Application Placement and Demand Distribution in a Global Elastic Cloud: A Unified Approach

[1]**Hangwei Qian**, [2]Michael Rabinovich

[1] VMware

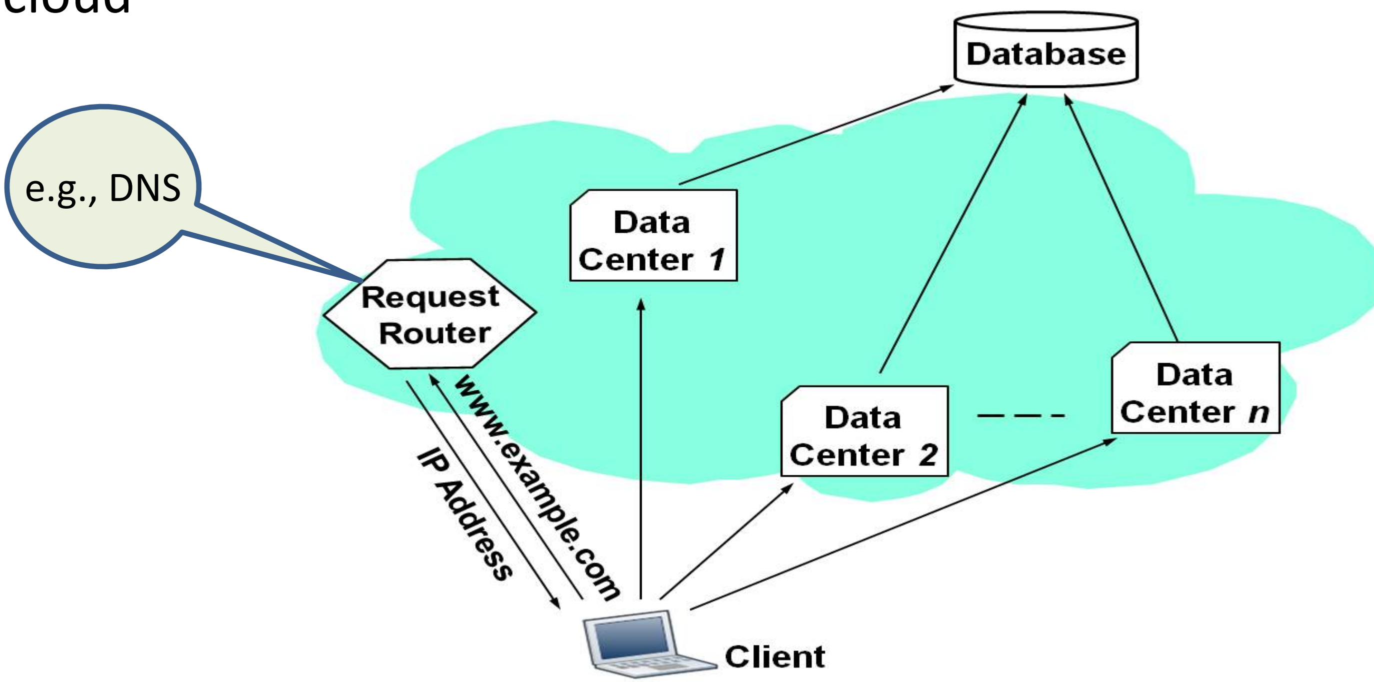
[2] Case Western Reserve University

Outline

- **Introduction**
  - System Environment
- **Unified Policy Computation**
  - Assumptions
  - Algorithm
- **Evaluation**
  - Simulation
  - Prototype testing (not discussed – see paper)
- **Summary**

Geo-Distributed Cloud Platforms

- Cloud providers deploy multiple data centers (DCs) around the world (Amazon/Google/Microsoft, etc.)
- Cloud Customers (i.e., application providers) deploy applications in the cloud



- Unpredictable load of the hosted applications: location/volume

Application Placement and Demand Distribution

- Resource auto-scaling in the cloud
  - *Application placement* – when/where to deploy an application instance
  - *Demand distribution* - how to distribute client requests among the instances
  - Only DC-level decisions – do not care about the number of application instances or request distribution inside data centers
- Existing approaches – address the two problems in isolation
  - Place applications assuming client requests go to closest data centers
  - Distribute client requests given the location of application instances
  - Do not consider back-end databases.
- Our approach
  - Unified: consider two problems together
  - Consider back-end databases
  - Address the scalability problem of computing a policy

Objectives

- Minimize overall user perceived response time
  - Minimize the overall network latency
  - Avoid data center overloading
- Minimize the number of application instances
  - Save resources and customer costs
- Minimize the number of placement changes
  - Reduce redeployment cost
  - Better cache behavior

Outline

- **Introduction**
  - System Environment
- **Unified Policy Computation**
  - Assumptions
  - Algorithm
- **Evaluation**
  - Simulation
  - Prototype testing (not discussed – see paper)
- **Summary**

Computing the Unified Policy for App Placement and Request Distribution

- Step I - optimal request distribution with full deployment
  - Full deployment - each application is deployed at each data center
  - Optimal request distribution - min-cost algorithm to solve centrally
- Step II - application placement policy
  - Calculate the amount of demand each data center receives for each application (from step I)
  - Remove underutilized instances (below some threshold)
- Step III – request distribution policy
  - Reassign demand for removed instances
  - Aggregate with demand for instances not removed in step II

Assumptions

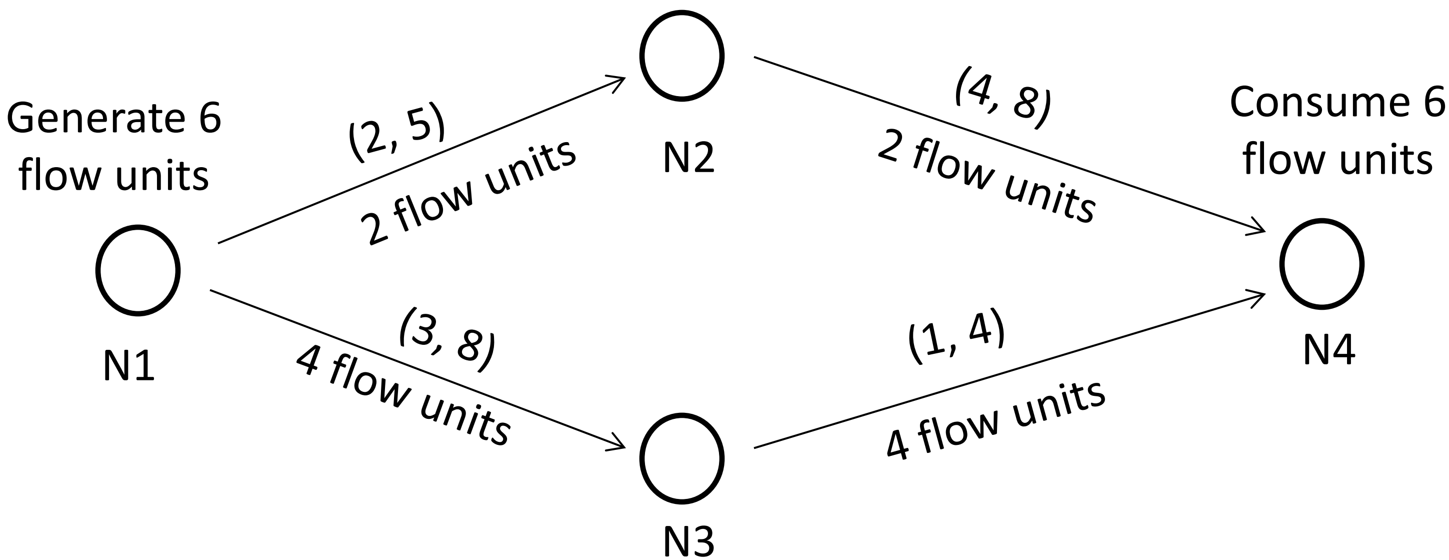
- Client Clusters (CC): group of clients sharing the same BGP prefix (~400K, network-aware clustering [SIGCOMM2000])
- Fixed back-end database location
- Aggregate distance -- simply sum up, though can easily be extended to more complex options
- Request rate as a metric for demand and data center load and capacity
  - Given demand pattern -- set of request rates from each client cluster for each application
  - Normalized request rate for different applications
  - As a measurement of data center capacity
- Notation: A - number of applications, C - number of client clusters, D – number of data centers

I - Optimal Request Distribution with Full Deployment

- Minimize overall network latency
- Avoid data center overloading
  - Limit the amount of total demand each data center receives (capacity limitation)
- Min-cost flow model
  - Source node, sink node, pair nodes (application, CC) and data center nodes
  - All nodes are balanced except the source and sink node
  - Minimize the cost when pushing all demands from source node to sink node

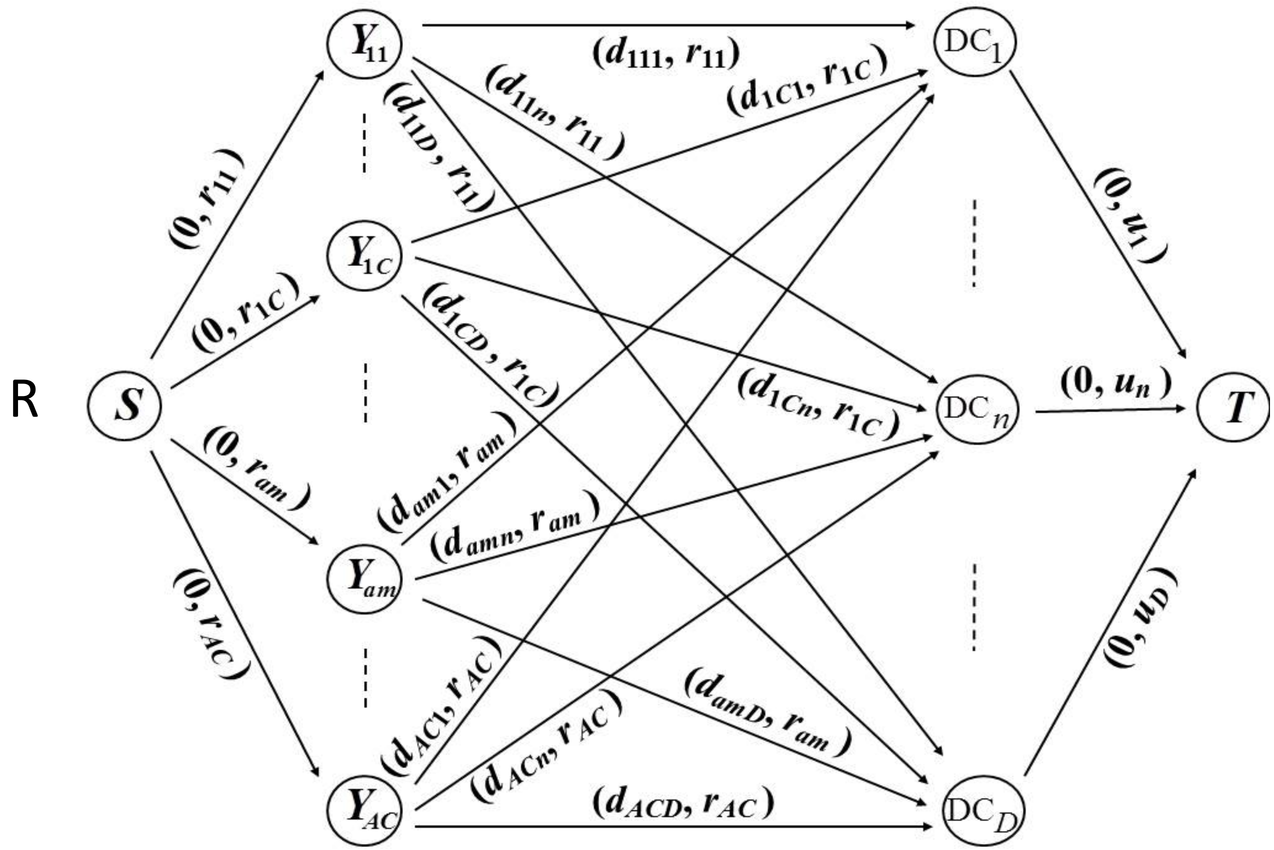
Simple Example

- Edge: cost, capacity
  - Supply node: generate flow (node N1)
  - Demand node: consume flow (node N4)
  - Balance node: neither (node N2 and N3)



Flow Model for Optimal Request Distribution

- Pair node (Y_{am}) – requests from client cluster *m* for application *a* (r_{am})
- Total amount of flow: $R = \sum_{a=1}^{A} \sum_{m=1}^{C} r_{am}$
- Move flow R from node S to node T with the minimum cost

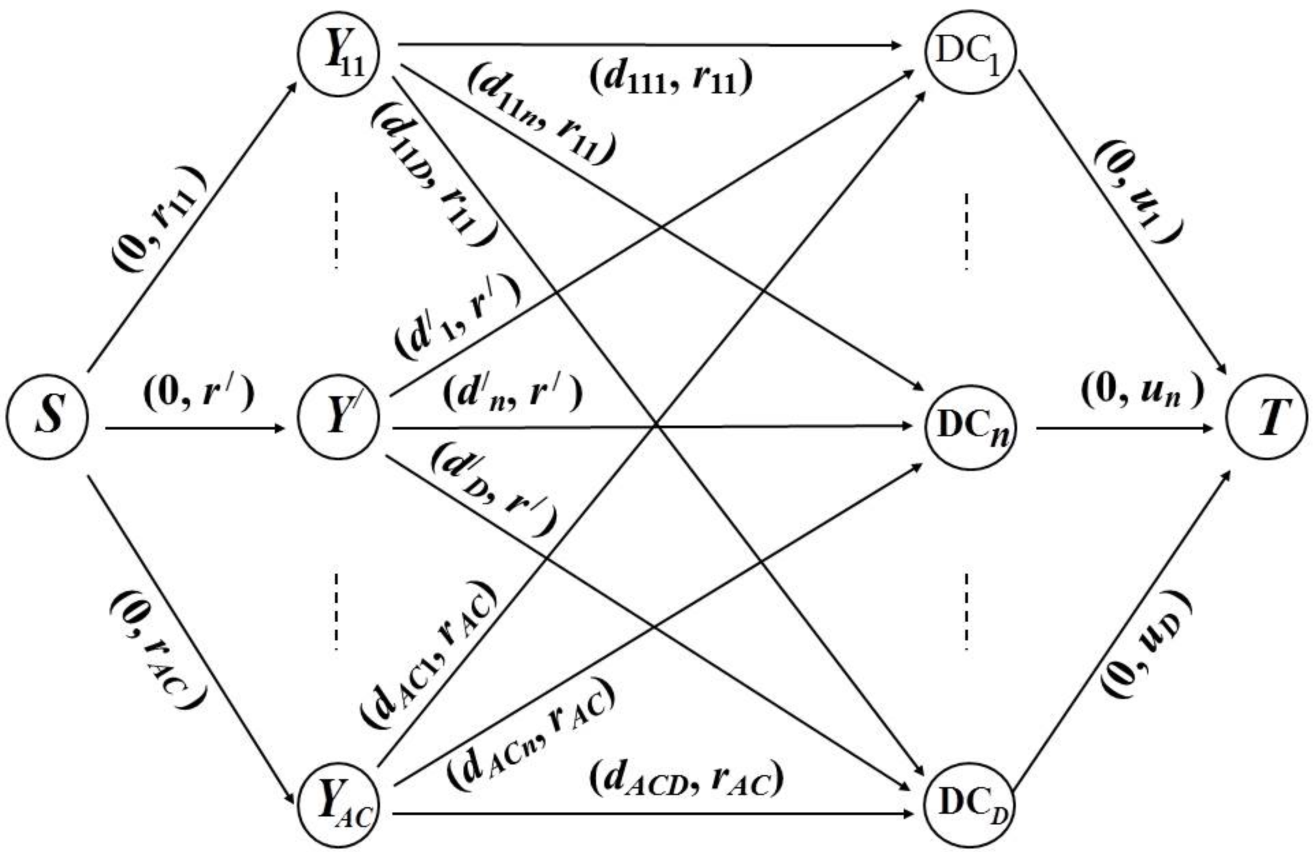


Permutation Prefix Clustering

- Scalability issue: A*C=100*400K=4*10^7 pair nodes
- Each pair node has permutation of preference of data centers {1,3,10,5,2,9,6,8,4,7}
- Merging pair nodes sharing prefix of certain length *L* of their permutations - if merge Y_{1C} and Y_{am} to Y'
  - Merged capacity: $r' = r_{1C} + r_{am}$
  - Merged cost: $d'_n = (d_{1cn} * r_{1C} + d_{amn} * r_{am}) / (r_{1C} + r_{am})$
- Trade-off between scalability and performance
  - Number of pair nodes: $\prod_{i=0}^{L-1}(D-i)$
  - Performance penalty

Merged Min-Cost Flow Model

- Total number of pair nodes: $20 * 19 * 18 = 6840,\ if\ L = 3\ and\ D = 20$



II - Application Placement

- Flow f_{na} : amount of requests DC *n* receives for each application *a* (obtained from step I)
- Deletion Threshold (*DT*): amount of requests worthy to deploy an application instance in the data centers.
- Normal flows: *if* $f_{na} \geq DT$
- Tiny flows: if $f_{na} < DT$
- Placement policy
  - Deploy application *a* at data center *n* for normal flow
  - Remove tiny flows unless it is the only instance for the applications

Reducing Placement Changes

- Hysteresis placement: add “stickiness” to previously deployed application instances
  - Smaller Deletion Threshold makes it harder to remove instances
  - Hysteresis ratio (HR): real Deletion Threshold = (Deletion Threshold) / (Hysteresis rate)
  - High HR for previously deployed application instances (>1)

III – Demand Distribution

- Redistribute the tiny flows (e.g., residual demand) to the data centers calculated placement policy
- Integrate the distribution of normal flows and tiny flows to get the final demand distribution policy

Outline

- **Introduction**
  - System Environment
- **Unified Policy Computation**
  - Assumptions
  - Algorithm
- **Evaluation**
  - Simulation
  - Prototype testing (not discussed – see paper)
- **Summary**

Cloud Model

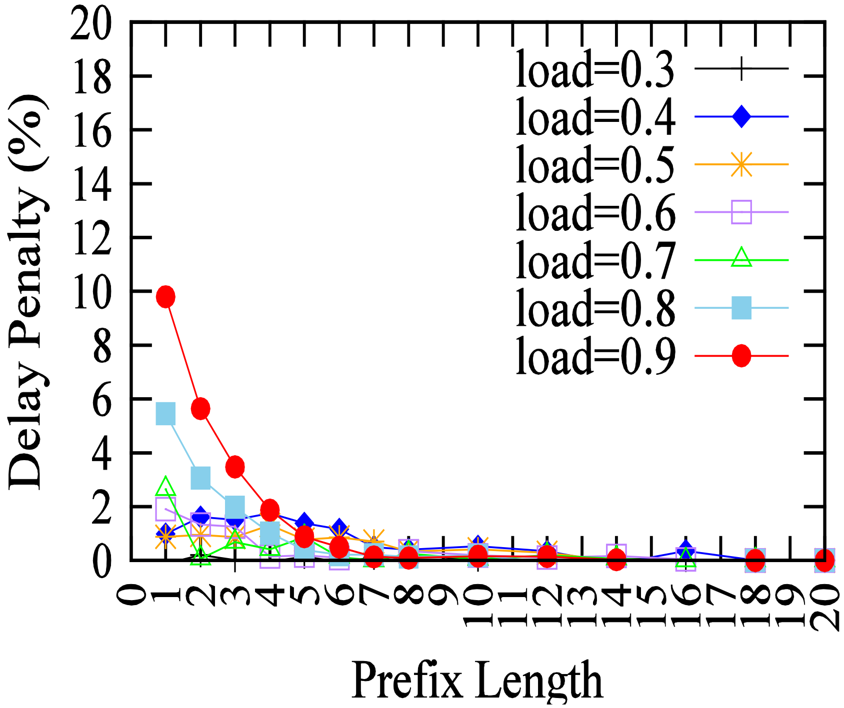
- Gnutella clients to mimic client clusters (~100K)
- Planetlab nodes (selected according to the distribution of clients) to mimic data centers (20)
- Planetlab nodes (randomly selected) to mimic back-end databases (100)
- "ping" network latency for the proximity among entities
- Each data center can deal with 10,000 req/s (200,000 req/s for all data centers)

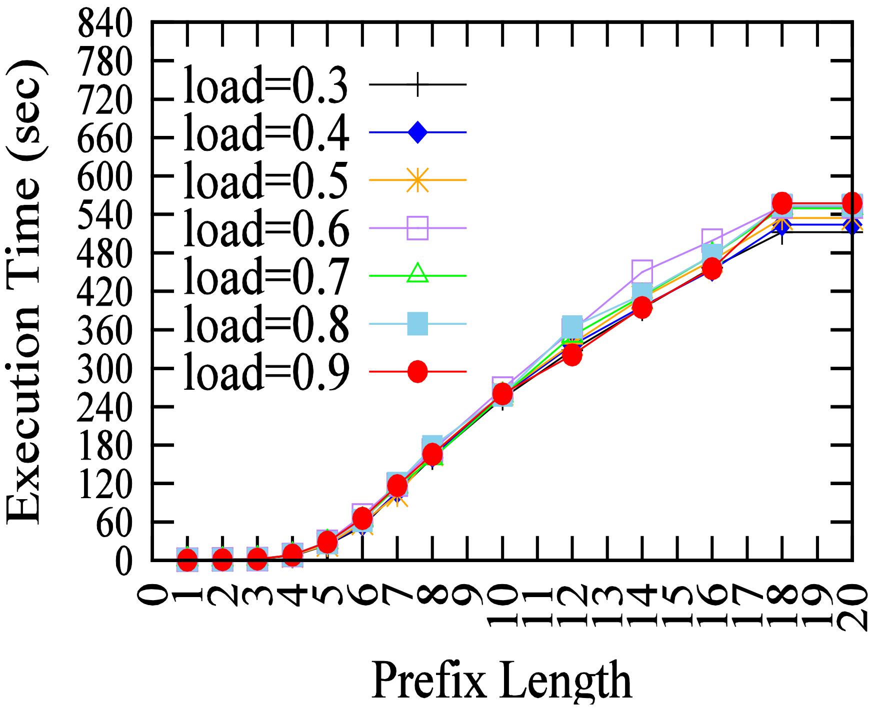
Experiment Setup

- Load factor, e.g., 0.5 (100,000 requests/s)
- Demand of different applications follows power law distribution with parameter 1
- Load generation (high-level)
  - For each request, select the application with power law
  - Select the client cluster it comes from
- CSIM: a discrete-event simulation tool

Prefix Clustering Evaluation

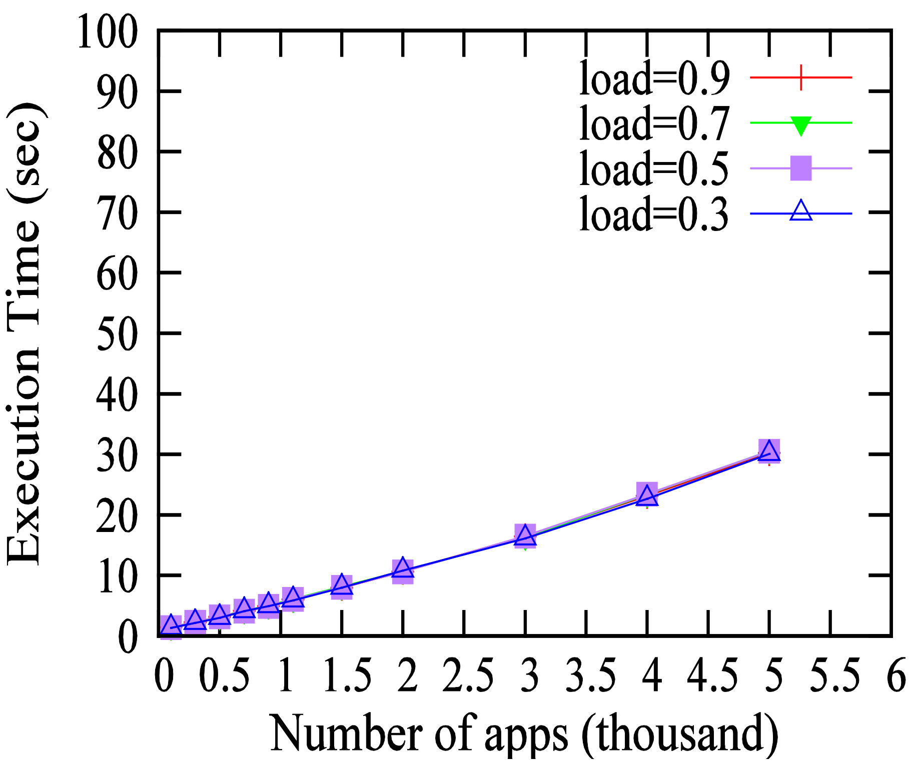
- Performance VS scalability: prefix length 3 is a good trade-off

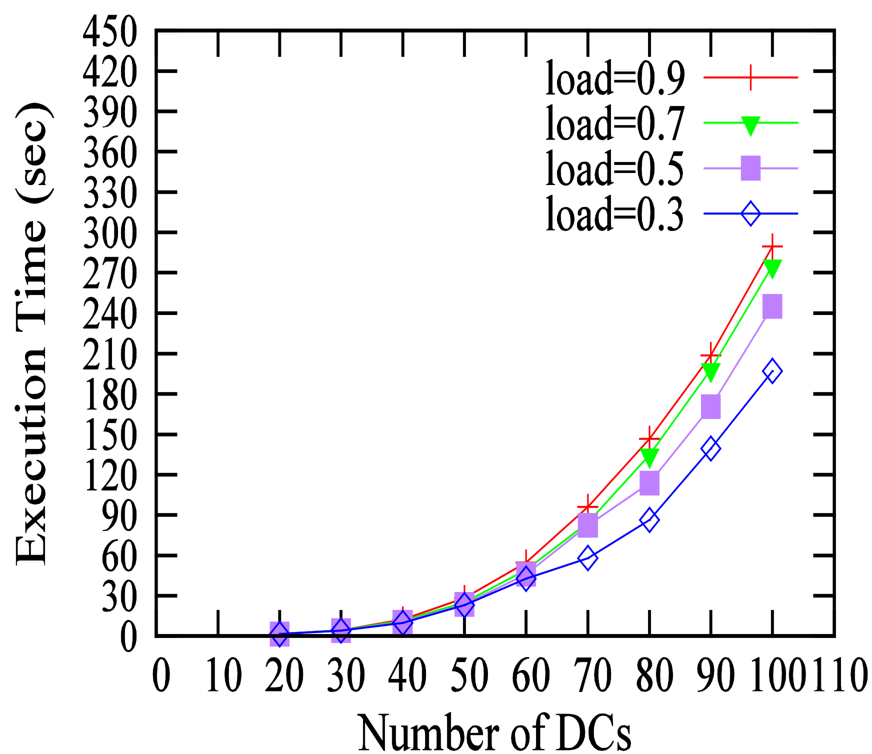




Scalability

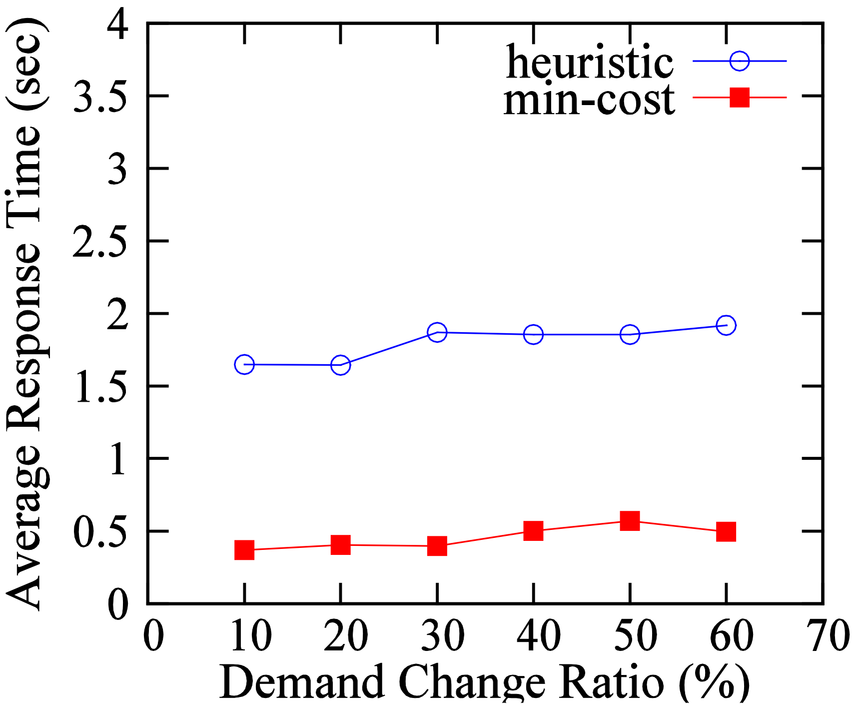
- Execution time *vs.* number of applications and data centers
  - Keep other parameters fixed

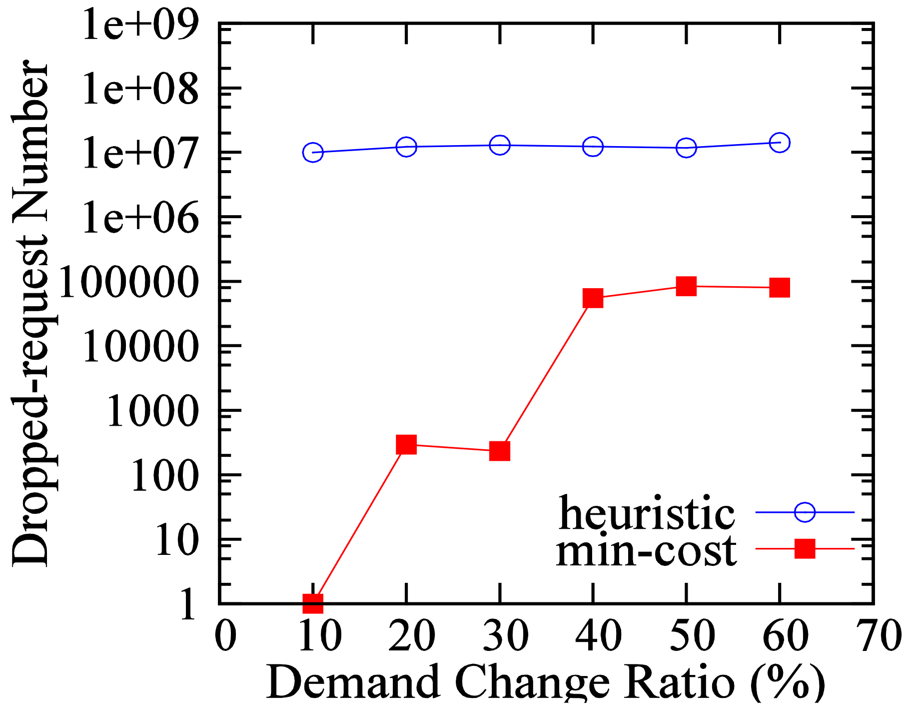




Policy Performance

- Compare with an existing method, which addressed both problems heuristically but in isolation
  - Update policy every 30 sec, and 900 seconds for the whole experiment
  - Workload changes randomly between +-Δ% from cycle to cycle (150 seconds)





Summary

- A unified approach to deal with the application placement and demand distribution problems together based on min-cost flow model
- Clustering technique to deal with the scalability issue
- Evaluations show that this approach is scalable and very effective

Thank you!