DEPO: A Platform for Safe DEployment of POlicy in a Software Defined Infrastructure

Aisha Syed (University of Utah)  
Vijay Gopalakrishnan (AT&T Research)  
Bilal Anwer (AT&T Research)  
Jacobus Van der Merwe (University of Utah)

Presented at ACM Symposium on SDN Research (SOSR), April 3-4, 2019, San Jose, CA, USA
Motivation

• With SDN and in-network clouds enabling NFV, we see trend towards a (mobile) software-defined infrastructure (SDI)
  o being embraced by telcos, network service providers, equipment vendors
Motivation

• With SDN and in-network clouds enabling NFV, we see trend towards a (mobile) software-defined infrastructure (SDI)
  o being embraced by telcos, network service providers, equipment vendors
Motivation

- With SDN and in-network clouds enabling NFV, we see trend towards a (mobile) software-defined infrastructure (SDI)
  - being embraced by telcos, network service providers, equipment vendors
Motivation

- With SDN and in-network clouds enabling NFV, we see trend towards a (mobile) software-defined infrastructure (SDI)
  - being embraced by telcos, network service providers, equipment vendors
Motivation

- With SDN and virtualization enabling NFV, we see a trend towards a (mobile) software-defined infrastructure (SDI) being embraced by telcos, network service providers, equipment vendors, and more.
- Automate network service instantiation and management.

SDN-enabled Network

Software Defined Core Infrastructure

Compute/ NFV/BBU Cloud Platform

Cloud RAN

SDN-enabled Network

Radio Access Network

Software Defined Radio

Mobile Device

Base Station
Motivation

• Push in industry and academia towards SDI control platforms
  o ONAP, OpenNFV, CORD, etc.
  o Automate network service instantiation and management
Motivation

• Push in industry and academia towards SDI control platforms
  ○ ONAP, OpenNFV, CORD, etc.
  ○ Automate network service instantiation and management

• Automation expected to be done through policies \((\text{condition } \rightarrow \text{action})\)
  ○ Unintended consequences
  ○ Performance degradation, SLA violations

Software Defined Infrastructure (SDI)
Motivation

- Push in industry and academia towards SDI control platforms
  - ONAP, OpenNFV, CORD, etc.
  - Automate network service instantiation and management

- Automation expected to be done through **policies** (*condition → action*)
  - Unintended consequences
  - Performance degradation, SLA violations

Software Defined Infrastructure (SDI)
Motivation

• Push in industry and academia towards SDI control platforms
  o ONAP, OpenNFV, CORD, etc.
  o Automate network service instantiation and management

• Automation expected to be done through policies \((\text{condition} \rightarrow \text{action})\)
  o Unintended consequences
  o Performance degradation, SLA violations

Software Defined Infrastructure (SDI)
Motivation

• Need to ensure safe policy deployment by **determining impact**
  o Earlier efforts focus on low-level ACL or routing policies (e.g., BGP, SDN rules)
  o SDI enables orchestration and service level policies
    ▪ Dynamic scaling, load balancing, orchestration and placement, migrations, edge cloud offloading, etc.
  o Static offline checks – not enough
  o Simulation – model may be incorrect or incomplete
Motivation

• Thus, there is a need for
  o an automated tool coupled with SDI control platforms
  o so policies can be tested for runtime impact before being deployed in production

➤ Required properties:
  ▪ Able to test SDI policies for runtime impact in varying environments
    ▪ E.g., varying traffic profiles, resource and service configurations, etc.
  ▪ Coupled with SDI control platform
  ▪ Automated
DEPO: A Platform for Safe DEployment of POlicy

• Captures runtime impact of SDI policies using an iterative emulations based approach

• **Required properties** and corresponding DEPO design principles
  ▪ **Ability to test SDI policies for runtime impact in varying environments**
    • emulations – realism with control
    • continuous impact learning – improves over time
  ▪ Coupled with SDI control platform
    • part of policy deployment workflow – comes after static testing, before production deployment
  ▪ Automated
    • knowledge based modeling
      – automates reasoning for test environment generation
      – automates ML model creation / statistical data analysis for learning impact
DEPO Context

Control Platform

Services and Managed Infrastructure

Software Defined Infrastructure (SDI)

Policies
DEPO Context

Control Platform

Templates → SDI State → Orchestrator

Software Defined Infrastructure (SDI)
DEPO Context

Control Platform

Policy Engine

Orchestrator

SDI State

Templates

Software Defined Infrastructure (SDI)

Policies
DEPO Context

Control Platform

Policy Engine

Orchestrator

SDI State

Templates

Software Defined Infrastructure (SDI)

Policies
DEPO Context

Software Defined Infrastructure (SDI)

Control Platform

- Policy Engine
- SDI State
- Orchestrator

Templates

Policies

Sandbox SDI

(Emulation Environment)

DEPO
DEPO Context

Control Platform

Policy Engine

SDI State

Orchestrator

Templates

Software Defined Infrastructure (SDI)

Sandbox SDI

(Emulation Environment)

DEPO

Policies
DEPO Context

Software Defined Infrastructure (SDI)

Control Platform
- Policy Engine
- Or orchestrator
- SDI State

Sandbox SDI

(Emulation Environment)

DePo

Policies

Emulate Learn Impact

Policy Stager
DEPO Context

Control Platform

Software Defined Infrastructure (SDI)

Policy Engine

Orchestrator

SDI State

Templates

Sandbox SDI

(Emulation Environment)

Policies

Emulate

Learn Impact

Policy Stager

Knowledge Base

Knowledge Graph

ML Models

DEPO
DEPO Context

Software Defined Infrastructure (SDI)

Control Platform
- Policy Engine
- Orchestrator

Policy Stager
- Emulate
- Learn Impact

Knowledge Base
- Knowledge Graph
- ML Models

DEPO

Policies

Domain Experts

Sandbox SDI

(Emulation Environment)
DEPO Context

Software Defined Infrastructure (SDI)

Control Platform
- Policy Engine
- Orchestrator
- SDI State

Policy Stager
- Emulate
- Learn Impact
- Knowledge Base
  - Knowledge Graph
  - ML Models

Domain Experts

Policies

Sandbox SDI

(Emulation Environment)
SDI Use-Case Services

- Use-case 1: Standard 4G LTE/EPC broadband service
SDI Use-Case Services

- Use-case 1: Standard 4G LTE/EPC broadband service
SDI Use-Case Services

- Use-case 1: Standard 4G LTE/EPC broadband service

- Use-case 2: 4G LTE/EPC service with edge cloud offloading (SMORE by Cho et al. ATC’14)
SDI Use-Case Services

• Use-case 1: Standard 4G LTE/EPC broadband service

• Use-case 2: 4G LTE/EPC service with edge cloud offloading (SMORE by Cho et al. ATC’14)
SDI Use-Case Services

- Use-case 1: Standard 4G LTE/EPC broadband service

- Use-case 2: 4G LTE/EPC service with edge cloud offloading (SMORE by Cho et al. ATC’14)

---

(a) Radio Access Network
(b) Core Mobile Network
(c) Low-latency App
(d) Edge Cloud at MTSO

Regular traffic
SDI Use-Case Services

- Use-case 1: Standard 4G LTE/EPC broadband service

- Use-case 2: 4G LTE/EPC service with edge cloud offloading (SMORE by Cho et al. ATC’14)
DEPO Component: Knowledge Graph (KG)

- Unit of data storage is a fact
  - 3-tuple of the form: `node1 relationship node2`

Example KG:

```
EPC isA Service
Service hasComponent NF
EPC hasComponent NF
EPC1 hasType EPC
EPC2 hasType EPC
VNF isA NF
MME isA VNF
MME1 hasType MME
EPC1 hasComponent MME1
Server isA ComputeNode
Server hosts VNF
Server1 isA Server
Server2 hosts MME1
MME.usage isA UsageVariable
MME1 usage hasValue 90% [timestamp=123]
```

Example Query:

```
SGW hasNeighbor X
```
DEPO Component: Knowledge Graph (KG)

- Unit of data storage is a **fact**
  - 3-tuple of the form: `node1 relationship node2`

```
EPC isA Service
Service hasComponent NF
EPC hasComponent NF
EPC1 hasType EPC
EPC2 hasType EPC
VNF isA NF
MME isA VNF
MME1 hasType MME
EPC1 hasComponent MME1
Server isA ComputeNode
Server hosts VNF
Server1 isA Server
Server2 hosts MME1
MME.usage isA UsageVariable
MME1.usage hasValue 90% [timestamp=123]
```

Example KG

Example Query

- `SGW_1` hostedOn `X` isA `VM`
- `Y` hostedOn `X`
- `X` isA `VM`
DEPO: Modeling Knowledge

- **Object**
  - hasObjectRelationship (e.g., hasComponent, hosts)
  - affects
- **Variable**
- **Event**
- **Mechanism**
- **Policy**

**Policy**
- Service
- Virtual
- Physical

**SDI Layers**

**Object**
- NF
  - isA
  - PNF
  - VNF
  - Forwarding Device
  - Non-Forwarding Device

**Service**
- Compute Node
  - hasComponent
  - Path
  - Link
  - Traffic Filter
  - hasChain
  - hosts

**Compute Node**
- VM
  - VNF
  - Server

**Topology**
- Chain
  - OR
  - hasComponent
DEPO: Impact Learning Approach

• Variables
  o configurable: num CPU, location, IP address, num of SGW, status
  o observed: usage (CPU, mem), throughput, response time
  o workload: rate of requests to SMORE VNF, UE attaches, traffic types
  o emulation environment parameters: subset of configurable/workload

• Mechanisms
  o potentially cause some change: start, stop, update, migrate

• Policies
  o higher level mechanism (if-then)

• Policy impact (statistically significant change – not categorized as good/bad)
  o Policy \(\rightarrow\) mechanism \(\rightarrow\) variable

Template

Variables
Mechanisms (lifecycle)
Policies
... Topology spec
Orchestration recipes
Implementation artifacts
...
• Policy writers may already know high-level impact — object-level relationships
• But non trivial to quantify impact at finer granularities — variable-level relationships
DEPO: Impact Learning Approach

1) Run emulation
   • Generating emulation environment suitable for testing a policy in sandbox SDI
   • Running emulation by running traffic through it and collecting logs

2) Learning impact
   • Analyze logs and annotate knowledge models

Diagram:
- New policy ----> Parse policy
  - Retrieve knowledge
    - Annotate knowledge models
    - Generate emulation parameters
    - Analyze
    - Run emulation

(1) Run emulation
(2) Learn impact
DEPO: Impact Learning Approach

Learning impact of policies

- Affected variables learned for SGW scale up policy

```when:
    SGW.usage >= THRESH  //THRESH = 70%
then:
    SGW.scaleUp()
```
DEPO: Impact Learning Approach

Learning impact of policies
• Affected variables learned for SGW scale up policy

```
when:
    SGW.usage >= THRESH  //THRESH = 70%
then:
    SGW.scaleUp()
```
DEPO: Impact Learning Approach

Learning impact of policies
• Affected variables learned for SGW scale up policy

```
when:
  SGW.usage >= THRESH  //THRESH = 70%
then:
  SGW.scaleUp()
```
DEPO: Impact Learning Approach

Learning impact of policies
- Affected variables learned for SGW scale up policy

```java
when:
  SGW.usage >= THRESH //THRESH = 70%
then:
  SGW.scaleUp()
```

SGW
DEPO: Impact Learning Approach

**Learning impact of policies**

- Affected variables learned for SGW scale up policy

```
when:  
    SGW.usage >= THRESH  //THRESH = 70%
then:  
    SGW.scaleUp()
```
DEPO: Impact Learning Approach

Learning impact of policies
- Affected variables learned for SGW scale up policy

```
when:
    SGW.usage >= THRESH  // THRESH = 70%
then:
    SGW.scaleUp()
```

- Is this a component of a service?
- Is this hosted on compute
DEPO: Impact Learning Approach

Learning impact of policies

- Affected variables learned for SGW scale up policy

```java
when:
    SGW.usage >= THRESH  //THRESH = 70%
then:
    SGW.scaleUp()
```

- Does it have neighbors?
- Topological, protocol peers
DEPO: Impact Learning Approach

**Learning impact of policies**

- Affected variables learned for SGW scale up policy

```plaintext
when: SGW.usage >= THRESH  //THRESH = 70%
then: SGW.scaleUp()
```

Diagram:

- EPC
- hasComponent
- eNodeB
- hasNeighbor
- SGW
- hosts
- MME
- VM
- hosts
- Server

Flow:

1. Run emulation
2. Learn impact
3. Parse policy
   - Retrieve knowledge
   - Annotate knowledge models
   - Generate emulation parameters
   - Analyze
   - Run emulation
DEPO: Impact Learning Approach

**Learning impact of policies**

- Affected variables learned for SGW scale up policy

```java
when: SGW.usage >= THRESH  //THRESH = 70%
then: SGW.scaleUp()
```

Observing impact results from emulation run later will refine this subgraph.
DEPO: Impact Learning Approach

Learning impact of policies

• Affected variables learned for SGW scale up policy

    when:
    
    SGW.usage \geq \text{THRESH} \quad // \text{THRESH} = 70\%

    then:
    
    SGW.scaleUp()
### Variable examples from our prototype SDI

<table>
<thead>
<tr>
<th>Server</th>
<th>VM</th>
<th>Switch/Link</th>
<th>VNF</th>
<th>Service</th>
</tr>
</thead>
<tbody>
<tr>
<td>location</td>
<td>location</td>
<td>location</td>
<td>location</td>
<td>topologyVars</td>
</tr>
<tr>
<td>status</td>
<td>status</td>
<td>status</td>
<td>status</td>
<td>numENB</td>
</tr>
<tr>
<td>rateStatusChange</td>
<td>rateStatusChange</td>
<td>ifaceVariables</td>
<td>type</td>
<td>numSGW</td>
</tr>
<tr>
<td>totalMem</td>
<td>totalMem</td>
<td>latency</td>
<td>totalMem</td>
<td>numPGW</td>
</tr>
<tr>
<td>allocatedMem</td>
<td>numCPU</td>
<td>totalMem</td>
<td>numCPU</td>
<td>numMME</td>
</tr>
<tr>
<td>numCPU</td>
<td>version</td>
<td>latency</td>
<td>version</td>
<td>numWebserver</td>
</tr>
<tr>
<td>numAllocatedCPU</td>
<td>type</td>
<td>totalMem</td>
<td>cpuUsage</td>
<td>latency, throughput</td>
</tr>
<tr>
<td>cpuOversubscription</td>
<td>memUsage</td>
<td>latency</td>
<td>memUsage</td>
<td>...</td>
</tr>
<tr>
<td>numAllocatedVM</td>
<td>cpuUsage</td>
<td>totalMem</td>
<td>throughput</td>
<td></td>
</tr>
<tr>
<td>numRunningVM</td>
<td>migrationVars</td>
<td>numFlaps</td>
<td>latency</td>
<td></td>
</tr>
<tr>
<td>version</td>
<td>numNeighborVM</td>
<td>latency</td>
<td>topologyVars</td>
<td>numUE</td>
</tr>
<tr>
<td>memUsage</td>
<td>osImage</td>
<td>numFlaps</td>
<td>cacheVars</td>
<td>rateOfRequests</td>
</tr>
<tr>
<td>type</td>
<td>memUsage</td>
<td>latency</td>
<td></td>
<td>interarrivalTime</td>
</tr>
<tr>
<td>numMigrationV</td>
<td>cpuUsage</td>
<td>latency</td>
<td></td>
<td>num/ rateOfMobility</td>
</tr>
<tr>
<td>memUsage</td>
<td>memUsage</td>
<td>latency</td>
<td></td>
<td>concurrentVNF/Service</td>
</tr>
<tr>
<td>propagationDelay</td>
<td>memUsage</td>
<td>latency</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Workload**
- numUE
- rateOfRequests
- interarrivalTime
- num/ rateOfMobility
- concurrentVNF/Service
**DEPO: Impact Learning Approach**

**Learning impact of policies**

- Affected variables learned for SGW scale up policy

  ```
  when:  
    SGW.usage >= THRESH  //THRESH = 70%
  then:  
    SGW.scaleUp()
  ```

- If no prior knowledge about impact is available

---

(1) **Run emulation**

(2) **Learn impact**

- Parse policy
- Retrieve knowledge
- Generate emulation parameters
- Annotate knowledge models
- Analyze
- Run emulation
DEPO: Impact Learning Approach

**Learning impact of policies**

- Affected variables learned for SGW scale up policy
  
  ```
  when: SGW.usage >= THRESH // THRESH = 70%
  then: SGW.scaleUp()
  ```

- If no prior knowledge about impact is available
  - Randomly pick a small set of emulation parameter values

- If prior knowledge about impact is available then greedily vary emulation parameters that cause the most (good/bad) impact for this policy
DEPO: Impact Learning Approach

**Learning impact of policies**

- Affected variables learned for SGW scale up policy

```
when:
    SGW.usage >= THRESH  //THRESH = 70%
then:
    SGW.scaleUp()
```
DEPO: Impact Learning Approach

Control Platform
- Policy Engine
- Orchestrator

Software Defined Infrastructure (SDI)

Sandbox SDI

Policy Stager
- Knowledge Base
  - Knowledge Graph
  - ML Models

Emulate
Learn Impact

Annotate
Analyze
New policy
Parse policy
Retrieve knowledge
Generate emulation parameters
Run emulation

Run emulation
(1)

Learn impact
(2)
DEPO: Impact Learning Approach

- Traffic generator parameters
  - Can be set based on past domain expertise
  - Or let DEPO consider these part of emulation parameters
DEPO: Impact Learning Approach

Learning impact of policies

• Affected variables learned for SGW scale up policy

```java
when: SGW.usage >= THRESH //THRESH = 70%
then: SGW.scaleUp()
```
(3) Learn Impact

• Logs collected include
  o Traces of policy executions
  o Configurable and observed variable logs for object instances – state changes in time

• Learn impact using
  1. Generating ML models
  2. Change analysis
3) Learn Impact

1) Change analysis

- Perform *before – after* change analysis for each variable to get course-grained impact info.
- Note statistically significant difference (e.g., 95% confidence) by comparing CDFs collected from before, and after policy triggerings.

---

**Time in emulation**

- **T**\_BEFORE
- **T**\_AFTER

**policy execution**
3) Learn Impact

1) Change analysis
   - Perform *before – after change analysis* for each variable to get course-grained impact info
   - Note statistically significant difference (e.g., 95% confidence) by comparing CDFs collected from before, and after policy triggerings
3) Learn Impact

1) Change analysis

- Perform *before – after change analysis* for each variable to get course-grained impact info
- Note statistically significant difference (e.g., 95% confidence) by comparing CDFs collected from before, and after policy triggerings
  - Kolmogorov-Smirnov 2-sample goodness of fit test
    - Generic test that works across variable types (makes no assumptions about distribution of data – worked well in our evaluations
    - Alternate specialized mechanisms can be plugged in here
  - Very parallelizable – large number of instance logs can be processed in parallel
- Output the list of impacted object instances
3) Learn Impact

Impact knowledge graph facts

PolicyAction: `scaleUp` SGW

Server:
- `server_numVM`
- `server_cpuUsage`
- `server_memUsage`
- `server_allocated_mem`
- `server_num_allocated_cpu`

VM:
- `vm_status`
- `vm_networkConfig`
- `vm_cpuUsage`
- `vm_memUsage`

EPC:
- `epc_numSGW`

SGW:
- `sgw_status`
- `sgw_networkConfig`
- `sgw_mmeNumConnections`
- `sgw_mmePercentOfConnectionsWorking`
- `sgw_cpuUsage`
- `sgw_memUsage`

MME:
- `mme_numSGW`
- `mme_sgwNumConnections`
- `mme_sgwPercentOfConnectionsWorking`

![Diagram showing the policy action `scaleUp` and its impacts on different components such as numSGW, status, cpuUsage, sgwPerc, and Connections Working.]
3) Learn Impact

2) Generating ML models

- Auto-generate ML models
  - Multiple linear regression
  - SVM
  - Random Forest

- Easier to auto generate as part of workflow
- Capture both linear and non-linear relationships
- Model for each impacted variable
  - Compute accuracy using training/test division of logs
  - Kfold cross validation for computing accuracy
  - Grid search for tuning on the model parameters
3) Learn Impact

2) Generating ML models

- Continuous learning from emulations improves impact knowledge over time
  - Leads to knowledge correction over time
Evaluations

• Sandbox SDI created in PhantomNet mobility testbed
  o Emulated RAN and core with multiple locations
  o Different servers and VM configs

• Extended SDI orchestrator and templates available in the community
  o Exposed more variables/mechanisms for EPC and SMORE services
  o Created SDI policies (Drools)
• Example policies

Listing 1: Update server
when:
  Server.updateAvailable == True AND
  Server.locatedAtEdge == True
then:
  Server.update()

Listing 2: Oversubscribe
when:
  Server.cpuUsage_10minAvg < THRESH1
  //THRESH1 = 50%
then:
  Server.setCPU_Oversub(oversubPerc = THRESH2) //THRESH2 = 50%

Listing 3: Scaling SGW
when:
  SGW.cpuUsage >= THRESH
  //THRESH = 70%
then:
  EPC.scaleUpSGW()

Listing 4: Scaling SMORE
when:
  SMORE_Webserver.cpuUsage_5minAvg
  >= THRESH //THRESH = 70%
then:
  SMORE.scaleUpWebserver()

Listing 5: SMORE caching
when:
  SMORE.subscriberLatency_5minAvg >= THRESH
  //THRESH = 30ms
then:
  SMORE.setCaching()

Listing 6: SMORE offload
when:
  EPC.subscriberLatency_10minAvg >= THRESH
  //THRESH = 30ms
then:
  EPC.augmentSMORE(subscriberList)
2) Generating ML models

- Continuous learning from emulations improves impact knowledge over time
  - Leads to knowledge correction over time
Evaluations

Impact knowledge graph facts

PolicyAction: scaleUpSGW

Server:
- server_numVM
- server_cpuUsage
- server_memUsage
- server_allocated_mem
- server_num_allocated_cpu

VM:
- vm_status
- vm_networkConfig
- vm_cpuUsage
- vm_memUsage

EPC:
- epc_numSGW

SGW:
- sgw_status
- sgw_networkConfig
- sgw_mmeNumConnections
- sgw_mmePercentOfConnectionsWorking
- sgw_cpuUsage
- sgw_memUsage

MME:
- mme_numSGW
- mme_sgwNumConnections
- mme_sgwPercentOfConnectionsWorking

Policy Action
scaleUpSGW

impacts

numSGW
- status
- cpuUsage
- sgwPerc
- Connections
- Working

EPC
- hasProperty

VM

Server

MME

SGW

status
Evaluations

- Example policies

**Listing 1: Update server**

```java
when:
    Server.updateAvailable == True AND Server.locatedAtEdge == True
then:
    Server.update()
```

**Listing 2: Oversubscribe**

```java
when:
    Server.cpuUsage_10minAvg < THRESH1
        //THRESH1 = 50%
then:
    Server.setCPU_Oversub(oversubPerc = THRESH2) //THRESH2 = 50%
```

**Listing 3: Scaling SGW**

```java
when:
    SGW.cpuUsage >= THRESH
        //THRESH = 70%
then:
    EPC.scaleUpSGW()
```

**Listing 4: Scaling SMORE**

```java
when:
    SMORE_Webserver.cpuUsage_5minAvg >= THRESH
        //THRESH = 70%
then:
    SMORE.scaleUpWebserver()
```

**Listing 5: SMORE caching**

```java
when:
    SMORE.subscriberLatency_5minAvg >= THRESH
        //THRESH = 30ms
then:
    SMORE.setCaching()
```

**Listing 6: SMORE offload**

```java
when:
    EPC.subscriberLatency_10minAvg >= THRESH
        //THRESH = 30ms
then:
    EPC.augmentSMORE(subscriberList)
```
Impact knowledge graph facts

PolicyAction: setCaching
- SMORE:
  - smore_cachingStatus
  - smore_latency
- SMORE_Loadbalancer:
  - smore_loadbalancer_cachingStatus
  - smore_loadbalancer_availableCacheSize
  - smore_loadbalancer_usedCacheSize
  - smore_loadbalancer_resourceFrequencyOfAccess
- SMORE_Webserver:
  - smore_webserver_status
  - smore_webserver_networkConfig
  - smore_webserver_loadbalancerConnectionStatus
  - smore_webserver_cpuUsage
  - smore_webserver_memUsage
  - smore_webserver_resourceFrequencyOfAccess
## Evaluations

### Impact knowledge graph facts

<table>
<thead>
<tr>
<th>PolicyAction: update</th>
<th>Server.[status, version, numVM, memUsage, cpuUsage, percentFailedMigrations], VM.[status, memUsage, cpuUsage], VNF.[status, memUsage, cpuUsage], Service.[smoreLatency]</th>
</tr>
</thead>
<tbody>
<tr>
<td>Server.migrateVM</td>
<td>Server.[numVM, memUsage, cpuUsage, percentFailedMigrations], VM.[status, memUsage, cpuUsage], VNF.[status, memUsage, cpuUsage], Service.[smoreLatency]</td>
</tr>
<tr>
<td>Server.sendVM</td>
<td>Server.[numVM, memUsage, cpuUsage], VM.[status, memUsage, cpuUsage], VNF.[status, memUsage, cpuUsage], Service.[smoreLatency]</td>
</tr>
<tr>
<td>Server.receiveVM</td>
<td>Server.[numVM, memUsage, cpuUsage, percentFailedMigrations], VM.[status, memUsage, cpuUsage], VNF.[status, memUsage, cpuUsage], Service.[smoreLatency]</td>
</tr>
<tr>
<td>VM.stop</td>
<td>Server.[memUsage, cpuUsage], VM.[status, memUsage, cpuUsage], VNF.[status, memUsage, cpuUsage], Service.[smoreLatency]</td>
</tr>
<tr>
<td>Server.installUpdate</td>
<td>Server.[status, version, memUsage, cpuUsage]</td>
</tr>
<tr>
<td>Server.reboot</td>
<td>Server.[status, memUsage, cpuUsage]</td>
</tr>
<tr>
<td>VM.start</td>
<td>Server.[memUsage, cpuUsage], VM.[status, memUsage, cpuUsage]</td>
</tr>
<tr>
<td>VNF.start</td>
<td>Server.[memUsage, cpuUsage], VM.[memUsage, cpuUsage], VNF.[status, memUsage, cpuUsage], Service.[smoreLatency]</td>
</tr>
</tbody>
</table>
Impact knowledge graph facts

PolicyAction: augmentSMORE

Server:
- server_numVM
- server_cpuUsage
- server_memUsage
- server_allocated_mem
- server_num_allocated_cpu

VM:
- vm_status
- vm_networkConfig
- vm_cpuUsage
- vm_memUsage

SMORE:
- smore_latency

SMORE_Loadbalancer:
- smore_loadbalancer_webserverNumConnections
- smore_loadbalancer_webserverPercentOfConnectionsWorking
- smore_loadbalancer_resourceFrequencyOfAccess

SMORE_Switch:
- smore_switch_status
- smore_switch_routeConfig

SMORE_Webserver:
- smore_webserver_status
- smore_webserver_networkConfig
- smore_webserver_loadbalancerConnectionStatus
- smore_webserver_cpuUsage
- smore_webserver_memUsage
- smore_webserver_resourceFrequencyOfAccess
Evaluations

- Variable examples from our prototype SDI

<table>
<thead>
<tr>
<th>Server</th>
<th>VM</th>
<th>Switch/Link</th>
<th>VNF</th>
<th>Service</th>
</tr>
</thead>
<tbody>
<tr>
<td>location</td>
<td>location</td>
<td>location</td>
<td>location</td>
<td>topologyVars</td>
</tr>
<tr>
<td>status</td>
<td>status</td>
<td>status</td>
<td>status</td>
<td>numENB</td>
</tr>
<tr>
<td>rateStatusChange</td>
<td>rateStatusChange</td>
<td>ifaceVariables</td>
<td>status</td>
<td>numSGW</td>
</tr>
<tr>
<td>totalMem</td>
<td>totalMem</td>
<td>latency</td>
<td>type</td>
<td>numPGW</td>
</tr>
<tr>
<td>allocatedMem</td>
<td>numCPU</td>
<td>latency</td>
<td>totalMem</td>
<td>numMME</td>
</tr>
<tr>
<td>numCPU</td>
<td>version</td>
<td>latency</td>
<td>numCPUs</td>
<td>numWebserver</td>
</tr>
<tr>
<td>numAllocatedCPU</td>
<td>type</td>
<td>latency</td>
<td>version</td>
<td>latency, throughput</td>
</tr>
<tr>
<td>cpuOversubscription</td>
<td>cpuUsage</td>
<td>latency</td>
<td>memUsage</td>
<td>...</td>
</tr>
<tr>
<td>numAllocatedVM</td>
<td>memUsage</td>
<td>latency</td>
<td>throughput</td>
<td></td>
</tr>
<tr>
<td>numRunningVM</td>
<td>migrationVars</td>
<td>latency</td>
<td>latency, throughput</td>
<td></td>
</tr>
<tr>
<td>version</td>
<td>numNeighorVM</td>
<td>latency</td>
<td>topologyVars</td>
<td></td>
</tr>
<tr>
<td>memUsage</td>
<td>osImage</td>
<td>latency</td>
<td>memUsage</td>
<td></td>
</tr>
<tr>
<td>cpuUsage, type</td>
<td>ifaceVars</td>
<td>latency</td>
<td>migrationVars</td>
<td></td>
</tr>
<tr>
<td>ifaceVars</td>
<td></td>
<td></td>
<td>cacheVars</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>concurrentVNF/Service</td>
<td></td>
</tr>
</tbody>
</table>

**Workload**

- numUE
- rateOfRequests
- interarrivalTime
- num/rateOfMobility
- concurrentVNF/Service
Evaluations

- Response time (s)
- Parameter (e.g. amount of CPU oversubscription)

- Good
- Moderate
- Bad
Determining environment variables that are major impact contributors

- Knowing this helps in greedily varying env variables that show policy’s worst impact
- For impact of server CPU oversubscription policy on service response time:

<table>
<thead>
<tr>
<th>Variables</th>
<th>Relative Importance</th>
</tr>
</thead>
<tbody>
<tr>
<td>VM.cpuUsage</td>
<td>0.254</td>
</tr>
<tr>
<td>VM.numNeighborVMs</td>
<td>0.002</td>
</tr>
<tr>
<td>Server.cpuUsage(aggregate)</td>
<td>0.298</td>
</tr>
<tr>
<td>Server.cpuOvsersubscription</td>
<td>0.289</td>
</tr>
<tr>
<td>Service.requestRate</td>
<td>0.157</td>
</tr>
</tbody>
</table>
Evaluations

Improved learning over time from emulations using iterative approach

• Decreasing average RMSE over emulation iterations
  • Averaged from models for service response time for the 6 policy examples
Evaluations

Performance and scalability

- Parallelizable and scalable DEPO process – reasonable bottleneck of emulation environment orchestration
Conclusion

• We presented DEPO
  • given SDI orchestration and service-level policies,
  • allows us to determine and quantify their impact on objects in an SDI.

• DEPO uses a systematic approach to modeling domain knowledge
  • Enables more informed policy writing and policy impact checking
  • Further annotate / verify initial model by incorporating knowledge learnt using emulations

• DEPO uses statistical analysis and machine learning to enable knowledge-based inference and reasoning
  • Describes how a given SDI policy affects the SDI objects

• We prototyped DEPO and evaluated it on a testbed with policies from realistic use case services in a sandbox SDI