Department of Computer Science
Computer Networks Research Group

Investigation of MANO scalability challenges

Management of ServiCes Across MultipLE clouds

Authors:

Ashwin Prasad Shivarpatna Venkatesh
Bhargavi Mohan
Deeksha Mysore Ramesh

Supervisors:

Prof. Dr. Holger Karl | Sevil Dräxler | Hadi Razzaghi Kouchaksaraei

Paderborn, May 3, 2019

Contents

List of Figures

1 Introduction

Scalability in the recent times has become one of the most important factors of the cloud environment. In this paper we discuss scalability, provide an insight about the effects of scaling and investigate some scaling approaches that could be incorporated to scale NFV management and network orchestration (MANO) system.

1.1 Definition of scaling

'Scalability' is defined in different ways in various academic work. Some of the definitions are listed below.

- "The ability of a particular system to fit a problem as the scope of that problem increases (number of elements or objects, growing volumes of work and/or being susceptible to enlargement)." [**?**]
- "Scalability of service is a desirable property of a service which provides an ability to handle growing amounts of service loads without suffering significant degradation in relevant quality attributes. The scalability enhanced by scalability assuring schemes such as adding various resources should be proportional to the cost to apply the schemes." [**?**]
- "Scalability is the ability of an application to be scaled up to meet demand through replication and distribution of requests across a pool or farm of servers." [**?**]
- "A system is said to be scalable if it can handle the addition of users and resources without suffering a noticeable loss of performance or increase in administrative complexity" [**?**]

1.2 Why does a MANO need scaling?

In recent years, distributed systems have gained an increase in the number of users and resources. Scaling such a system is an important aspect when large user requests have to be served without compromising system performance or increase in administrative complexity. In terms of MANO, when there are a large number Network Service(NS) instantiation of various network functions, they need to be instantiated considering all the relevant metrics of the system.

System load: In a distributed system, the system load is the large amount of data that is to be managed by network services increasing the total number of requests for service. The load on a MANO can be defined in terms of it's load on NFV Orchestrator (NFVO) to process large number of tasks like on-boarding, instantiation and monitoring of VNFs. The NFVO of a MANO receives monitoring information which also increases the load on NFVO triggering it to scale the network service across multiple MANOs in a distributed system [**?**].

1.3 Metrics to assess a scalable system

In this section, a few metrics that are important in terms of a MANO server are introduced.

- **Speedup** Speedup measures how the rate of doing work increases with the number of processors, compared to one processor, and has an ideal linear speedup value. [**?**]
- **Response time**: Service response time of a MANO is a time period from when a service invocation message is arrived to a MANO on the provider side to when a response for the invocation is returned to the service consumer.
- **Throughput**: It is a metric which measures the efficiency of a MANO to handle service invocations within a given time.
- **Cost**: High scalability under high service loads is an expensive affair. There is always some additional cost involved in planning a scalability strategy.
- **Performance**: MANO should be able to handle the growing amount of service loads. Scalability should take into account MANOs' ability to manage high service loads without deteriorating Qos.
- **Fault tolerance**: This refers to the ability of a MANO to continue operating without interruption when it's components fail

Lifecycle Management & service provisioning: To provision a network service, the NFVO of a MANO's functionality include instantiation, global resource management scaling in/out , event correlation and termination of services. These functionalities form the lifecycle of NS. With the increase in instantiation of NS over a distributed network, the lifecycle management of each service is a overhead, hence increasing the provision time. This can be better handled when the MANO can be scaled out. To manage services with a closer proximity of geographical region, MANOs could be scaled in.

2

Scalability techniques

This chapter discusses three types of scalability techniques that could be adopted in any of the scalability approaches which are discussed in the chapter 3.

2.1 Service replication

A technique to clone services running on other nodes to stabilize the service load among different nodes without causing any damage to the ongoing operations. Services that are replicated secure additional resources provided by the new nodes for handling larger service load. In other words, service replication enhances service scalability and reduces the risk of QoS degradation by handling larger service loads. In a case study, Falatah and Omar [**?**], performed an analysis by varying the system load. Firstly, a variable for service load is set. The service load is a number of service invocations within a unit time. For the case study, the unit time was set to be 500ms. That is, if ten invocations occur within 500ms, then the service load is 10. To show an effectiveness of service replication, they simulate the service replication scheme for seventeen different volumes of service load. On each service load, conventional service system is compared with service replication strategy in terms of average response time.

Service replication is easier when the server is stateless, but the MANO will have a database, user session and various managed services. Simply replicating servers cannot be the solution as multiple MANOs using a single database will lead to a bottleneck. Hence, database clustering or database replication is also needed to maintain uniformity across the databases. Service replication increases availability and parallelism.

2.2 Service migration

Service migration is a strategy of placing a service on a different node when a particular node cannot provide high QoS due to a hardware/software problem or due to the physical distance between consumers and providers. After service migration, the migrated service performs the same role that it was supposed to performed on the unstable node and the unstable node is removed from the list of service nodes. The removal of this unstable node reduces overall QoS degradation. Lee and Kim [**?**] conducted a simulation where the service was migrated to a node that is located closer to consumer. There is also an assumption made that the response time is directly proportional to the distance. Hence, a service is migrated to the fastest node in terms of response time.

In terms of MANO, it manages VIMs in different locations. There is a close association between MANOs and VIMs to handle the lifecycle management of an NS. Mere migration of a MANO server closer to the operator will not make the communication time between MANO and VIM faster. Instead, migrating the instance of MANO closer to the VIMs will improve the overall NS lifecycle management time.

2.3 Service system scaling - dynamic node scaling

To scale a distributed system with nodes spread over different locations, management components which help in monitoring the status of all these nodes are needed. Lee and Kim [**?**] propose two key components to manage service scalability, Global Scalability Manager (GSM) and Regional Scalability Manager (RSM).

The key role of GSM is to manage service scalability. It balances service load in the system by obtaining the current status of nodes that are listed in the service system and designs a scalability strategy.

RSM component is installed on all the service nodes. It observes the status of its node and communicates it to the GSM. RSM also executes the scalability strategy based on instructions from the GSM. These two components assist the service system to add or remove new nodes dynamically at run-time.

Service scalability assuring process (Figure 2.1)

1. Define metrics for scalability measure. This metric are used to decide the raw data to be collected from services and to compute scalability in further steps.

2. Certain techniques from QoS monitored services are used to collect the set of raw data items [**?**] [**?**].

3. Analyze scalability metrics. If the metrics indicate an acceptable scalability level then continue step 2 and 3 where if the metrics indicate a need to repair the below averaged scalability then execute the following steps.

4. Develop a remedy plan for improving the below averaged scalability considering the current status of monitored service. Scalability assuring strategies like service replication and/or service migration could be adopted depending on the complexity and nature of the suffered service.

5. The selected scalability strategy is executed and this is quite often an automated process.

6. Inspect the result of the strategy and understand from the entire procedure as to how to improve the scalability. If the outcome of the process is valuable, then both the consequence and the remedy plan are logged for future uses thus making it a smart scalability framework.

Similarly In MANO's context, the metrics defined in section 1.3 could be the basis for deciding the scalability strategy. GSM and RSM can be developed as part of the MANO's ecosystem. The scalability strategy can be a hybrid of the approaches discussed in this chapter. GSM can also be responsible for state management of the MANO instances, that is, when scaling down a MANO, the metadata and network services managed by this MANO instance can be transferred to the next best node with sufficient resources.

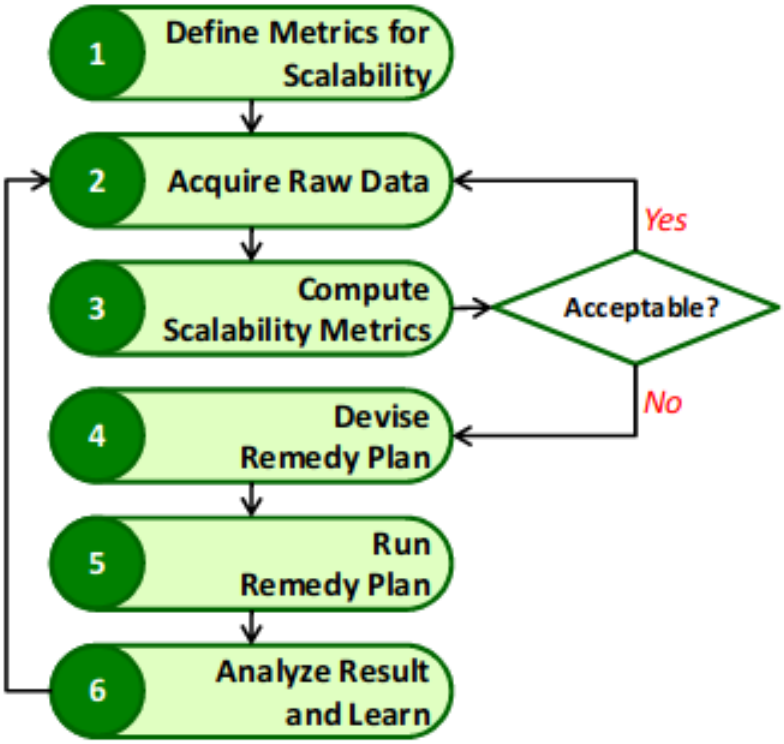

Figure 2.1: Service scalability Assuring Process from [**?**]

3

Scalability approaches

Some of the scaling approaches from various academic work are discussed in this chapter. In each section a brief introduction about each scalability approach and the relavance of a specific approach to our context of research (MANO scalability) is discussed.

3.1 Proactive scaling

Proactive scaling also known as scheduled scaling is mostly done in a cloud by scaling at predictable, fixed intervals or when big traffic stream is anticipated. A well-designed proactive scaling system enables providers to schedule capacity alterations based on a plan. With scheduled scaling, one can set when to increase the capacity or number of servers and when to decrease them. To implement proactive scaling, providers should understand the expected traffic flow, which means the providers should have some kind of statistics which indicates the desired traffic and deviation from expected traffic [**?**][**?**]. This type of scaling is suitable for servers that will have increased load during known days.

In MANOs' terms, the MANO should be able to use these statistics and decide a scaling action. At specified times, it should scale up with the values for normal, minimum or maximum traffic surges.

3.2 Reactive scaling

A reactive scaling strategy also known as auto-scaling adjusts its capacity by scaling up or down dynamically based on system metrics. Cloud providers require periodic acquisition of performance metrics for maintaining QoS. In addition, reactive scaling enables a provider to react quickly to unexpected demand. The crudest form of reactive scaling is utilization-based, that is, when CPU, RAM or some other resource reaches a certain level of utilization, the provider adds more of that resource to the environment [**?**][**?**]. This type of scaling is suitable for servers that will have increased load during a few unpredictable days.

A "Scalability Manager" can be developed that could be installed in each instance of the MANO. This will help make scalability decisions based on dynamically collected metrics. It is the responsibility of the scalability manager to scale MANOs and redirect the requests to other instances. This can be done when the NS instances that are assigned to a particular MANO reaches the threshold.

3.3 Predictive scaling

This type of scaling uses machine learning to predict the traffic stream of a server/application beforehand so that the capacity changes can be done accordingly, It collects data from all the VM instances and various other data points and uses well trained machine learning models to actually predict the flow of traffic. This model would make use of one day's data and then the data is re-evaluated every 24 hours. This type of scaling strategy will be useful where servers are affected with cyclic periodic loads.

In terms of MANO, this type of scaling can use existing monitoring data from MANO instances and predict the load on the server with the help of a machine learning model.

3.4 Hierarchical service placement

Service placement in a centralized network model lacks scalability. The service orchestrator in the centralized network model has a detailed view of all nodes/servers. A centralized placement algorithm maintaining all the information of all users and nodes is not a feasible approach. Maini et al. [**?**] investigates a hierarchical solution where the overall orchestration domain is split into geographical sub-domains.

In this model, the authors refers to a node as an Execution Zone (EZ - Services will be deployed in datacenters/clouds). The high-level orchestrator has limited visibility of EZ and user demands within a sub-domain - it sees only the aggregate of user demands and the aggregate of EZ capacities within a particular sub-domain. The high-level orchestrator places service instances at the coarse granularity of sub-domain only. Subsequent sub-domain orchestrators undertake a further placement algorithm with the scope of that sub-domain to determine in which specific EZs what QoS instances should be placed to meet the specific demand pattern of user requests within that sub-domain [**?**].

There are many ways of sub-dividing an overall orchestration domain into sub-domains. One option is to map sub-domains onto the same geographical area covered by resolution domains: the entity responsible for resolving user requests to EZs with available session slots. Another option is to consider multiple hierarchical levels of service orchestration and placement. The author models two levels of analysis.

4

Hierarchical orchestration

The orchestration of a network service in a MANO is one of the responsibilities of NFVO. The VNFM along with NFVO are responsible for managing the VNF instances and its lifecyle. To achieve end-to-end provisioning, a single MANO platform is not feasible in most of the cases. A multi-domain NFVI that allows interaction of multiple administrative domains at different levels is required. To satisfy the requirements of a multi-domain, multi-vendor, multi-technology interoperability environments, the orchestration of network services can be done in a hierarchical fashion or in a peer-to-peer fashion. [**?**]

In a peer-to-peer orchestrating model, the MANOs are interconnected to each other in an arbitrary fashion to provide end-to-end network services. This model is preferred when there is no cross-domain control or cross-domain visibility is required, thus limiting the scaling possibilities of a MANO. [**?**]

In a hierarchical orchestrating model, one of the MANO acts as a global orchestrator to the underlying MANO with a parent/child hierarchy. This architecture uses a common API between the parent MANO and child MANOs, hence maintaining a good overview of the global system. Each hierarchical level deals with a minimum of one orchestrator. The orchestrator at any given level is managed by the higher level orchestrator. Some of the factors influencing hierarchical orchestration are the geographical spread and placement of MANOs across regions, scalability constraints of peer-to-peer architectures, different administrative requirements from the operators with different layers of abstraction. This architecture is preferred because of its broader scope of network services from the lower MANOs and also for better abstraction. The number of levels of hierarchy in this type of orchestration is based on the geographical region and network domain. The hierarchical levels are also based on abstraction required for the provisioning of network services [**?**].

The overall orchestration of the hierarchical model is split based on its geographical domains known as execution zones (EZ). The splitting of EZ further is based on the geographical area the domain covers. The grouping of few data centers which are geographically close for one execution zone. The number of data centers in a region is proportional to the number of execution zones required for the region. Supposedly, here, the whole world is split into 11 execution zones based on their geographical regions like, Western Europe (EUW), Eastern Europe (EUE), Central Asia (ASC), Southern Asia (ASS), Pacific Asia (ASP), Africa (AFR), Northern North America

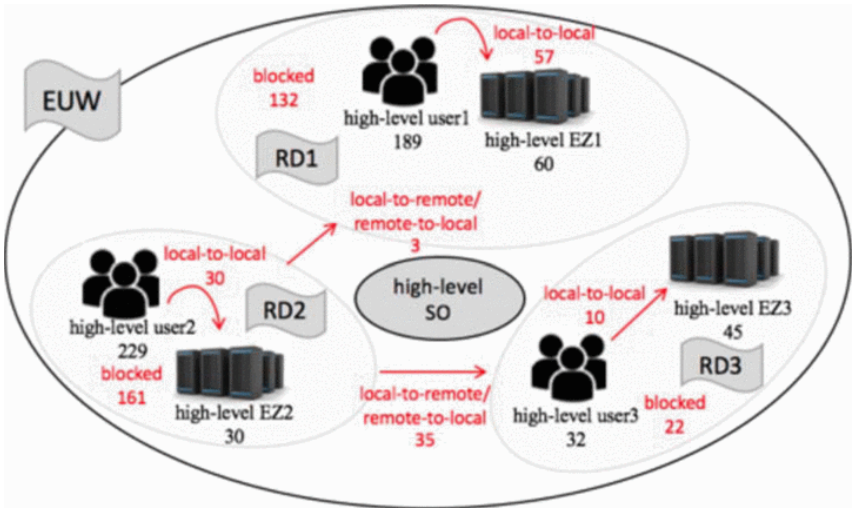

Figure 4.1: An example of the placement decision made by the high-level orchestrator [**?**]

(NAN), Southern North America (NAS), Eastern South America (SAE), Western South America (SAW) and Oceania (OCE). Considering one of the regions, like Western Europe (EUW), it is further split into lower levels based on the resolution domains (RD). The lower levels can further be split based on the granularity of orchestration required[**?**].

Each RDs constitutes number of EZs located at a specific location for users. Here, as an example, the first level(level 0) split is based on 3 resolution domains, RD1, RD2, RD3. With a limited visibility of EZs for the high level orchestrator makes a placement decision. The RDs are divided based on local-local requests of a EZ, remote-local requests of EZ, and local-remote requests of EZ. The next level of split is influenced by the scalable property of orchestrators in first level split [**?**].

Thus, the above investigation of MANO scalability can infer the conclusions about optimal number of MANOs in such a system and also the optimal number of hierarchical levels. Considering the above example, the MANOs are initially placed in 3 RDs and that forms the first level of hierarchy. The number of MANOs and the level of hierarchy can increase further depending on the user requests in that location. When there is a demand for more number of MANOs to complete the service requests. The existing MANOs are designed to scale more MANOs using service replication/service migration techniques in a hierarchical fashion.

In the further chapters, we discuss about the SCrAMbLE plug-in architecture. This plug-in could be installed in SONATA/OSM MANO to make use of the services that is developed by Translator, Splitter and Adaptor. For the ease of plug-in development, we choose the top most MANO to be a SONATA(phishahang) MANO. Considering the same example of EUW region, 3 of the SONATA MANOs could be placed in 3 different RDs intially. These MANOs depending on the service requests will call OSM MANOs using the Adaptor services. When there is a situation when all the 3 MANOs reach a threshold where in they can not serve any more requests, additional MANOs are installed with a load balancer to further scale MANO using service replication techniques. The scaled(child) MANOs are also installed with SCrAMbLE plug-in and the same services can be made use to serve more requests. The scaling of any MANOs is handled by a component called scalability manager. This could be a part of MANO. We do not discuss the implementation details of the scalability manager at this point as it is still a work in progress.

5

Effects of scaling

Scaling affects the system properties in many ways, this chapter discusses some of the effects of scaling.

5.1 Availability

Availability describes how often a service can be used over a defined period of time. Scalability approaches such as service replication increases the availability of a system.

How to estimate the availability of a system? Most service outages are the result of misbehaving equipment. These outages can be prolonged by misdiagnosis of the problem and other mistakes in responding to the outage in question. Determining expected availability as stated in [**?**] involves two variables:

1. The likelihood that one will encounter a failure in the system during the measurement period.
2. How much downtime is expected in the event the system fails.The mathematical formulation of the availability of a component is:

$$a = (p - (c * d))/p \tag{5.1}$$

where a = expected availability
c = the % of likelihood that there is a server loss in a given period
d = expected downtime from the loss of the server
p = the measurement period

5.2 Reliability

Reliability is often related to availability, but it's a slightly different concept. Specifically,reliability refers to how well one can trust a system to protect data integrity and execute its transactions [**?**].

The cloud presents a few issues outside the scope of the application code that can impact a system's reliability. Within the cloud, the most significant of these issues is how persistent data is managed. In particular, any time one loses a server, loss or corruption of data becomes a concern.

5.3 Heterogeneity

Heterogeneity refers to the state of being diverse. The scaling in a distributed system is also affected by the heterogeneity of systems involved. The administrative dimension of the scaling constitutes to the problem regarding heterogeneity focusing of both hardware and also software required, to deliver the services efficiently. One of the solutions to such a problem is coherence. In a coherence system, the different administrative systems have a common interface [**?**].

Administration in a MANO framework: The administrative domain in an NFV architectural framework is majorly divided into Infrastructure domain and Tenant domain. Infrastructure domains are defined based on the criteria like type of resource such as networking, compute and storage in traditional data center environments, by geographical locations or by organization The tenant domains are defined based on the criteria like by the type of network service, etc. In a MANO, multiple infrastructure domains may co-exist, providing infrastructure to a single or multiple tenant domain. The VNFs and Network Services reside in the tenant domain which consumes resources from one or more infrastructure domains [**?**].

Multi-MANO interworking: To achieve a better provisioning of network services in a multiple MANO system, two or more Service Platforms (SPs) cooperate or one orchestrator leverages on the NFV interface on the other orchestrator to instantiate functions, services. The infrastructure domain of a MANO is segmented to accommodate the demands of separate organization hence deploying a hierarchy of service platforms that need to collaborate in order to deploy NFV end-to-end services. The interaction between the two MANOs is achieved by mapping the services and infrastructure domains of the MANO.

In a hierarchical placement of the two MANO service platforms, it either supports complete outsourcing of a network service for deployment in a lower service platform or split the service deployment across two MANO SPs. Hence, the NFVO of the upper MANO constitutes a resource orchestrator (RO) along with network service orchestrator (NSO) to facilitate the services.

According to [**?**], for a network service to support across multiple administrative domains, they require coordination and synchronization between multiple involved infrastructure domains which are performed by one or more orchestrators. The ETSI approaches for multiple administrative domains are depicted in the figure below.

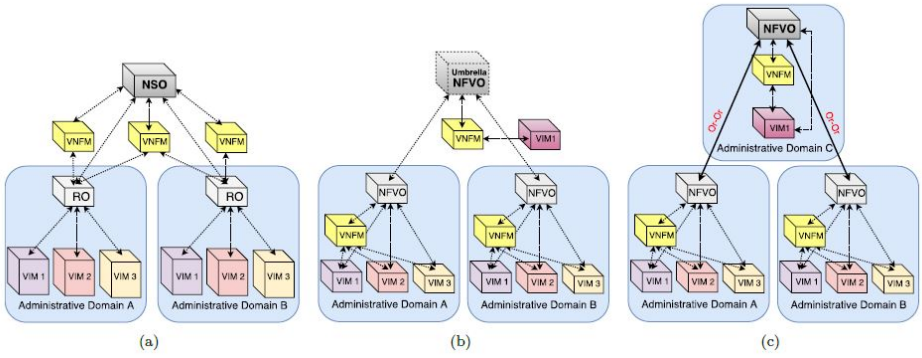

Figure 5.1: ETSI approaches for multiple administrative domains. Adapted from [**?**]

In the above figure 3.1, (a) refers to a approach in which the orchestrator is split into two components (NSO and RO), (b) refers to a approach with multiple orchestrators and a new reference point: Umbrella NFVO and (c) refers to a approach that introduces hierarchy and the new reference point Or-Or.

6

Scaling a network service

The scaling of a network service plays a key role while handling the system load on a MANO or for a better performance. The network service contains a NSD that limits the instantiation levels of an NS instance, by defining them as the discrete set of levels addressing the number of instantiation levels required while scaling a network service [**?**].

According to [**?**], the deployment flavors of an NSD contains information about the instantiation levels permitted for an NS instance, with the help of information from VNFDs and VLDs. The VNF flavour of the VNFD specifies which part of VNFCs are to be deployed. The NS flavour selects the part of VNFs and VLs to be deployed as a part of NS. With this information it defines the instantiation levels. The below figure shows the scaling attributes of an NSD.

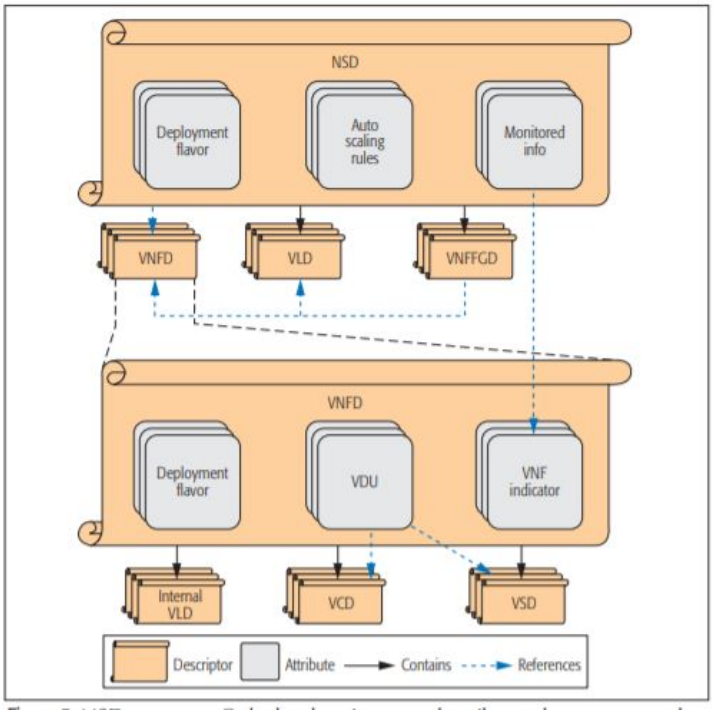

Figure 6.1: NSD structure. Adapted from [**?**]

7

Capacity of NFVOs and VNFMs

In distributed NFVI environment or when network services are spanned over large geographical areas, the number of VNF instances are likely to increase. In a MANO, during the increase on the system load, the number of requests for a service increases, and in turn the number of VNF instances. Hence, there should be adequate number of VNFMs and NFVOs to manage the VNF instances. To determine the capacity of NFVO and VNFM, the Integer Linear Programming (ILP) formula is proposed [**?**].

The below formula determines the optimal number of NFVOs and VNFMs required in a distributed system [**?**].

Consider the NFVI modeled as a graph $G = (P, E)$ where P is the set of NFVI-PoP nodes and E is the set of edges linking them, such that $E = \{(p, q) \mid p \in P, q \in P, p \neq q\}$. We use $\delta_{p,q}$ to represent the network delay of an edge $(p, q) \in E$. Let V represent the set of VNF instances in the system. The location of a VNF instance $v \in V$ is defined by $l_{v,p} \in \{0, 1\}$ such that $l_{v,p}$ equals to 1 only when v is placed at $p \in P$. We define M to represent the set of VNFMs that can be used to manage the VNF instances. We also use φ to denote the capacity of a VNFM. It represents the maximum number of VNF instances that can be managed by a VNFM.

Decision Variables:

$h_p \in \{0,1\} : (1)$ indicates that a NFVO is placed at $p \in P$, (0) otherwise.

$r_{q,p} \in \{0,1\} : (1)$ specifies that $q \in P$ is assigned to the NFVO which is placed at $p \in P$, (0) otherwise.

$x_{m,p} \in \{0,1\} : (1)$ designates that $m \in M$ is placed at $p \in P$, (0) otherwise.

$y_{v,m,p} \in \{0,1\} : (1)$ indicates that $v \in V$ is assigned to $m \in M$ which is placed at $p \in P$, (0) otherwise.

Mathematical Model:

$$\textit{Minimize} \quad \sum_{p \in P} h_p + \sum_{m \in M} \sum_{p \in P} x_{m,p} \tag{1}$$

Subject to:

$$\sum_{p \in P} r_{q,p} = 1, \quad \forall q \in P \tag{2}$$

$$r_{q,p} \leq h_p, \quad \forall q, p \in P \tag{3}$$

$$r_{p,p} = h_p, \quad \forall p \in P \tag{4}$$

$$\sum_{p \in P} x_{m,p} \leq 1, \quad \forall m \in M \tag{5}$$

$$\sum_{m \in M} \sum_{p \in P} y_{v,m,p} = 1, \quad \forall v \in V \tag{6}$$

$$y_{v,m,p} \leq x_{m,p}, \quad \forall v \in V, m \in M, p \in P \tag{7}$$

$$l_{v,q}\, y_{v,m,\acute{p}}\, r_{\acute{p},p} \leq r_{q,p}, \quad \forall v \in V, m \in M, q, \acute{p}, p \in P \tag{8}$$

$$\sum_{v \in V} \sum_{m \in M} \sum_{q \in P} y_{v,m,q}\, r_{q,p} \leq \Phi\, h_p, \quad \forall p \in P \tag{9}$$

$$\sum_{v \in V} y_{v,m,p} \leq \varphi\, x_{m,p}, \quad \forall m \in M, p \in P \tag{10}$$

8 Scramble architecture

Scramble is a tool to realize scalability with hierarchical orchestration. While the higher level orchestrator (Parent MANO) still controls the life-cycle of the deployed network services, scramble acts as a bridge between parent MANO and lower level orchestrator (child MANO) thus allowing interconnection between different MANO's to provide end-to-end service.

8.1 Architecture

The services of scramble resides as a plugin within MANO frameworks such as SOTANA/Pishahang and Open Source MANO (OSM) and enables cross MANO communication. Service requests received by the higher level MANOs in the hierarchy can be routed to lower level MANOs using scramble plugin based on monitoring parameters.

Each child MANO's in-turn contains scramble plugin which allows multiple other MANO's to be added as a child hence achieving hierarchical orchestration (See Figure 8.1).

Scramble is composed of three main services.

- Translator
- Splitter
- Wrapper(adaptor)

Each of these services are further explained in-detail in Chapters 11, 10, 9.

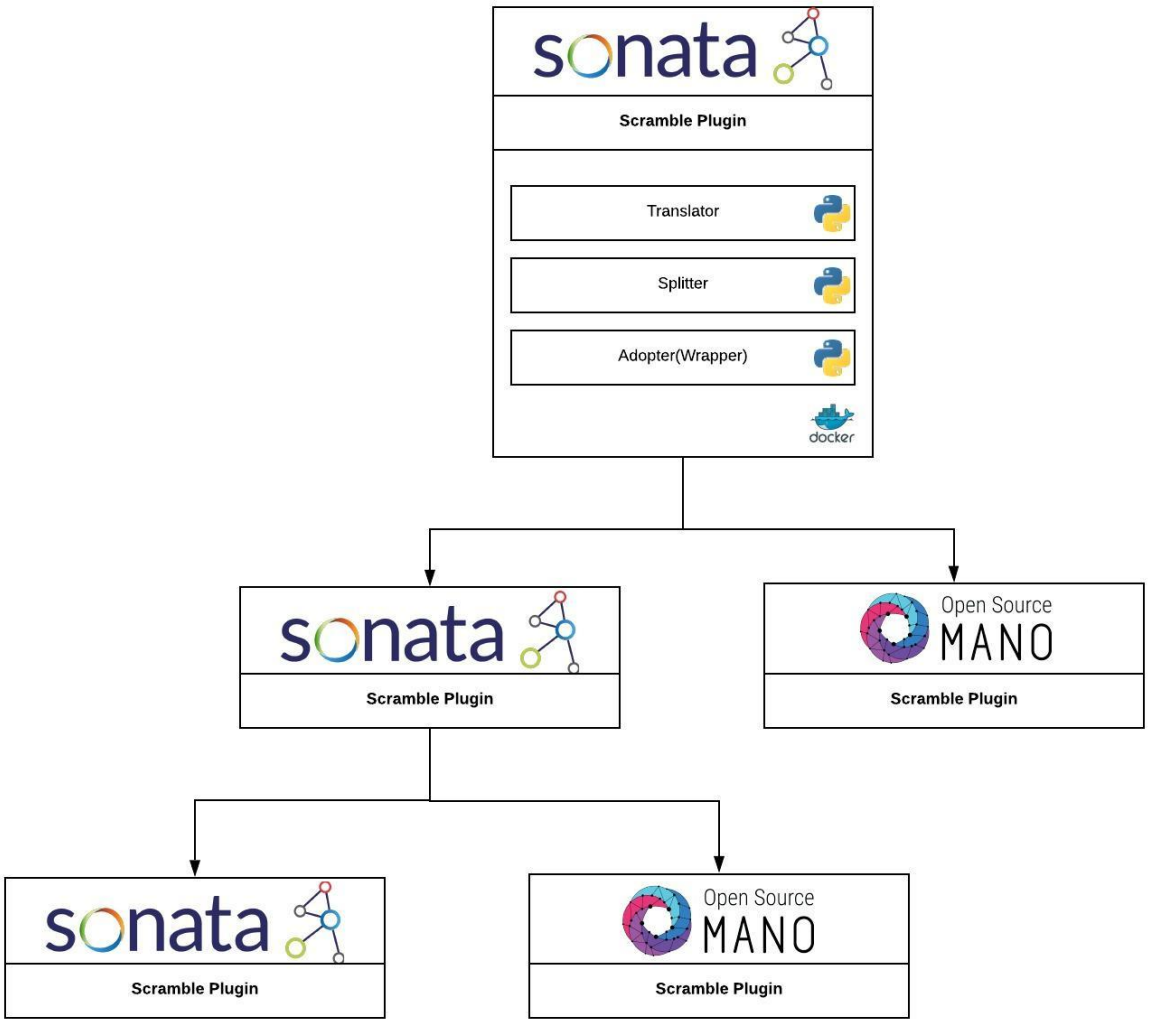

Figure 8.1: Scramble Architecture

9 Python MANO Wrappers (Adaptor)

Python MANO Wrappers (PMW) is a uniform python wrapper library for various implementations of NFV Management and Network Orchestration (MANO) REST APIs. PMW is intended to ease the communication between python and MANO by providing a unified, convenient and standards oriented access to MANO API.

To achieve this, PMW follows the conventions from the ETSI GS NFV-SOL 005 (SOL005) RESTful protocols specification. This makes it easy to follow and the developers can use similar processes when communicating with a variety of MANO implementations.

PMW is easy to use and well documented. Code usage examples are available along with the detailed documentation at the following link `https://python-mano-wrappers.readthedocs.io/en/adaptor/`.

PMW in scramble helps in inter communication of different instances of MANO, thereby creating opportunity for more advanced feature set, for example, hierarchical scaling. Operations such as on-boarding of NSD and VNFD, instantiation and termination of NS can be performed with ease.

9.1 Architecture

Standards based approach is a fundamental design principle behind PMW's design. A Common interface template is defined in compliance with SOL005 which contains the blueprint for all the methods mentioned in the standards. These methods are divided into different sections as per SOL005 into the following:

- **auth:** Authorization API
- **nsd:** NSD Management API
- **nsfm:** NS Fault Management API
- **nslcm:** Lifecycle Management API
- **nspm:** NS Performance Management API
- **vnfpkgm:** VNF Package Management API

In the figure 9.1, different sections of PMW are visualized. As part of the scramble project, support for Open Source MANO (OSM) and Sonata is developed. This is represented by the dotted lines to OSM and Sonata modules. These modules are based on the common interface and implement the methods it has defined.

9.2 Usage

Installation

PMW can be installed using pip:

```
pip install python-mano-wrappers
```

Basic examples

Listing 9.1: Fetching Auth token

```
import wrappers

username = "admin"
password = "admin"
mano = "osm"
# mano = "sonata"
host = "vm-hadik3r-05.cs.uni-paderborn.de"

if mano == "osm":
   _client = wrappers.OSMClient.Auth(host)
elif mano == "sonata":
   _client = wrappers.SONATAClient.Auth(host)

response = _client.auth(username=username, password=password)

print(response)
```

Listing 9.2: Instantiating a NS in OSM

```
from wrappers import OSMClient

USERNAME = "admin"
PASSWORD = "admin"
HOST_URL = "vm-hadik3r-05.cs.uni-paderborn.de"

osm_nsd = OSMClient.Nsd(HOST_URL)
osm_nslcm = OSMClient.Nslcm(HOST_URL)
osm_auth = OSMClient.Auth(HOST_URL)

_token = json.loads(osm_auth.auth(username=USERNAME, password=PASSWORD))
_token = json.loads(_token["data"])

_nsd_list = json.loads(osm_nsd.get_ns_descriptors(token=_token["id"]))
_nsd_list = json.loads(_nsd_list["data"])
_nsd = None
```

```
for _n in _nsd_list:
   if "test_osm_cirros_2vnf_nsd" == _n['id']:
      _nsd = _n['_id']

response = json.loads(osm_nslcm.post_ns_instances_nsinstanceid_instantiate(
                      token=_token["id"],
                      nsDescription=NSDESCRIPTION,
                      nsName=NSNAME,
                      nsdId=_nsd,
                      vimAccountId=VIMACCOUNTID))

response = json.loads(response["data"])

print(response)
```

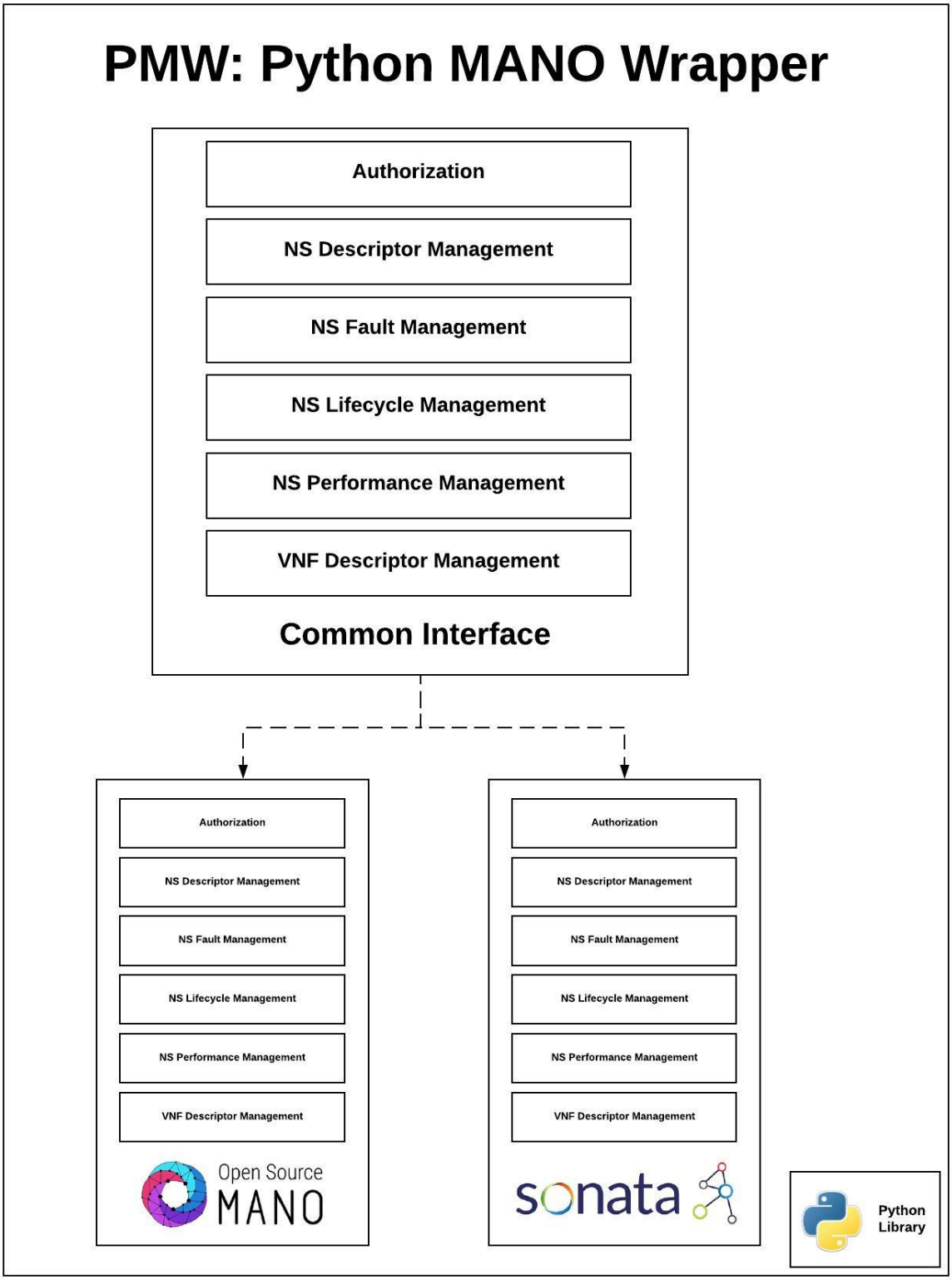
PMW: Python MANO Wrapper
Authorization
NS Descriptor Management
NS Fault Management
NS Lifecycle Management
NS Performance Management
VNF Descriptor Management
Common Interface
Authorization
NS Descriptor Management
NS Fault Management
NS Lifecycle Management
NS Performance Management
VNF Descriptor Management
Open Source MANO
Authorization
NS Descriptor Management
NS Fault Management
NS Lifecycle Management
NS Performance Management
VNF Descriptor Management
sonata
Python Library

Figure 9.1

10

Scramble splitter architecture

Communication between MANO frameworks and Scramble happens through plug-ins. Plug-ins will have some random logic to generate a multiple set of Network Functions into which a Network Service needs to be divided. The multiple set of NFs is then forwarded to Gateway micro-service along with the NS file which just receives the request and forward it to the Main-Engine. The Main Engine acts as an interface to all three utilities including Splitter. Main engine is responsible for interoperability and communication between all utilities. All three utilities are containerized using docker for easy distribution and scaling. Once the Splitter receives the request for splitting along with the parameters it splits the NSD into sub NSDs.

10.1 Architecture

Before actual splitting of NSD, Splitter first validates if the the multiple set it received is valid or not. Following things are validated by the splitter.

- The total number of NFs mentioned in all the set matches the number of NFs defined in the NSD.
- There are no invalid NFs in the sets received.

After validation the actual splitting starts. We have created classes for different sections of a NSD which encapsulate all the attributes and its values into a single unit which makes it very easy to process. Once the objects are set they are passed to different splitting functions based on there type. We have two different processing units for OSM and SONATA. Following are some functions responsible for splitting the NSD.

- **Set General information:** This function copies all the general information from the main NSD to the sub NSDs. Information includes Vendor, author, Version, Descripton etc.
- **Split Network Functions:** This function splits the Network functions from NSD to sub sets according to the request parameter received.
- **Split Virtual Links:** When a NSD is splitted into different parts, its topology changes. Change in topology results in changing of Virtual Links. For example if A, B and C are three Nfs and we are splitting them in such a way so that A and B remain in one NSD

and C in separate NSD. A virtual link between B and C now does not make sense. So this link should be broken down and B's output should be connected to the external end point which was connected to C's input earlier. This function splits these kind of Virtual Links.

- **Split Forwarding Graph:** As explained in the above section, once the topology changes, the respective Forwarding graph also changes. Split forwarding graph pulls out the set of connection points and newly created virtual links and sets them in the sub NSDs.

Once the Splitting is done, create file is responsible for creating YAML files depending on the number of sub NSDs created. These files are saved in the file system which can be downloaded or moved forward to the adopter for deployment purpose. Following figure 10.1 graphically represents the splitting architecture.

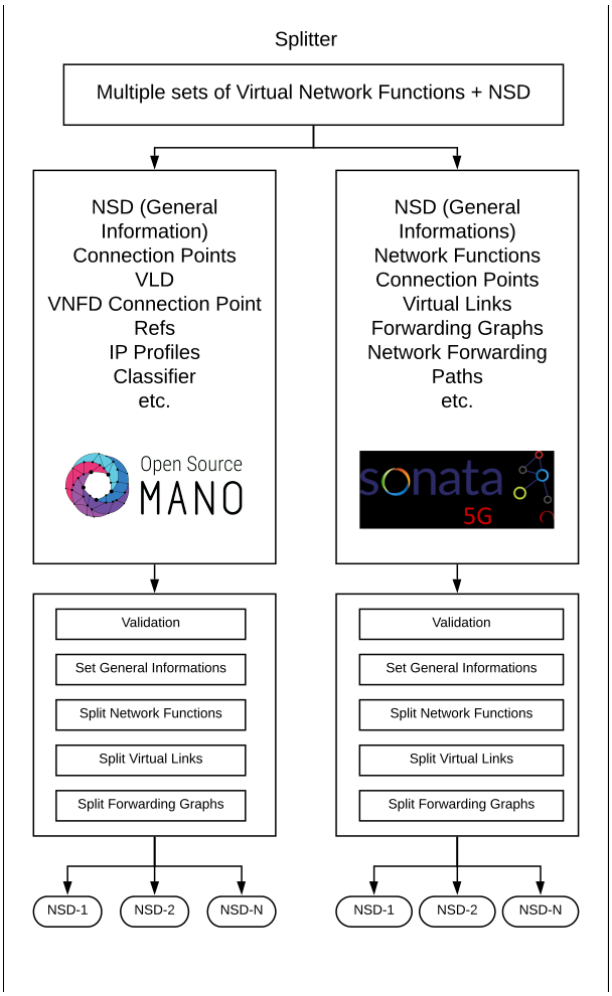

Figure 10.1: Scramble Architecture

11 Scramble translator architecture

In a hierarchical architecture involving different MANOs, there is a need of conversion of network descriptors to schemas of respective MANO. Service Descriptor Translator (SDT) serves the purpose of translating network descriptors, namely NSDs and VNFDs from schema of SONATA Pishahang to that of OSM and vice versa.

In a scenario, where a parent MANO, say Pishahang decides to deploy one of the network services in its lower hierarchy MANO, say OSM, the NSD and VNFD(s) need to be converted to the descriptor schema of OSM. In such an event, the Scramble plugin calls the translator service and sends the descriptors to the SDT, where the translation of the descriptors takes place and the translated descriptors are sent to Adaptor utility for deployment in appropriate MANO. Figure 11.1 gives a high level view of Translator.

Scramble plug-in installed within the parent MANO forwards the network descriptors with service request to Scramble Main-Engine. Main-Engine checks the service request and sends the network descriptors to Translator along with the information of destination schema. On receiving the network descriptors, SDT translates the same to requested schema and calls the validator function to validate the translated network descriptors. Once validation is complete, the translated descriptors will be sent to Adaptor for deployment.

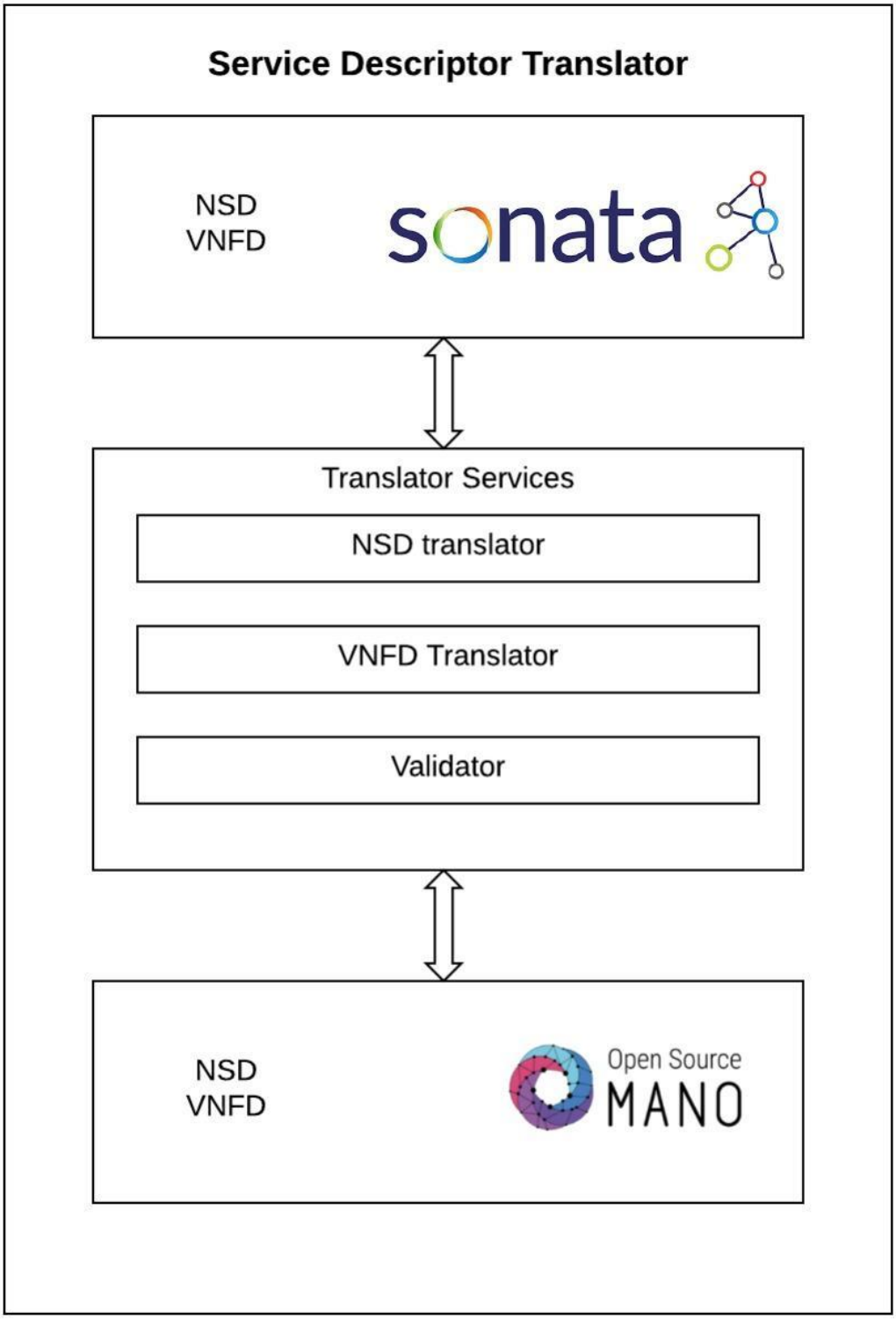
Service Descriptor Translator
NSD
VNFD
sonata
Translator Services
NSD translator
VNFD Translator
Validator
NSD
VNFD
Open Source
MANO

Figure 11.1

12 Conclusion

The main goal of this paper is to aggregate the factors affecting the scalability of a MANO. Scalability is an important factor in cloud environments which accommodates the demanding needs and also not affecting the system's performance. The report further states the system load in terms of the load on the NFVO of a MANO. The scaling in a distributed MANO system is affected by the availability of a service and its reliable quotient. The report further describes the effect of diverse administrative domains of a MANO framework and how it can be addressed by a coherence mean. Scalability approaches from various research papers are discussed. This paper also discusses the architecture design of Translator, Splitter and Adaptor.

The report further addresses the effects involved to scale a MANO, also briefing about scaling a network service. The further steps involved in the research is narrowing down to one of the scaling approaches to scale a MANO.

Bibliography