage /MB
- Cache Memory Usage /KB
- Cache Disk Usage /MB
- Response Time /ns

3.5 InfluxDB

InfluxDB is an open source time series platform in a single binary a multi-tenanted time series database. InfluxDB is a purpose-built to collect, store, process and visualize the metrics collection. It provides query language to take querying measurements, series and points. Database is designed in such a way that it can handle high write and query loads. It stores time series data consisting keys value pairs which called field set and timestamp. It uses the Network Time Protocol to synchronize time between hosts. If no timestamp is provided, InfluxDB uses the local server's timestamp.

InfluxDB Architecture Figure 4 shows architecture of InfluxDB. InfluxDB also uses 'Digital Twin' as the data generator. This data is collected by OPCUA connector which is acting as a server. We are using a python script that will act as a client for sending data (JSON Format) towards the third component, Telegraf.

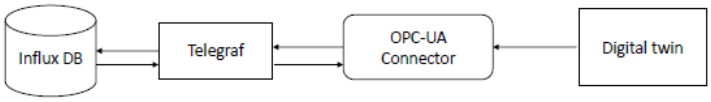

Figure 4: Influx DB VNF architecture

Telegraf is a plugin-driven agent which has input output plugins and used for collecting and writing metrics into Influx DB in batches. On the other way around, it also listens request from Influx DB and submits metrics to database. Influx DB stores data in 'shard groups', which are temporal blocks acting as a storage engine. These 'shard groups' and 'shard duration' are handled by 'retention policy' and stores data with timestamps within specific time interval called 'shard duration'. (In this setup, data is retaining for 7 days into Influx DB). Once data is being stored into Influx DB, a response arrives to OPCUA server. Each time OPCUA receives an acknowledgement, it updates performance metrics for that particular message into a .csv file.

Metrics Selection The following metrics are extracted from InfluxDB container and OPCUA connector and they are combined with timestamps to predict storage consumption.

- VNF RunTime /s
- Influx Shard Disk /Bytes
- CPU Utilization %
- Influx Write Request /s
- Ingestion Rate Bytes /s
- Influx Write Request /bytes

- Influx Write Request Duration /ns
- Influx Heap in Use /Bytes
- Response Time /ms
- Influx Disk /Bytes
- Influx Mem Stats System
- Memory Available %
- Disk Used %
- Influx Write Request Active /s
- Message Size /Bytes

4 Analysis

4.1 Storage behavior of VNFs

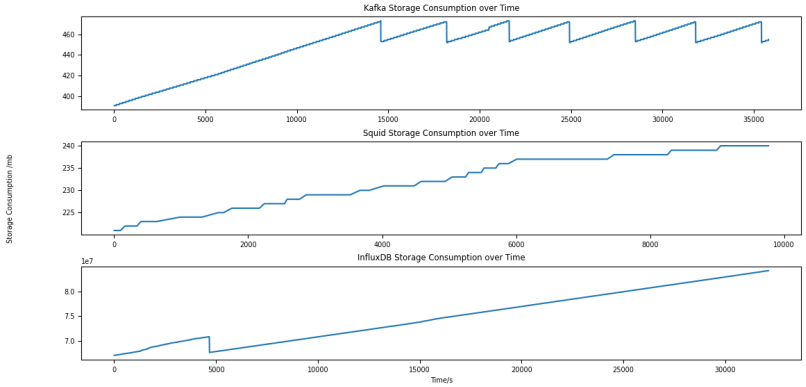

Figure 5: Storage consumption behavior of VNFs over time

Figure 5 shows the storage consumption behavior of all three VNFs over time. Kafka storage grows linearly until it reaches the `retention bytes` and then Kafka removes the oldest segment. This behavior is visible on-wards from `time=14600` where the `storage consumption=473 MB`.

Unlike Kafka, storage consumption of Squid and InfluxDB VNF gradually increases for each new request sent to the origianl server. However, there are some time intervals, where the storage consumption stays constant because of the original server going down. When the server is up again, storage consumption starts growing in a similar manner.

4.2 Confidence Interval

To measure the certainty of observed storage consumption data, 95% confidence interval is calculated for the VNFs for multiple test runs. Figure 6, 7 and 8 depicts the mean and interval range of storage consumption for all VNFs.

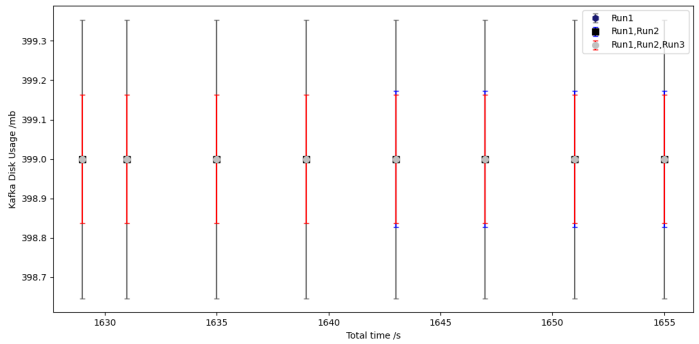

Figure 6: Confidence Interval (95%) of Kafka for three test runs

4.3 Multiple Regression Analysis

The storage behavior analysis of VNFs further comprises two steps. First, to find out which metric/metrics can realize the storage demand well. We found out that more than one predictor variable can predict the storage behavior way better than only one predictor variable. Different combinations of the VNF metrics have been tried and the ones with the most precise predictions are chosen to use in following stages for each of the VNFs.

The storage behavior of all three VNFs have been analyzed with supervised ML algorithms. In total, three regression techniques are used, such as, Linear Regression, Polynomial Linear Regression and Support Vector Regression. Our second task is to find out which regression algorithm is performing better in comparison to others. The selected algorithm and combinations of metrics will be exploited later by the *Storage allocator* to allocate storage on a live run dynamically.

Apache Kafka According to table 1, the following combinations of metrics resulted in with the lowest error:

- Memory Consumption /MB
- CPU Utilization %
- Response Time /ms
- Topic Usage /MB

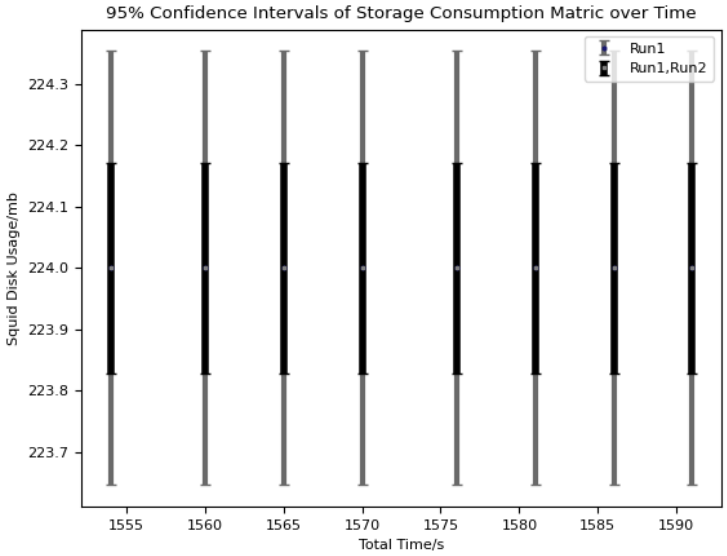

Figure 7: Confidence Interval (95%) of Squid for two test runs

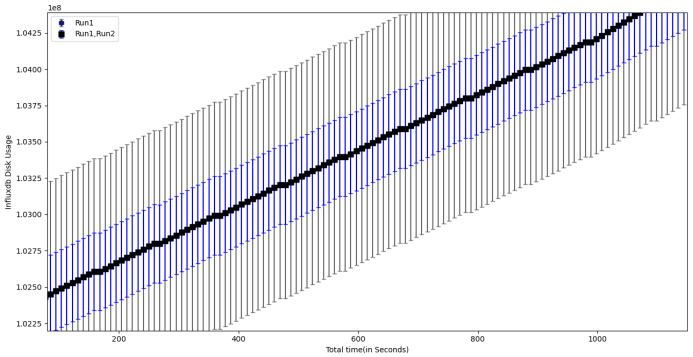

Figure 8: Confidence Interval (95%) of Influx DB for two test runs

Table 1: Predictor Metrics Selection (Kafka)

Model	CPU Utilization %, Response Time/ms, Memory Consumption/KB, Topic Usage/MB %		CPU Utilization %, Response Time/ms, Memory Consumption/KB, Topic Usage/MB, VNF Runtime/s, Ingestion Rate/s,		Memory Consumption/MB, CPU Utilization %		Topic Usage CPU Utilization %	
	RMSE	R^2	RMSE	R^2	RMSE	R^2	RMSE	R^2
Multiple Linear	0.494	0.99	0.496	0.99	19.65	0.38	0.493	0.99
Multiple Polynomial (Order=4)	1.45	0.996	7.47	0.91	17.84	0.49	.494	0.99
SVR (kernel=rbf, C=1e4,γ = 0.1)	1.84	0.99	23.86	-.001	15.03	0.60	0.750	0.997

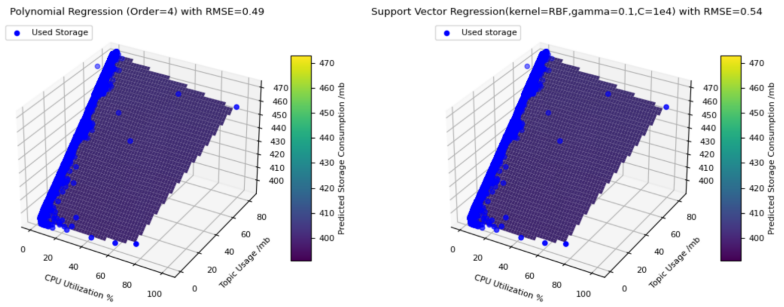

Figure 9: Used storage vs. Predicted storage of Kafka for multiple regression methods

It is also notable that among these four matrics, `Topic Usage` plays a vital role in predicting storage consumption of Kafka.

Figure 9 shows the used and predicted storage consumption of Kafka predicted with `Topic Usage` and `CPU Utilization` metrics for Multiple Polynomial (Order=4) and Multiple SVR regression methods.

Table 2: Predictor Metrics Selection (Squid)

Model	CPU Utilization, VNF Runtime/s, Ingestion Rate/s, Cache Disk Usage /MB, Memory Consumption/KB, Cache Memory/KB %		CPU Utilizatione % , VNF Runtime/s, Ingestion Rate/s, Cache Disk Usage /MB		Memory Consumption/KB, CPU Utilizatione % , Cache Disk Usage /MB		Cache Disk Usage /MB, CPU Utilizatione % , Cache Memory/KB	
	RMSE	R^2	RMSE	R^2	RMSE	R^2	RMSE	R^2
Multiple Linear	0.710	0.987	0.722	0.987	1.061	0.972	1.084	0.971
Multiple Polynomial (Order=4)	0.448	0.995	0.467	0.994	0.725	0.987	.592	0.992
SVR (kernel=rbf, C=1e4,γ = 0.1)	0.469	0.994	0.442	0.995	0.700	0.987	0.590	0.992

Squid From table 2, it is apparent that,`Cache Disk Usage`, `Memory Consumption`,`VNF Runtime` and `Ingestion Rate` are the most prominent metrics among others in predicting storage demand of Squid VNF. However, we will consider as many *predictor metrics* as possible with a considerable amount of error because it may not be possible to predict all of them accurately with `Ingestion Rate` and `Pod Runtime` in the latter analysis part. Hence, the predictor metrics for Squid are:

- Ingestion Rate MB/s
- VNF Runtime /s
- Memory Consumption /KB
- Cache Disk Usage /MB
- CPU Utilization %
- Cache Memory Usage /KB

Figure 10 shows the used and predicted storage consumption of Squid predicted with Cache Disk Usage and VNF Runtime metrics for Multiple Polynomial (Order=4) and Multiple SVR methods.

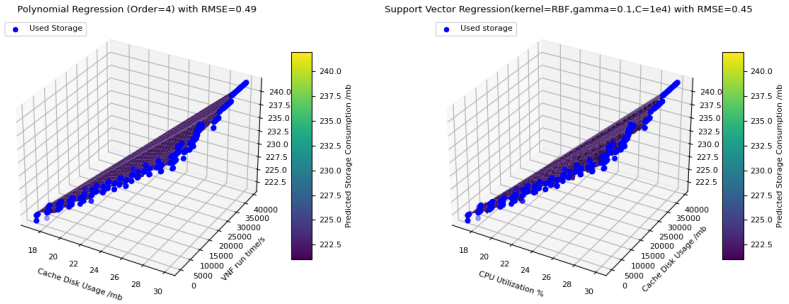

Figure 10: Used storage vs. Predicted storage of Squid for multiple regression methods

InfluxDB According to the Influx DB analysis table, it is observed that following metrics combinations provide low error rate which can be used for the further analysis of storage prediction.

- Influx Write Request /s
- Influx Write Request /MB
- Influx Write Request Duration /s
- Ingestion Rate
- VNF Runtime /s

Metrics / Regression Algorithms	Influx Write Request/S, Influx Write Request/MB, Influx Write Request Duration/S, CPU Usage, Response Time/MS	VNF Run Time/S, Ingestion rate, Influx Write request/MB, Influx Write Request Duration/S, CPU usage	Influx Write Request/MB, Influx Write Request/S, Influx Write Request Duration/S	CPU Usage, Response Time/MS, Influx Heap in Use/Bytes	VNF Run Time/S, Ingestion rate, Write Request /MB	Influx Write Request/S, Influx Write Request/MB
Multiple Linear	RMSE=8.582 R^2=0.30	RMSE=8.441 R^2= 0.32	RMSE=8.566 R^2= 0.31	RMSE=5.041 R^2=0.72	RMSE=8.442 R^2=0.33	RMSE=8.848 R^2=0.26
Multiple Polynomial (Order=3)	RMSE=9.851 R^2= 0.08	RMSE=4.361 R^2= 0.82	RMSE=5.205 R^2=0.74	RMSE=6.714 R^2=0.57	RMSE=6.022 R^2=0.66	RMSE=6.958 R^2=0.54
Multiple SVR	RMSE= 1.712 R^2=0.97	RMSE=0.858 R^2=0.99	RMSE=1.675 R^2=0.97	RMSE=4.891 R^2=0.77	RMSE=0.712 R^2=1.00	RMSE=1.688 R^2=0.97

Figure 11: Predictor metrics selection InfluxDB

4.4 Generic ML model and Cross Validation

While allocating storage in a live run for a future point of time, the selected ML model will not have knowledge of the *predictor metrics* such as `Response Time`, `CPU Utilization` etc. The storage allocator should be able to extract some parameters from the VNF environment, and then predict the other *predictor metrics* from those parameters. Finally, it can exploit all these *key* and *predictor metrics* to predict storage allocator for a future time. This situation implies the need of a generic ML model which further contains multiple *sub-models* to predict the *predictor metrics*.

Which parameters to extract from the live run? As we are optimizing storage consumption, the most significant parameters would be data/message size or amount. In this particular scenario, messages generated by `Digitaltwin` are of equal size (864 bytes). So, first parameter that will be considered is message amount in terms of ingestion rate. As the expected behavior of the dynamic storage allocator is to allocate storage for a future point of time, the other parameter that is being considered here is VNF Runtime. These two metrics will be referred as the `key metrics` in the rest of the documentation.

Generic ML model for Kafka After analysis of storage consumption of Kafka, four *predictor metrics* were selected. At this step of the analysis, it is examined how well the SVR model can predict these four *predictor metrics* by using the *key metrics*.

Table 3: Estimation accuracy of Apache Kafka predictor metrics

Model	Response Time /ms %		CPU Utilization%		Memory Consumption/MB		Topic Usage/MB	
	RMSE	R^2	RMSE	R^2	RMSE	R^2	RMSE	R^2
Multiple Linear	0.02	0.75	0.48	0.53	234.13	0.37	11.07	0.80
Multiple Polynomial (Order=4)	0.02	0.76	0.47	0.64	161.07	0.70	5.78	0.94
SVR (kernel=rbf, C=1e4,γ = 0.1)	0.09	-1.3	0.76	-.01	279.17	-1.8e^5	25.24	-0.09

According to table 3, it is possible to predict only `Topic Usage` and `Response Time` more accurately. But when applied as a predictor variable in the final generic ML model of Kafka, `Response Time` does not help much. The probable reason behind that is this metric particularly captures the response time of only one message, not the overall accumulated messages till a certain point of time. Hence, this metric is skipped as well. So, the generic ML model of Kafka further contains two sub-models in total, such as:

- Sub-model 1 (Ingestion Rate, VNF Runtime) → Topic Usage
- Sub-model 2 (VNF Runtime, Topic Usage) → Total Disk Usage

To avoid over-fitting of sub-models, they are cross validated for all the regression algorithms with a split of 80:20 for training and test data. With polynomial algorithm, the `degree` is varied from 2 to 8, and the best sub-model 1 is generated with `degree=4` with the following training and test error:

Error on training data: RMSE: 6.13, R^2 : 0.93
Error on test data: RMSE: 5.78, R^2 : 0.94

Sub-model 2 is not well predicted with Polynomial regression algorithm. For the validation of Support Vector Regression models (kernel = rbf), Grid Search algorithm is used with fold, `k=5` and the following hyper-parameter ranges:

```
C= 0.001,0.01,0.1,1,10,100,1000,10000,100000
gamma= 1e-3, 0.005,0.01,0.05, 0.1,0.5
epsilon= 0.001, 0.005, 0.01, 0.05, 0.1, 0.3, 0.5, 1,3,5
```

Both of the best SVR sub-models, which are trained to avoid over-fitting and found with Grid Search method contains the following hyper-parameters:

```
C= 1000, gamma= 0.001, epsilon=0.001
```

Table 4: Estimation accuracy of Squid predictor metrics

Model	CPU Utilization %		Cache Memory/KB		Cache Disk Usage /MB		Memory Consumption /MB	
	RMSE	R^2	RMSE	R^2	RMSE	R^2	RMSE	R^2
Multiple Linear	0.06	0.61	1174.53	0.74	2.95	0.52	8392.93	0.23
Multiple Polynomial (Order=4)	0.05	0.67	896.9	0.46	1.02	0.94	5376.41	0.68
SVR (kernel=rbf, C=1e4,γ = 0.1)	0.09	0.01	1098.47	0.32	4.47	0.09	9135.5	-0.15

Generic ML model for Squid Among the six *predictor variables* of Squid, Cache Disk can be predicted better with the *key metrics* according to table 4.

Hence, the generic ML model of Squid further comprises two sub-models, such as:

1. Sub-model 1 (Ingestion Rate, VNF Runtime) → Cache Disk

2. Sub-model 2 (VNF Runtime, Cache Disk) → Total Squid Disk Usage

Similar to Kafka sub-models, Squid sub-models have also been validated. The best model with Polynomial regression is of `degree = 4`. The best SVR sub-models contains the following hyper-parameters:

Sub-model 1: `C= 10, gamma= 0.001, epsilon=0.005`
Sub-model 2: `C= 10, gamma= 0.001, epsilon=0.001`

Generic ML model for InfluxDB After analyzing our ML models with following predictor variables with VNF Run Time and Ingestion rate we observed that Influx Write Request /MB and Influx Write Request Duration /s is providing low error rates. Therefore we have chosen two final sub models with lower error rates that are following :

	Influx Write Request/s	Influx Write Request/MB	Influx Write Request Duration/S
Multiple Linear	RMSE=0.357 R^2=1.00	RMSE=0.0004 R^2=1.00	RMSE=0.543 R^2=1.00
Multiple Polynomial (Order=3)	RMSE= 0.337 R^2=1.00	RMSE= 0.0001 R^2=1.00	RMSE= 0.201 R^2=1.00
Multiple SVR (gamma=0.0001, C=100000, epsilon=0.0001)	RMSE=3.381 R^2=1.00	RMSE=0.298 R^2=0.0	RMSE= 0.211 R^2=1.00

Figure 12: Estimation accuracy of InfluxDB

- Sub-model 1 (Ingestion Rate, VNF Runtime) → Influx Write /MB
- Sub-model 2 (Influx Write /MB) → Influx Disk Usage

Corresponding to other VNF models, Influx DB has also been Cross validated. The best model with Polynomial regression is of `degree = 3`. The best SVR sub-models holds the following hyper-parameters :

Sub-model 1 and 2: `gamma=0.0001, C=100000, epsilon=0.0001`
The training error is: RMSE: 0.000971 R^2 : 1.0
The testing error is: RMSE: 0.000975 R^2: 1.0

5 Dynamic Storage Allocation

The scheme behind analyzing the VNF's storage characteristic is to devise a dynamic storage allocator which will be optimized in terms of storage cost and downtime. Figure 13 illustrates the workflow of the storage allocator.

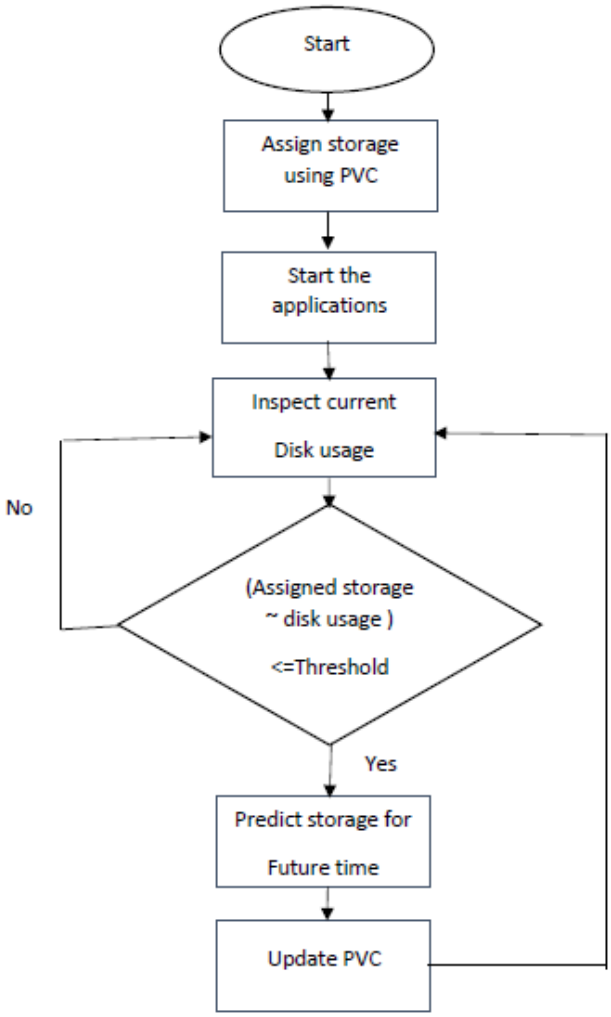

Figure 13: Work flow of storage allocator model

At first, it creates a persistent volume and assigns the lowest required storage for a certain application to execute through PVC. While creating the storage class, the `allowVolumeExpansion` field is set to `true` so that the online volume expansion of the PVC is possible. The next step of the allocator is to start the applications using API (Kubernetes Python Client). Once the application is up and running, the allocator examines the disk usage periodically and compares it with a predefined threshold.

The threshold is set in such a way so that the assigned storage is greater than the current disk usage. When the threshold is reached, the allocator predicts the storage demand for a future point of time using the generic ML model of that particular VNF. To predict the storage demand, the allocator needs two key metrics, `VNF Runtime` and `Ingestion rate` of the current environment. The allocator is provided with an `update interval`, which is the periodic inspection duration. It then calculates the `VNF Runtime` by adding the `update interval` with the `current runtime` of pod. The allocator can extract the other `key metric`, `Ingestion rate` from the environment.The generic ML models to predict storage demand, which are trained and validated beforehand on section 4.4, are loaded by the allocator to determine the storage demand from them.
After predicting storage demand, the allocator updates the current PVC and expands the volume for the VNF. The whole process is repeated periodically to make sure that the VNF is assigned with the required amount of storage for a certain point of time. Though for Influx DB we are using a `window` concept for that we are considering window size of 1 hour three times. That means each window is of 1 hour, we are taking the mean of ingestion rates for each window. As an example, the mean for window 1 is 6.6, for window 2 is 9.0 and for window 3 it is 8.2. Then at the end again we are taking the mean of these values then the result we are getting that we are considering the ingestion rate for entire set. Then multiplying with pod runtime we are getting the `total message count`.

6 Evaluation

6.1 Comparison of Different ML Models

Different combinations of the selected sub-models from section 4.4 are deployed within the storage allocator. These models are then evaluated by extracting three storage consumption data, which are:

1. Used: Actual used storage of a VNF at a particular time.
2. Predicted: The predicted storage amount at a particular time by that specific model.
3. Allocated: The amount of storage assigned inside a PVC at a specific time.

Kafka: Figure 14 depicts the performance of kafka Storage allocated for four different ML models where each of the model comprises two different sub models. Although each of these four models uses same input metrics `VNF Runtime` and `Ingestion Rate` as sub-model 1, different regressor methods were used to build them. Model 1, on the top-left corner uses Polynomial regressor (order=4) to predict `Topic Usage` from the input metrics. Then it uses sub-model 2 which applies SVR regressor on `Topic Usage` to predict `Total Disk Usage`. On the other hand, Model 2 uses SVR regressor in sub-model 1 which improves the prediction result compared to Model 1. With Model-2, a volatile storage prediction is observed which happens due to the fluctuation of `Ingestion rate` MB/s. To improve that situation, in case of Model-3 and Model-4, an average `Ingestion Rate` MB/min is considered.

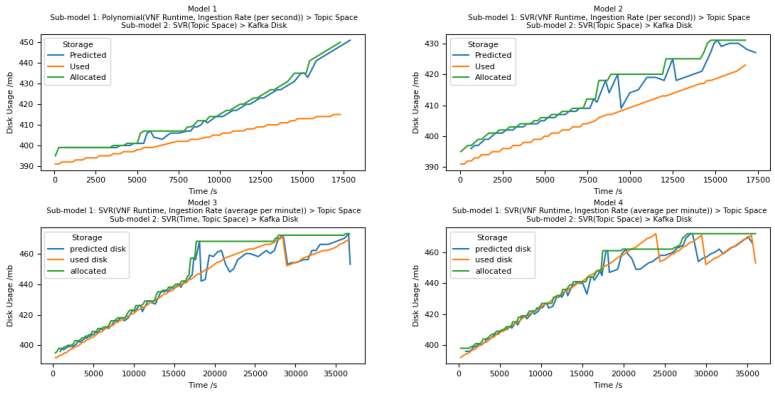

Figure 14: Performance Comparison of Storage Allocator Models (Kafka-VNF)

Among these four different models, Model 3 shows better performance which uses `VNF Runtime` and `Topic Usage` as input metrics to predict `Total Disk Usage` in Sub-model 2 with SVR algorithm. Table 5 contains the Root

Table 5: Storage Allocator Models Evaluation (Kafka-VNF)

Model 1		Model 2		Model3		Model4	
RMSE	R^2	RMSE	R^2	RMSE	R^2	RMSE	R^2
14.143	-2.67	6.14	0.050	5.27	0.944	4.92	0.96

Mean Square and R^2 error for Kafka models.While interpreting the error values to select a best storage allocator model, we need to consider one with lower RMSE error value and a R^2 error value closer to 1. The reason behind choosing a model with such error values are two folds; While a lower RMSE value indicates a smaller average deviation of the predicted disk usage from the actual used usage, a higher R^2 value indicates a better fit for the model. So, if a model has an R^2 of 0.60, it means 60% of the data fit the regression model. Based

on these interpretations, Among these four models, Model 4 is the best model both in terms of RMSE and R^2 error as it comprises lowest RMSE error with highest R^2 error.

Squid Storage allocator of Squid is experimented with four different models and figure 15 shows their performances. The `Ingestion Rate` for all of the test runs is in terms of per second and all four models contain the similar Sub-model 1 which predicts `Cache Disk Usage` from `VNF Runtime` and `Ingestion Rate` metrics. Model 1 on the top-left corner contains two SVR sub-models where Sub-model 2 predicts Total Disk Usage of Squid VNF from the predicted `Cache Disk Usage` of Sub-model 1. Instead of using a second SVR sub-model, Model 2 adds the required binary files for Squid application with the predicted `Cache Disk Usage`.

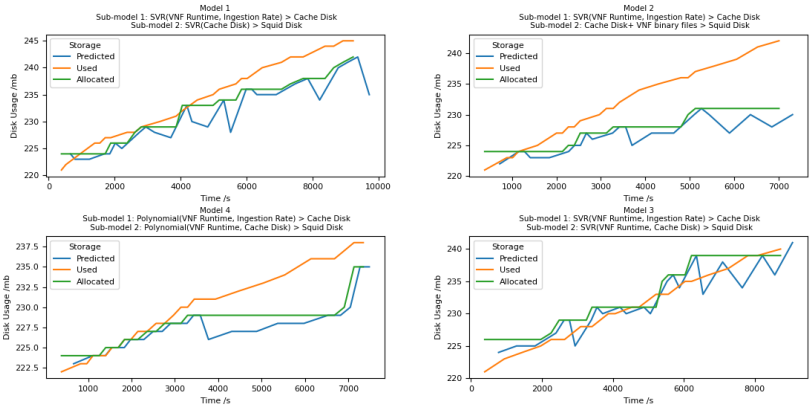

Figure 15: Performance Comparison of Storage Allocator Models (Squid-VNF)

Model 3 and Model 4 contain (in the bottom) the same predictor metrics and outputs for their sub-models but with different regression methods. Sub-model 2 of both these models uses two predictor variables `VNF Runtime` and `Cache Disk Usage` to predicti `Total Disk Usage` of Squid VNF. Table 6

Table 6: Storage Allocator Models Evaluation (Squid-VNF)

Model 1		Model 2		Model3		Model4	
RMSE	R^2	RMSE	R^2	RMSE	R^2	RMSE	R^2
6.92	-0.37	4.22	0.64	3.77	0.36	2.70	0.80

contains the Root Mean Square and R2 error for each of the models discussed above. Among these 4 models, Model 3 is the best as it has the lowest RMSE and highest R2 error.

InfluxDB Influx DB allocator has compared with two Models. Figure 16 shows Sub-model 1 SVR Regressor-1 (Ingestion Rate, VNF RunTime)-Influx Write Request/MB Sub-model 2 Regressor-2 (Influx Write Request/MB)-Influx Disk Usage

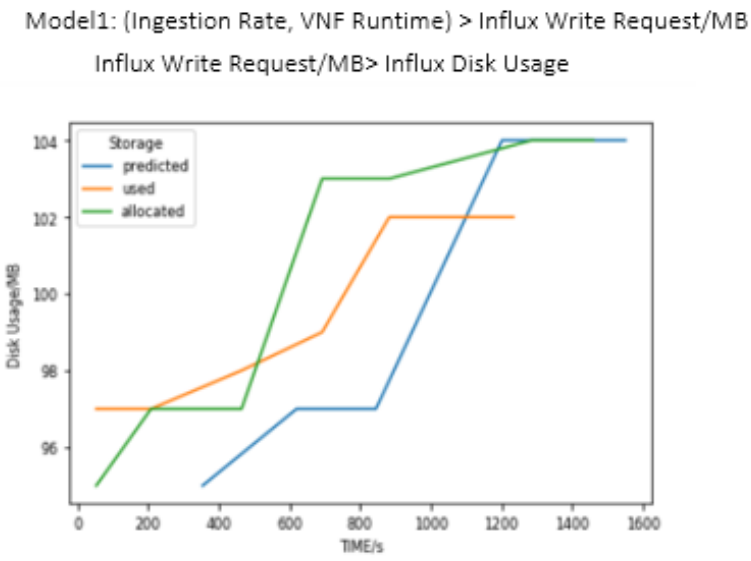

Figure 16: Performance Comparison of Storage Allocator Models (InfluxDB-VNF)

This figure contains ingestion rate per second and predictor variables predicting 1 or 2 MB less than what is the actual usage is and after sometime its being constant at the end time its predicting more 2 or 3 MB than actual usage is.Comparatively this model is better in prediction than the other model.

The figure 17 shows Sub-model 1 SVR Regressor-1 (Ingestion Rate, VNF RunTime)-Influx Write Duration/s Sub-model 2 Regressor-2 (Influx Write Request Duration/s)-Influx Disk Usage. where in this model's prediction is being very less than what is the actual usage is though slowly its increasing but still not near to what is the actual usage is.

6.2 Confidence Interval of Predictions

This section shows and explains the confidence intervals of multiple independent experiments of the selected models for different VNFs. The intervals explain the likelihood of the model to predict with accuracy.

Kafka Figure 18 shows the confidence interval of Model 3 for Kafka-VNF for 4 independent experiments. Because of more stabilized `Ingestion Rate` the

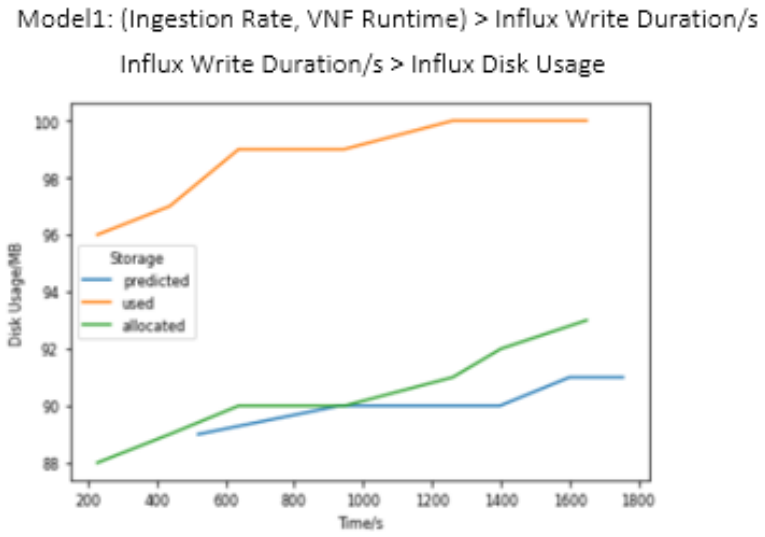

Figure 17: Performance Comparison of Storage Allocator Models (Influx DB-VNF)

prediction values does not differ much with respect to time.

Squid The confidence interval of Squid-VNF is also drawn for the selected model, Model-4 for 4 independent experiments which is illustrated in figure 19. Due to a fluctuating per second ingestion rate unlike Kafka, the prediction values differ for a certain point of time in case of Squid VNF. Hence, the confidence interval is not always decreasing for the latter experiments.

InfluxDB Figure 20 shows the confidence interval for InfluxDB VNF for 2 independent experiments. With a different `Ingestion Rate` the prediction values are varying over time.

6.2.1 Storage Allocator Performance within Service Function Chain

Figure 21 shows the implementation of VNFs within a Service Chain Function (SFC). Unlike previous architecture, Digital Twin sends continuous messages which is stored in Kafka topic through OPCUA connector. On the other hand, with the help of `Kafka Python API`, `HTTP Server` stores the messages in html file format from kafka topic. These requests are being sent to squid during the experiment through `squidclient`.

Figure 22 compares the storage consumption behavior of individual VNFs and when they are implemented within SFC.

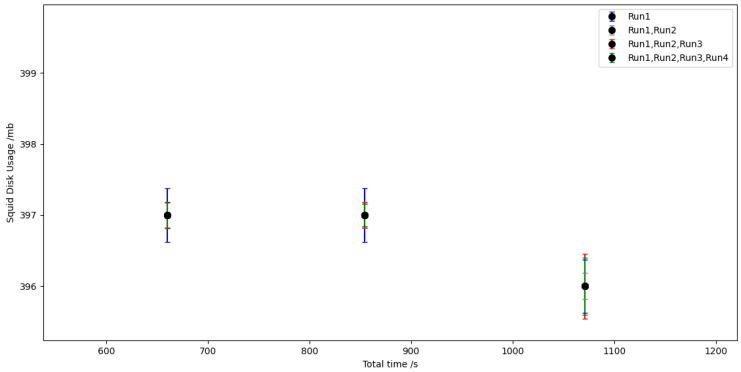

Figure 18: Prediction Intervals of Kafka-VNF

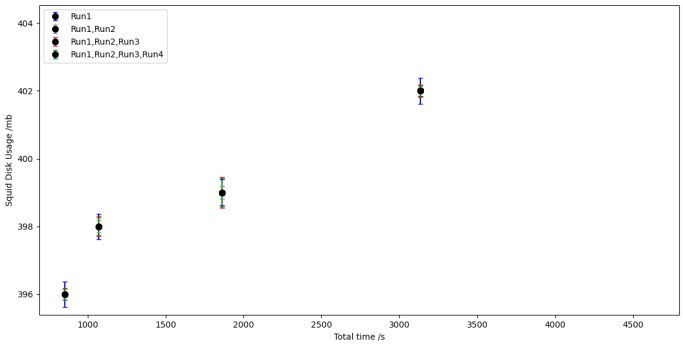

Figure 19: Prediction Intervals of Squid-VNF

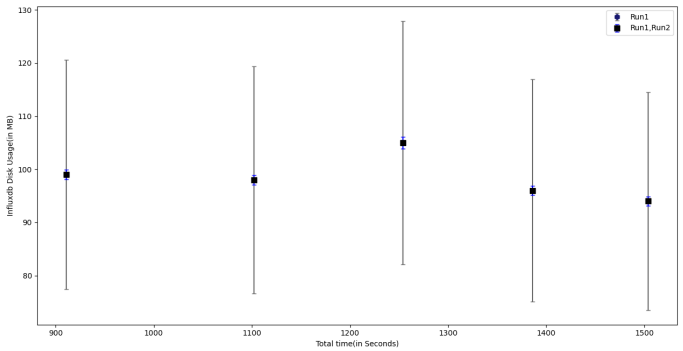

Figure 20: Prediction Intervals of Influx DB-VNF

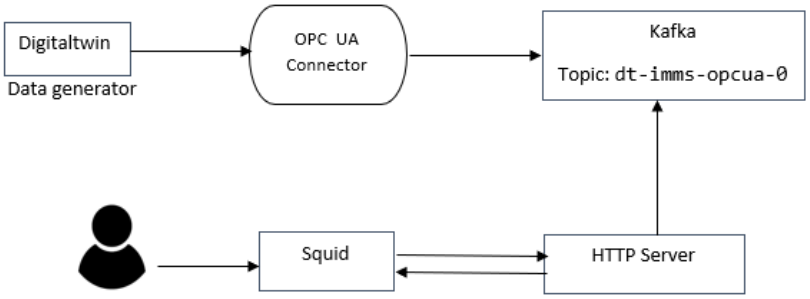

Figure 21: Service Function Chain Architecture

Top left plot shows the behavior of individual Kafka VNF, where the allocator successfully allocates storage for future point of time based on storage demand. But when it is implemented in SFC (top right) the prediction storage consumption shows volatility (almost 4 MB) as the ingestion rate fluctuates.

The two plots in the bottom show the storage consumption behavior of Squid VNF. After implementing squid on SFC, the ingestion rate of squid increased from 2 MB/s to 3 MB/s which increased the overall storage consumption compared to prediction result. Moreover, as the Squid VNF was trained for storage consumption upto 242 MB, once it exceeds that point, the storage allocator fails to predict the storage demand.

From Squid's best model (bottom-left), the highest deviation of the predicted storage was 5 MB. This amount is assigned as the headroom of Squid while allocating storage in SFC to make sure that the VNF does not run out of storage.

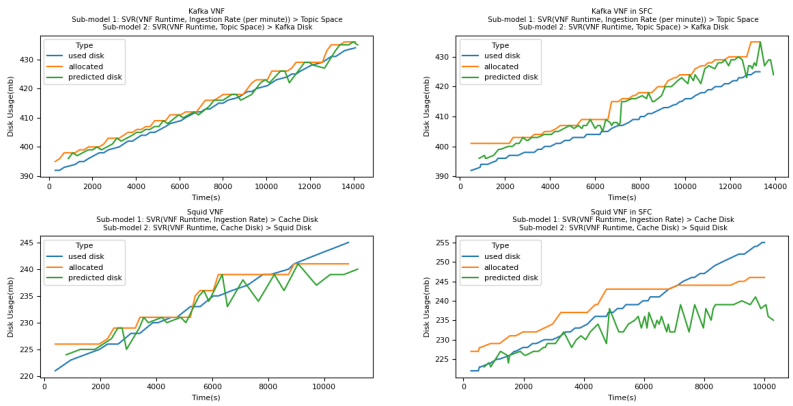

Figure 22: Performance Comparison of Storage allocator Model between individual VNFs and implemented within SFC

7 Pitfalls

7.1 Kafka Listeners and Advertised Listeners

While implementing Kafka VNF, there was a situation where the OPCUA connector, which was running on a separate Kubernetes node failed to access Kafka. Although the connector was able to connect from the same node to the Kafka server. We found out that the connection problem was raised from a very interesting fact about Kafka. Kafka needs `listeners` and `advertised.listeners` to communicate with its clients. `Listeners` are what interfaces Kafka binds to, `Advertised_Listeners` are how clients can connect to Kafka.

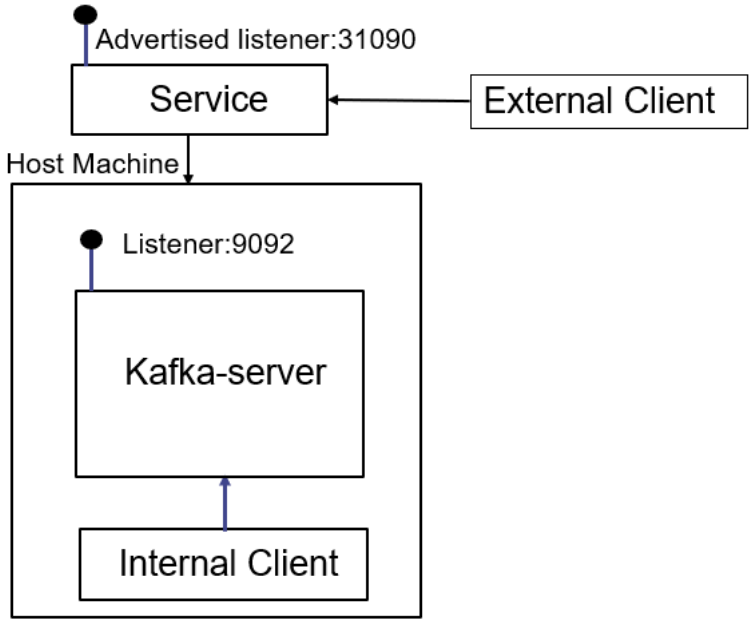

Figure 23: Listeners and Advertised Listeners in Kafka

When everything is implemented locally, Kafka sets up both of the `listeners` and `advertised.listeners` as `localhost` and to its default port `9092`. But, unlike that in a distributed environment, there are external clients which need to access Kafka server. For those clients to be able to connect with Kafka, a different listener is required containing Kubernetes service's `external IP` and a different external port, which is `31090` in this scenario. This situation is depicted in figure 23.

7.2 Storage Metric Selection

We have considered three different ways to collect the storage consumption metric of the VNFs. They are as follows:

1. **GlusterFS free space** : We tried to find out the free space of GlusterFS node with `gluster volume status` command but it turned out later that it is not possible to capture fine grained storage usage with the Gluster command. We need the GlusterFS usage value atleast in MegaByte format to be able to analyze the storage behavior pattern.

2. **Monitoring GlusterFS brick usage**: Next we tried collecting the disk usage of different files in a GlusterFS volume brick, which was a bit more

fine grained but the change in the brick usage was very random. So, we could not use that either.

3. **Disk usage of VNF from container**: Finally, we used the disk usage of the individual VNF container with `du -sh /` command to capture the storage usage. It captures the size of incoming messages/data of the VNF as well as all the required system files of the VNF and the storage usage is fine grained enough to analyze later.

7.3 Kubernetes Volume (GlusterFS/hostPath)

Kubernetes supports several volume plug-ins. We considered two of them and decided to proceed with GlusterFS for the following reasons:

- Our storage allocator needs to be able to dynamically allocate volume based on demand of the VNFs. But `HostPath` does not support online volume expansion. As a result, with HostPath as volume, it requires a considerable amount of downtime to recreate the PV and PVC and to assign the VNF to the new PVC again.
- GlusterFS: Starting from 1.8 version, Kubernetes allows dynamic resizing of the volumes. GlusterFS is one of such volume plugins. However, it is possible to only expand the mounted volume with GlusterFS. Online volume shrinking is not yet supported by Kubernetes.

8 Future Work

8.1 Volume Shrinkage

While allocating storage to VNFs, it may happen that a VNF requires a lower amount of storage than before. But as Kubernetes does not support the volume shrinkage feature yet, it can be integrated with the current storage allocator by following a rather manual strategy. The steps to shrink volume are as below:

1. Take a snapshot of the current `PersistentVolumeClaim` (PVC) that a VNF is assigned to.
2. Create a new PVC with the new storage amount (lower than before) and clone the snapshot of the old PVC to the new PVC.Delete the VNF and the associated PVC (old).
3. Create the VNF again remounted to the new PVC with a reduced storage.

8.2 Train Individual SFC Model

The storage allocator model for individual VNFs failed to give better prediction when the VNFs were implemented in SFC. One reason behind the unexpected

storage behavior of VNFs might be when two different VNFs are implemented within a SFC, it affects their ingestion rates. This situation can be improved by training both of the models for SFC implementation in later phases. Also, the existing models are hoped to perform better when trained with different ingestion rates.