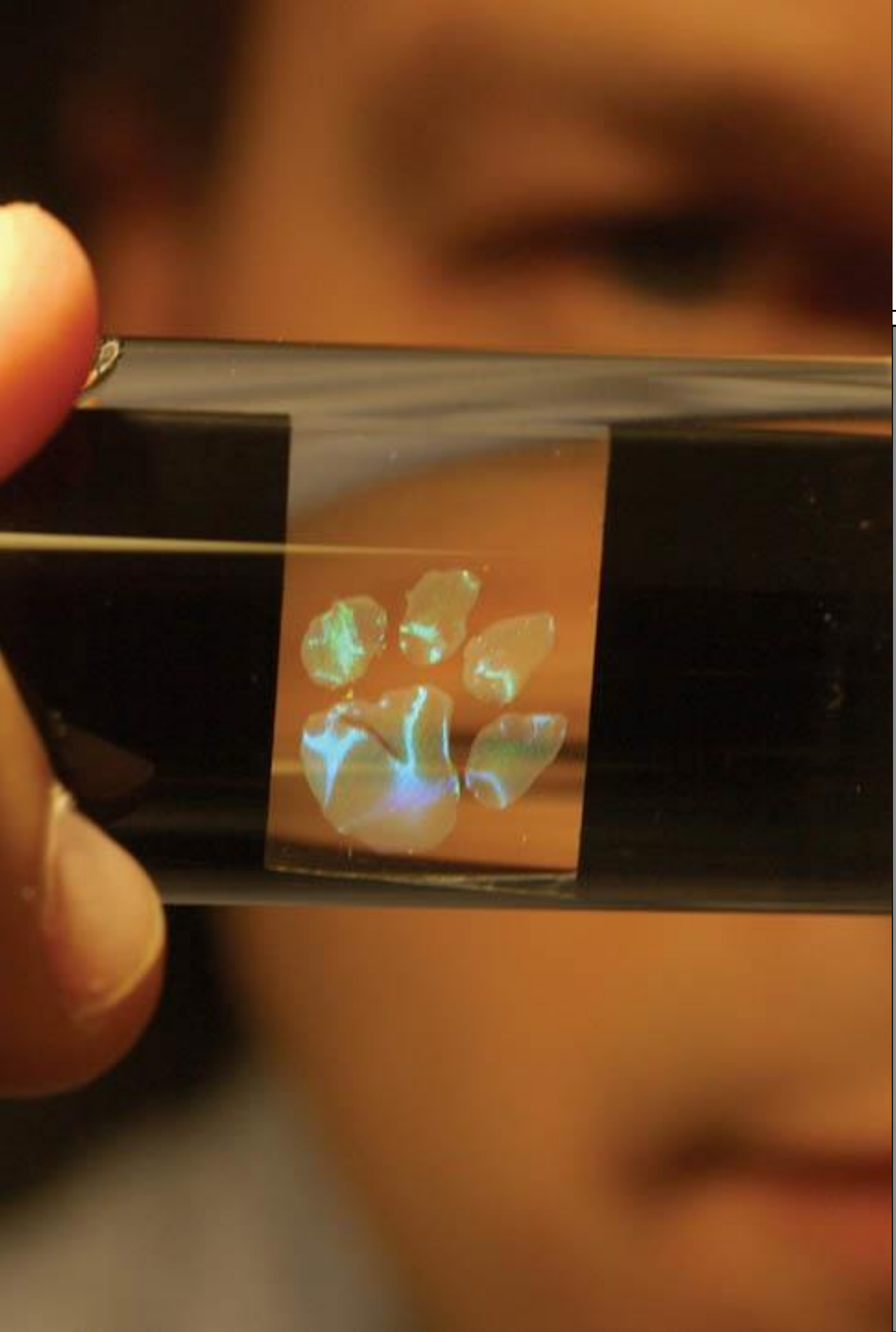


Consolidating Complementary VMs with Spatial/Temporal-awareness in Cloud Datacenters

Liuhua Chen and Haiying Shen
Dept. of Electrical and Computer Engineering
Clemson University, SC, USA

Outline

- Introduction
- System Design
  - Motivation
  - Patterns detection
  - Allocation policy
- Performance Evaluation
- Conclusions

Introduction

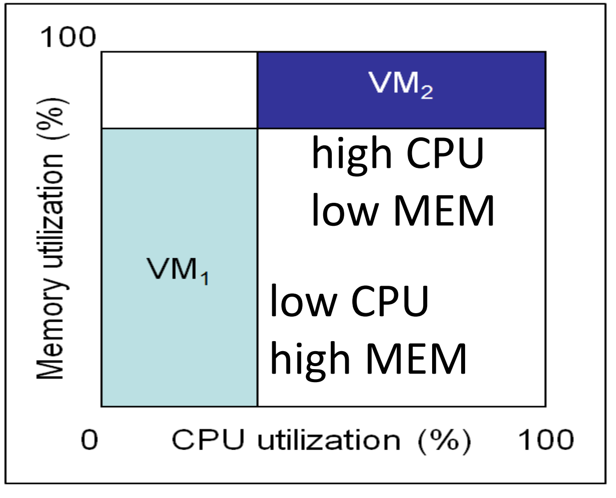
- The scale of cloud datacenters has been growing
- Energy consumption becomes critical concerns
- Resource provisioning should both maximize energy efficiency and satisfy Service Level Agreements (SLAs)

Introduction (cont.)

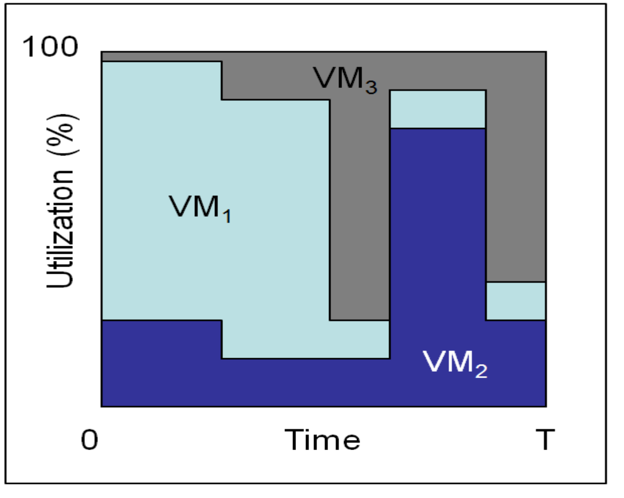
- Static provisioning
  - Allocates physical resources to VMs based on static VM resource demand
  - Cannot fully utilize resource
- Dynamic provisioning
  - Handles the PM resource constraint through live VM migrations
  - Produces migration overhead
- Our goal
  - Further reduce the number of PMs (energy efficiency)
  - Reduce the number of VM migrations (SLA)

Introduction (cont.)

- We propose an initial VM allocation mechanism that consolidates complementary VMs
  - Spatial complementary: total demand of each resource dimension nearly reaches PM capacity
  - Temporal complementary : total demand reaches PM capacity during lifetime period



Spatial



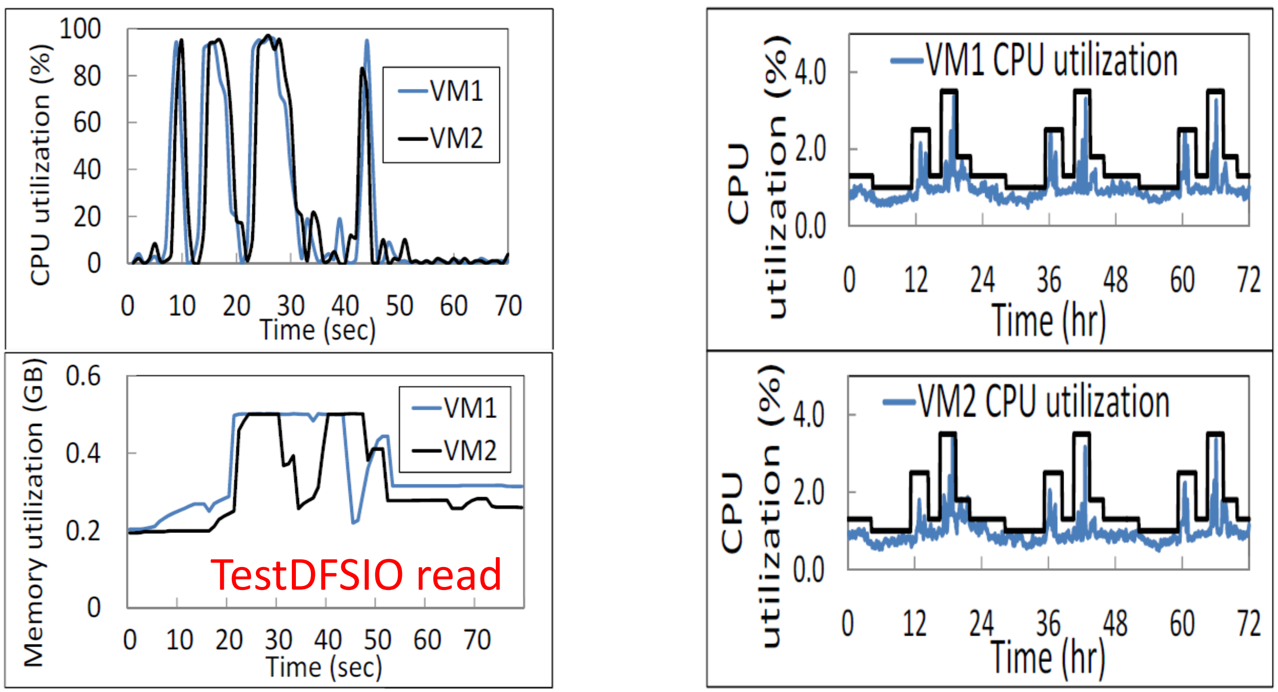
Temporal

System Design: Motivation

- Can we predict the resource demand?
  - VMs running the same short-term job
  - VMs running long-term applications
  - Experiments confirm the above observations
- How to predict the resource demand?
  - Precise prediction
    - Complex and costly
    - Prediction for individual VM cannot represent general case
  - Utilization patterns
    - Achieve balance between simplicity and precision
- How to consolidate?
  - Spatial/Temporal-awareness VM allocation algorithm

System Design: Motivation (cont.)

- Utilization pattern exists for VMs running the same short-term job
- Utilization pattern repeats for VMs running long-term job



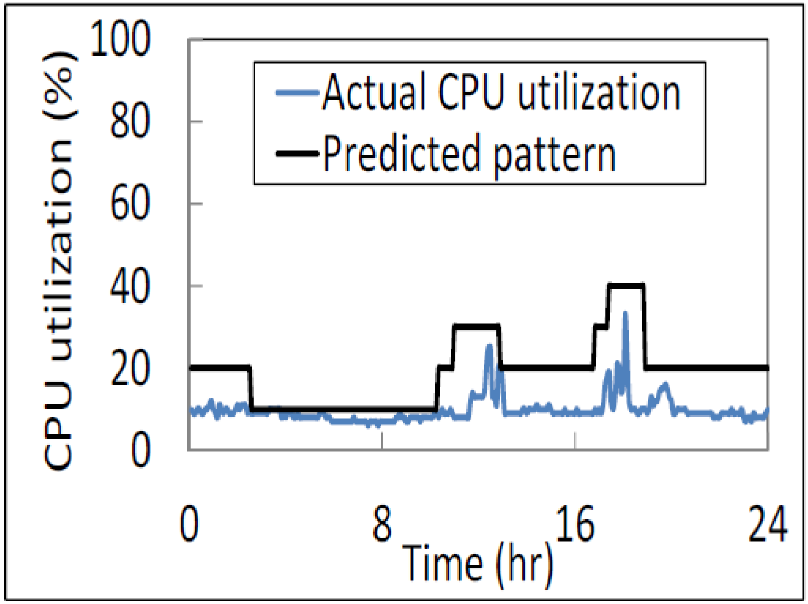
System Design: Patterns detection

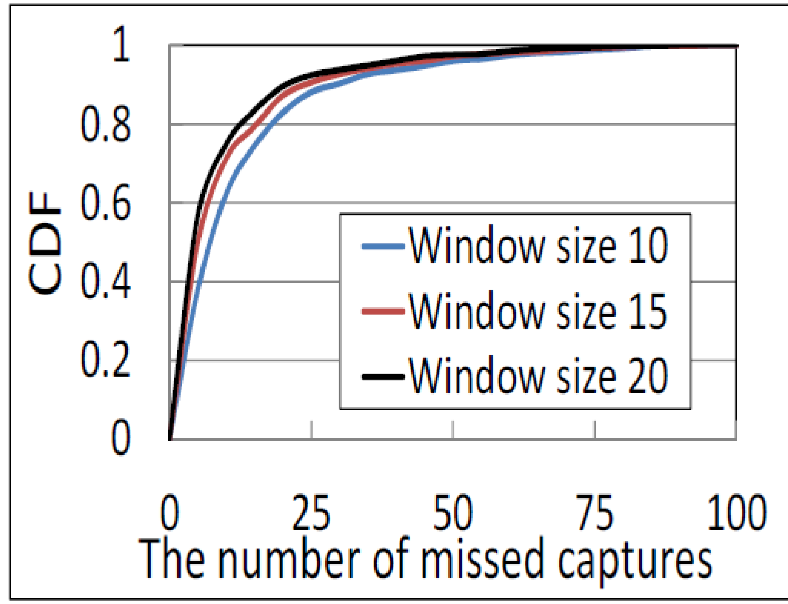
Algorithm 1 VM resource demand pattern detection.

Input: $\mathcal{D}_i(t)$ $(i = 1, 2, ..., N)$: Resource demands of a set of VMs
Output: $\mathcal{P}(t)$: VM resource demand pattern
  / Find the maximum demand at each time */*
  $\mathcal{E}(t_j) = \max_{i \in N}\{\mathcal{D}^i(t_j)\}$ for each time t_j
  / Smooth the maximum resource demand series */*
  $\mathcal{E}(t_j) \leftarrow$ LowPassFilter($\mathcal{E}(t_j)$) for each time t_j
  / Use sliding window to derive pattern */*
  $\mathcal{P}(t_j) = \max_{t_j \in [t_j, t_j + Window]}\{\mathcal{E}(t_j)\}$ for each time t_j
  / Round the resource demand values */*
  $\mathcal{P}(t_j) \leftarrow$ Round($\mathcal{P}(t_j)$) for each time t_j
  return $\mathcal{P}(t)$ $(t = T_0, ..., T_0 + T)$

System Design: Patterns detection

- Performance of patterns detection algorithm





Google Cluster trace

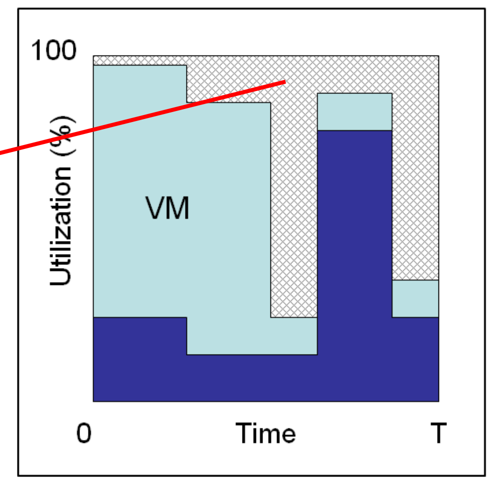
Bigger window size generates fewer missed captures
but leads to more resource provisioning

System Design: Allocation policy

- Classical d-dimensional vector bin-packing

Algorithm 2 Pseudocode for initial VM allocation.

Input: $\mathcal{P}_i(t)$: Predicted resource demand of requesting VM i
$\mathcal{R}_j(t)$: Residual resource capacity of m host candidates
Output: Allocated host of the VM
M=Double.MAX_VALUE //initialize the distance M
for $j = 1$ to m **do**
  if CheckValid($\mathcal{P}(t)$,$\mathcal{R}_j(t)$)==**false then**
    continue
  else
    for $k = 1$ to d **do**
      $E_j^k = E_j^k + \frac{1}{T \cdot H_j^k} \int_{T_0}^{T_0+T} P^k(t)dt$
      $M_j + = \{w_k(1 - E_j^k)\}^2$
    end for
    if $M_j < M$ **then**
      $M = M_j$
      AllocatedHost=host j
end for
return AllocatedHost



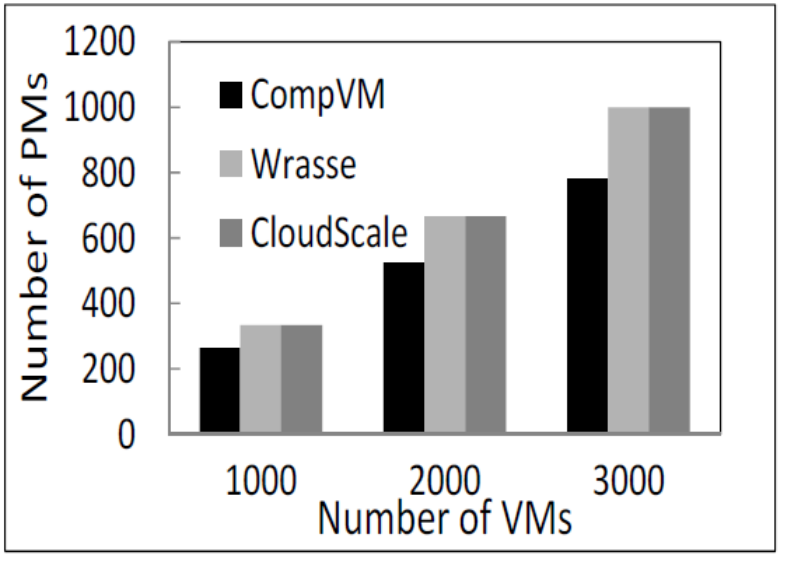
Performance Evaluation: Simulation

- Comparison methods
  - Wrasse [1]: Static provisioning
  - CloudScale [2]: Dynamic provisioning
- Simulation tool
  - CloudSim
- Scale
  - Allocating 1000~3000 VMs
- Traces
  - PlanetLab trace [3]
  - Google Cluster trace [4]

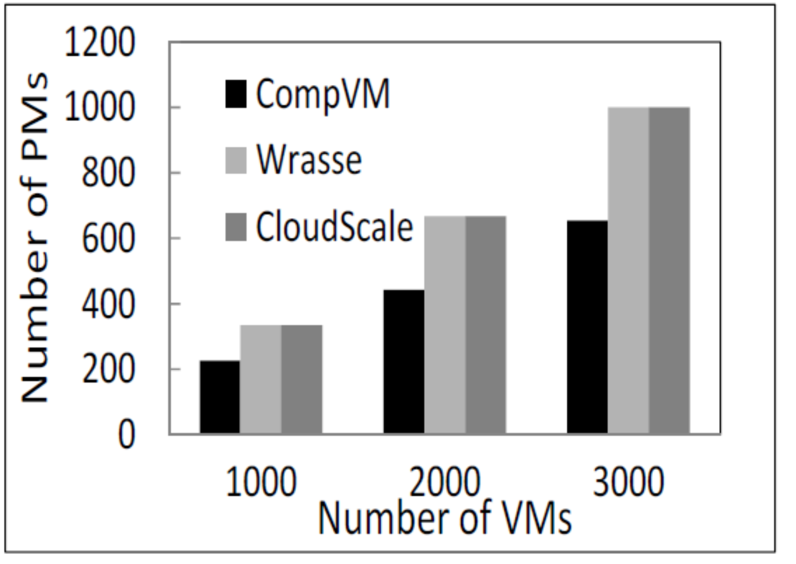
[1] A. Rai, R. Bhagwan, and S. Guha, "Generalized resource allocation for the cloud." in Proc. of SOCC, 2012.
[2] Z. Shen, S. Subbiah, X. Gu, and J. Wilkes, " Elastic resource scaling for multi-tenant cloud systems." in Proc. of SOCC, 2011.
[3] http://www.cloudbus.org/cloudsim/
[4] https://code.google.com/p/googleclusterdata/

Performance Evaluation: Simulation

- The number of PMs needed



PlanetLab trace



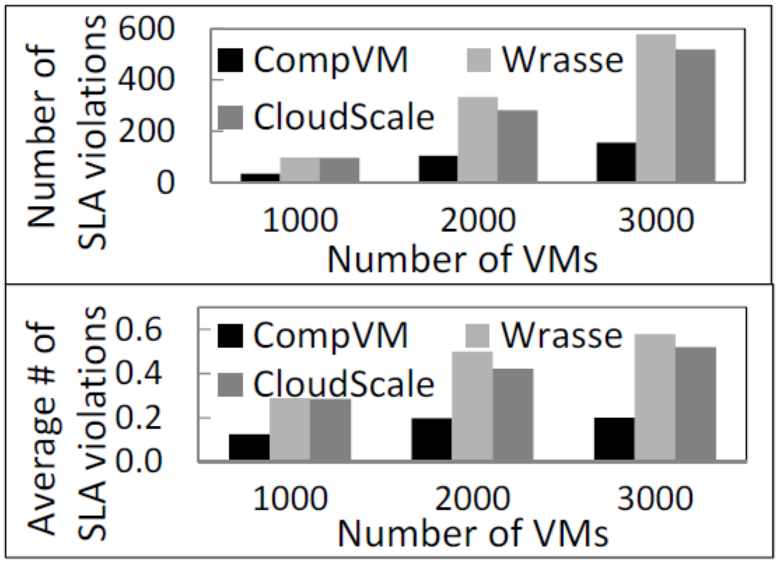
Google Cluster Trace

Result: CompVM < Wrasse = CloudScale

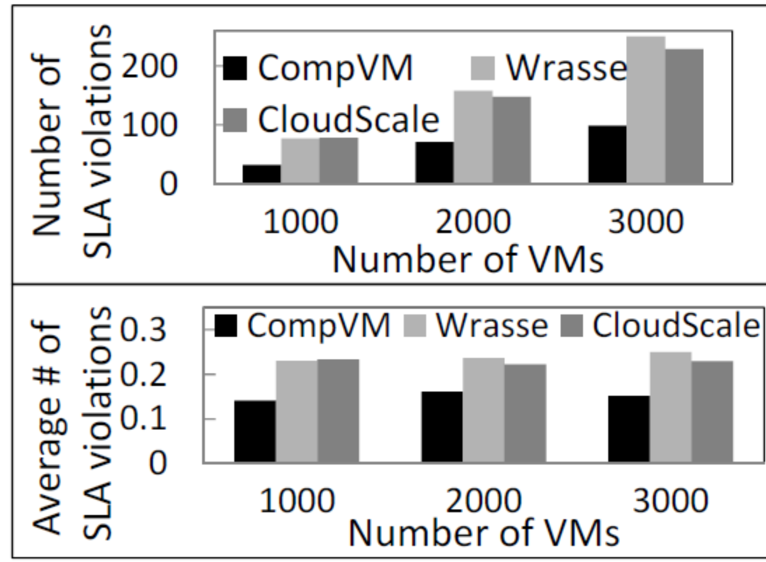
Reason: Wrasse and CloudScale use First Fit to select PM for VM during VM initial placement, without considering complementary VMs

Performance Evaluation: Simulation

- The number of SLA violations



PlanetLab trace



Google Cluster Trace

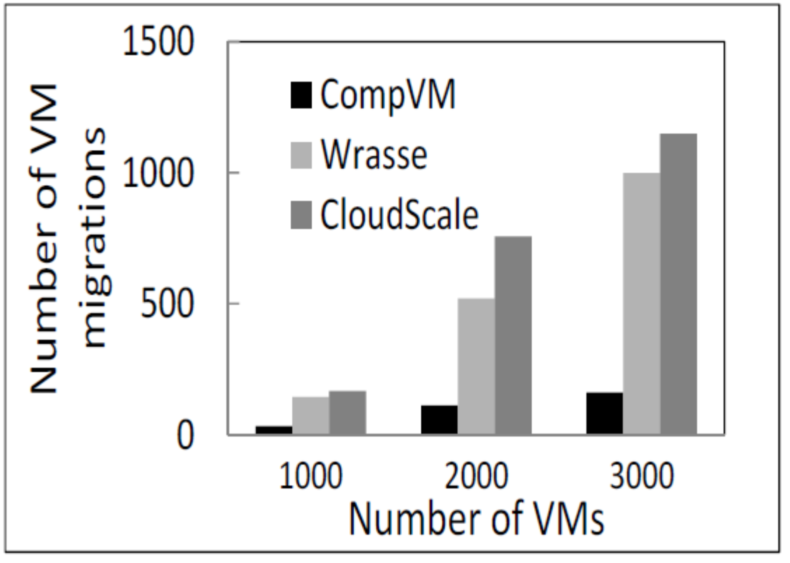
Result: CompVM < CloudScale < Wrasse

Reason: CloudScale predicts demands and migrates VM based on prediction

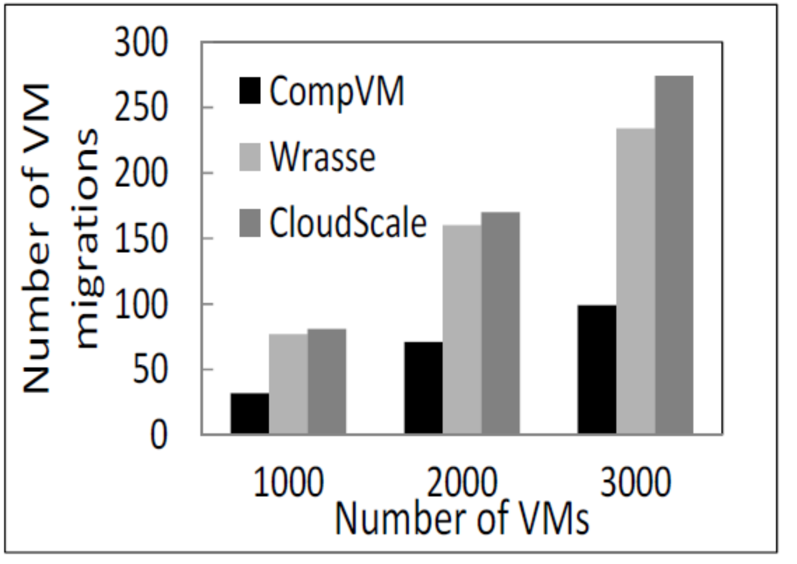
Wrasse migrate VM based on static VM demands as initial placement

Performance Evaluation: Simulation

- The number of migrations



PlanetLab trace



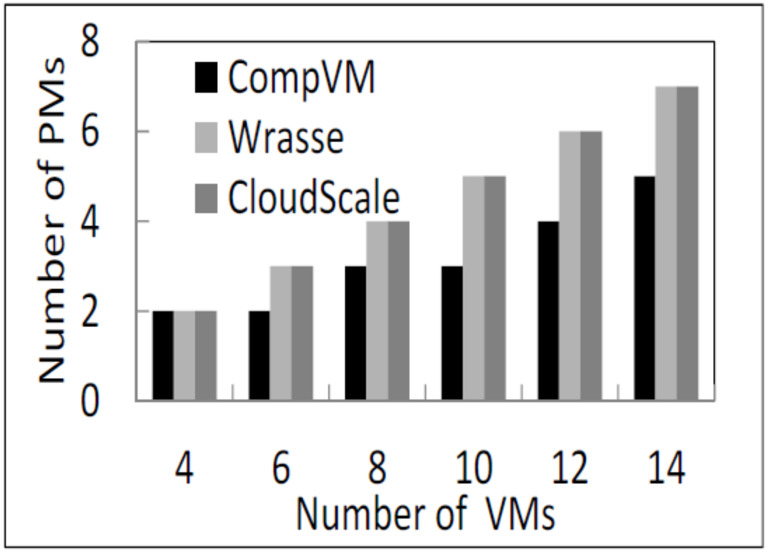
Google Cluster Trace

Result: CompVM < Wrasse < CloudScale

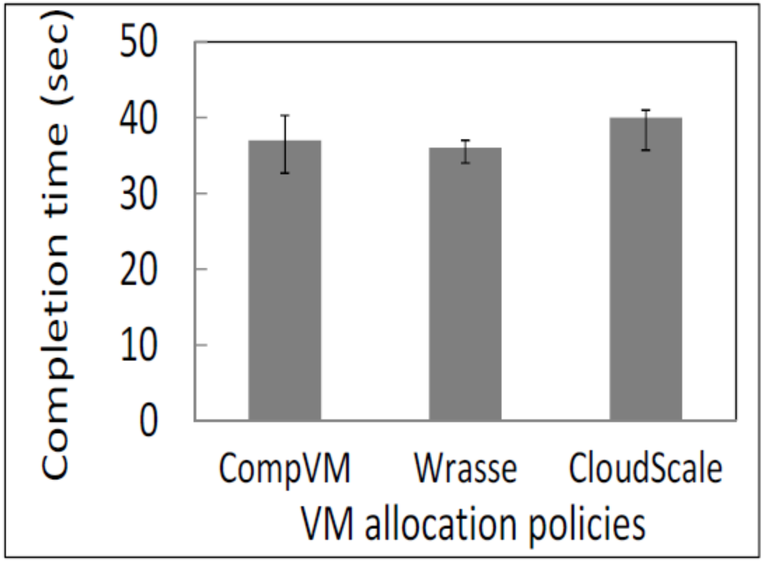
Reason: CompVM outperforms the others due to fewer number of SLA violations
CloudScale has higher number than Wrasse because it triggers VM migration upon a predicted SLA violation, which may not actually occur

Performance Evaluation: Testbed

- The number of VMs and completion time



VMs workloads are generated by workload generator



5 VMs collaboratively running the WordCount Hadoop benchmark

Conclusions

- Studied VMs running short-term MapReduce jobs
- Studied VM resource utilization traces
- Proposed an initial VM allocation mechanism for cloud datacenters that consolidates complementary VMs with spatial/temporal-awareness
- Conducted both trace driven simulation and real testbed experiments

Thank you!

Questions & Comments?

Liuhua Chen

liuhuac@clemson.edu

PhD candidate

Pervasive Communication Laboratory

Clemson University