A Model Driven Intelligent Orchestration Approach to Service Automation in Large Distributed Infrastructures*

Xi Yang
University of Maryland
College Park, Maryland
USA
maxyang@umd.edu

Tom Lehman
University of Maryland
College Park, Maryland
USA
tlehman@umd.edu

Raj Kettimuthu
Argonne National Laboratory
Argonne, Illinois
USA
kettimut@anl.gov

Linda Winkler
Argonne National Laboratory
Argonne, Illinois
USA
winkler@mcs.anl.gov

Eun-Sung Jung
Hongik University
Seoul
South Korea
ejung@hongik.ac.kr

ABSTRACT

Today's scientific computing applications and workflows operate on heterogeneous and vastly distributed infrastructures. Traditional human-in-the-loop service engineering approach met its insurmountable challenge in dealing with these very complex and diverse networked systems, including conventional and software defined networks, compute, storage, clouds and instruments. Orchestration is the key to integrate and coordinate the networked multi-services and automate end-to-end workflows. In this work, we present a model driven intelligent orchestration approach to this end-to-end automation, which is built upon a semantic modeling solution that supports the full stack of service integration, orchestration, abstraction, and intent and policy representation. We also present the design of a real-world orchestrator called StackV that is able to accommodate highly complex application scenarios such as Software Defined ScienceDMZ (SD-SDMZ) and Hybrid Cloud Inter-Networking (HCIN) by implementing this approach.

CCS CONCEPTS

• **Networks systems** → **Network Services**; *Programmable Networks*; Cloud Computing • **Architectures** → Distributed architectures

AI-Science'18, June 11, 2018, Tempe, AZ, USA © 2018 Association for Computing Machinery. ACM ISBN 978-1-4503-5862-0/18/06...$15.00
https://doi.org/10.1145/3217197.3217207

KEYWORDS

ACM proceedings, text tagging

ACM Reference format:

X. Yang, T. Lehman, R. Kettimuthu, L. Winkler, E. Jung. *AI-Science'18*, June 11, 2018, Tempe, AZ, USA © 2018 Association for Computing Machinery. ACM ISBN 978-1-4503-5862-0/18/06. https://doi.org/10.1145/3217197.3217207

1 INTRODUCTION

More and more information technology consumers, including those for large scientific computing workflows, are looking for integrated services through single or consolidated channels, interfaces or portals. The practices of procuring separate compute, storage and network services and then plumbing in-house integration are becoming obsolete. This trend has firstly been manifested by the success of public cloud platforms such as Amazon AWS, where consumers can buy a wide range of information services and use them in integrated fashion through only a few mouse clicks. Fueled by growing interest from a wide spectrum of service providers, this trend will continue to proliferate through the entire information technology ecosystem.

One critical part of the ecosystem is the underlying infrastructures, a.k.a. cyberinfrastructures, which include the Internet, telecommunication networks, high-performance computing and storage systems, public and enterprise clouds, data centers, scientific instruments, etc. Built upon these infrastructures currently are diverse and heterogeneous networked multi-services. Both the commercial IT and scientific research communities have emerging advanced applications that increasingly revolve around the integration of big data, computation, and high performance networking. Their highly complex workflows and demanding performance requirements pose great challenges for these traditional services. At University of Maryland/Mid-Atlantic Crossroads (UMD/MAX) and Argonne National Laboratory (ANL), we were funded by Department of Energy under the Resource Aware Intelligent Network Services (RAINS) project to develop a collaborative, multi-service orchestration system. In this work, the definition of Orchestration is to provide intelligence to integrate and automate these networked multi-services into simple, abstract forms that support a wide range of high demanding science applications and workflows.

The key to realize this goal is technology that enables flexible, owner controlled resource descriptions. We refer to this as Modeling (Description and Abstraction). This is important because collaborative distributed systems will need to use compatible methods, semantics, and syntax for resource description to allow the higher-level orchestration functions to reason about relationships between resources and services. This is what will ultimately provide the value added services for applications workflows.

Two challenges we face in realizing the orchestration goal are control automation and distributed coordination. Automation in any complex system requires forming a control loop. In one direction, control operation results in state changes in the infrastructures. In the other, control feedback, a.k.a. telemetry, is desired to provide state awareness back to the orchestration layer. Unified resource modeling can provide semantics in both directions. With a proper level of abstraction, the orchestration intelligence can learn dynamic resource and service states and create new services with reduced chance of conflict and better efficiency. The same modeling semantics can also serve to synchronize the orchestration intent to resource and service states in the underlying infrastructures and thus close the control loop. We call this a model driven intelligent orchestration approach. Applying this approach helps solve the challenge of distributed coordination as well. When all resource owners use unified, extensible models to describe their resources and services and make state changes, the interface can be greatly simplified. We then effectively create a thin-API to introduce universal programmability to all the parties. Any parties can engage in free-form provider-consumer relationships for any As-a-Service transactions and thus decentralize the service integration, orchestration and instantiation processes. In a sense, "modeling is the starting point for everything".

A full-stack model driven orchestration approach provides a general solution for a variety of use cases. One use case is Software Defined Science DMZ (SD-SDMZ). UMD/MAX developed the SD-SDMZ as a pilot for providing flexible edge services that included traditional data transfer functions. Built upon technologies similar to those used in modern data centers, this allows scientific computing users to dynamically instantiate resources such as Data Transfer Nodes (DTN), networked storage, Globus End Points and fast data paths in dedicated virtual network enclosures. SD-SDMZ offers an "As-a-Service" model in a "cloudified" multi-tenancy environment that can be shared by a well networked region instead of by a single campus. Another use case is Hybrid Cloud Inter-Networking (HCIN) which bridges multi-provider cloud resources including cloud direct connects, hypervisor bypass interfaces, on-premise compute clusters and software defined local and inter-cloud networks. The HCIN services co-allocate these resources and bring the public and private clouds and networks together to provide on-demand high-performance hybrid cloud experience to end users. The use cases are many, including but not limited to end-to-end networking, data movement and storage co-scheduling, scientific HPC workflow, and multi-cloud orchestration. The scope of resources ranges from commodity network, compute and storage infrastructures, to on-premise facilities, to purpose-built instruments, to datasets and applications, and to custom DevOps procedures. Generalizability of this approach comes from the full-stack modeling that uses consistent solution for resource and service description, for service abstraction and intent representation and for orchestration intelligence in between.

We implemented a full-stack model driven orchestrator called StackV for real-world service operations. The modeling solution in StackV is based on Semantic Web standards, namely W3C Resource Description Framework (RDF) [1] and Web Ontology Language (OWL) [2]. Southbound and northbound APIs and orchestration intelligence are designed around a common set of RDF/OWL ontologies. StackV uses the model data in its native data structure without having to bind to fixed schema. Service orchestration and instantiation uses hot-pluggable driver modules, two-phase commit distributed transactions and model based dynamic computation workflows. These make the system highly extensible and scalable. In this work, we will present some architectural details and walk through the design with practical use cases.

2 MODELING NETWORKED RESOURCES AND SERVICES

2.1 Multi-Resource Markup Language (MRML)

Through the RAINS project we developed the Multi-Resource Markup Language (MRML) [3]. MRML is a network centric multi-resource service modeling solution, extended from the Network Markup Language (NML) [4]. Networks are the central components of information infrastructures. Network topological structures are well suited for graph based modeling, as manifested from day one of network research. NML is a standard developed by the Open Grid Forum (OGF) for describing network resources and services. It supports both XML and RDF/OWL representations. In RDF/OWL form, it introduces an NML ontology that defines concepts such as Topology, Node, Port, Link, SwitchingService and AdaptationService as well as the relationships for interconnecting these resources and services. Our work through an earlier DOE project extended NML for multi-layer networks [5]. Aiming at describing heterogeneous cyberinfrastructures, MRML defines an extensible ontology for network, compute, storage and interconnects by further extending the NML.

Compared to other modeling languages such as YANG [6], MRML has some unique features. YANG is specialized in network configuration and state manipulation. It is often used as a southbound data modeling solution paired with the NETCONF protocol [7]. In comparison, MRML is a protocol-neutral full-stack solution, meaning its models can be used for both southbound and northbound as well as for internal reasoning and computation.

2.2 Service-Oriented Modeling Stack

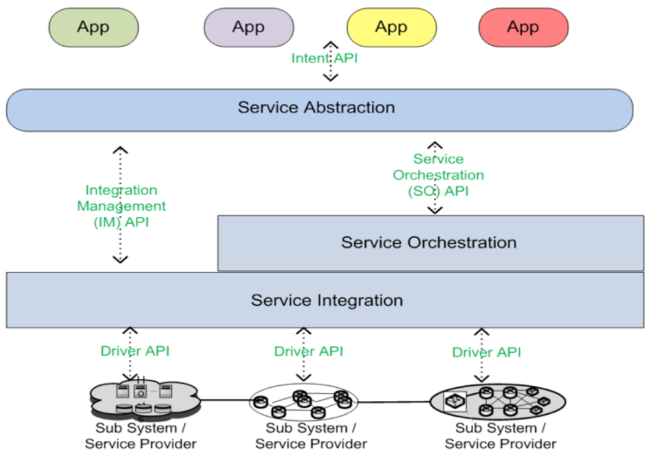

App
App
App
App
Intent API
Service Abstraction
Integration Management (IM) API
Service Orchestration (SO) API
Service Orchestration
Service Integration
Driver API
Driver API
Driver API
Sub System / Service Provider
Sub System / Service Provider
Sub System / Service Provider

Figure 1: Service-oriented modeling stack illustrated by StackV system function layers

In practice, we use MRML for resource and service management, instantiation, integration, orchestration, abstraction, intent and policy, all the way from bottom up to the very top of the stack. We use MRML to carry semantics at every layer that provides programmability through an API. Figure 1 illustrates the service-oriented modeling stack in contingency with function layers of a real-world orchestrator StackV that we will further describe in later sections. The Service Integration layer interacts with different sub-systems, namely service providers and domains, through Driver API, to pull their resource models and push model change “deltas”. Through this layer, sub-system models are integrated into a service integration model, a.k.a. system model, for the entire infrastructure and exposed as “full view” to service Integration Management API. It also conducts two-phase commit distributed transactions to decompose and push a system model “delta” to related sub-systems. At the top, the Service Abstraction layer interprets service intent as presented by an application into a service abstraction model using the MRML ontologies. For a complex service, the Service Abstraction layer will work with the Service Orchestration layer to compile and compute the service abstraction into detailed system model delta and push down to the Service Integration layer for instantiation. The API also provides methods to revert, modify and verify a service through its life cycle. These are all model driven, automated and programmability-ready to integrate into applications and DevOps procedures.

2.3 Service Integration Modeling

The MRML ontologies and future extensions are meant to describe arbitrary information infrastructures and services. Currently we use them to model multi-layer local and wide area networks, SDN, public and private clouds, attached network fabric, HPC clusters and parallel storage systems. Models of these diverse subsystems are generated in real time by pluggable modules and assembled into integrated model for the entire infrastructure in the Service Integration layer.

As an example UMD/MAX is a regional optical network with multi-100G capabilities at Layers 1, 2 and 3. This includes a 10G DirectConnect to the Amazon AWS US East Region. There is also an on-premise OpenStack cloud, which has compute nodes equipped with Single Root I/O Virtualization (SR-IOV) interfaces for hypervisor bypass. We are able to create dedicated layer-2 paths between the public and private clouds for up to 10Gbps. As illustrated in Figure 2, we use StackV to integrate these subsystems using three pluggable drivers: AWS Driver, OpenStack Driver and Generic SDN Driver. Through standard RDF/OWL rendering, the MRML models pulled from the three drivers are combined into a complete MRML model, which provides all the details of resources and services for the entire infrastructure.

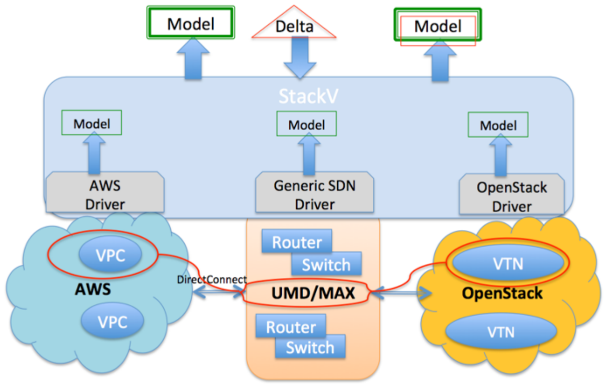


Figure 2: Service integration modeling for cloud inter-networking

Here we only describe at high level the MRML model pulled from the Generic SDN Driver to illustrate the general modeling solution. We model Layer-2 connectivity of the UMD/MAX network as one Topology. The topology model consists of Node and BidirectionalPort instances. All border ports are included in a single SwitchingService, which provides SwitchingSubnet instances to represent dynamic VLAN circuits. Edge ports peering with AWS DirectConnect and OpenStack SR-IOV interfaces have the connectivity described through a property isAlias. This allows the Service Integration layer to connect the three subsystem models into a whole. In this work we have used several ontology elements, whose meanings are straightforward by names. Detailed ontology definitions can be found in [3] and [4].

2.4 Service Abstraction Modeling

We use the same modeling to describe the services for end users and applications, only that the description is about abstract resources that reflect high level intent. An intent represents users definition of a service instance that is customized to support a specific set of application functions for a specific service life cycle. This users definition is in a domain-specific, human-readable, format and simplified language, which may vary in syntax and semantic form. These different forms can then be translated into a unified abstraction using MRML. The MRML ontologies are well designed to accommodate abstract, virtualized and hierarchical resources and services which can apply to any system layers. This is why we call it a "full-stack" modeling solution. As part of the modeling solution, we developed a Simple Policy Annotation (SPA) ontology [8] to describe the user intent for service orchestration policies, such as resource inter-dependency, workflow order and service chaining. SPA is a component ontology of MRML, and is only used to facilitate orchestration computation at abstract level. For example, it can define a policy that applies to a virtual server, with constraints that specify where and how to schedule this server during the orchestration.

On top of the internetworked cloud infrastructures, we create a virtual inter-cloud network that consists of AWS Virtual Private Cloud (VPC), OpenStack Virtual Tenant Network (VTN) and dedicated VLAN connection across the SDN in between. The virtualization is illustrated as the connected red bubbles in Figure 2. This virtual inter-cloud network "intent" is represented as an MRML Topology in Service Abstraction modeling. The Topology consists of two sub-level Topology instances one for VPC and the other for VTN, each having its own Node, BidirectionalPort, SwitchingService, SwitchingSubnet, RoutingService and RoutingTable instances and other elements, all in abstract forms, as none is available in the detailed MRML model at the Service Integration layer. The SPA statements are added to this abstraction model to describe the following intent policies: (a) Allocate VPC and VTN with contained compute, storage and network resources, (b) Compute a VLAN path between AWS DirectConnect and OpenStack cluster switch, (c) Stitch VPC gateway to a DirectConnect VLAN, (d) Stitch virtual machine SR-IOV interfaces to OpenStack cluster switch VLAN, and (e) The VPC gateway, SR-IOV interfaces and VLAN(s) in c and d depend on the results of a and b. Based on the MRML and SPA ontologies the service abstraction modeling thus describes the service intent in standard RDF/OWL format, which is consistent to other layers in the stack.

3 STACKV: MODEL DRIVEN SERVICE ORCHESTRATION

3.1 Service Orchestration Workflow

StackV is a general-purpose orchestrator for networked multi-services and is implemented based on the full-stack model driven intelligent orchestration approach. The software is written in Enterprise Java and deployed as a suite of Web Services. It provides APIs at all the model driven function layers as previously illustrated in Figure 1. Figure 3 shows the service orchestration workflow of StackV. From very top of the stack, applications communicate to the orchestrator with highly abstract service request, namely Intent. The StackV provides both GUI based Web Portal and REST API to help applications describe the intent. This is called "Service Model Description and Abstraction". Outcome of the procedure is a formal MRML model that consists of abstract resources annotated with SPA policy statements. The abstract model data are then fed to a Dynamic Compile procedure and compiled into a model based computation workflow. A computation workflow consists of a variety of Model Computation Elements (MCE) as intelligence functions assembled into an execution tree. Each MCE uses system model data, service model data and policy data as input and accomplishes a specific function such as resource placement and connection computation. The output will be more detailed service model data, which could be used as input for another MCE. When the computation workflow finishes successfully, a System Model Delta will be created that provides detailed model statements about what need to change in the underlying infrastructures to satisfy the intent.

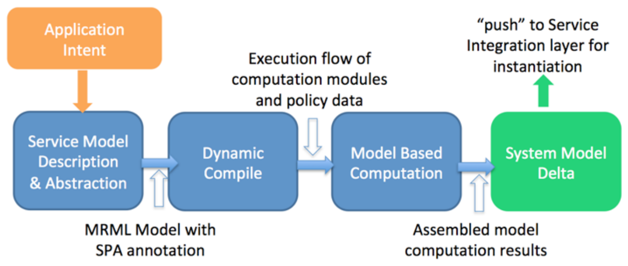

Application Intent
Execution flow of computation modules and policy data
"push" to Service Integration layer for instantiation
Service Model Description & Abstraction
Dynamic Compile
Model Based Computation
System Model Delta
MRML Model with SPA annotation
Assembled model computation results

Figure 3: StackV service orchestration workflow

3.2 Application Intent and Policy

In StackV the Intent API is responsible for interpreting application intent and policy into service abstraction model. StackV front end provides both web portal and REST API to support extensible intent as a service. For example, it supports the intent of layer-2 VLAN connection through a Dynamic Network Connection (DNC) service, and virtual cloud with gateway to external layer-2 stitching point through a Virtual Cloud Network (VCN) service that consists of on-demand networked storage and DTN nodes. Both are application scenarios of the SD-SDMZ use case.

Using VCN as an example, an application intent specifies the cloud provider identifier, subnets, placement of VMs in subnets, attached Internet and private gateways, custom routes, and optionally external layer-2 stitching port. The intent is firstly translated into abstract resource statements based on the MRML modeling. Then the SPA based policy statements are added to augment the service abstraction model. The resulting abstraction model typically consists of Policy Action, Policy Data and Policy Dependency statements. In the VCN service abstraction, policy actions include virtual cloud creation, VM placement, storage allocation and mount, Globus end point installation, dedicated layer-2 network connection and virtual cloud to external network stitching. The policy data include the user data for resource dimensioning such as number of subnets, VMs, and amount of storage, and the intermediate variables for output of one policy action into another policy action. For example, the VM placement action will import policy data from the virtual cloud creation action to learn available subnets for the placement. The policy dependency statements define how an abstract resource depends on the policy actions and how policy actions depend on each other.

3.3 Model Based Computation

Based on the modeling stack described in Section II.B., the StackV service orchestration uses model based computation to transform a service abstraction model into a service integration model, Only the latter can be used to drive actual resource (de)allocation and states change. StackV leverages the open source Apache Jena [9] libraries to handle all its model data in native RDF/OWL format. The benefits of model based computation include eliminating conversion between interface, internal and persistence data structures, leveraging standard tools for data query, navigation, transformation and reasoning, and maintaining consistent data semantics through all the computation modules.

Model Computation Element (MCE) is the basic computation module. The input and output of an MCE are both model data based on RDF/OWL, MRML and SPA ontologies. In the compiled computation execution workflow, each MCE instance computes for a specific purpose. We still use VCN as an example. A virtual cloud creation MCE takes in the initial service abstraction model and resource dimensioning policy data. It maps the abstract virtual cloud topology under a specific cloud provider topology found in the full system model from the Service Integration layer. Then the MCE creates the model statements for adding a virtual cloud, contained subnets, routes and gateways to an updated service abstraction model and exports it together with some intermediate policy data. Depending MCEs such as the VM placement and layer-2 connection then use this new service abstraction model (more detailed than the original one) and policy data as input to perform their own computation. StackV has implemented sophisticated logic to concatenate MCEs and merge computation results. The basic idea of this technique is to use SPARQL [11] queries to "shape" the output of a upper stream MCE into custom JSON format and use JSONPath [12] queries to extract information and "fit" to the input of downstream MCEs. Success in finishing the computation workflow means StackV has resolved all model abstractions and policy annotations in the final product, and has turned an application intent into a System Model Delta. This "delta" (as shown in Figure 1) can be pushed down to the Service Integration layer for instantiation.

4 APPLICATION SCENARIOS CASE STUDY

The StackV system provides a thin-API that is syntax simple and semantics rich for service orchestration. Users can use either the StackV front end or their own code to interpret service intent into service abstraction model in uniform format. This gives them the flexibility to represent arbitrary service requests. In practice, we created a few abstraction templates for common service requests so users can quickly sketch out a request and then extend, tailor or change to any combination of resources, connectivity, and dependencies they desire. The orchestration layer is very general (only limited by the types of available MCEs) in handling the variations. In this work, we provide detailed case study of application scenarios for two use cases, SD-SDMZ and HCIN, that we have practiced in real world operations. Compared to the typically manual, time consuming operations for these applications scenarios, StackV can have services ready in a few minutes while providing users with better awareness of both the services and supporting infrastructures.

4.1 Software Defined Science DMZ (SD-SDMZ)

The traditional Science DMZ architecture is based on bare metal servers providing a fixed set of services typically focused on data movement between HPC facilities. This Science DMZ concept has been proven a well thought out architecture that has greatly facilitated data movement services at a large number of university campuses and Department of Energy laboratories. However, we identified a class of users, and associated services, where this fixed architecture and pre-defined service model was not as flexible as desired for all use cases. As a result, the Software Defined Science DMZ (SD-SDMZ) concept was developed. The SD-SDMZ is a re-imagining of the traditional Science DMZ based on virtualization and cloud technologies. Built upon technologies similar to those used in modern data centers, the SD-SDMZ provides scalable, multi-tenancy environment, where each user or user group can allocate dedicated resources and use the resources in an isolated, independent and elastic fashion. In addition external resources such as direct connections to public clouds along with research and education network services from Internet2 and ESnet are integrated. The result is an edge facility where users can obtain traditional Science DMZ data transfer service, and/or have specialized topologies built which integrate on-premise cloud based compute/storage, and external connections to public clouds, local HPC, or other remote destinations. Embedding this resource in the University of Maryland (UMD) Mid-Atlantic Crossroads (MAX) regional network allows the SD-SDMZ to leverage rich connectivity to advance cyberinfrastructure to provide new and flexible services.

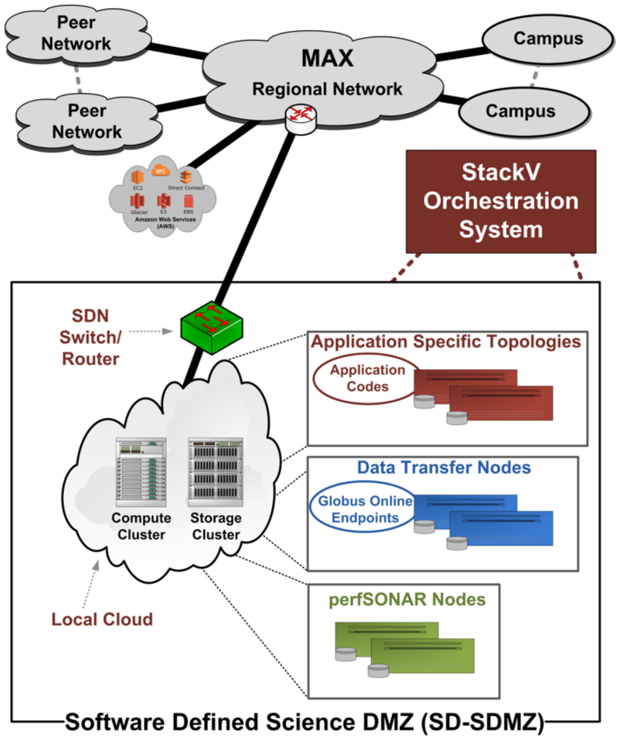
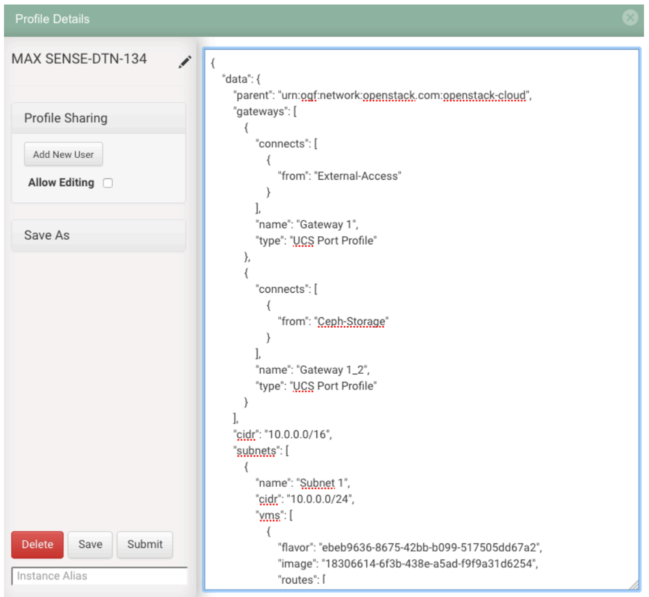


Figure 4: StackV backed SD-SDMZ at UMD/MAX

UMD/MAX has deployed a ScienceDMZ for regional network connectors to utilize. Motivated by a desire to have a flexible and extensible infrastructure, we based this upon a SDN and Cloud technologies. This includes OpenStack [10] driven Cisco Unified Computing System (UCS) clusters, Ceph [13] parallel filesystem, SDN enabled network elements, and direct integration with local HPC high performance filesystem. This UMD ScienceDMZ provides all of the standard ScienceDMZ services including DTNs, Globus Endpoints, integration with local HPC, high-speed layer3/layer2 flow termination, and a security perimeter. It has the capability to terminate 100G layer2/layer3 flows from Interent2 or ESnet to research project specific virtual machine topologies with high speed SR-IOV [14] interfaces to storage and external network locations. By leveraging the StackV intelligent orchestration solution, we have transformed this into an SD-SDMZ infrastructure as depicted in Figure 4. Any UMD/MAX user within our regional network can create dedicated virtual SDMZ services on demand through the StackV self-service portal.

As a specific service instance example, we have created a DTN cluster with five VM based DTNs for UMD campus researchers. Each DTN node has 16 cores, 32GB memory, a management network interface and two 10G SR-IOV interfaces. One SR-IOV interface faces external 100G network and binds to auto-registered Globus endpoint. The other faces internal Ceph storage network that mounts CephFS. With a predefined service profile, this DTN cluster with 5x10G network capability and parallel networked file system was created in a few minutes. We automate Kerberos based authentication on all DTN nodes, tied with the Globus authentication so that authorized users can access the storage and data transfer services instantaneously.

We have also predefined “scale-out” service profiles that help create additional DTN instances, as partly illustrated in Figure 5.



```
{
  "data": {
    "parent": "urn:ogf:network:openstack.com:openstack-cloud",
    "gateways": [
      {
        "connects": [
          {
            "from": "External-Access"
          }
        ],
        "name": "Gateway 1",
        "type": "UCS Port Profile"
      },
      {
        "connects": [
          {
            "from": "Ceph-Storage"
          }
        ],
        "name": "Gateway 1_2",
        "type": "UCS Port Profile"
      }
    ],
    "cidr": "10.0.0.0/16",
    "subnets": [
      {
        "name": "Subnet 1",
        "cidr": "10.0.0.0/24",
        "vms": [
          {
            "flavor": "ebeb9636-8675-42bb-b099-517505dd67a2",
            "image": "18306614-6f3b-438e-a5ad-f9f9a31d6254",
            "routes": [
```

Figure 5: A “scale-out” DTN service profile in StackV portal that serves the UMD/MAX SD-SDMZ

These profiles will have SR-IOV interfaces sharing network connectivity with the existing cluster. Whenever the five-node cluster gets too crowded, we can instantiate new service instances that add DTNs nodes to linearly scale the cluster until reach the limit of external network and internal storage.

4.2 Hybrid Cloud Inter-Networking (HCIN)

Cloud inter-networking in most cases is used to connect a public cloud with a private cloud to form application specific virtualization, namely hybrid cloud service. This requires co-allocation of cloud resources at both clouds and network resources in between to stitch them together. This is complicated by the diverse set of services and technologies, including Amazon Web Services (AWS) DirectConnect, wide and local area networks, cloud network fabric and SR-IOV interfaces. In Section II.C, we described a HCIN infrastructure we operate at UMD/MAX. As a real-world application, we have provided domain scientists a HCIN hybrid cloud solution. With the model driven orchestration system, we are moving from manual provisioning and stitching of various resources to fully automated services. In one application scenario, a scientist wanted to run large-scale simulations using Hadoop cluster. They started with a few high-end local nodes for prototyping but desired many more in the hybrid cloud for production work loads. A HCIN hybrid cloud service with high performance compute, storage and network resources are ideal for them to achieve the elasticity around prototyping and production cycles. Figure 5 shows a diagram of the abstraction model we composed based on their intent and sent to the service orchestration layer for compile and execution. In this example, the execution order of MCEs is below:

1. Virtual_Cloud_Network MCEs compute model statements for adding AWS and OpenStack virtual networks, subnets, gateways and routes.
2. VM_Placement MCEs add statements to place VMs into specific AWS subnets or to OpenStack subnets and hosts.
3. L2_Path_Computation MCE computes layer2 path and add statements for all the VLANs on the switches, DirectConnect and UCS fabric between AWS VPC and OpenStack hosts from 1 and 2.
4. SRIOV_Stitching uses the results of 2 and 3 to add model statements for OpenStack VM SR-IOV interfaces and for their connectivity to layer2 VLAN path.
5. In parallel to 4, DirectConnect_Stitching uses the results of 2 and 3 to to add statements for AWS VPC to attach to a DirectConnect VLAN and then to the layer2 path.
6. NFV_Quagga_BGP MCE adds model statements for deploying Quagga [15] based virtual router on an OpenStack VM Linux using the results of 4 and 5.
7. Networked_Block_Storage MCE adds model statements for creating Ceph [13] Block Devices (RBD) and attaching and mounting to designated VMs with traffic routed to SR-IOV interfaces. This depends on 2, 4 and 6.

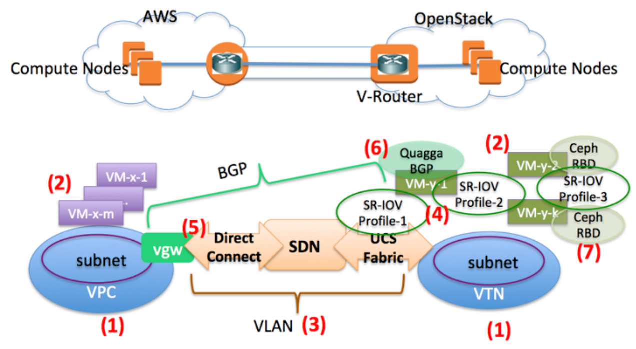


Figure 6: An example HCIN hybrid cloud service intent (top) and abstraction model with orchestration workflow annotations (bottom: the red numbers correspond to execution order of MCEs).

In the application scenario illustrated in Figure 6, an OpenStack VM is allocated as the master Hadoop node and all other nodes in the OpenStack and AWS clouds are slave nodes. The master VM also serves as the virtual router to bring together AWS and OpenStack VMs from separate networks through BGP. This creates a dedicated, secure virtual infrastructure contained by layer-2/layer-3 isolation. Between the compute and storage servers are SR-IOV, Quality of Service (QoS) enforced 10G/100G networks and AWS DirectConnect. This provides guaranteed high bandwidth and low latency that facilitate deterministic I/O performance. From users and applications perspective, these bring total transparency and on-premise experience, as consistent to what they typically have in local high-performance clusters.

4.3 Service Topology for Data Transfer

As an example orchestrated service, the SD-SDMZ DTN and HCIN functions can be combined to instantiate a service topology focused on data transfer. The result can be a custom configuration of DTNs, storage system connections, and public cloud resource integration over direct network paths. Standard data movement systems, such as Globus [16] can then be utilized for data transfers. Figure 7 depicts an example of such a service topology. These types of topologies could be built for dedicated use by a single user. In this mode of operation, SD-SDMZ infrastructure resources would be released for others after work is complete. In another mode of operation, this topology may be run by the facility operators and made available on a more general use basis. For this case, the orchestration functions would be utilized for dynamic scaling of service resources based on overall facility load and expected activity.

5 CONCLUSIONS

In this work we presented a model driven intelligent orchestration approach to service automation for large distributed infrastructures. This approach provides a viable path to building general purpose service orchestrators on top of today's increasingly complex and diverse networked systems. We leveraged the Semantic Web standards and developed the Multi-Resource Markup Language and Simple Policy Annotation to provide a formal modeling solution. This modeling solution is extensible and capable of integrating a wide spectrum of resources and services from many service providers and network domains. Based on the modeling foundation, we have developed an intelligent orchestration solution that uses pluggable Model Computation Elements (MCE) to construct flexible intelligence workflows that can solve a large set of end-to-end computation, co-scheduling and automation problems, and can be extended to handle a lot more.

This approach uses the same modeling solution for a full stack of service functions including intent interpretation, service abstraction, service orchestration, computation and integration. This means high-level application intent can be translated into low-level resource state changes with consistent semantic representation and rendering through all the function layers. At the same time, applications gain awareness of low-level services and infrastructures bottom up via the same modeling stack. The model based telemetry data can be fed into the control intelligence and operation analytics. This completes the control-feedback loop and makes continuous automation and optimization possible. We believe this represents a promising direction for future networked multi-services management and operation.

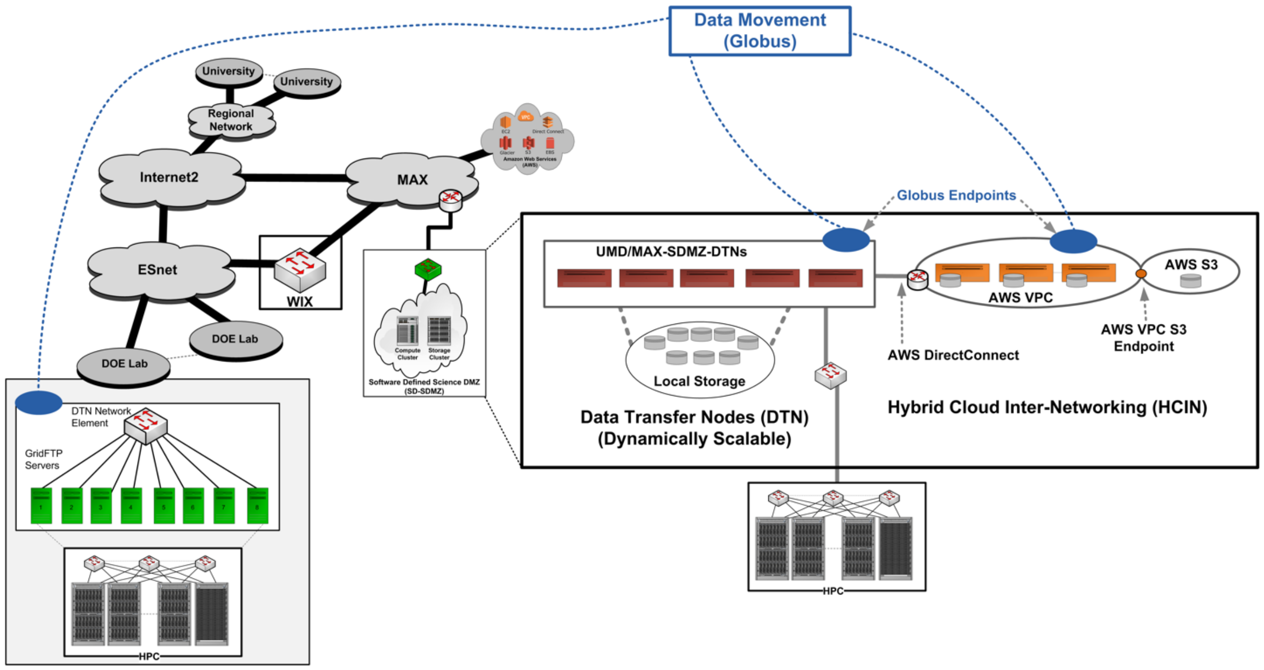


Figure 7: Example Service Topology with focus on Data Movement

In this work we also presented our full-stack model driven orchestrator called StackV. We walked through architectural details of its service orchestration workflow, application intent and policy handling and model based computation processes. Through case study of SD-SDMZ and HCIN application scenarios, we demonstrated that real-world implementation of the model driven intelligent orchestration approach is not only possible but also brings practical benefits to our daily operations.

ACKNOWLEDGMENTS

This work was supported in part by the U.S. DOE Office of Science under award numbers DE-SC0010716 and DEAC02-06CH11357, entitled "Resource Aware Intelligent Network Services (RAINS)", and by U.S. National Science Foundation under award number 1659356, entitled "CC* Integration: Regional Embedded Cloud for As-a-Service Transformation (RECAST)".

REFERENCES

[1] World Wide Web Consortium, "Resource Description Framework (RDF)", W3C Recommendation. Available at: http://www.w3.org/RDF/.
[2] World Wide Web Consortium, "Web Ontology Language (OWL)", W3C Recommendation. Available at: www.w3.org/2004/OWL/.
[3] RAINS Project, "The Multi-Resource Markup Language (MRML) Ontology - Extension of NML", Available at: https://github.com/RAINS-Project/nml-mrs-model/blob/master/schema/rdf-owl/nml-mrs-ext-v2.owl
[4] J. van der Ham, F. Dijkstra, R. Lapacz, J. Zurawski, "Network Markup Language Base Schema version 1", OGF GFD-R-P.206, May 2013.
[5] T. Lehman, X. Yang, N. Ghani, et al., "Multilayer Networks: An Architecture Framework," IEEE Communications Magazine, 49-5, May 2011.
[6] Internet Engineering Task Force, "YANG - A Data Modeling Language for the Network Configuration Protocol (NETCONF)", IETF RFC 6241, October 2010.
[7] Internet Engineering Task Force, "Network Configuration Protocol (NETCONF)", IETF RFC 6020, June 2011.
[8] RAINS Project, "The Multi-Resource Service (MRS) Simple Policy Annotation (SPA) Ontology", Available at: https://github.com/RAINS-Project/nml-mrs-model/blob/master/schema/rdf-owl/mrs-spa-v1.owl.
[9] Apache Software Foundation, "Apache Jena: A free and open source Java framework for building Semantic Web and Linked Data applications", Available at https://jena.apache.org.
[10] OpenStack, "OpenStack Open Source Cloud Computing Software," Avaialbe at: http://www.openstack.org/software/
[11] SPARQL Query Language for RDF, https://www.w3.org/TR/rdf-sparql-query/
[12] JSONPath - XPath for JSON, http://goessner.net/articles/JsonPath/
[13] S. Weil, S. Brandt, E. Miller et al., "Ceph: A Scalable, High-Performance Distributed File System", in OSDI, pp. 307-320, 2006.
[14] Intel Corp. whitepaper, "PCI-SIG SR-IOV Primer: An Introduction to SR-IOV Technology," http://www.intel.com/content/www/us/en/pci-express/pci-sig-sr-iov-primer-sr-iov-technology-paper.html
[15] Quagga Routing Software Suite, Available at: http://www.quagga.net.
[16] Globus, Research Data Management Services, http://www.globus.org