Why do systems need to find nodes?
Use Cases

Cloud Management

VM Provisioning
Use Cases

Cloud Management

VM Provisioning

VM Migration
Use Cases

Cloud Management

VM Provisioning
VM Migration
Monitoring
Use Cases

Cloud Management
- VM Provisioning
- VM Migration
- Monitoring

NVF Automation
Use Cases

Cloud Management
VM Provisioning
VM Migration
Monitoring

NVF Automation
Geo-distributed VNF
Service Chain Placement
Use Cases

Cloud Management
- VM Provisioning
- VM Migration
- Monitoring

NVF Automation
- Geo-distributed VNF
- Service Chain Placement

Required information is assumed available

But HOW is node information collected?
Outline

- How do systems find nodes?
- Limitations of current approaches
- FOCUS design
- Evaluation
- Conclusion
Looking Under The Hood

How do systems search for nodes?
Node finding in OpenStack
Hosts
Hosts

RabbitMQ Server

Push

Hosts
$ openstack server create --flavor FLAVOR_ID --image IMAGE_ID

Hosts

RabbitMQ Server

Dequeue

Central DB

Push
```
$ openstack server create --flavor FLAVOR_ID --image IMAGE_ID
```
$ openstack server create --flavor FLAVOR_ID --image IMAGE_ID
Provision VMs

$ openstack server create --flavor FLAVOR_ID --image IMAGE_ID

Dequeue

Query

Central DB

Push

Hosts

Placement Service

RabbitMQ Server
Limitations of Current Approaches
Hosts
---
Central DB
---
RabbitMQ Server
---
Placement Service
---
Provision VMs
---
Query
---
Dequeue
---
Push

Hard to scale > 100s of nodes!
Hard to scale > 100s of nodes!

- Hosts
- Central DB
- Placement Service
- RabbitMQ Server
  - Dequeue
  - Push
- Provision VMs
- Query
- Bottleneck
- Hard to scale > 100s of nodes!
What if we have a data center with 1000s of servers?
What if we have a data center with 1000s of servers?

Create clusters!
Independent clusters acting as their own datacenters
This increases operational complexity!

Now we manage several entities (instead of one)
Multi-vendor/site cloud

Finding nodes across cloud sites/vendors is even harder!
Scalable and generic search service for distributed systems
Main Components
Main Components

Query Processing with Directed Pulling
Main Components

Query Processing with Directed Pulling

Gossip-based Node Coordination
Main Components

- Query Processing with Directed Pulling
- Gossip-based Node Coordination
- Easy-to-integrate Query Interface
Main Components

- Query Processing with Directed Pulling
- Gossip-based Node Coordination
- Easy-to-integrate Query Interface
Abstractions
Abstractions

Node Attributes
Abstractions

- Node Attributes
  - Static
  - Dynamic
Abstractions

Node Attributes
- Static
  - Never change
- Dynamic
Abstractions

Node Attributes

- Static
  - Never change
    - # cpu cores, arch, etc

- Dynamic
Abstractions

Node Attributes

- Static
  - Never change
    - \# cpu cores, arch, etc

- Dynamic
  - Frequently change
Abstractions

Node Attributes

Static
- Never change
  - # cpu cores, arch, etc

Dynamic
- Frequently change
  - Usage: cpu, ram, disk, bandwidth, etc
Abstractions

Node Attributes

- Static
  - Never change
    - # cpu cores, arch, etc

- Dynamic
  - Frequently change
    - Usage: cpu, ram, disk, bandwidth, etc
Abstractions

Node Attributes
- Static
  - Never change
    - # cpu cores, arch, etc
- Dynamic
  - Frequently change
    - Usage: cpu, ram, disk, bandwidth, etc

Query Structure
Abstractions

Node Attributes

- Static
  - Never change
    - # cpu cores, arch, etc
- Dynamic
  - Frequently change
    - Usage: cpu, ram, disk, bandwidth, etc

Query Structure

- Attribute List
Abstractions

Node Attributes
- Static
  - Never change
    - # cpu cores, arch, etc
- Dynamic
  - Frequently change
    - Usage: cpu, ram, disk, bandwidth, etc

Query Structure
- Attribute List
  - name (string)
  - upper bound (int)
  - lower bound (int)
Abstractions

Node Attributes
- Static
  - Never change
  - # cpu cores, arch, etc
- Dynamic
  - Frequently change
  - Usage: cpu, ram, disk, bandwidth, etc

Query Structure
- Attribute List
  - name (string)
  - upper bound (int)
  - lower bound (int)
  - limit (int)
  - freshness (int)
Query Processing with Directed Pulling
Attribute-based Grouping
Attribute-based Grouping

Group nodes according to their attribute values.
Attribute-based Grouping

*cpu_usage {50-100}%*

Group nodes according to their attribute values
Attribute-based Grouping

Group nodes according to their attribute values

- **cpu_usage {50-100}%**
- **cpu_usage {0-50}%**
Attribute-based Grouping

Group nodes according to their attribute values

cpu_usage {50-100}%
cpu_usage {0-50}%
avail_RAM {4-8}GB
Attribute-based Grouping

- **cpu_usage**: {50-100}%
- **cpu_usage**: {0-50}%
- **avail_RAM**: {4-8}GB
- **cpu_cores**: {8-12}

Group nodes according to their attribute values.
Nodes

cpu_usage \{50-100\}\% 

cpu_usage \{0-50\}\% 

avail_RAM \{4-8\}\text{GB}
Nodes

Groups metadata

- cpu_usage {50-100}%
- cpu_usage {0-50}%
- avail_RAM {4-8}GB
Query: {get nodes with cpu_usage < 50 and avail_RAM > 4GB}
Query: \{get nodes with cpu_usage < 50 and avail_RAM > 4GB\}
Query: \{get nodes with cpu_usage < 50 and avail_RAM > 4GB\}
Query: \{get nodes with cpu_usage < 50 and avail_RAM > 4GB\}
Gossip-based Node Coordination
Gossip-based Coordination

cpu_usage {50-100}%
Gossip-based Coordination

Nodes in a group are connected through a p2p gossip channel

cpu_usage {50-100}%
Gossip-based Coordination

Nodes in a group are connected through a **p2p gossip** channel.

Nodes exchange membership information.

`cpu_usage {50-100}%`
Gossip-based Coordination

**cpu_usage** \{50-100\}%

Nodes in a group are connected through a **p2p gossip** channel

Nodes exchange membership information

One node pushes group info to the FOCUS server
Gossip-based Coordination

**cpu_usage {50-100}%**

Nodes in a group are connected through a **p2p gossip** channel

- Nodes exchange membership information
- One node pushes group info to the FOCUS server
- Queries are propagated via **gossip channel**
Dynamic Groups Management

Operations flow in FOCUS

Node | FOCUS | P2p group
Operations flow in FOCUS

Node  FOCUS  P2p group

register
Dynamic Groups Management

Operations flow in \textit{FOCUS}

Node \textit{FOCUS} P2p group

- register
- refer to group(s)
Dynamic Groups Management

Operations flow in FOCUS

- Node
- P2p group

- Register
- Refer to group(s)
- Join p2p group(s)
Dynamic Groups Management

Operations flow in FOCUS

Node

- register
- refer to group(s)
- join p2p group(s)
- state change
- leave p2p group(s)

P2p group
Dynamic Groups Management

Operations flow in FOCUS

Node
- register
- refer to group(s)
- join p2p group(s)
- state change
- leave p2p group(s)

P2p group
Dynamic Groups Management

Operations flow in FOCUS

Node

- register
- refer to group(s)
- join p2p group(s)
- state change
- leave p2p group(s)

P2p group

- query
Dynamic Groups Management

Operations flow in FOCUS

Node

register

refer to group(s)

join p2p group(s)

state change

leave p2p group(s)

P2p group

query

disseminate query
to nodes
Dynamic Groups Management

Operations flow in FOCUS

Node

register
refer to group(s)
join p2p group(s)
state change
leave p2p group(s)

FOCUS

P2p group

query
response

disseminate query to nodes
Implementation & Evaluation
Implementation

**FOCUS Agent**

**Node Manager**

**p2p Agent**

**Registrar**

**Dynamic Groups Manager**

**Query Router**

**Cache**

**REST API**

**Query**

**Java**

1.9K LoC

1.2K LoC
Evaluation

- Deployed in Amazon EC2
- 4 regions: Canada, California, Ohio, Oregon
- In each region: 8 VMs (4 vCPUs, 16GB RAM)
- FOCUS server running in California (same VM config)
- Testing up to 1600 simulated node agents
**FOCUS** vs. Other Approaches

Measuring BW Consumption at the Query Server
(frequency = 1 query/update per second)

- **Naive Push/Pull**
- **Static Hierarchy**
- **RabbitMQ (Publish)**
- **RabbitMQ (Subscribe)**
Focus vs. Other Approaches

Measuring BW Consumption at the Query Server (frequency = 1 query/update per second)

- Naive Push/Pull
- Static Hierarchy
- RabbitMQ (Publish)
- RabbitMQ (Subscribe)

Adding a layer of intermediate nodes acting as aggregators

Bandwidth Consumption (KBps) vs. Number of Nodes

- Focus
- Naive Push/Pull
- Static Hierarchy
- RabbitMQ - pub
- RabbitMQ - sub
**FOCUS vs. Other Approaches**

Measuring BW Consumption at the Query Server (frequency = 1 query/update per second)

- **Naive Push/Pull**
- **Static Hierarchy**
- **RabbitMQ (Publish)**
- **RabbitMQ (Subscribe)**

Nodes publish their state (i.e., fancy push)
**FOCUS** vs. Other Approaches

Measuring BW Consumption at the Query Server (frequency = 1 query/update per second)

- Naive Push/Pull
- Static Hierarchy
- RabbitMQ (Publish)
- RabbitMQ (Subscribe)

Nodes subscribe for queries

![Graph showing bandwidth consumption vs. number of nodes for different approaches.](image-url)
Focus vs. Other Approaches

Measuring BW Consumption at the Query Server (frequency = 1 query/update per second)

- Naive Push/Pull
- Static Hierarchy
- RabbitMQ (Publish)
- RabbitMQ (Subscribe)

![Graph showing bandwidth consumption vs. number of nodes for different approaches. The graph indicates 95% improvement for Focus compared to other approaches.](image)

95% improvement
FOCUS with Real-world Cloud Traces*

Focus on Real-world Cloud Traces*

75K OpenStack VM placement requests

with Real-world Cloud Traces*

75K OpenStack VM placement requests
Replayed at accelerated rate (15,000x)

75K OpenStack VM placement requests

Replayed at accelerated rate (15,000x)

Latency stabilizes after 600 nodes
→ because group size is capped (~150 nodes per group)

Microbenchmarks

Resource usage of the FOCUS server (40 queries/s)
Microbenchmarks

Resource usage of the FOCUS server (40 queries/s)

Overhead imposed by node agent (KBps)
Microbenchmarks

Resource usage of the FOCUS server (40 queries/s)

Overhead imposed by node agent (KBps)

Query response time for different group sizes
Conclusion

- **Current systems’ scalability is limited**
  - This is due to tightly-coupled node management
Conclusion

• Current systems’ scalability is limited
  - This is due to tightly-coupled node management

• FOCUS is scalable search service
  - Employs a *loosely-coupled* node management (p2p)
  - *Scales* better than current approaches (15x improvement)
  - Imposes *minimal* overhead on nodes
  - *Integrates* well with current systems
Thank You!

Questions?