NFV Research Group
Internet-Draft
Intended status: Informational
Expires: March 11, 2017

N. Figueira
Brocade
R. Krishnan
Dell
D. Lopez
Telefonica
S. Wright
AT&T
D. Krishnaswamy
IBM

September 7, 2016

Policy Architecture and Framework for NFV Infrastructures
draft-irtf-nfvrg-nfv-policy-arch-04

Abstract

A policy architecture and framework is discussed to support NFV environments, where policies are used to enforce business rules and to specify resource constraints in a number of subsystems. This document approaches the policy framework and architecture from the perspective of overall orchestration requirements for services involving multiple subsystems. The framework extends beyond common orchestration constraints across compute, network, and storage subsystems to include energy conservation. This document also analyses policy scope, global versus local policies, static versus dynamic versus autonomic policies, policy actions and translations, policy conflict detection and resolution, interactions among policies engines, and a hierarchical policy architecture/framework to address the demanding and growing requirements of NFV environments. These findings may also be applicable to cloud infrastructures in general.

Copyright Notice

Conventions used in this document

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT",
"SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this
document are to be interpreted as described in RFC 2119.

Table of Contents

1. Introduction 3
2. Policy Intent Statement versus Subsystem Actions and
Configurations 4
3. Global vs Local Policies 5
4. Static vs Dynamic vs Autonomic Policies 7
5. Hierarchical Policy Framework 7
6. Policy Conflicts and Resolution 9
6.1. Soft vs Hard Policy Constraints 11
7. Policy Pub/Sub Bus 12
7.1 Pub/Sub Bus Name Space 16
8. Examples 17
8.1 Establishment of a Multipoint Ethernet Service 17
8.2 Policy-Based NFV Placement and Scheduling 20
8.2.1 Policy Engine Role in NFV Placement and Scheduling . . . 20
8.2.2 Policy-based NFV Placement and Scheduling with
OpenStack 21
9. Summary 25
10. IANA Considerations 25
11. Security Considerations 25
12. Contributors 26

13. References 26
13.1. Normative References 26
13.2. Informative References 26
Acknowledgements 28
Authors' Addresses 28

1. Introduction

This document discusses the policy architecture and framework to support Network Function Virtualization (NFV) [11] infrastructures. In these environments, policies are used to enforce business rules and to specify resource constraints, e.g., energy constraints, in a number of subsystems, e.g., compute, storage, network, and etc., and across subsystems. These subsystems correspond to the different "infrastructure domains" identified by the NFV ISG Infrastructure Working Group [8][10][7].

The current work in the area of policy for NFV is mostly considered in the framework of general cloud services, and typically focused on individual subsystems and addressing very specific use cases or environments. For example, [11] addresses network subsystem policy for network virtualization, [19] and [20] are open source projects in the area of network policy as part of the OpenDaylight [21] software defined networking (SDN) controller framework, [18] specifies an information model for network policy, [13] focuses on placement and migration policies for distributed virtual computing, [24] is an open source project in OpenStack [22] to address policy for general cloud environments.

This document approaches policy, policy framework, and policy architecture for NFV services from the perspective of overall orchestration requirements for services involving multiple subsystems, and can be applied to the general case of any cloud-based service. The analysis extends beyond common orchestration constraints across compute, network, and storage subsystems to also include energy conservation constraints applicable to NFV and other environments. The analysis in this document also extends beyond a single virtual Point of Presence (vPoP) or administrative domain to include multiple data centers and networks forming hierarchical domain architectures [12]. The focus of this document is not general policy theory, which has already been intensively studied and documented on numerous publications over the past 10 to 15 years (see [18], [27], [17], [28], and [3] to name a few). This document's purpose is to discuss and document a policy architecture that uses known policy concepts and theories to address the unique requirements of NFV services including multiple vPoPs and networks forming hierarchical domain architectures [12].

With the above goals, this document analyses policy scope, global versus local policies, static versus dynamic versus autonomic policies, policy actions and translations of actions, policy conflict detection and resolution (which can be relevant to resource management in service chains [16]), the interactions among policies engines from the different vPoPs and network subsystems, and a hierarchical policy architecture/framework to address the demanding and growing requirements of NFV environments. These findings may also be applicable to cloud infrastructures in general.

2. Policy Intent Statement versus Subsystem Actions and Configurations

Policies define which states of deployment are in compliance with the policy, and, by logic negation, which ones are not. The compliance statement in a policy may define specific actions, e.g., "a given customer is [not allowed to deploy VNF X]", where VNF refers to a Virtual Network Function, or quasi-specific actions, e.g., "a given customer [must be given platinum treatment]." Quasi-specific actions differ from the specific ones in that the former requires an additional level of translation or interpretation, which will depend on the subsystems where the policy is being evaluated, while the latter does not require further translation or interpretation.

In the previous examples, "VNF X" defines a specific VNF type, i.e., "X" in this case, while "platinum treatment" could be translated to an appropriate resource type depending on the subsystem. For example, in the compute subsystem this could be translated to servers of a defined minimum performance specification, while in the network subsystem this could be translated to a specific Quality of Service (QoS) level treatment.

The actions defined in a policy may be translated to subsystem configurations. For example, when "platinum treatment" is translated to a specific QoS level treatment in a networking subsystem, one of the outcomes (there can be multiple ones) of the policy could be the configuration of network elements (physical or virtual) to mark that customer's traffic to a certain DSCP (DiffServ Code Point) level (Figure 1). Some may refer to the QoS configuration above as a policy in itself, e.g., [27], [28], [4], and [17]. In this document, such domain configurations are called policy enforcement technologies to set them apart from the actual policy intent, e.g., "a given customer must be given platinum treatment" as in the above example.

Describing intent using a high-level policy language instead of directly describing configuration details allows for the decoupling of the desired intent from the actual configurations, which are subsystem dependent, as shown in the previous example (Figure 1). The

translation of a policy into appropriate subsystem configurations requires additional information that is usually subsystem and technology dependent. Therefore, policies should not be written in terms of policy enforcement technologies. Policies should be translated at the subsystems using the appropriate policy provides a few examples where the policy "a given customer must be given platinum treatment" is translated to appropriate configurations at the respective subsystems.

The above may sound like a discussion about "declarative" versus "imperative" policies. We are actually postulating that "imperative policy" is just a derived subsystem configuration using an appropriate policy enforcement technology to support an actually intended policy.

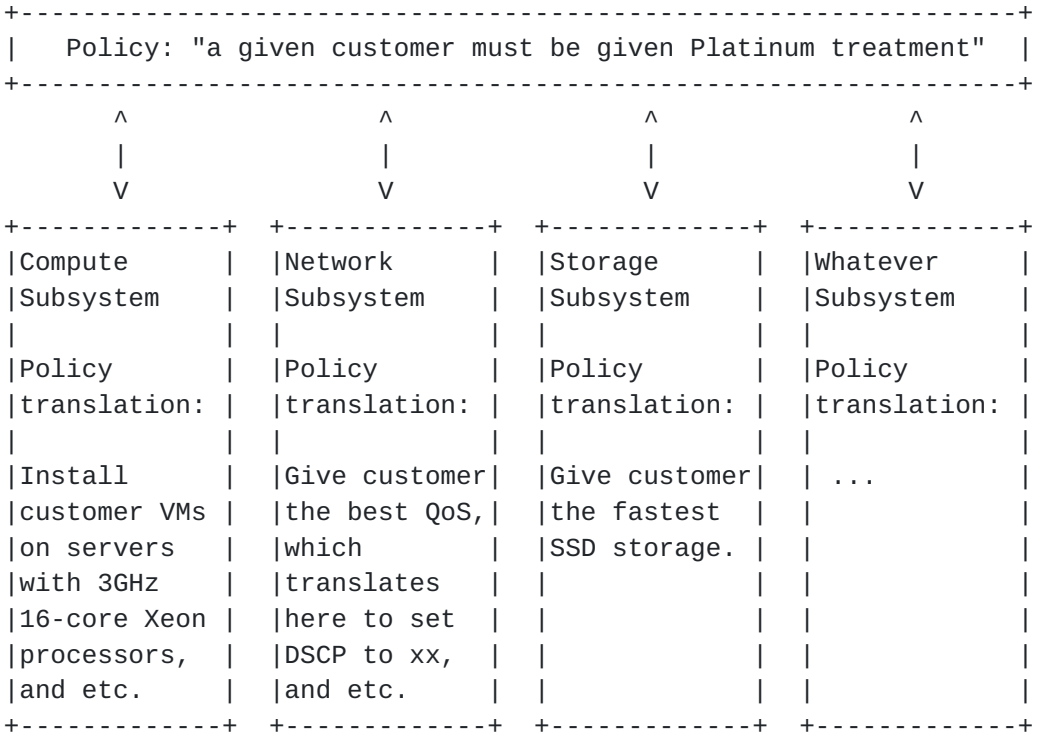

Figure 1: Example of Subsystem Translations of Policy Actions

3. Global vs Local Policies

Some policies may be subsystem specific in scope, while others may have broader scope and interact with multiple subsystems. For example, a policy constraining certain customer types (or specific customers) to only use certain server types for VNF or Virtual Machine (VM) deployment would be within the scope of the compute subsystem. A policy dictating that a given customer type (or specific

customers) must be given "platinum treatment" could have different implications on different subsystems. As shown in Figure 1, that "platinum treatment" could be translated to servers of a given performance specification in a compute subsystem and storage of a given performance specification in a storage subsystem.

Policies with broader scope, or global policies, would be defined outside affected subsystems and enforced by a global policy engine (Figure 2), while subsystem-specific policies or local policies, would be defined and enforced at the local policy engines of the respective subsystems.

Examples of sub-system policies can include thresholds for utilization of sub-system resources, affinity/anti-affinity constraints with regard to utilization or mapping of sub-system resources for specific tasks, network services, or workloads, or monitoring constraints regarding under-utilization or over-utilization of sub-system resources.

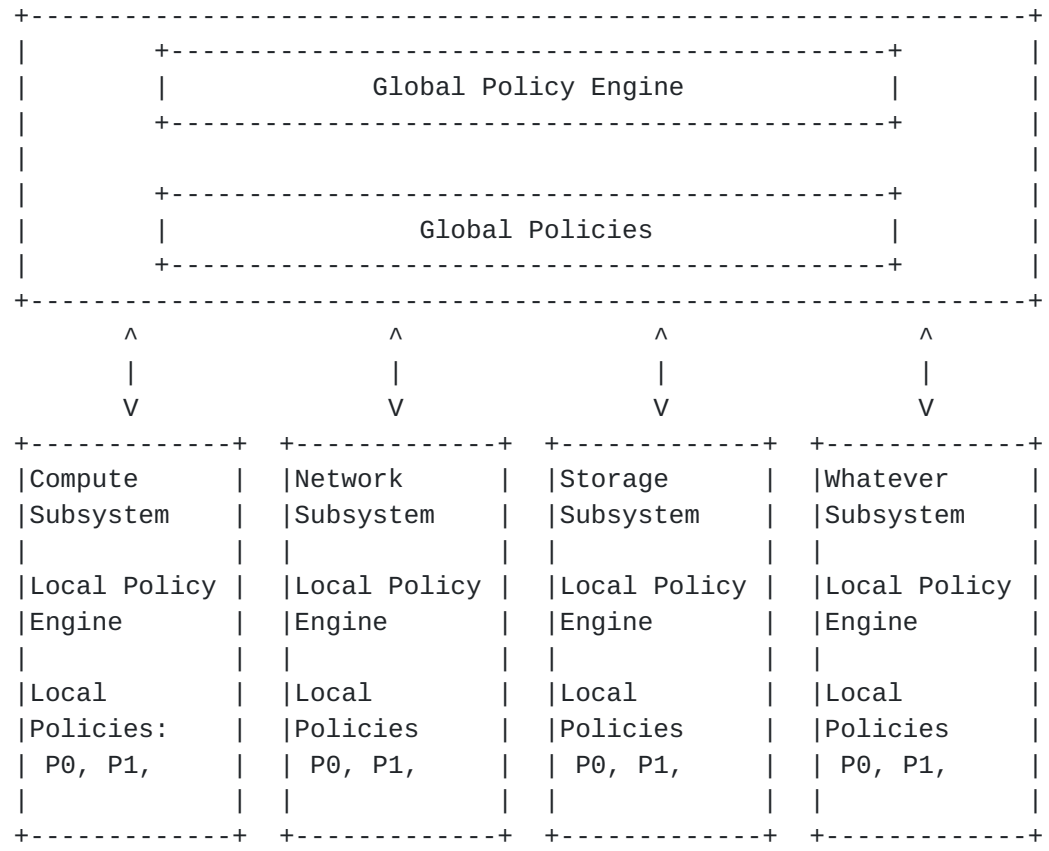

Figure 2: Global versus Local Policy Engines

4. Static vs Dynamic vs Autonomic Policies

Policies can be defined based on a diverse set of constraints and sources, e.g., an operator's energy cost reduction policy based on the time-varying energy rates imposed by a utility company supplying energy to a given region or an operator's "green" policy based on the operator's green operational requirements, which may drive a reduction in energy consumption despite of potentially increased energy costs.

While "static" policies can be imposed based on past learned behavior in the systems, "dynamic" policies can be define that would override "static" policies due to dynamically varying constraints. For example, if a system needs to provided significantly additional scaling of users in a given geographic area due to a sporting or a concert event at that location, then a dynamic policy can be superimposed to temporarily override a static policy to support additional users at that location for a certain period of time. Alternatively, if energy costs significantly rise on a particular day, then an energy cost threshold could be dynamically raised to avoid policy violation on that day.

Support for autonomic policies may also be required, such as an auto-scaling policy that allows a resource (compute/storage/network/energy resource) to be scaled up or down as needed. For example, a policy specifying that a VNF can be scaled up or down to accommodate traffic needs.

5. Hierarchical Policy Framework

So far, we have referenced compute, network, and storage as subsystems examples. However, the following subsystems may also support policy engines and subsystem specific policies:

- SDN Controllers, e.g., OpenDaylight [21].
- OpenStack [22] components such as, Neutron, Cinder, Nova, and etc.
- Directories, e.g., LDAP, ActiveDirectory, and etc.
- Applications in general, e.g., standalone or on top of OpenDaylight or OpenStack.
- Physical and virtual network elements, e.g., routers, firewalls, application delivery controllers (ADCs), and etc.
- Energy subsystems, e.g., OpenStack Neat [25].

Therefore, a policy framework may involve a multitude of subsystems. Subsystems may include other lower level subsystems, e.g., Neutron [26] would be a lower level subsystem in the OpenStack subsystem. In

other words, the policy framework is hierarchical in nature, where the policy engine of a subsystem may be viewed as a higher level policy engine by lower level subsystems. In fact, the global policy engine in Figure 2 could be the policy engine of a Data Center subsystem and multiple Data Center subsystems could be grouped in a region containing a region global policy engine. In addition, one could define regions inside regions, hierarchically, as shown in Figure 3.

Metro and wide-area network (WAN) used to interconnect data centers would also be independent subsystems with their own policy engines.

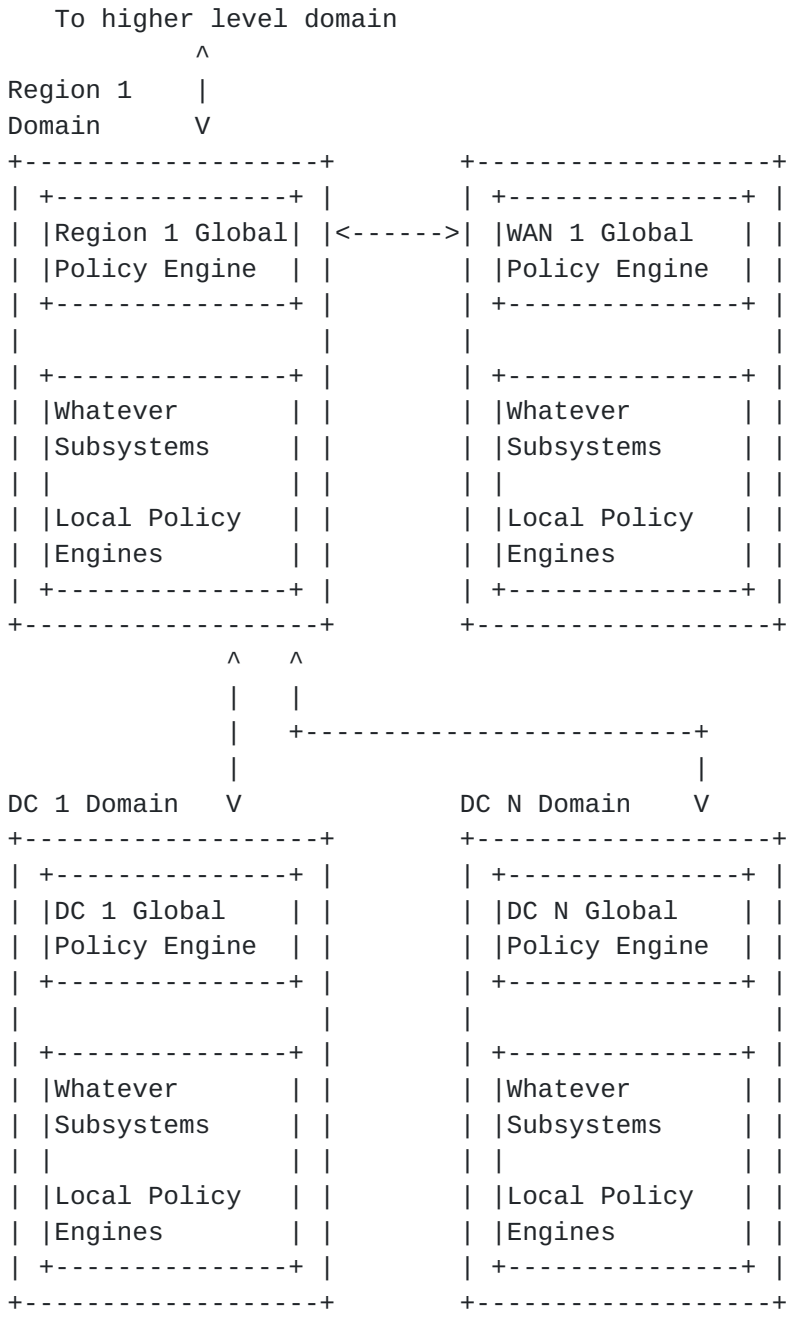

Figure 3: A Hierarchical Policy Framework

6. Policy Conflicts and Resolution

Policies should be stored in databases accessible by the policy engines. For example, the local policies defined for the Compute subsystem in Figure 2 would be stored in a database accessible by the local policy engine in that subsystem.

As a new policy is added to a subsystem, the subsystem's policy engine should perform conflict checks. For example, a simple conflict would be created if a new policy states that "customer A must not be allowed to use VNF X", while an already existing policy states that "customer A is allowed to use VNF X". In this case, the conflict should be detected and an appropriate policy conflict resolution mechanism should be initiated.

The nature of the policy conflict resolution mechanism would depend on how the new policy is being entered into the database. If an administrator is manually attempting to enter that policy, the conflict resolution could entail a warning message and rejection of the new policy. The administrator would then decide whether or not to replace the existing policy with the new one.

When policies are batched for later inclusion in the database, the administrator should run a preemptive conflict resolution check on those policies before committing to include them in the database at a future time. However, running a preemptive conflict resolution check does not guarantee that there will be no conflicts at the time the batched policies are actually included in the database, since other policies could have been added in the interim that cause conflicts with those batched policies.

To avoid conflicts between batched policies waiting for later inclusion in the database and new policies being immediately added to the database, one could run a preemptive conflict resolution check against database policies and also batched policies every time new policies are added to the database. However, this may not be sufficient in case of separate administrative domains. A region administration could define batched polices to be pushed to the Compute subsystem of a Data Center at a later time. However, the Compute subsystem may be a separate administrative domain from that of the region administrative domain. In this case, the Compute subsystem may not be allowed to run preemptive policy conflict checks against the batched policies defined at the region administrative domain. Thus, there is a need for a reactive policy conflict resolution mechanism besides preemptive techniques.

The above discussions implicitly assumed that policies are individually evaluated for conflicts and individually committed without regard to other policies. However, a set of policies could be labeled as part of a same "Commit Group", where the whole set of policies in the Commit Group must be committed for a desired result to be obtained. In this case, the conflict resolution mechanism would need to verify that none of the policies in the Commit Group conflicts with currently committed policies before the Commit Group is added (in other words, committed) to the policy database.

The Commit Group conflict detection mechanism and subsequent addition to the database should be implemented as an atomic process, i.e., no changes to the policy database should be allowed by other processes until either the whole Commit Group is checked and committed or a conflict is detected and the process stopped, to avoid multiple writers issues.

The above described atomic Commit Group conflict detection and policy commit mechanism would eliminate the need for Commit Group rollback. A rollback could be required if policies in a Commit Group were to be checked for conflicts and committed one by one, since the detection of a subsequent policy conflict in the Commit Group would require the rollback of previously committed policies in that group.

6.1. Soft vs Hard Policy Constraints

Policies at any level in the policy hierarchy can be either soft or hard. A soft policy imposes a soft constraint in a system that can be violated without causing any catastrophic failure in the system. A hard policy imposes a hard constraint in the system that must not be violated. An example of a soft constraint is the degree of underutilization (for example, a 40% utilization threshold could be used as a soft constraint) of compute servers with regard to CPU utilization. In such a case, when this soft constraint is violated, the system continues to function, although it may consume more energy (due to a non-linear dependence of the energy utilization as a function of the CPU utilization) compared to a task allocation across multiple servers after workload consolidation is performed. Alternatively, a soft constraint could be violated if the network bandwidth exceeds a certain fraction (say 80%) of the available bandwidth, the energy utilization exceeds a certain value, or the memory utilization falls below a certain value (say 30%). An example of a hard constraint could be to disallow a desired mean CPU utilization across allocated workloads in a compute subsystem to exceed a certain high threshold (such as 90%), or to disallow the number of CPUs allocated for a set of workloads to exceed a certain value, or to disallow the network bandwidth across workloads to exceed a certain value (say 98% of the maximum available bandwidth).

When considering policy conflicts, violation of policies across hard policy constraints is undesirable, and must be avoided. A conflict resolution could be possible by relaxing one or more of the hard constraints to an extent that is mutually satisfactory to the imposers of the policies. Alternatively, as discussed earlier, a new hard policy that conflicts with existing policies is not admitted into the system. Violation of policies across soft policy constraints or between one or more hard policy constraints and one or more soft policy constraints can be allowed, such that one or more soft policy

constraints are violated without hard constraints being violated. Despite soft constraints being violated, it is desirable to have a region of operating conditions that would allow the system to operate. For admission of new policy constraints, whether hard or soft, one should ensure that the overall system has a feasible region for operation given the existing constraints, and the new constraints under consideration. When such a feasible region is not possible, one can consider relaxing one or more of the existing or new constraints to allow such policies to be admitted.

7. Policy Pub/Sub Bus

In the previous section, we considered policy conflicts within a same level subsystem. For example, new local policies added to the Compute subsystem conflicting with existing local policies at that subsystem. However, more subtle conflicts are possible between global and local policies.

A global policy may conflict with subsystems' local policies. Consider the following Compute subsystem local policy: "Platinum treatment must be provided using server of type A."

The addition of the Global policy "Platinum treatment must be provided using server subtype A-1" would intrude into the Compute subsystem by redefining the type of server to be used for a particular service treatment. While one could argue that such global policy should not be permitted, this is an event that requires detection and proper resolution. A possible resolution is for the Compute subsystem to import the more restrictive policy into its local database. The original local policy would remain in the database as is along with the new restrictive policy. The local policy engine would then enforce the more restricted form of the policy after this policy change, which could make already existing resource allocations non-compliant and requiring corrective actions, e.g., Platinum treatment being currently provided by a server of type A instead of a server of type A-1.

If the new Global policy read "Platinum treatment must be provided using server of types A or B" instead, the Compute subsystem would not need to do anything different, since the Compute subsystem has a more restrictive local policy in place, i.e., "Platinum treatment must be provided using server of type A."

The above examples demonstrate the need for subsystems to subscribe to policy updates at the Global policy level. A policy publication/subscription (pub/sub) bus would be required as shown in Figure 4.

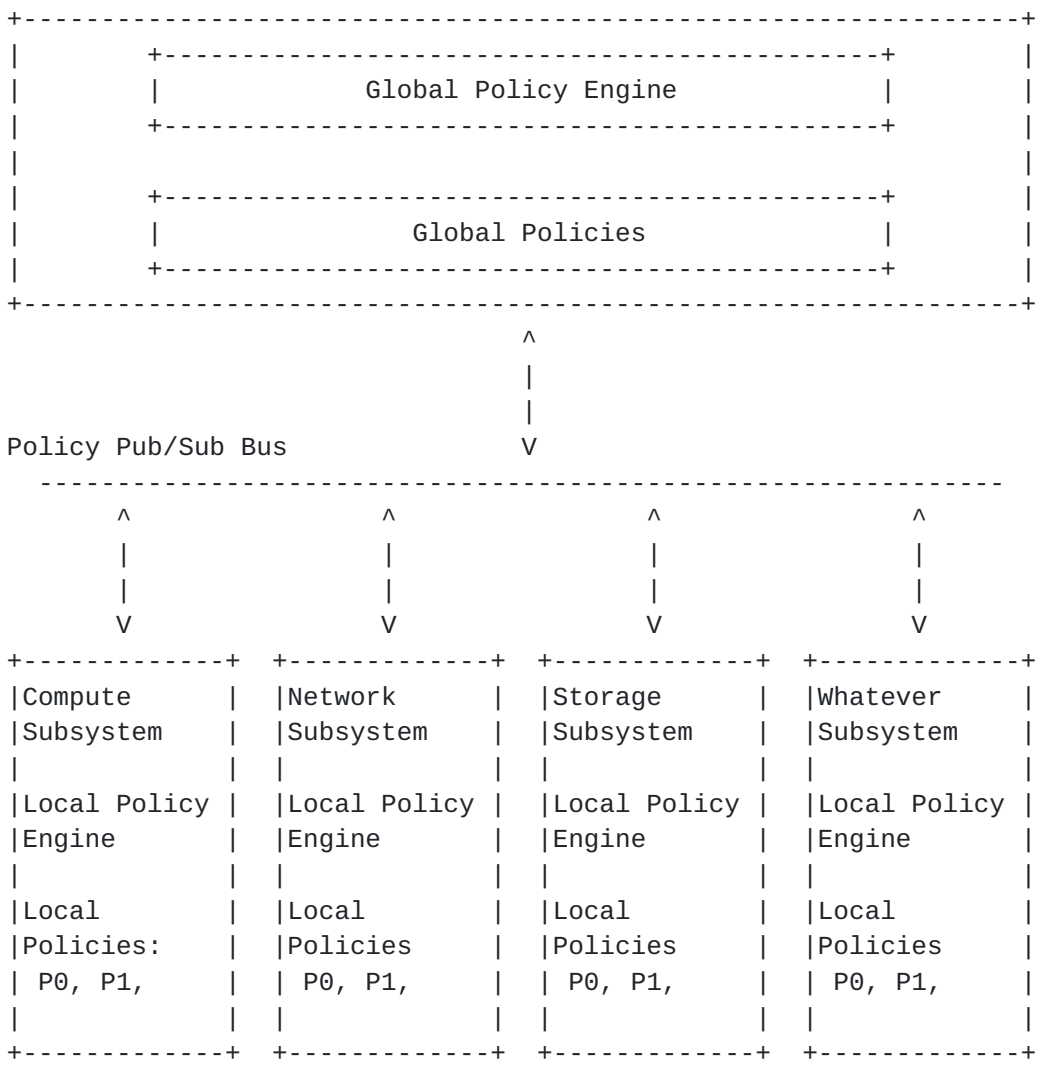

Figure 4: A Policy Pub/Sub Bus

A policy conflict may force policies to change scope. Consider the following existing policies in a Data Center:

Compute subsystem policy: "Platinum treatment requires a server of type A or B."

Storage subsystem policy: "Platinum treatment requires a server storage of type X or Y."

Now consider the outcome of adding the following new Global policy: "Platinum treatment requires a server of type A when storage of type X is used or a server of type B when storage of type Y is used."

This new Global policy intrudes into the Compute and Storage subsystems. Again, one could argue that such global policy should not

be permitted. Nevertheless, this is an event that would require detection and proper resolution. This Global policy causes a conflict because the Compute and Storage subsystems can no longer independently define whether to use a server of type A or B or storage of type X or Y, respectively. If the Compute subsystem selects server of type A for a customer and the Storage subsystem selects storage of type Y for that same customer service the Global policy is violated. In conclusion, if such global policy is permitted, the Compute and Storage subsystems can no longer make such selections. A possible conflict resolution is for the Compute and Storage subsystems to relegate policy enforcement for such resources to the Global policy engine. In this example, the Global Policy engine would need to coordinate with the Compute and Storage subsystems the selection of appropriate resource types to satisfy that policy.

That suggests that the policy pub/sub bus should in fact be an integral part of the northbound service interfaces (NBI) of the subsystems in the hierarchy. Such issue was analyzed in [12], where the concepts of service capability, service availability, and service instantiation were introduced to enable a higher-level subsystem to properly select services and resources from lower-level subsystems to satisfy existing policies.

The above example demonstrates again the need for subsystems to subscribe to policy updates at the higher policy level (the Global policy level in this example) as shown in Figure 4.

If, as demonstrated, a Global policy may "hijack" or "nullify" local policies of subsystems, what exactly makes the scope of a policy local versus global then?

Proposition: A Local Policy does not affect the compliance state imposed by global Policies or the local policies of other subsystems.

The above non-exhaustive examples demonstrate that global and local policies may conflict in subtle ways. Policy conflicts will also policy framework requires a policy pub/sub bus between all levels to allow for conflict detection, conflict information propagation, and conflict resolution (Figure 5).

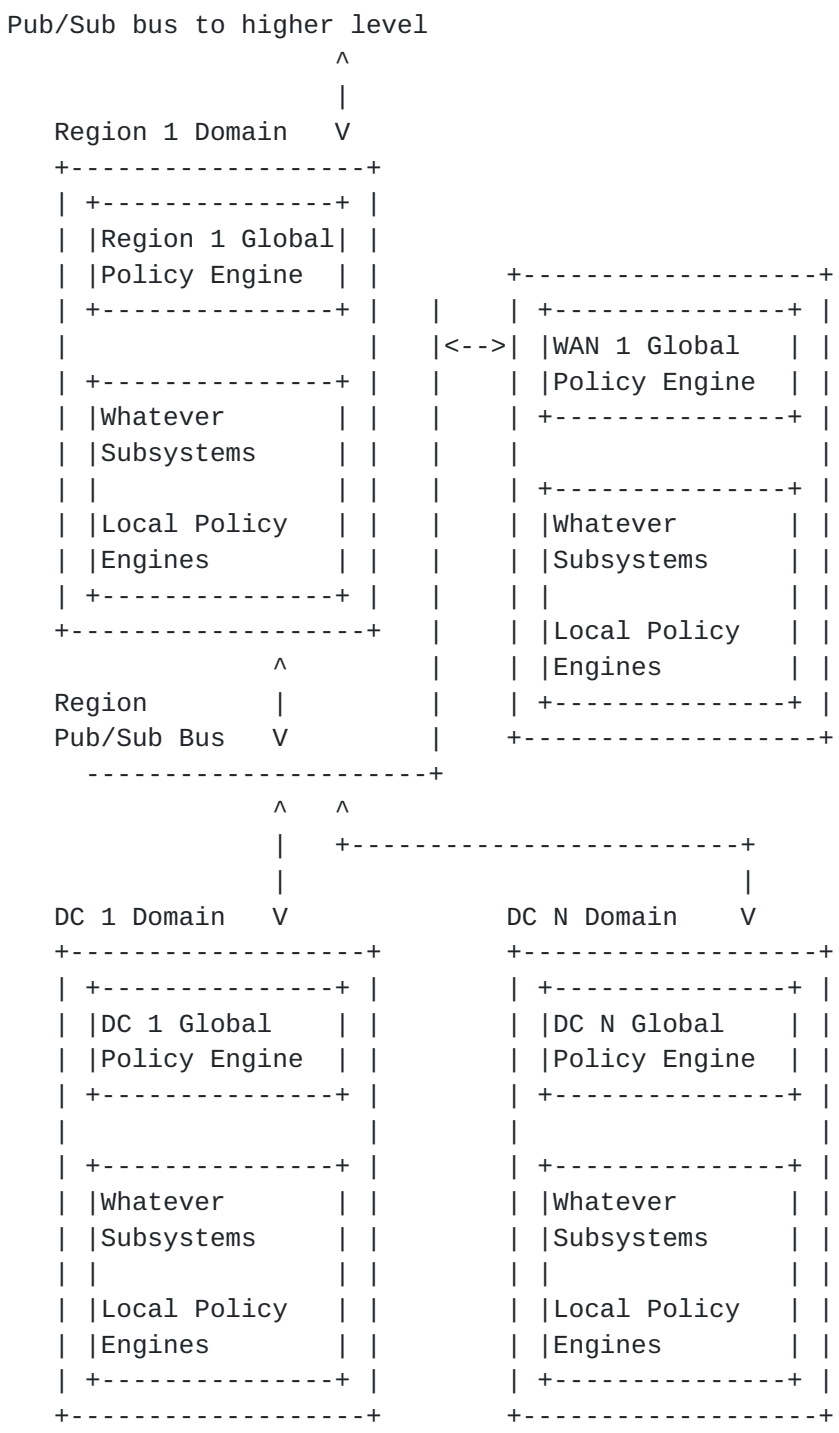

Figure 5: Pub/Sub Bus - Hierarchical Policy Framework

7.1 Pub/Sub Bus Name Space

As described above, a higher tier policy engine would communicate policies to lower tier policy engines using a policy pub/sub bus. Conversely, lower tier policy engines would communicate their configured policies and services to the higher tier policy engine using the same policy pub/sub bus. Such communications require each policy pub/sub bus to have a pre-defined/pre-configured policy "name space". For example, a pub/sub bus could define services using the name space "Platinum", "Gold", and "Silver". A policy could then be communicated over that pub/sub bus specifying a Silver service requirement.

In a hierarchical policy framework, a policy engine may use more than one policy pub/sub bus, e.g., a policy pub/sub bus named "H" to communicate with a higher tier policy engine and a policy pub/sub bus named "L" to communicate with lower tier policy engines. As the name spaces of policy pub/sub buses H and L may be different, the policy engine would translate policies defined using the policy pub/sub bus H name space to policies defined using the policy pub/sub bus L name space, and vice-versa. For example, suppose that the policy pub/sub bus H name space defines service levels named Platinum, Gold, and Silver and that the policy pub/sub bus L name space does not define such service levels, but defines QoS levels High, Medium, and Low. The policy engine would translate a policy to support Silver service, which is written using the policy pub/sub bus H name space, to an appropriate policy (or set of policies) written using the policy pub/sub bus L name space, e.g., QoS level Low.

The described policy framework does not preclude use of a single/same name space throughout the hierarchy. However, to promote scalability and limit complexity, the name spaces of higher tier policy pub/sub buses should be limited to support higher level policies, since the higher the degree of specificity allowed at the higher tiers of the policy hierarchy the higher the operational complexity.

8. Examples

8.1 Establishment of a Multipoint Ethernet Service

Consider a service provider with an NFV infrastructure (NFVI) with multiple vPoPs, where each vPoP is a separate administrative domain. A customer "Z" requests the creation of a "multipoint Silver Ethernet service" between three of its sites, which are connected to service provider's vPoPs A, B, and C. The customer request is carried out using a service provider self-service web portal, which offers customers multiple service type options, e.g., point-to-point and multipoint Ethernet services, and multiple service levels per service type, e.g., Platinum, Gold, and Silver Ethernet services, where the different service levels may represent different service specifications in terms of QoS, latency, and etc. The web portal relays the request to a service provider's OSS/BSS. The service request is stored as a service policy that reads as: "multipoint Silver Ethernet service between vPoPs A, B, and C for customer Z".

The OSS/BSS subsystem would communicate the service request and requirements as a policy to a global NFV Orchestrator (NFVO) subsystem using the name space of the pub/sub bus between these two subsystems (see Section 7.1). For example, the OSS/BSS could translate "Silver" service level into a policy defined using a Network Service (NS) Flavor ID, as defined by the name space of pub/sub bus between the OSS/BSS and the NFVO.

The service provider's vPoP NFV infrastructure architecture may vary depending on the size of each vPoP and other specific needs of the service provider. For example, a vPoP may have a local NFVO subsystem and one or more local Virtual Infrastructure Manager (VIM) subsystems (as in Figure 6). In this case, the global NFVO subsystem would communicate the service request and requirements as a policy to the local NFVOs of vPoPs A, B, and C.

At each vPoP, the local NFVO (and VNF Managers) would carry out the requested service policy based on the local configurations of respective subsystems and current availability of resources. For example, the requested service may translate in vPoP A to use a specific vCE (virtual customer edge) VNF type, say vCE_X, while in vPoP B it may translate to use a different vCPE VNF type, say vCPE_Y, due to local subsystem configurations (refer to Section 2 for a discussion on subsystem actions and configurations). Similarly, the local VIM interaction with the vPoP's compute, network, and storage subsystems may lead to local configurations of these subsystems driven by the translation of the policies received by the respective subsystems (see Section 3 for a discussion on global versus local policies). Note that the original policy at the OSS/BSS level is

translated throughout the policy hierarchy by respective policy engines to fit the name spaces of the associated pub/sub buses in the hierarchy.

The global NFVO subsystem could also communicate a policy defining the requirements to create a multipoint Ethernet service between vPoPs A, B, and C to a WAN infrastructure management (WIM) subsystem (not shown in Figure 6). The WIM subsystem could oversee a hierarchy of other subsystems, e.g., SDN multi-domain architecture of controllers deployed as a hierarchy of network regions (see [12]). Network subsystems would translate locally received policies to local configurations (again, refer to Section 2 for a discussion on subsystem actions and configurations).

As depicted in Figure 6, policy communications would employ a policy pub/sub bus between the subsystems' policy engines in the policy hierarchy (see Section 7). The global NFVO subsystem should have visibility into the policies defined locally at each vPoP to be able to detect any potential global policy conflicts, e.g., a local vPoP administrator could add a local policy that violates or conflicts with a global policy. In addition, the global NFVO subsystem would benefit from being able to import the currently configured services at each vPoP. The global NFVO would use such information to monitor global policy conformance and also to facilitate detection of policy violations when new global policies are created, e.g., a global level administrator is about to add a new global policy that, if committed, would make certain already configured services a violation of the policy. The publication of subsystem service tables for consumption by a global policy engine is a concept used in the Congress [24] OpenStack [22] project.

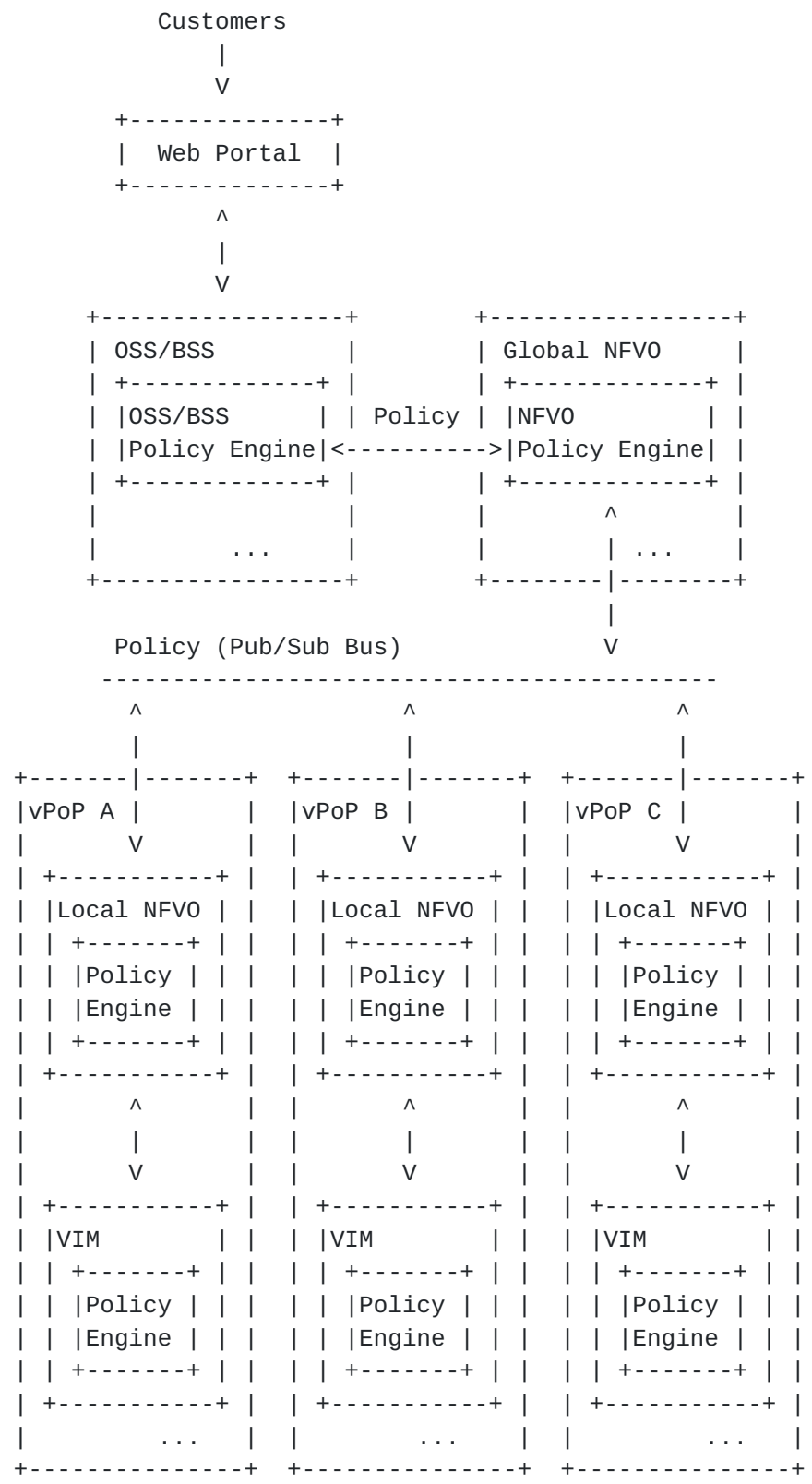

Figure 6: Simplified view of a service provider's NFV Architecture:
Multipoint Ethernet Service Example

8.2 Policy-Based NFV Placement and Scheduling

One of the goals of NFV is to allow a Service Provider (SP) offer the NFV infrastructure as a service to other Service Providers as Customers - this is called NFVIaaS [6]. In this context, it may be desirable for a Service Provider to run virtual network elements (e.g., virtual routers, virtual firewalls, and etc.) as virtual machine instances inside the infrastructure of another Service Provider. In this document, we call the former a "customer SP" and the latter an "NFVIaaS SP."

There are many reasons for a customer SP to require the services of an NFVIaaS SP, including: to meet performance requirements (e.g., latency or throughput) in locations where the customer SP does not have physical data center presence, to allow for expanded customer reach, regulatory requirements, and etc.

As VNFs are virtual machines, their deployment in such NFVIaaS SPs would share some of the same placement restrictions (i.e., placement policies) as those intended for Cloud Services. However, VNF deployment will drive support for unique placement policies, given VNF's stringent service level specifications (SLS) required/imposed by customer SPs. Additionally, NFV DCs or NFV PoPs [8] often have capacity, energy and other constraints - thus, optimizing the overall resource usage based on policy is an important part of the overall solution.

This section describes an example [15] of a global policy written in Datalog [3] applicable to compute to promote energy conservation for the NFVIaaS use case in an OpenStack framework. The goal of that global policy is to address the energy efficiency requirements described in the ETSI NFV Virtualization Requirements [9].

A related energy efficiency use case using analytics-driven policies in the context of OpenStack Congress [24] policy as a service was presented and demonstrated at the Vancouver OpenStack summit [14], where the Congress policy engine delegated VM placement to a VM placement engine that migrated under-utilized VMs to save energy.

8.2.1 Policy Engine Role in NFV Placement and Scheduling

A policy engine may facilitate policy-based resource placement and scheduling of VMs in an NFVIaaS environment. In this role, a policy engine (Figure 7) would determine optimized placement and scheduling choices based on the constraints specified by current resource placement and scheduling policies. The policy engine would evaluate such policies based on event triggers or programmable timers.

In one instantiation, a policy engine would interface with a "Measurement Collector" (e.g., OpenStack Ceilometer [23]) to periodically retrieve instantaneous per-server CPU utilization, in order to compute a table of per-server average CPU utilization. In an alternative instantiation, the Measurement Collector could itself compute per-server average CPU utilization and provide that information to the policy engine. The latter approach would reduce overhead, since it would avoid too frequent pulling of stats from the Measurement Collector.

Other average utilization parameters such as VM CPU utilization, VM Memory utilization, VM disk read IOPS, Network utilization/latency, and etc. could also be used by the policy engine to enforce other types of placement policies.

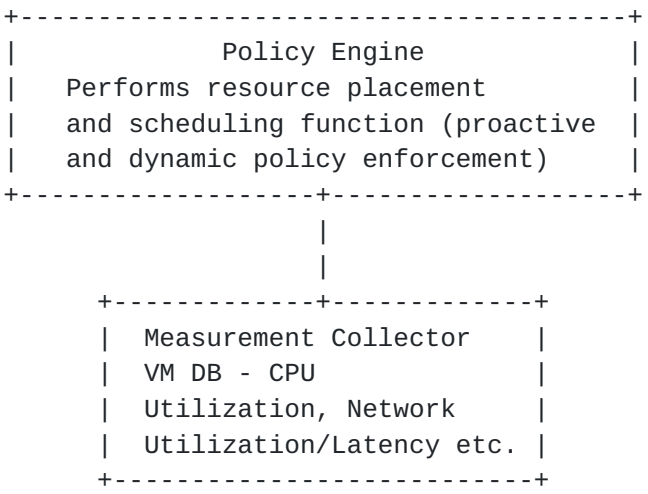

Figure 7: NFVIaaS Architecture for Policy Based Resource Placement and Scheduling

In the ETSI NFV Architectural Framework [7], the Policy Engine is part of the Orchestrator and the Measurement Collector is part of the Virtual Infrastructure Manager (VIM).

8.2.2 Policy-based NFV Placement and Scheduling with OpenStack

Consider an NFVIaaS SP that owns a multitude of mini NFV data centers managed by OpenStack [22] where:

- The Policy Engine function is performed by OpenStack Congress [24].

- The Measurement Collector function is performed by OpenStack Celiometer [23].

- The Policy Engine has access to the OpenStack Nova database that stores records of mapping of virtual machines to physical servers.

An exemplary mini NFV DC configuration is used in this example, as described below:

- 210 physical servers in 2U rack server configuration spread over 10 racks.

- 4 types of physical servers each with a different system configuration and from a particular manufacturer. It is possible that the servers are all from the same or different manufacturers. For the purpose of this example, a server "type 1" is described. Server type 1 has 32 virtual CPUs and 128GB DRAM from manufacturer x. Assume 55 physical servers of type 1 per mini NFV DC.

- 2 types of instances large.2 and large.3, which are described in Table 1. Each parameter has a minimum guarantee and a maximum usage limit.

Instance Type	Virtual CPU Units Minimum Guarantee /Maximum Usage	Memory (GB) Minimum Guarantee /Maximum Usage	Minimum/Maximum Physical Server Utilization (%)
large.2	0/4	0/16	0/12.5
large.3	0/8	0/32	0/25

Table 1: NFVIaaS Instance Types

For the purpose of this example, the Mini NFV DC topology is considered static -- the above topology, including the network interconnection, is available through a simple file-based interface.

Policy 1 (an exemplary NFV policy): In a descriptive language, Policy 1 is as follows - "For physical servers of type 1, there can be at most only one active physical server with average overall utilization less than 50%." Policy 1 is an example of reactive enforcement. The goal of this policy is to address the energy efficiency requirements described in the ETSI NFV Virtualization Requirements [9].

Policy 2 (another exemplary NFV policy): Policy 2 is designed to protect NFV instances from physical server failures. Policy 2 reads as follows in a descriptive language - "Not more than one VM of the same high availability (HA) group must be deployed on the same physical server". Policy 2 is an example of proactive and reactive enforcement.

Note that there may be cases where there may not be any placement solution respecting both policies given the current DC load. To avoid such cases, Policy 1 could be relaxed to: "Minimize the number of physical servers with average overall utilization less than 50%".

Policy 1 calls for the identification of servers by type. OpenStack Congress would need to support server type, average CPU utilization, and be able to support additional performance parameters (in the future) to support additional types of placement policies. OpenStack Congress would run the policy check periodically or based on trigger events, e.g., deleting/adding VMs. In case OpenStack Congress detects a violation, it would determine optimized placement and scheduling choices so that current placement and scheduling policy are not violated.

A key goal of Policy 1 is to ensure that servers are not kept under low utilization, since servers have a non-linear power profile and exhibit relatively higher power consumption at lower utilization. For example, in the active idle state as much as 30% of peak power is consumed. At the physical server level, instantaneous energy consumption can be accurately measured through IPMI standard. At a customer instance level, instantaneous energy consumption can be approximately measured using an overall utilization metric, which is a combination of CPU utilization, memory usage, I/O usage, and network usage. Hence, Policy 1 is written in terms of overall utilization and not power usage.

The following example addressed the combined effect of Policy 1 and Policy 2.

For an exemplary maximum usage scenario, 53 physical servers could be under peak utilization (100%), 1 server (server-a) could be under partial utilization (62.5%) with 2 instances of type large.3 and 1 instance of type large.2 (this instance is referred as large.2.X1), and 1 server (server-b) could be under partial utilization (37.5%) with 3 instances of type large.2. Call these three instances large.2.X2, large.2.Y and large.2.Z

One HA-group has been configured and two large.2 instances belong to this HA-group. To enforce Policy 2 large.2.X1 and large2.X2 that belong to the HA-group have been deployed in different physical servers, one in server-a and a second in server-b.

When one of the large.3 instances mapped to server-a is deleted from physical server type 1, Policy 1 will be violated, since the overall utilization of server-a falls to 37.5%, since two servers are underutilized (below 50%).

OpenStack Congress, on detecting the policy violation, would use various constraint based placement techniques to find placements for physical server type 1 to address Policy 1 violation without breaking Policy 2. Constraint based placement involves a convex optimization framework [5]. Some of the algorithms that could be considered include linear programming [1], branch and bound [2], interior point methods, equality constrained minimization, non-linear optimization, and etc.

Various new placements are described below:

1) New placement 1: Move 2 of three instances of large.2 running on server-b to server-a. Overall utilization of server-a - 62.5%. Overall utilization of server-b - 25%. large.2.X2 must not be one of the migrated instances.

2) New placement 2: Move 1 instance of large.3 to server-b. Overall utilization of server-a - 12.5%. Overall utilization of server-b - 62.5%.

A third solution consisting of moving 3 large.2 instances to server-a cannot be adopted, since this violates Policy 2. Another policy minimizing the number of migrations could allow choosing between solutions (1) and (2) above.

New placements 2 and 3 could be considered optimal, since they achieve maximal bin packing and open up the door for turning off server-a or server-b and maximizing energy efficiency.

To detect violations of Policy 1, an example of a classification rule is expressed below in Datalog, the policy language used by OpenStack Congress.

The database table exported by the Resource Placement and Scheduler for Policy 1 is described below:

- server_utilization (physical_server, overall_util): Each database entry has the physical server and the calculated average overall utilization.

- vm_host_mapping(vm, server): Each database entry gives the physical server on which VM is deployed.

- anti-affinity_group(vm, group): Each entry gives the anti-affinity group to which a VM belongs.

Policy 1 in a Datalog [3] policy language is as follows:

```
    error (physical_server) :-
      nova: node (physical_server, "type1"),
      resource placement and scheduler:
        server_utilization (physical_server, overall_util < 50)
```

Policy 2 (in Datalog policy language):

```
    error(vm) :-
      anti-affinity_group(vm1, grp1),
      anti-affinity_group(vm2, grp2),
      grp1 != grp2,
      nova: vm host mapping(vm1, server-1),
      nova:  vm host mapping(vm2,server-2),
      server-1 == server-2
```

9. Summary

This document approached the policy framework and architecture from the perspective of overall orchestration requirements for services involving multiple subsystems. The analysis extended beyond common orchestration for compute, network, and storage subsystems to also include energy conservation constraints. This document also analyzed policy scope, global versus local policies, policy actions and translations, policy conflict detection and resolution, interactions among policies engines, and a hierarchical policy architecture/framework to address the demanding and growing requirements of NFV environments, applicable as well to general cloud infrastructures.

The concept of NFV and the proposed policy architecture is applicable to service providers and also enterprises. For example, an enterprise branch office could have capacity and energy constraints similar to that of many service provider NFV vPoPs in constrained environments. This is an aspect that would be worth examining in detail in future work.

10. IANA Considerations

This draft does not have any IANA considerations.

11. Security Considerations

Security issues due to exchanging policies across different administrative domains are an aspect for further study.

12. Contributors

In addition to the authors listed on the front page, the following individuals contributed to the content of Section 8.2 ("Policy-Based NFV Placement and Scheduling"):

Tim Hinrichs
Styra
tim@styra.com

Ruby Krishnaswamy
Orange
ruby.krishnaswamy@orange.com

Arun Yerra
Dell Inc.
arun.yerra@dell.com

13. References

13.1. Normative References

13.2. Informative References

[1] Alevras, D. and Padberg, M. "Linear Optimization and Extensions: Problems and Solutions," Universitext, Springer-Verlag, 2001.

[2] Brassard, G. and Bratley, P., "Fundamentals of Algorithmics," .

[3] Ceri, S. et al., "What you always wanted to know about Datalog (and never dared to ask)," IEEE Transactions on Knowledge and Data Engineering, (Volume: 1, Issue: 1), August 2002

[4] "Common Information Model (CIM)," DTMF, http://www.dmtf.org/standards/cim

[5] "Convex Optimization," https://web.stanford.edu/~boyd/cvxbook/bv_cvxbook.pdf

[6] ETSI GS NFV 001 v1.1.1 (2013-10): "Network Function Virtualisation (NFV); Use Cases," http://www.etsi.org/deliver/etsi_gs/NFV/001_099/001/01.01.01_60/gs_NFV001v010101p.pdf

[7] ETSI GS NFV 002 v1.2.1 (2014-12): "Network Function Virtualisation (NFV); Architectural Framework," http://www.etsi.org/deliver/etsi_gs/NFV/001_099/002/01.02.01_60/gs_nfv002v010201p.pdf

[8] ETSI GS NFV 003 V1.2.1 (2014-12): "ETSI NFV: Terminology for Main Concepts in NFV," http://www.etsi.org/deliver/etsi_gs/NFV/001_099/003/01.02.01_60/gs_NFV003v010201p.pdf

[9] ETSI GS NFV 004 v1.1.1 (2013-10): "Network Function Virtualisation (NFV); Virtualization Requirements," http://www.etsi.org/deliver/etsi_gs/NFV/001_099/004/01.01.01_60/gs_NFV004v010101p.pdf

[10] ETSI GS NFV-INF 001 v.1.1.1 (2015-01): "Network Function Virtualisation (NFV); Infrastructure Overview," http://www.etsi.org/deliver/etsi_gs/NFV-INF/001_099/001/01.01.01_60/gs_NFV-INF001v010101p.pdf

[11] ETSI NFV White Paper: "Network Functions Virtualisation, An Introduction, Benefits, Enablers, Challenges, & Call for Action," http://portal.etsi.org/NFV/NFV_White_Paper.pdf

[12] Figueira, N. and Krishnan, R., "SDN Multi-Domain Orchestration and Control: Challenges and Innovative Future Directions," CNC VIII: Cloud and Multimedia Applications, IEEE International Conference on Computing (ICNC), February 2015

[13] Grit, L. et al., "Virtual Machine Hosting for Networked Clusters: Building the Foundations for "Autonomic" Orchestration," Virtualization Technology in Distributed Computing, 2006. VTDC 2006.

[14] Krishnan, R. et al., "Helping Telcos go Green and save OpEx via Policy", Talk and demo at the Vancouver OpenStack summit. Video Link: https://www.openstack.org/summit/vancouver-2015/summit-videos/presentation/helping-telcos-go-green-and-save-opex-via-policy

[15] Krishnan, R. et al., "NFVIaaS Architectural Framework for Policy Based Resource Placement and Scheduling," https://datatracker.ietf.org/doc/draft-krishnan-nfvrg-policy-based-rm-nfviaas/

[16] Lee, S. et al., "Resource Management in Service Chaining", https://datatracker.ietf.org/doc/draft-irtf-nfvrg-resource-management-service-chain/

[17] Moore, B., et al., "Information Model for Describing Network Device QoS Datapath Mechanisms," RFC 3670, January 2004

[18] Moore, B. et al., "Policy Core Information Model -- Version 1 Specification," RFC 3060, February 2001

[19] "OpenDaylight Group Based Policy," https://wiki.opendaylight.org/view/Project_Proposals:Group_Based_Policy_Plugin

[20] "OpenDaylight Network Intent Composition Project," https://wiki.opendaylight.org/index.php?title=Network_Intent_Composition:Main#Friday_8AM_Pacific_Time

[21] "OpenDaylight SDN Controller," http://www.opendaylight.org/

[22] "OpenStack," http://www.openstack.org/

[23] "OpenStack Celiometer," http://docs.openstack.org/developer/ceilometer/measurements.html

[24] "OpenStack Congress," https://wiki.openstack.org/wiki/Congress

[25] "OpenStack Neat," http://openstack-neat.org/

[26] "OpenStack Neutron," https://wiki.openstack.org/wiki/Neutron

[27] "Policy Framework Working Group," IETF, http://www.ietf.org/wg/concluded/policy.html

[28] Westerinen, A. et al., "Terminology for Policy-Based Management," RFC 3198, November 2001

Acknowledgements

The authors would like to thank the following individuals for valuable discussions on some of the topics addressed in this document: Uwe Michel, Klaus Martiny, Pedro Andres Aranda Gutierrez, Tim Hinrichs, Juergen Schoenwaelder, and Tina TSOU.

Authors' Addresses

Norival Figueira
Brocade Communications Systems, Inc.
nfigueir@Brocade.com

Ram (Ramki) Krishnan
Dell
Ramki_Krishnan@Dell.com

Diego R. Lopez
Telefonica I+D
diego.r.lopez@telefonica.com

Steven Wright
AT&T
sw3588@att.com

Dilip Krishnaswamy
IBM Research
dilikris@in.ibm.com