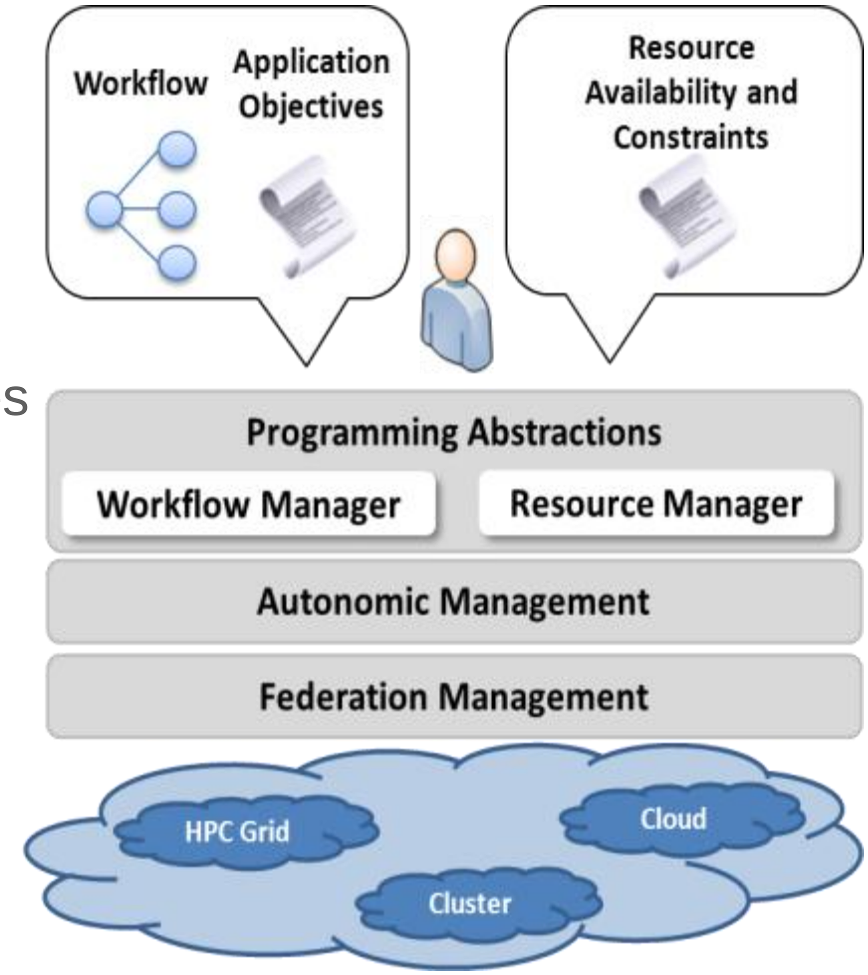
Software-Defined Federation

Moustafa AbdelBaky, Javier Diaz-Montes, and Manish Parashar

NSF Cloud and Autonomic Computing Center (CAC)
Rutgers Discovery Informatics Institute (RDI^2)
Rutgers, The State University of New Jersey

Software Defined Federation

- Combine ideas from federated computing, cloud computing, and software defined environments
- Create a nimble and programmable environment that autonomously evolves over time, adapting to:
  - Changes in the infrastructure
  - Application requirements
- Independent control over application and resources



Programmatic Provisioning

- Provision and federate an appropriate mix of resources on-the-fly
  - Enable the creation and modification of these federations programmatically
  - Separate the control plane from the execution plane
  - Provide programming abstractions to support the continuous execution of applications

Dynamic Provisioning

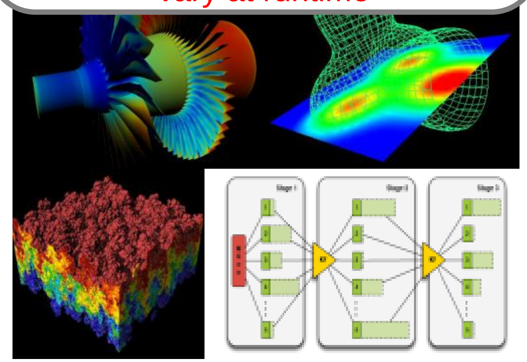
- Declarative specification to define availability as well as policies and constraints to regulate resource usage
  - Customized views of the federation for different projects or situations
  - Specify how to react to unexpected changes in the resource availability or performance or application behavior
- Evolve in time and space -- the evaluation of these policies and constraints provides a set of available resources during runtime

Software-defined Ecosystem

Scientific Applications & Workflows

- Workflow definition
- Objectives (deadline, budget)
- Requirements (throughput, memory, I/O rate)
- Defined in terms of science (e.g., precision, resolution)

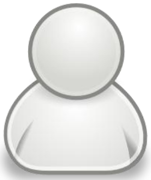
- vary at runtime -



Autonomic Manager

- Identify utility of federation
- Negotiate with application
- Ensure applications' objectives and constraints
- Adapt and reconfigure resources and network on the fly

User/Provider



Define federation programmatically using rules and constraints

- Availability
- Capacity & Capability
- Cost
- Location
- Access policy

- vary at runtime -

Synthesize a space-time federated ACI

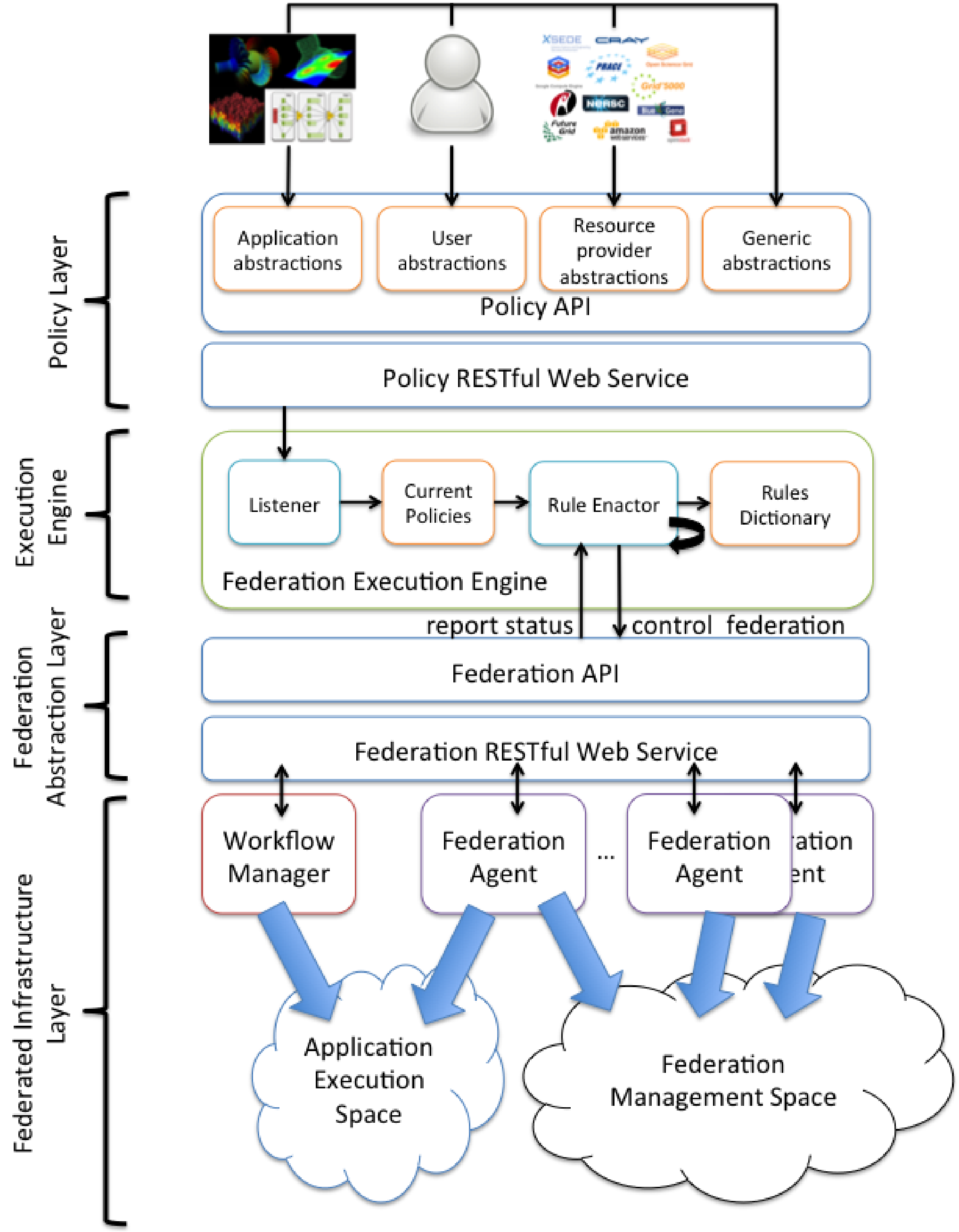
Exposed as a uniform resource to the application/workflow

Software-Defined Federated Cyber-infrastructure

RULE ENGINE BASED SOFTWARE-DEFINED FEDERATION

Architecture

- Policy Layer
- Execution Engine
- Federation Abstraction Layer
- Federated Infrastructure Layer

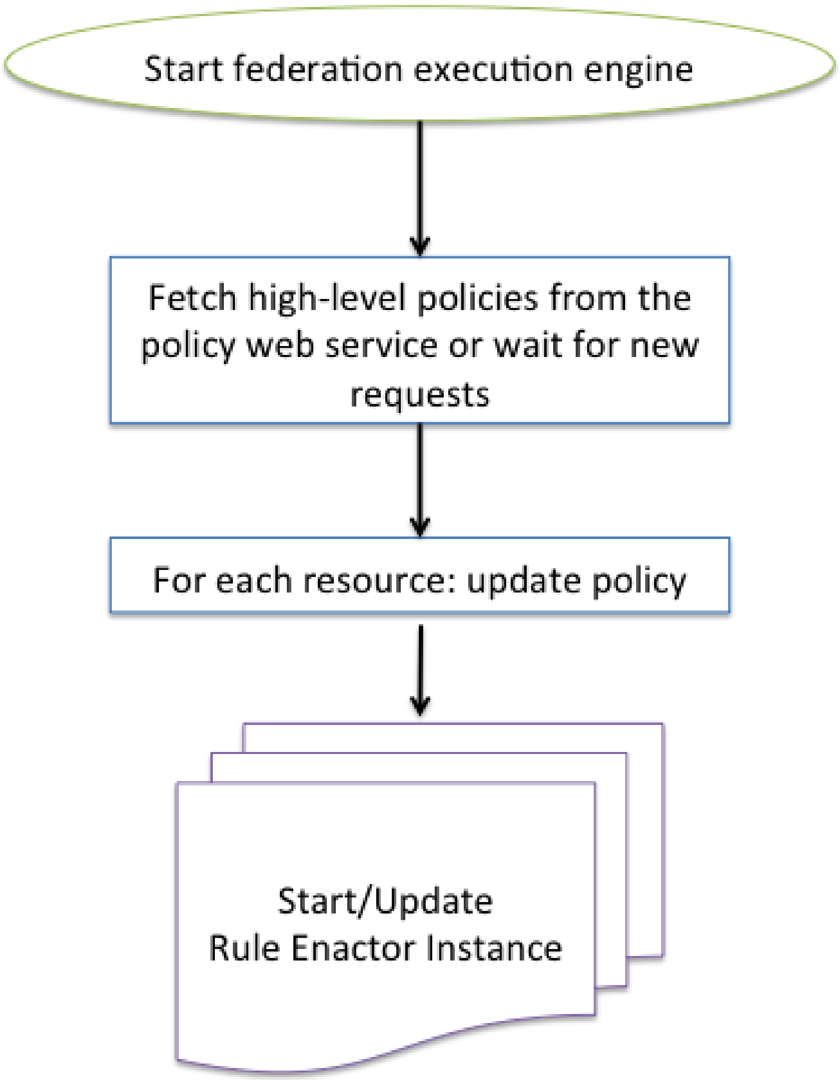


Policy Layer

- The policy layer provides mechanisms for expressing the attributes of the federation in terms of resource availabilities and constraints
- Supports different types of policies that are tailored to meet the needs of the different actors (e.g., users, applications, and resource providers)
- Generic Policies
  - Direct declaration of resources over time
- User Policies
  - Expose resources in terms of cost or deadline
- Application Policies
  - Expose resources in terms of type or capacity
- Resource Provider Policies
  - Expose resources in terms of utilization

Execution Engine

- A rule engine enables the policy-based management of the federation process
  - Translates the high-level policies at runtime into a set of resources (***recipes***)
  - Ensures the orchestration of federated sites over time according to these recipes using the federation abstraction layer
  - Executes the application on top of the resulting federated infrastructure
  - Monitors the composition of the federation over time and modifying it as necessary based on existing and new policies



Federation Abstraction Layer

- Exposes federation mechanisms as uniform programming abstractions and supports the addition/removal of sites, scale up/down of resources within a site, discovery of sites and resources, etc.
- Provides abstractions for monitoring the status of the federated infrastructure, e.g., the available sites, number of available resources, number of resources running applications, etc.
  - Resource description operations
  - CometCloud federation agent operations
  - Application execution operations
  - Status operations

Use Case Scenario – User Driven Federation

- A user has an application that she would like to execute on a set of available resources
- These resources can be owned by the user (e.g. local machine or clusters), shared (e.g. allocations on a supercomputer), or paid per usage (e.g. cloud resources)
- The objective defined for this application is maximizing throughput, i.e., aggregating as much computational power from the federation as possible
- Using our SDF framework, the user can specify the list of available resources and their usage policy in two separate methods.
  - Scenario 1: The user can declare a strict description policy that specifies the exact composition of the federation over time
  - Scenario 2: The user defines the desired behavior of the federation but not its exact composition over time.

Experimental Summary

- Run on Future Systems always
- Run on Spring daily from 11:05:00 to 11:40:00
- Run on Green from 02/28/2015 11:15:00 to 02/28/2015 11:30:00
- Run on Chameleon when the dynamic price is less than $0.1 per hour

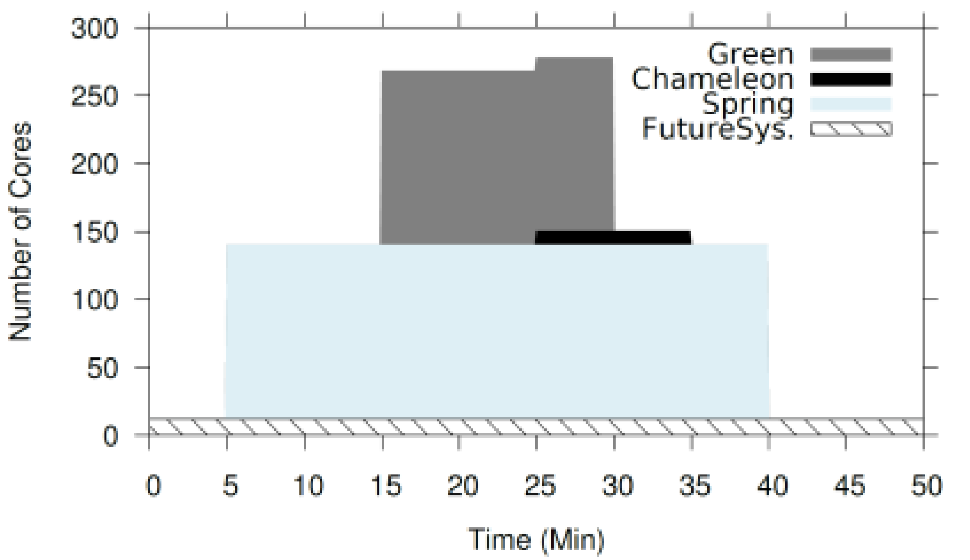
Table 1: Resources available at each site and their characteristics.

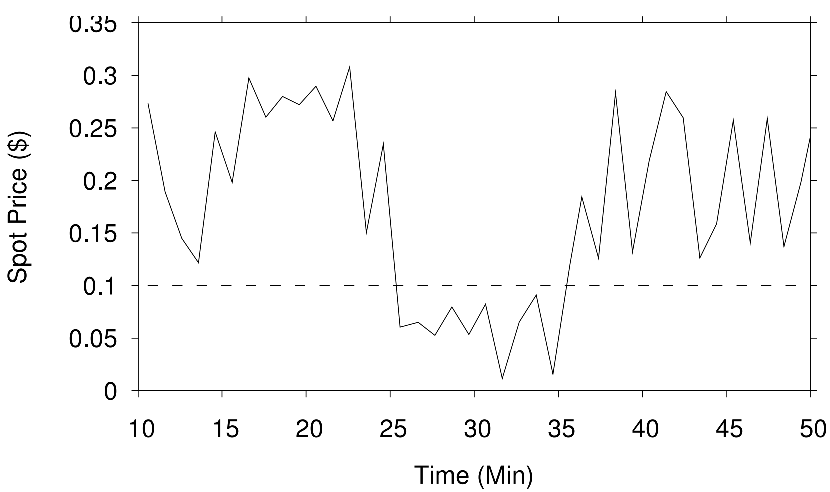
Future Systems - OpenStack Cloud				
Resource	#Cores	Memory	Performance	Max. VMs‡
VM_Medium	2	4 GB	1.36	3
VM_Small	1	2 GB	0.69	6
Spring - HPC Cluster				
Resource†	#Cores	Memory	Performance	Max. Machine‡
Bare-metal	8	24 GB	1.42	16
Green - HPC Cluster				
Resource†	#Cores	Memory	Performance	Max. Machine‡
Bare-metal	8	24 GB	0.42	16
Chameleon - OpenStack Cloud				
Resource	#Cores	Memory	Performance	Max. VMs‡
VM_Medium	2	4 GB	1	3
VM_Small	1	2 GB	0.5	4

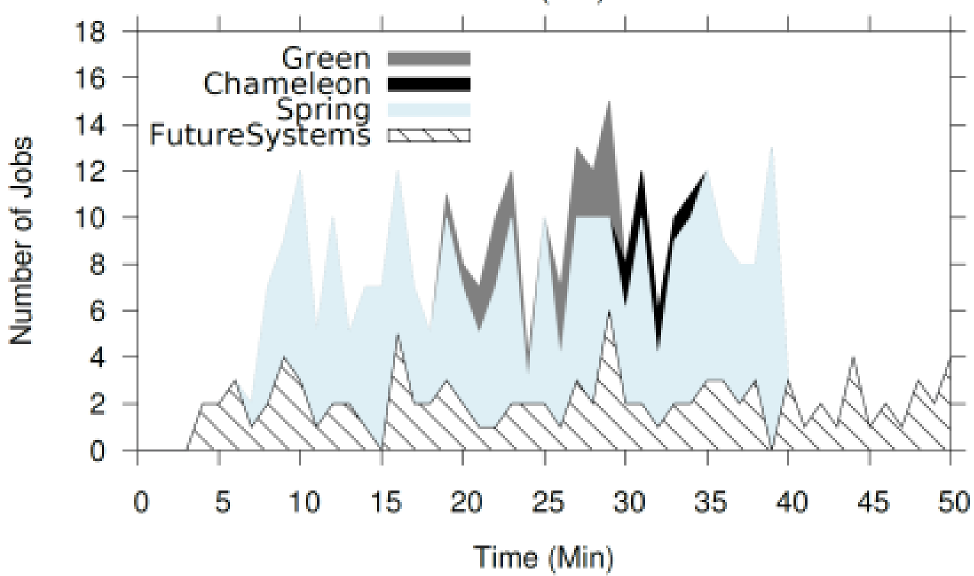
Note: ‡ – Maximum number of available VMs/bare-metal per type

Results

- Dynamic policies
- Resource allocation
- Throughput
- Cost-based allocation
- Small experiment but..







CONSTRAINT PROGRAMMING BASED SOFTWARE-DEFINED FEDERATION

Approach

I. Separate resource selection from application scheduling

II. Build a constraint programming model to specify finer grained user/provider requirements for resource provisioning

- Example Constraints: Availability, Capacity, Utilization, Cost, Performance, Security, Power, Overhead, Waste, …
- Ability to add or remove new/existing constraints

III. Deploy applications using a resource-selection aware scheduler

IV. The entire process is continuously repeated to allow for dynamic adaptation.

Architecture