CloudMirror: Application-Driven Bandwidth Guarantees in Datacenters

HP Labs, [1]University of Edinburgh, [2]DataBricks

JK Lee Yoshio Turner Myungjin Lee[1] Lucian Popa[2] Sujata Banerjee Joon-Myung Kang Puneet Sharma







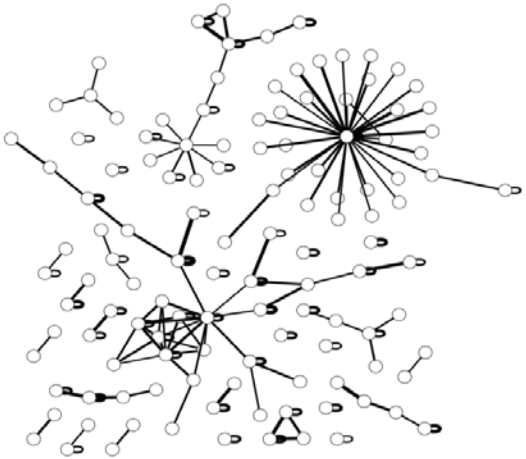
CloudMirror Motivation

Emerging Cloud applications are diverse

Complex combinations of interacting service components

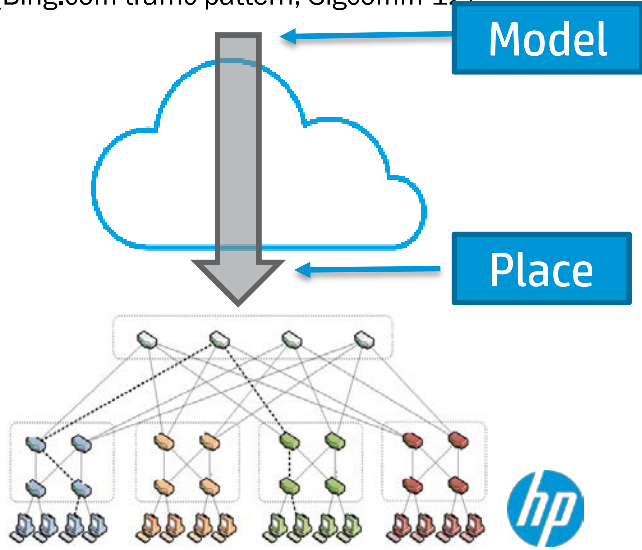
Require predictable performance:

via guaranteed bandwidth and high availability



[Bing.com traffic pattern, Sigcomm'12]

Need to accurately represent and efficiently map application requirements onto shared physical network



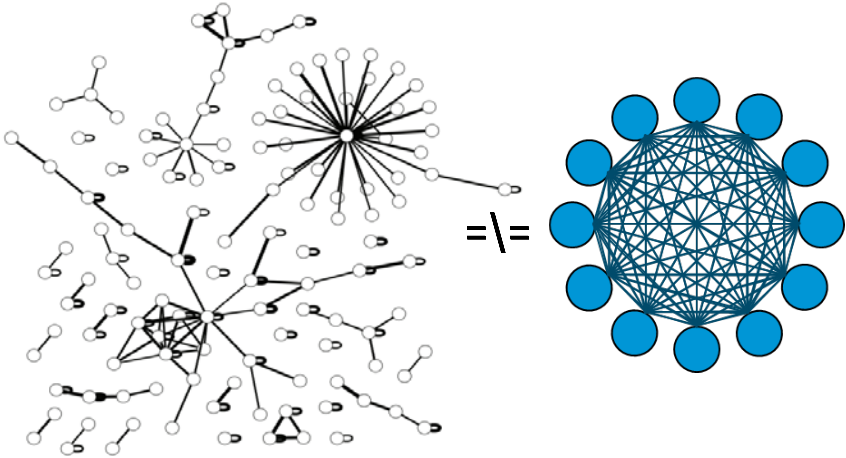
CloudMirror Motivation

Emerging Cloud applications are diverse

Complex combinations of interacting service components

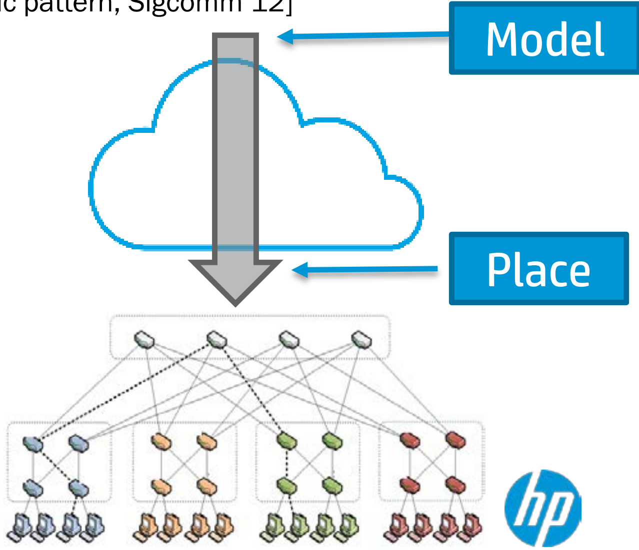
Require predictable performance:

via guaranteed bandwidth and high availability



[Bing.com traffic pattern, Sigcomm'12]

Need to accurately represent and efficiently map application requirements onto shared physical network



Existing models and systems fall short!

Have focused on batch applications like MapReduce, Hadoop w/

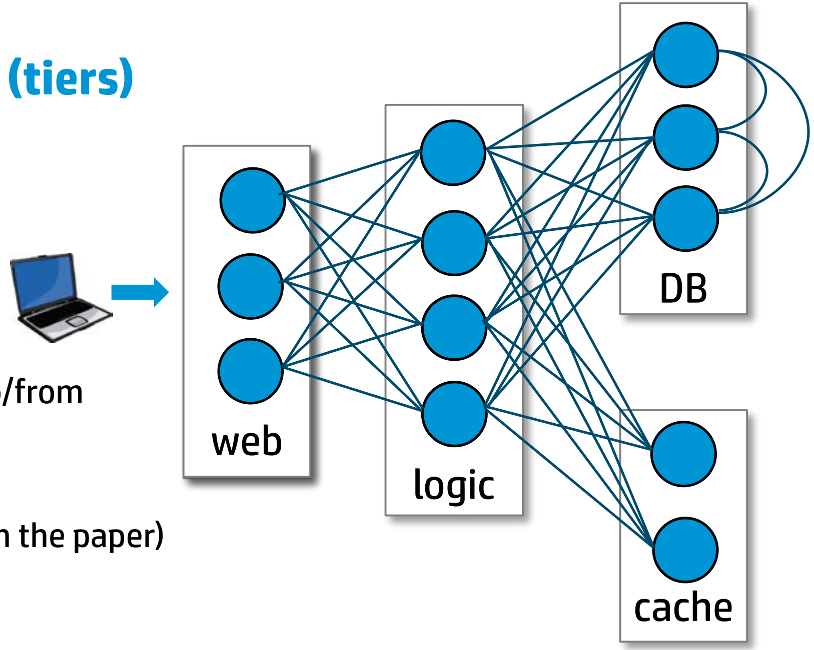
- All-to-all traffic patterns

Interactive online applications

E.g., 3-tier web, enterprise ERP, realtime analytics

Composed of communicating service components (tiers)

Diverse, complex structure



High BW requirement

- Facebook [Farrington, OI'13]

  One HTTP request triggered >500 internal calls

  "*Experiences 1000x more traffic inside its datacenters*" than traffic to/from outside users
- Widely used data-intensive framework

  Redis, VoltDB require up to 10x BW than Hadoop, Hive (benchmarks in the paper)

Delay-sensitive

- Amazon – "*Every 100ms latency costs 1% in (e-commerce) sales*"
- Insufficient bandwidth hurts response time

CloudMirror Goals and System Components

Bandwidth Model

- Accurate to complex applications
- Flexible to elastic scaling
- Intuitive

Tenant Application Graph (TAG)

↓

VM Placement

- Guarantee bandwidth and High-Availability
- Efficient to network and other resources

***CloudMirror* Algorithm**

↓

Runtime Enforcement

- Work-conserving, practical
- Easily guarantee the model

Leverage *ElasticSwitch* [Sigcomm'13]

CloudMirror Goals and System Components

Focus of this talk

Bandwidth Model

- Accurate to complex applications
- Flexible to elastic scaling
- Intuitive

Tenant Application Graph (TAG)

VM Placement

- Guarantee bandwidth and High-Availability
- Efficient to network and other resources

CloudMirror Algorithm

Runtime Enforcement

- Work-conserving, practical
- Easily guarantee the model

Leverage *ElasticSwitch* [Sigcomm'13]

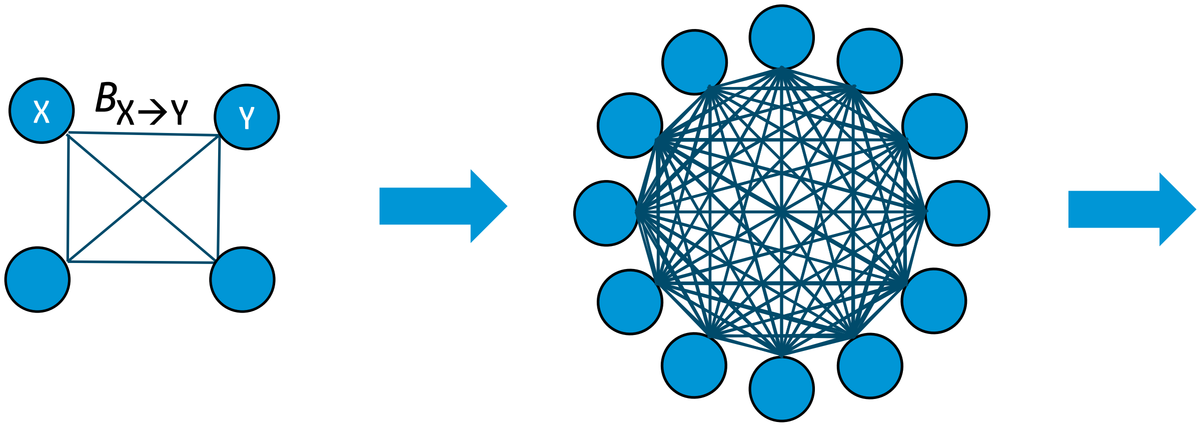
Prior work: Pipe model

Specifies every VM-to-VM pair bandwidth

O(n^2) pipes for n VMs

Not scalable, too slow to compute valid VM placements

- O(n^3) or higher time complexity, for resource-efficient placements

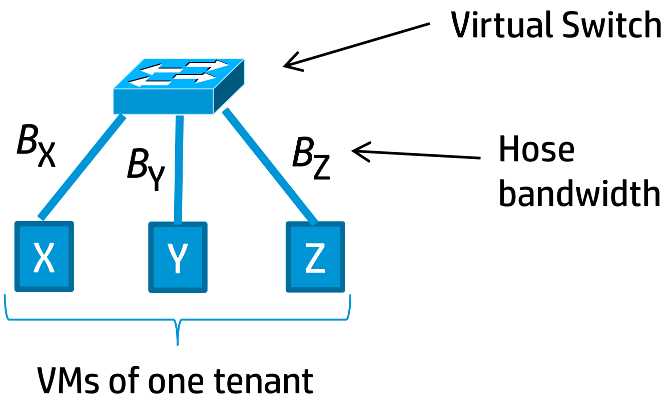


100 VMs
: up to 9,900 pipes

Prior work: Hose model [N. Duffield, Sigcomm'99]

Specifies per-VM aggregate bandwidth

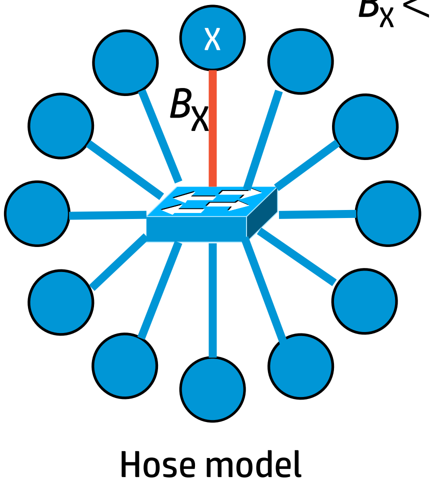
Simple, scalable

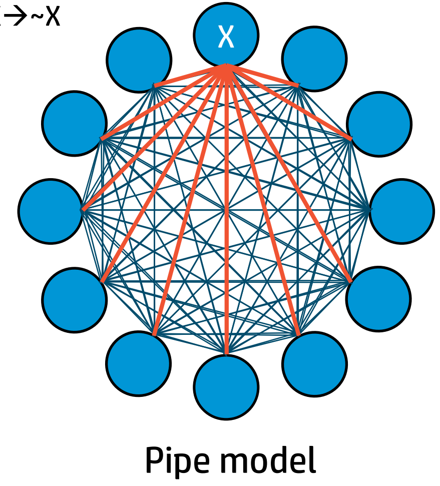


Aggregates BW demands towards all other VMs

Statistical multiplexing gain:

$B_X < \sum B_{X \to \sim X}$





Flat all-to-all communication structure fits well for batch applications like Hadoop

Hose model is unfit

Applications have diverse & complex structures

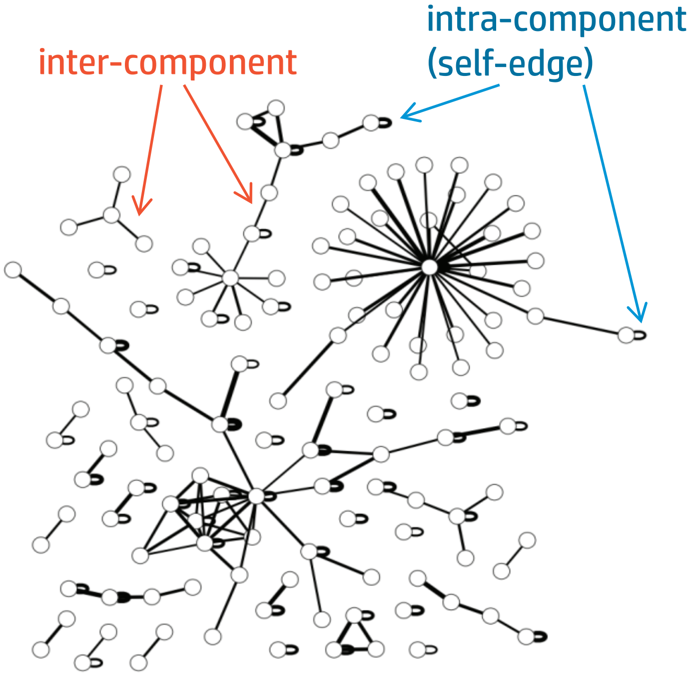
Star, linear, mesh, … from bing.com datacenter [Bodik, Sigcomm'12]

Inter-component traffic dominates

Hose aggregates BW towards different components

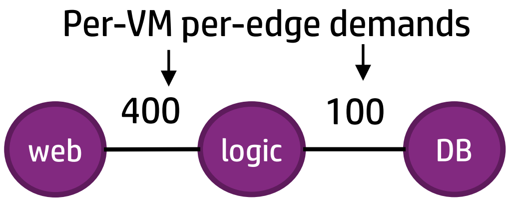
Too coarse-grained

Prevents accurate and efficient guarantees on infrastructure

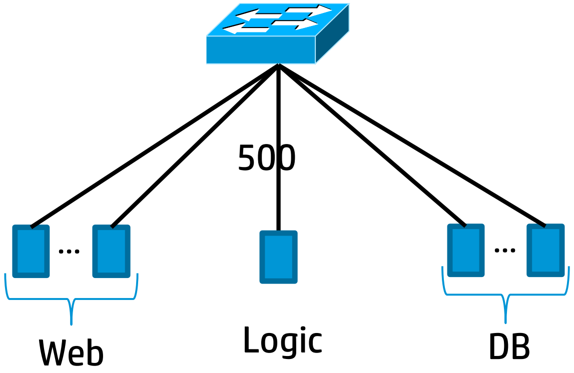


Hose is too coarse-grained

3-tier web example



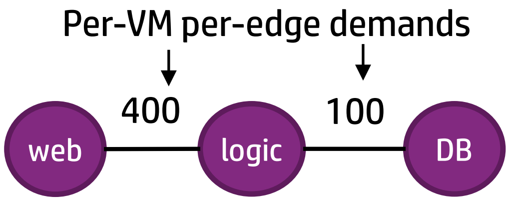
Hose model



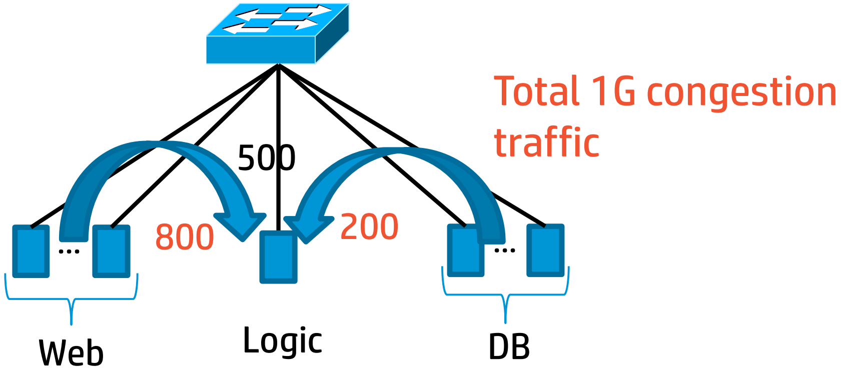
400+100 = 500 Mbps Hose guarantee for a Logic VM

Hose is too coarse-grained

3-tier web example



Hose model

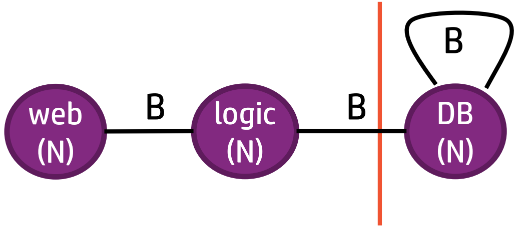


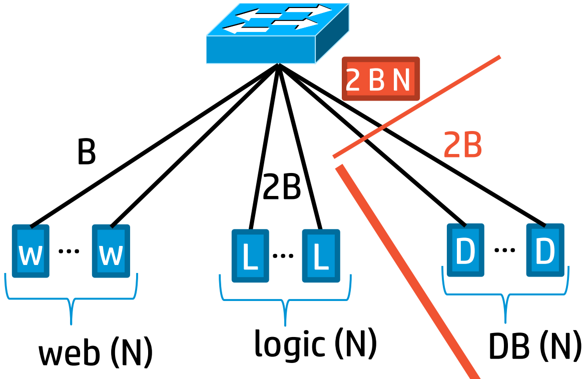
400+100 = 500 Mbps Hose guarantee for a Logic VM

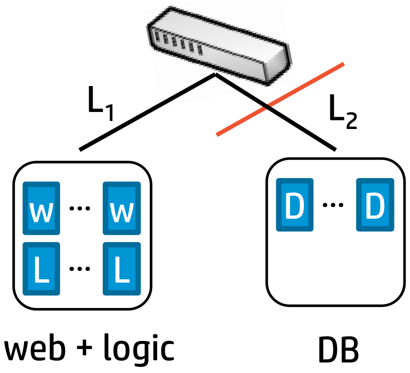
At congestion, TCP-like fair allocation would split 500 into 300:200

Failing to provide 400:100 that application requires

Hose over-provisions physical link bandwidth







Physical deployment example

N: # VMs in each tier

B: per-VM per-edge bandwidth

Hose model reservation at L_2 : $2B \cdot N$

logic - DB demand = $B \cdot N$

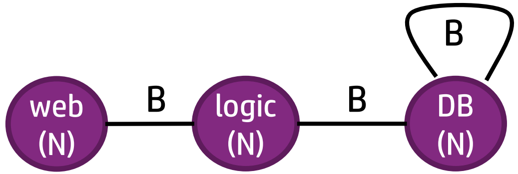
2X overprovision by Hose Model

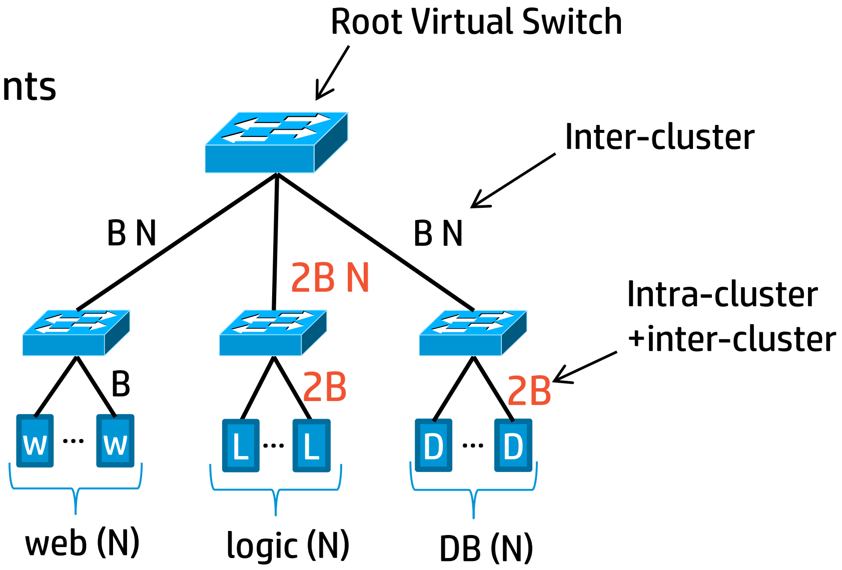
Virtual Oversubscribed Cluster (VOC) [Ballani, Sigcomm'11]

a.k.a Virtual Tree [NSDI'13]

2-level hierarchical hose model

To better model applications having multiple components





VOC model

Aggregates demands towards different clusters

→ too coarse-grained

Lessons

1. Aggregate pipes (like Hose)

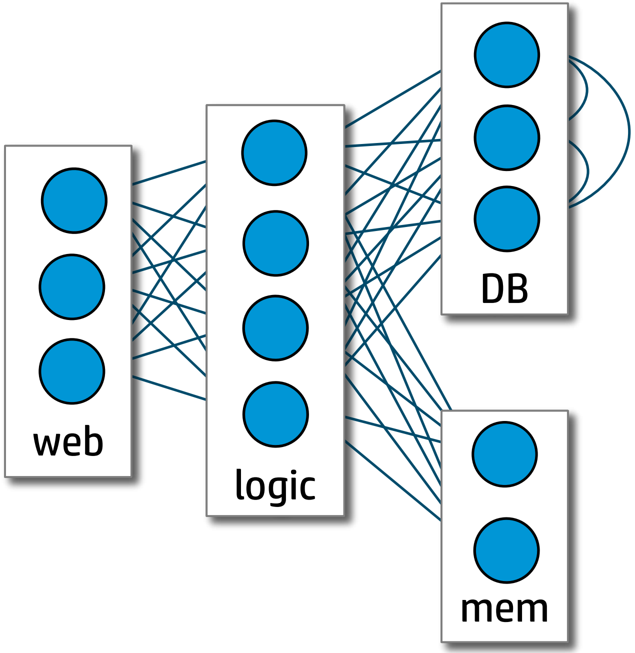
Model simplicity

Multiplexing gain

2. Preserve inter-component structure (like Pipe)

Accurately capture application demands

Efficiently utilize network resources



Lessons

1. Aggregate pipes (like Hose)

Model simplicity

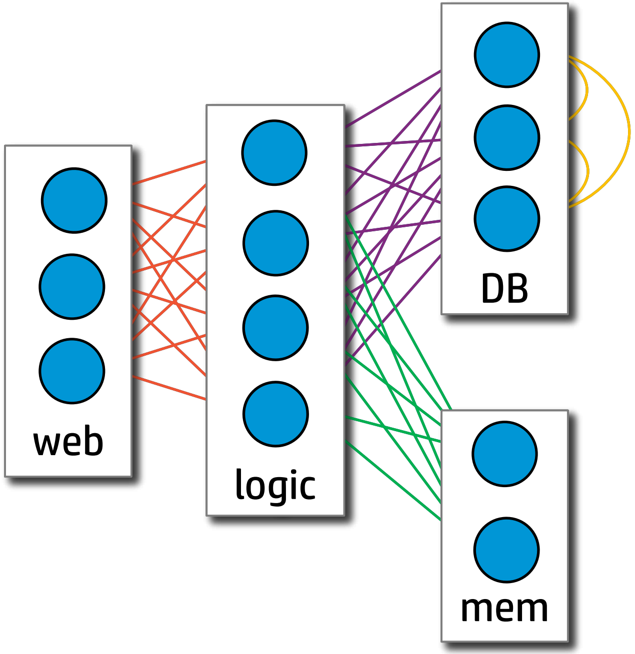
Multiplexing gain

2. Preserve inter-component structure (like Pipe)

Accurately capture application demands

Efficiently utilize network resources

Solution:
aggregating pipes only between a pair of communicating tiers (same-color pipes)



Lessons

1. Aggregate pipes (like Hose)

Model simplicity

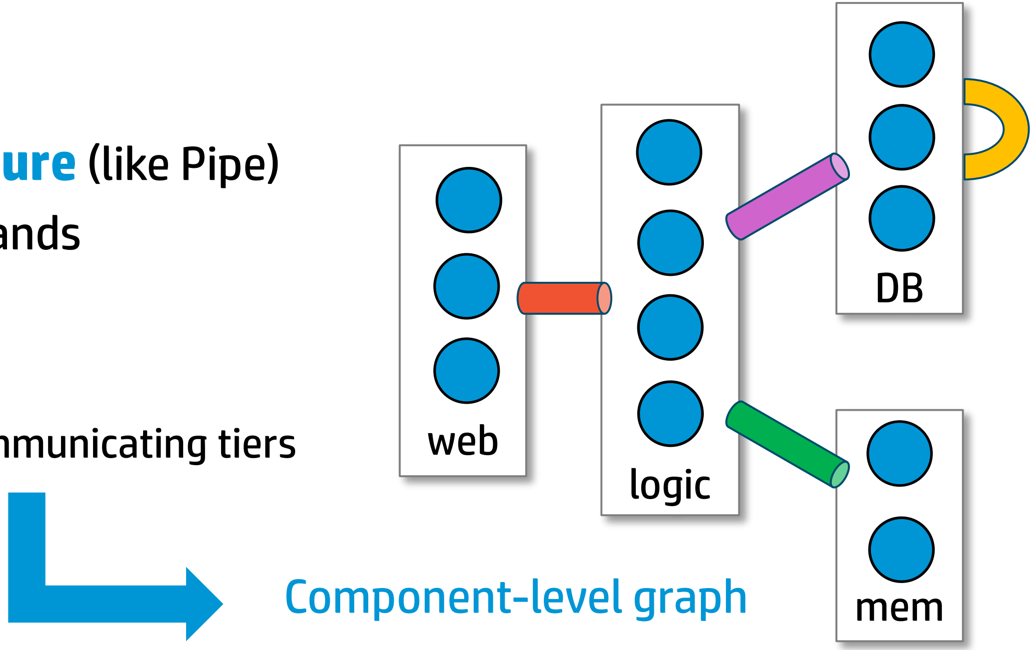
Multiplexing gain

2. Preserve inter-component structure (like Pipe)

Accurately capture application demands

Efficiently utilize network resources

Solution:
aggregating pipes only between a pair of communicating tiers



Tenant Application Graph (TAG)

Vertex = application component (or tier)

A group of VMs performing the same function

Vertex size N = # of VMs

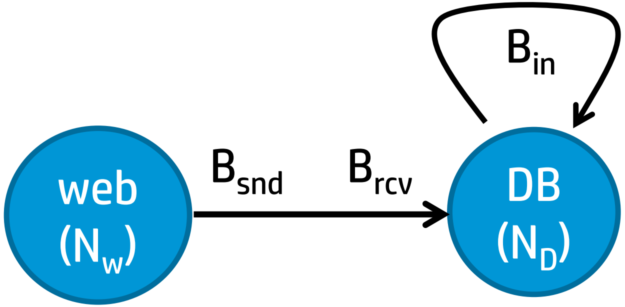
Two types of edges

Directional edge between two vertices: inter-component

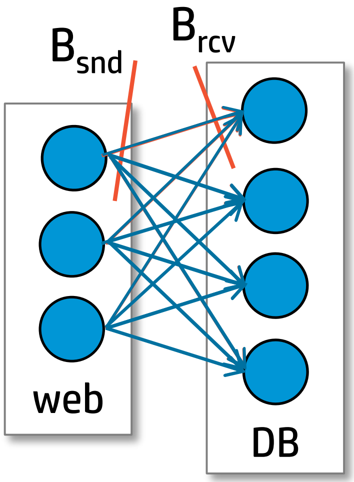
- B_{snd} = per-VM sending bandwidth (VM-to-component aggregation)
- B_{rcv} = per-VM receiving bandwidth (component-to-VM aggregation)

Self-edge: intra-component

- B_{in} = per-VM sending/receiving for traffic between the same tier VMs



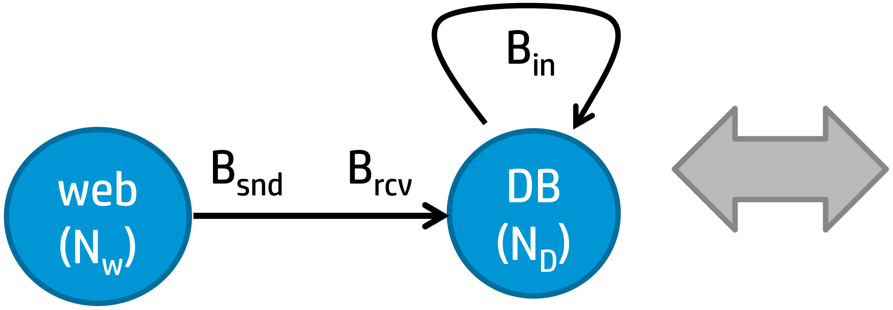
TAG model



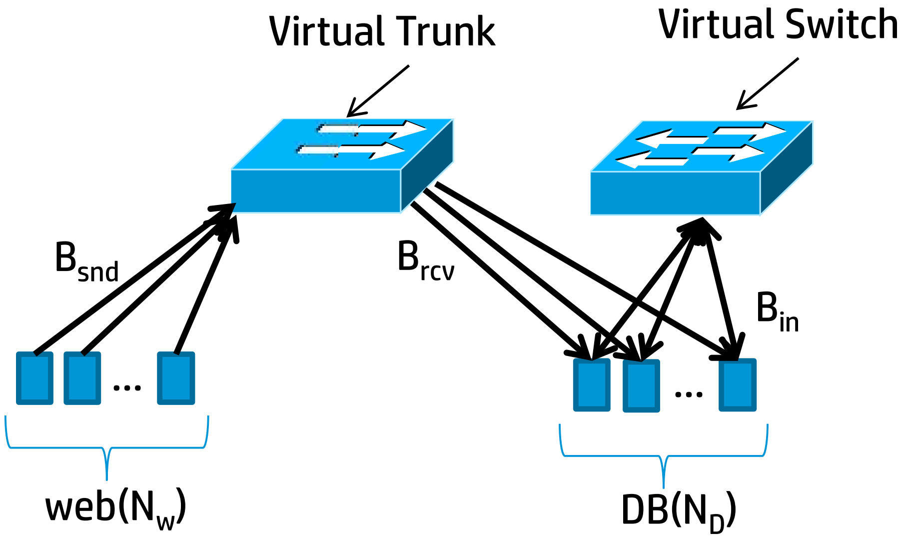
Abstract models in TAG

Self-edge ↔ Hose

Directional edge ↔ directional Hose, Virtual Trunk



TAG model



Total guarantee of virtual trunk

$= \min(B_{snd} \cdot N_w, B_{rcv} \cdot N_D)$

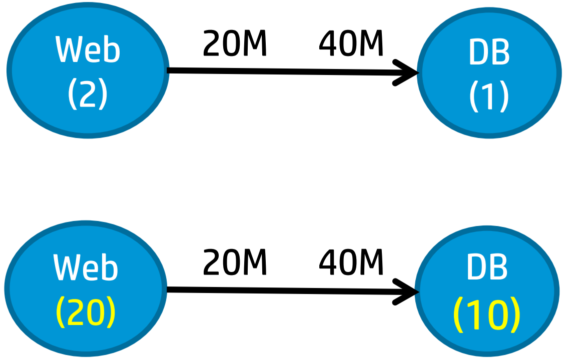
TAG is Flexible

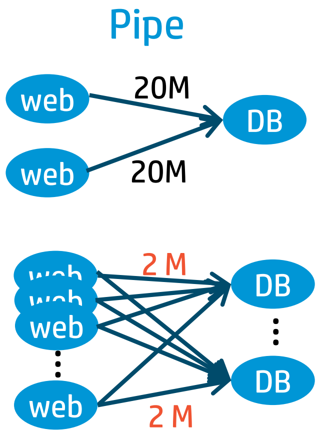
Per-VM aggregation from/to each other component

Flexible to dynamic load distribution between communicating tiers

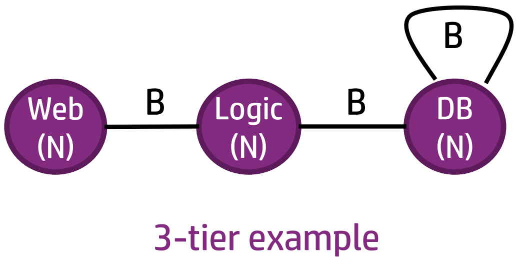
Flexible to elastic tenant scaling

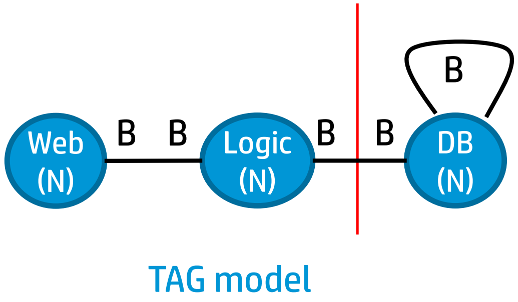
- TAG *per-VM* bandwidth guarantees invariant to scaling
- Pipe reservations need to be updated while scaling

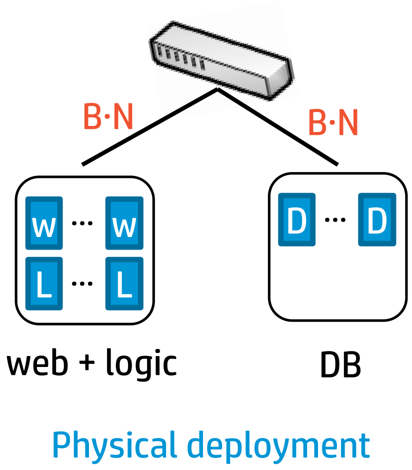




TAG is accurate and resource-efficient





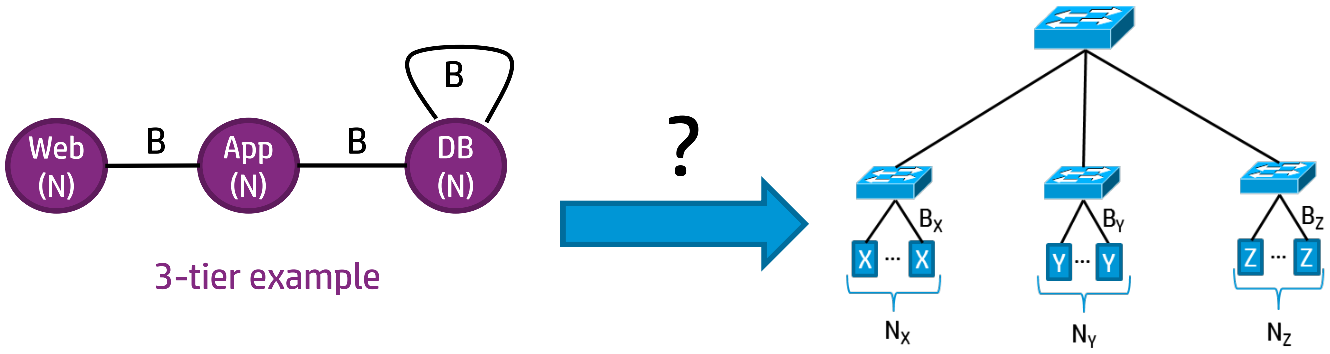


TAG requires less or equal BW than VOC

- Equal when there is no inter-component demand
- Mathematically proven

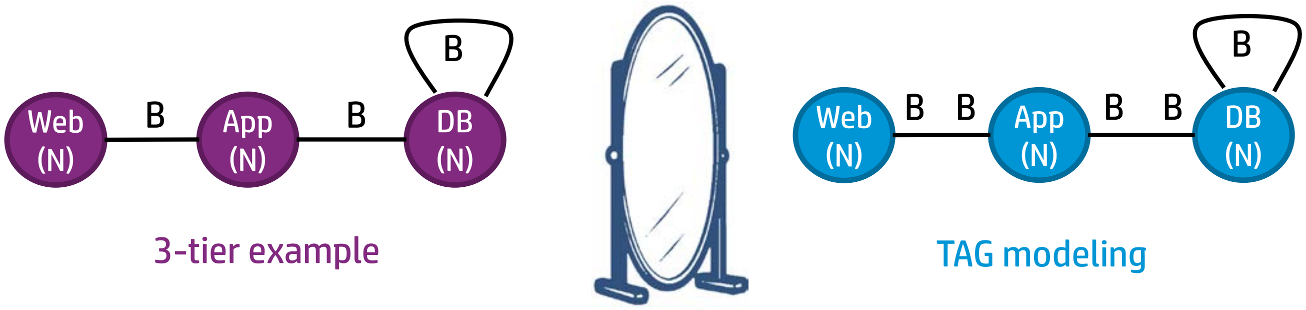
TAG is Intuitive

TAG is easy to use because it directly mirrors application structure



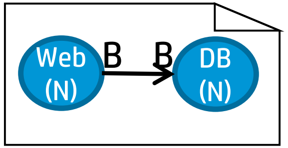
TAG is Intuitive

TAG is easy to use because it directly mirrors application structure



CloudMirror operation

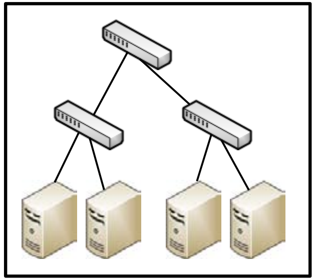
TAG spec



Available VM slots

host1	10
host2	50
host3	25

Network topology &
BW reservation state



VM placement
BW reservation
Admission control

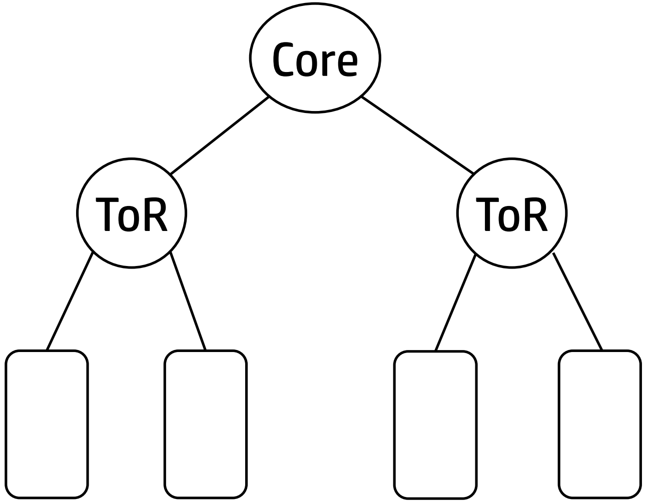
VM placement

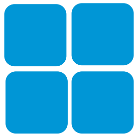
Goal

Deploy as many tenants as possible onto a tree-shaped topology while guaranteeing SLAs

➔ NP-hard problem

Prior heuristics = colocation, localize traffic and save core bandwidth





Client requests 4 VMs

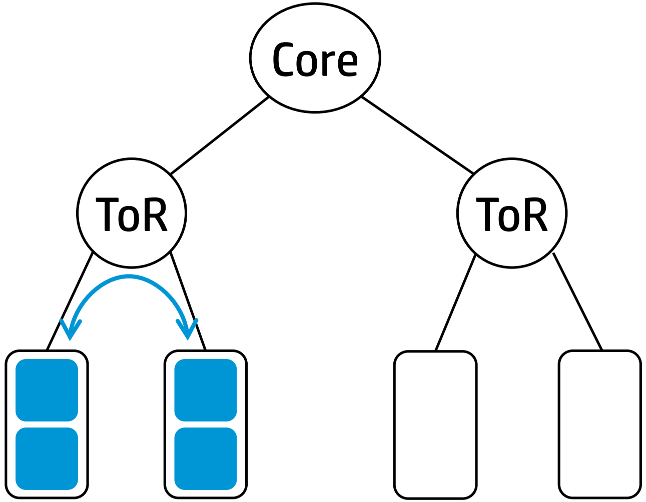
VM placement

Goal

Deploy as many tenants as possible onto a tree-shaped topology while guaranteeing SLAs

➔ NP-hard problem

Prior heuristics = colocation, localize traffic and save core bandwidth

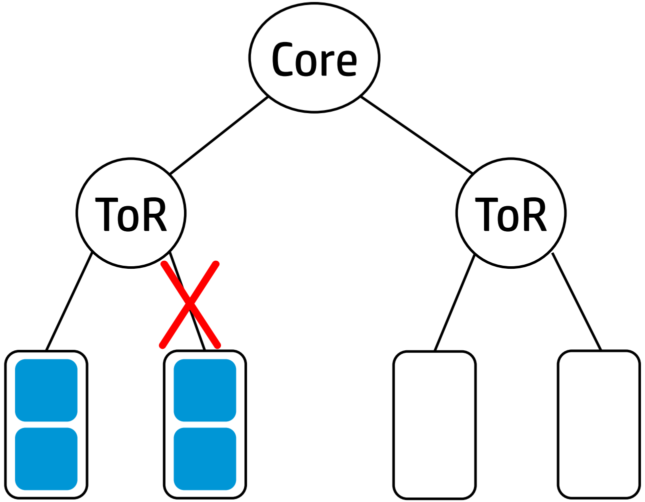


CloudMirror also takes colocation approach:

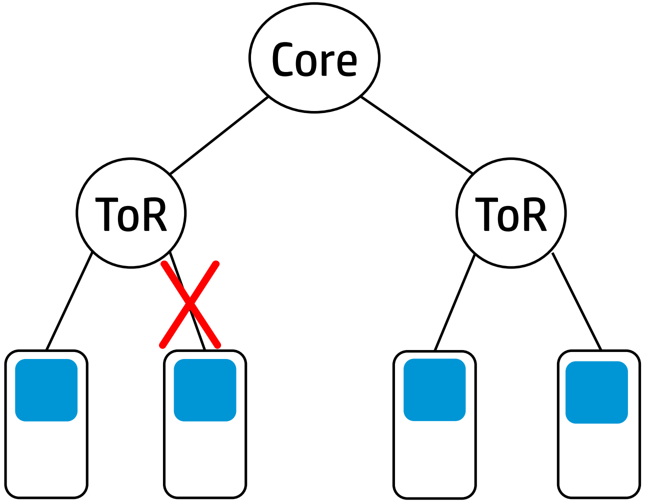
BW saving benefit using TAG $\geq$ VOC, Hose

Disadvantages of colocation

Colocation hurts high-availability [Bodik, Sigcomm'12]



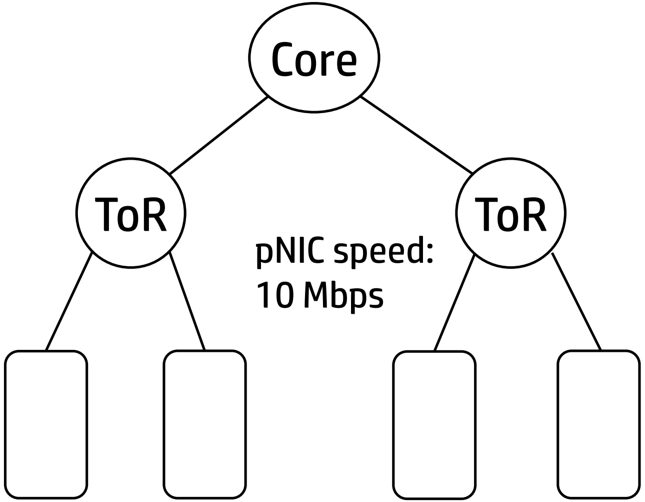
A server failure → 50% survive

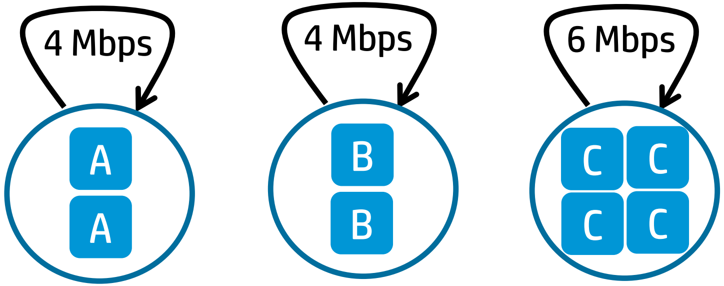


A server failure → **75%** survive

Disadvantages of colocation

Colocation can hurt efficient resource utilization

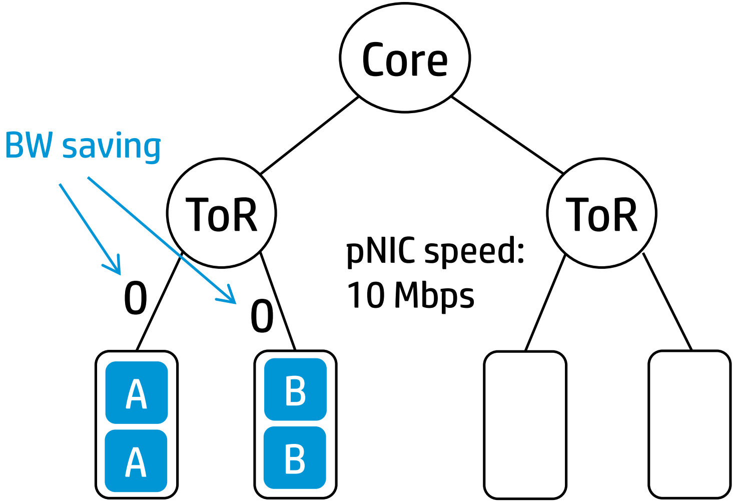




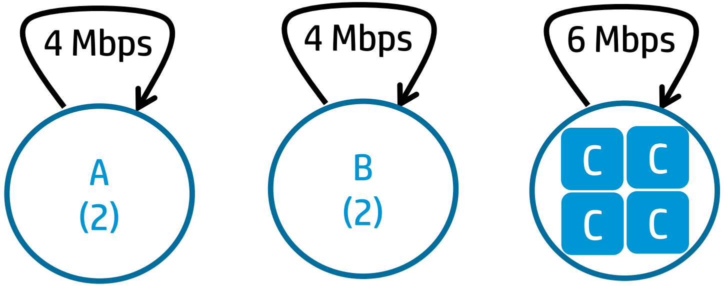
Client requests 3 hose components

Disadvantages of colocation

Colocation can hurt efficient resource utilization



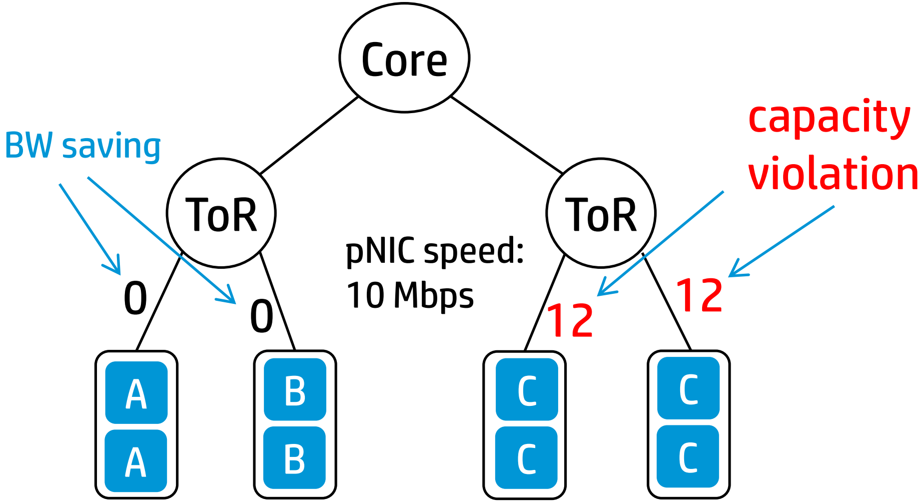
Colocation for saving bandwidth of A and B



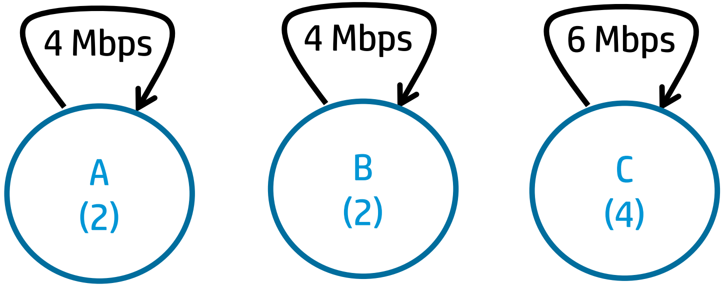
Client request

Disadvantages of colocation

Colocation can hurt efficient resource utilization



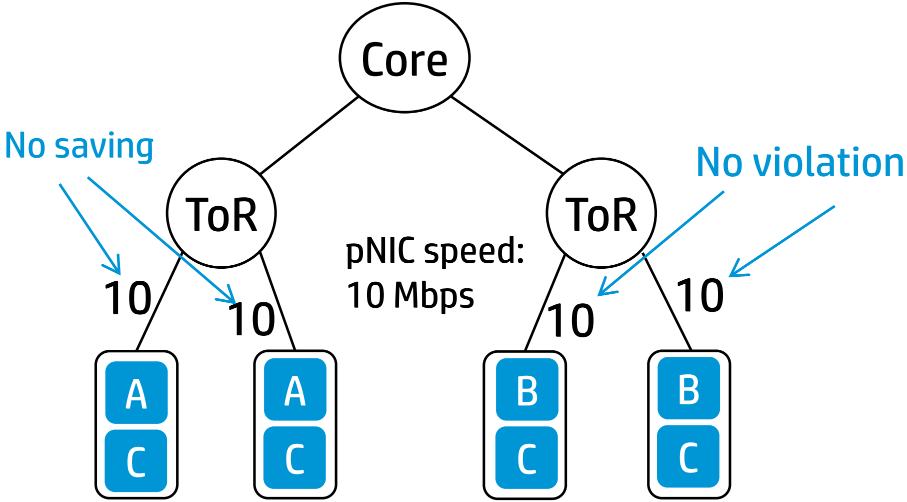
Colocation for saving bandwidth of A and B

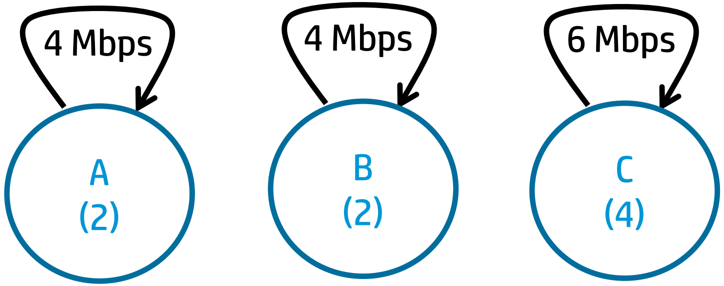


Client request

Disadvantages of colocation

Colocation can hurt efficient resource utilization





Client request

Valid placement: Colocate high-BW VMs (C) and low-BW VMs (A,B) though they don't talk to each other

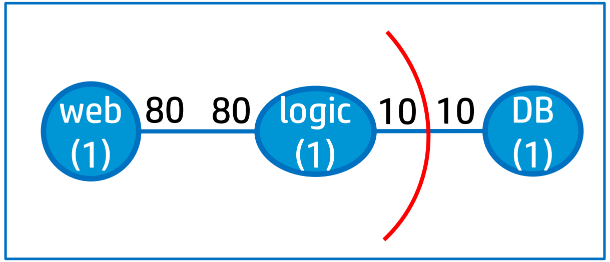
➔ **balanced, efficient utilization of network and VM slots**

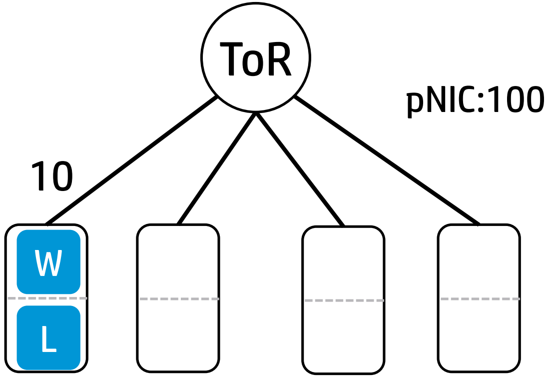
CloudMirror approach

Identify tiers that would benefit from ***colocation***

More than ½ VMs of communicating tiers should be colocated

Iterate over tier pairs: $O(T^2)$, T = # tiers



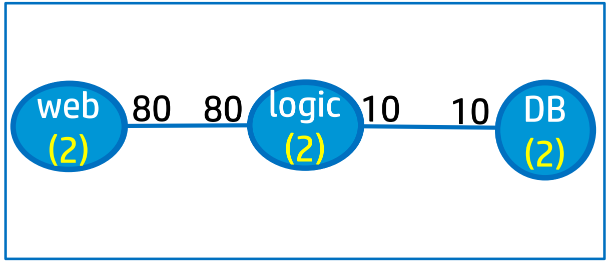


CloudMirror approach

Identify tiers that would benefit from *colocation*

More than ½ VMs of communicating tiers should be colocated

Iterate over tier pairs: $O(T^2)$, T = #tiers

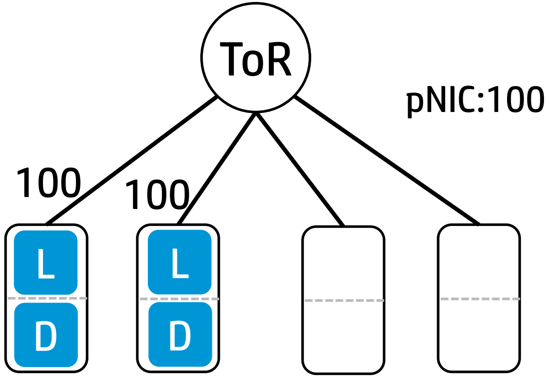


For tiers w/o BW saving benefit (too large to colocate)

Balance resource utilizations over subtrees and resource types

Multi-dimensional knapsack, Multi-resource 'VM' packing

- Our greedy heuristic: $O(N)$, N = #VMs
- VM scheduling/packing makes more sense than task/flow level scheduling for online-applications that handle continuous streams of user requests or realtime data

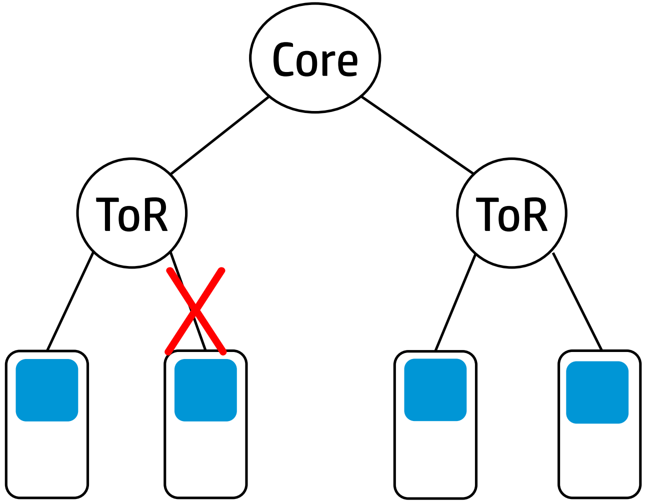


Achieving High-Availability

1) Guarantee survivability for requesting tiers

Limit #VMs colocated in the same subtree

- E.g., 75% survivability, 4-VM tier: at most 1 VM per subtree



A server failure → 75% survival guaranteed

Achieving High-Availability

1) Guarantee survivability for requesting tiers

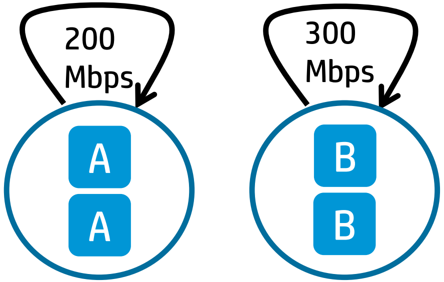
Limit #VMs colocated in the same subtree

- E.g., 75% survivability, 4-VM cluster: at most 1 VM per subtree

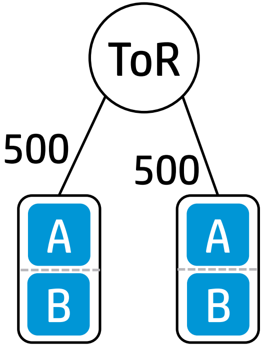
2) Opportunistically improve HA for others

Even for tiers with BW saving benefit,

distribute VMs when BW is not a bottleneck



Client request



Link capacity: 1Gbps

Evaluation

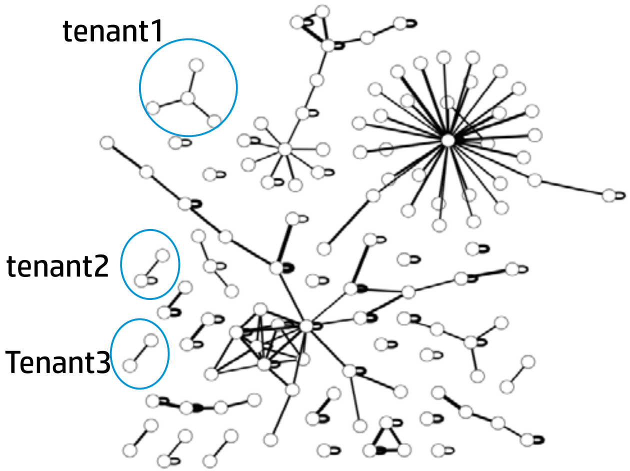
Methodology

Simulate a stream of (Poisson) tenant arrivals and departures

Workload: bing.com data

Various cluster sizes, communication patterns

Tenant: a set of connected components



Source: [Bodik, Sigcomm'12]

3-level tree topology

2048 hosts, 25 VM slots per host

Baseline

Model: VOC (2-level hose)

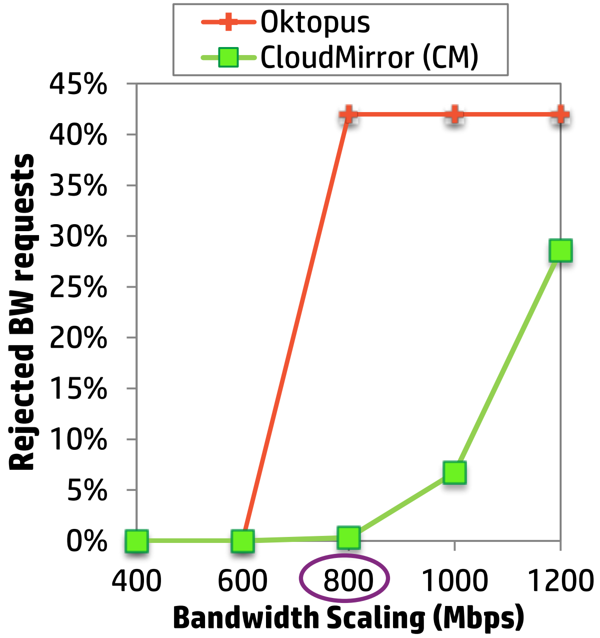
Algorithm: Oktopus [Sigcomm'11]

Resource efficiency

Metric: Rejected BW requests

Oktopus rejects 42% more BW requests

(Synthetic workload: up to 100% difference)

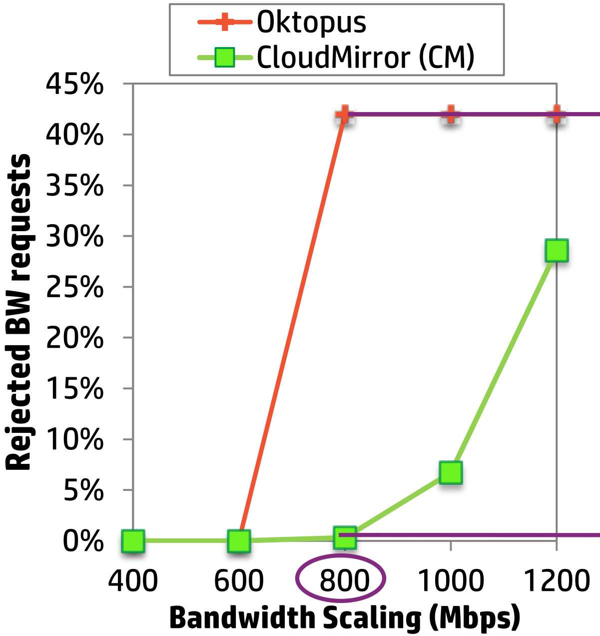


Resource efficiency

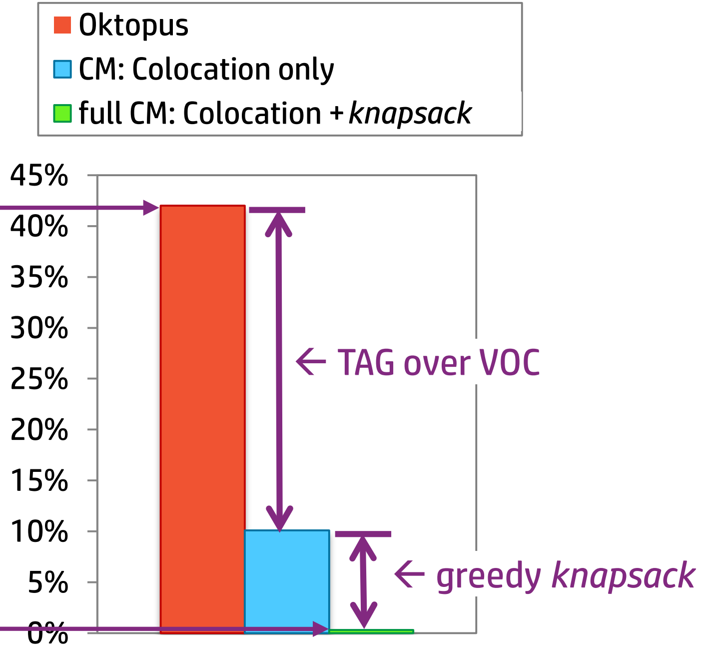
Metric: Rejected BW requests

Oktopus rejects 42% more BW requests

(Synthetic workload: up to 100% difference)



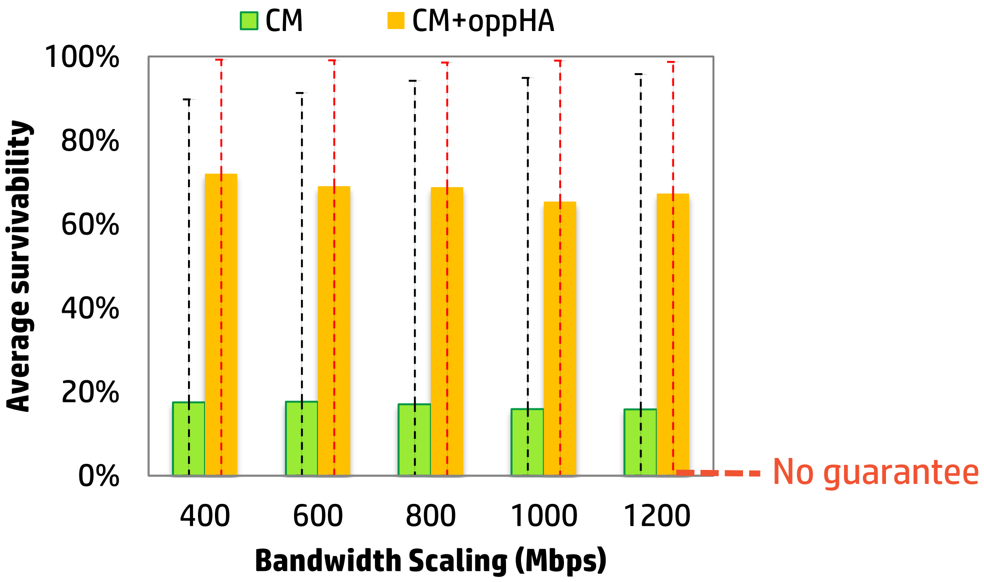
Dissect benefit of CloudMirror



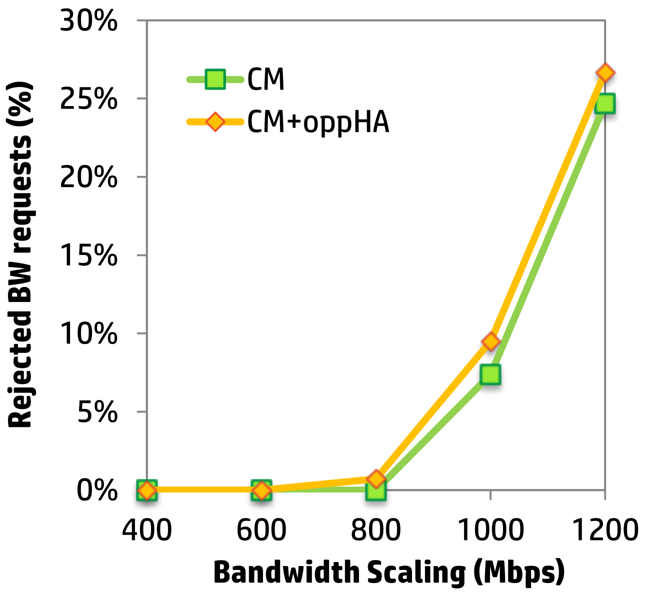
Opportunistic High-Availability

Achieved survivability

Over deployed application components



Rejected BW requests



CM+oppHA achieves high average survivability w/ marginal increase of rejected BW requests

CloudMirror recap

TAG models application structure

Intuitive, flexible and efficient

Algorithm efficiently places TAGs on tree-shaped topology

with High-Availability supports

Feasibility test with *ElasticSwitch* [Sigcomm'13] for TAG enforcement

30 lines of patch to support TAG

Ongoing work

Full implementation in *OpenStack*

Automatic generation of TAG models

Discussed in the paper

Generic graph model

Specify other SDN requirements and policies besides bandwidth