Azure's VM Allocator Internals

Marcus Fontoura
Microsoft

Thomas Moscibroda, Yang Chen, et al.
Microsoft Research
Mark Russinovich, Jim Johnson, Nar Ganapathy,
Ashwin Kulkarni, et al.
Azure

Microsoft

Motivation

- What is Azure
- A bit of history
- Where we are going

Azure Global Footprint

Operational

Announced/Not Operational

* Operated by 21Vianet

- 100+ datacenters
- Top 3 networks in the world

Azure Scale

~100,000 New Azure customer subscriptions/month	20 Million SQL database hours used every day	>60 Trillion Storage objects in Azure	>7 Trillion Storage transactions every month
425 Million Azure Active Directory Users	60 Billion Hits to Websites run on Azure Web App Service	57% Of Fortune 500 Companies use Microsoft Azure	>1 Trillion Messages delivered every month with Event Hubs

Resource utilization in Azure

- Each 1% of utilization gain results in millions of $ savings

Resource utilization in Azure

- Each 1% of utilization gain results in millions of $ savings

VM allocation algorithms are crucial for operating Azure effectively!

AZURE INTERNALS

Virtual Machine Types

- Azure currently has three VM families:

A: High-Value

Type	Cores	RAM
A0	1	0.768
A1	1	1.75
A2	2	3.5
A3	4	7
A4	8	14
A5	2	14
A6	4	28
A7	8	56
A8	8	56
A9	16	112
A10	8	56
A11	16	112

High Memory

Infiniband

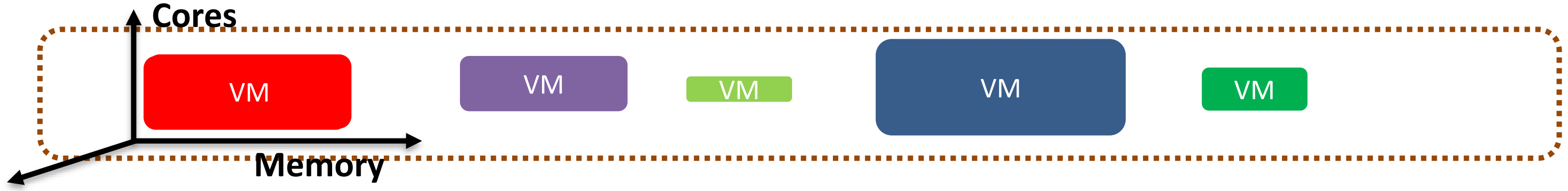
Faster CPUs

D: Low-Latency, SSD

Type	Cores	RAM
D1	1	3.5
D2	2	7
D3	4	14
D4	8	28
D11	2	14
D12	4	28
D13	8	56
D14	16	112

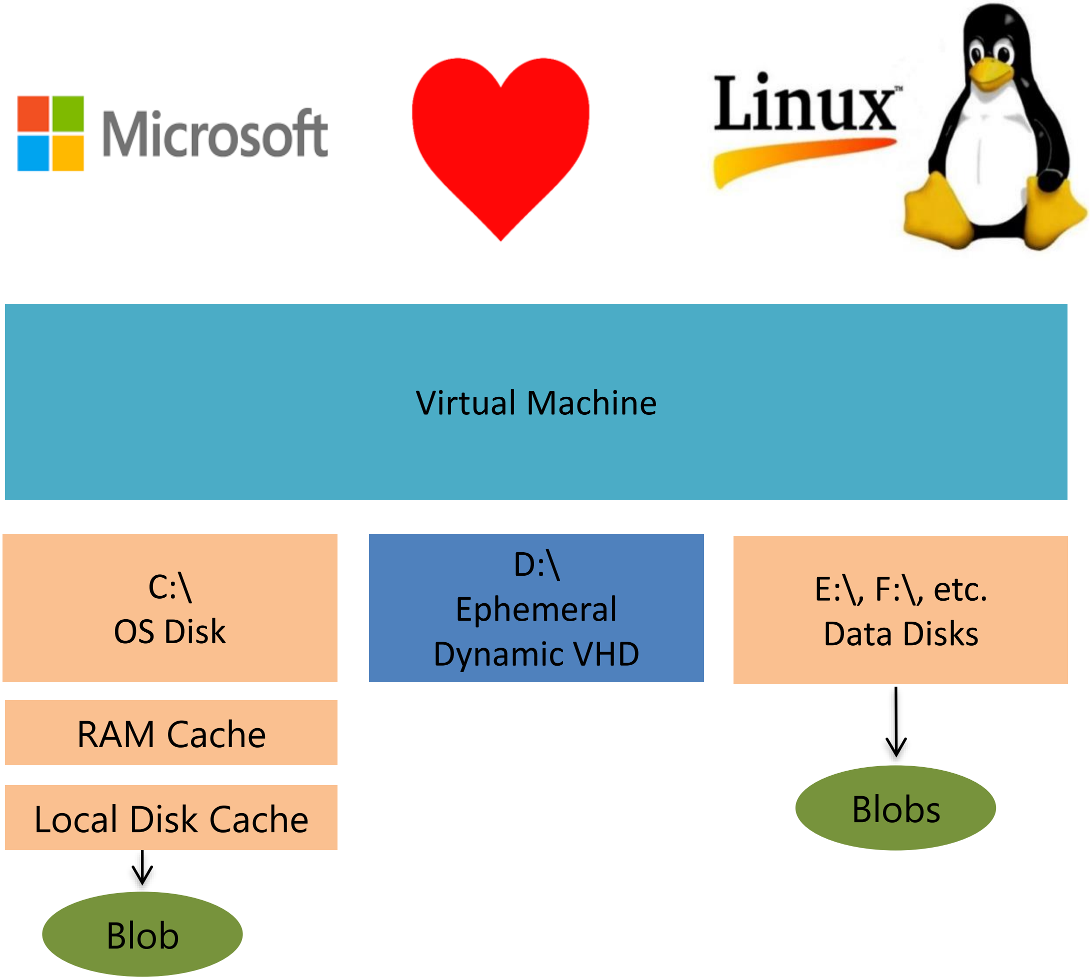
G: Extreme Performance, SSD

Type	Cores	RAM
G1	2	28
G2	4	56
G3	8	112
G4	16	224
G5	32	448



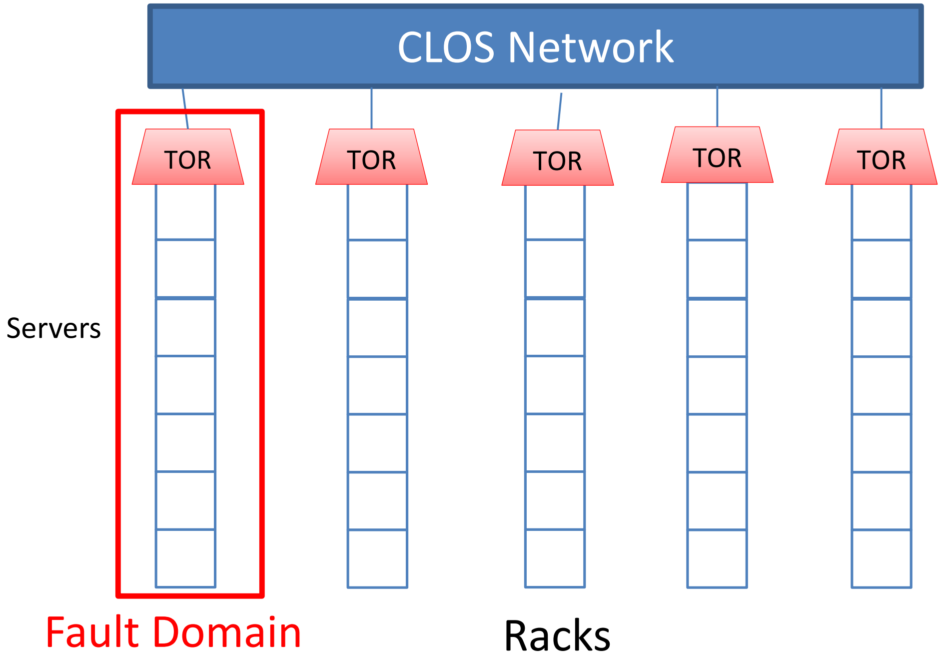
Virtual Machine Architecture

- Network, local and remote storage are additional allocation dimensions
- Ephemeral storage: uses local storage bandwidth and space
  - Backed by local HDD or SSD
- Persistent storage: uses network bandwidth
  - Cached on local server RAM, HDD or SSD
  - Backed by Azure Storage page blobs
  - “S” variants (e.g. “DS14”) can use SSD-backed Premium Storage



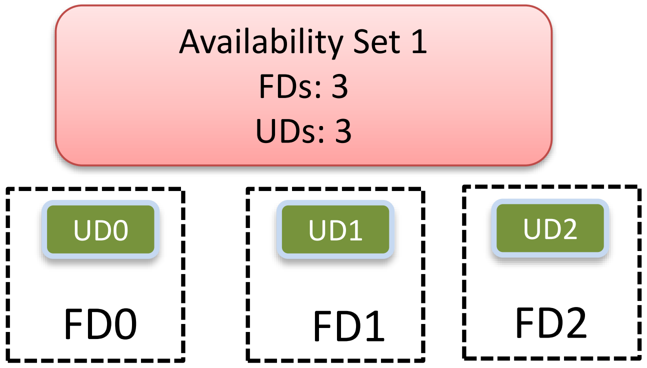
Availability Domains - FDs and UDs

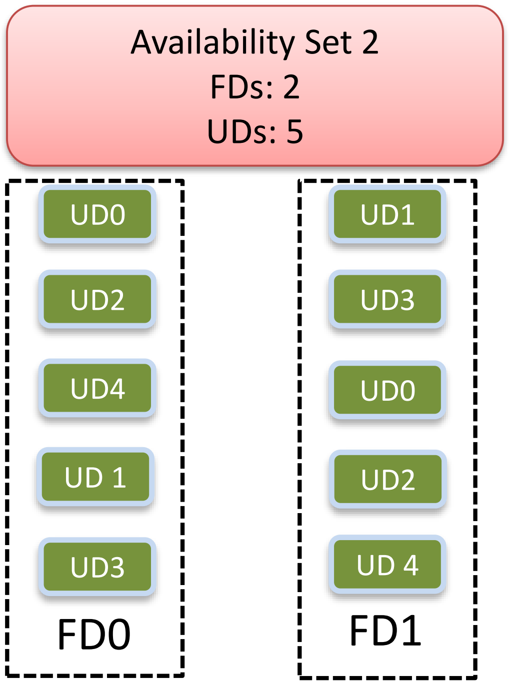
- Fault domain: largest scope single-point of failure in a datacenter
  - SPoFs: server, TOR, PDU => rack
- Update domain: group of servers that can be updated in parallel
  - Periodic host software (e.g. hypervisor and OS) require reboots
  - Some VMs may not wish to be rebooted concurrently



Availability Set - FD and UD Constraints

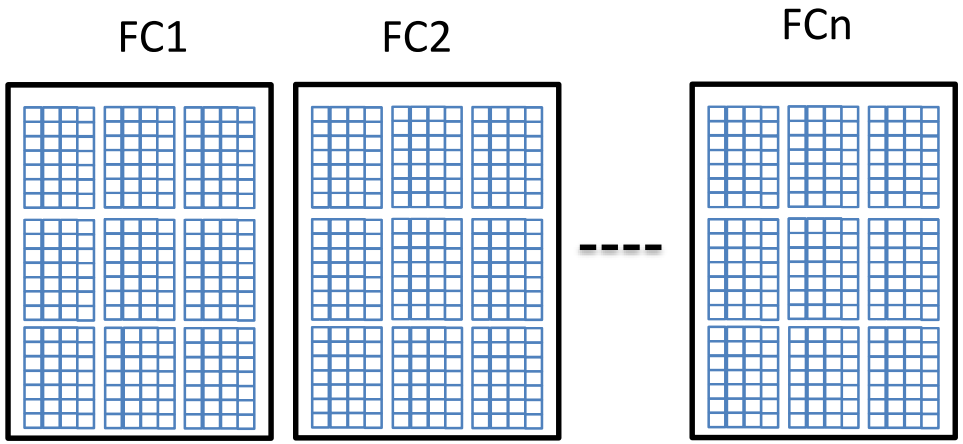
- Availability Sets group collections of VMs with related availability constraints
  - Up to 3 FDs, up to 20 UDs M
  - More FDs available for infrastructure
- Examples:
  - 3 VMs performing Paxos replication: 3 FDs
  - 10 VMs serving web requests: 90% availability goal





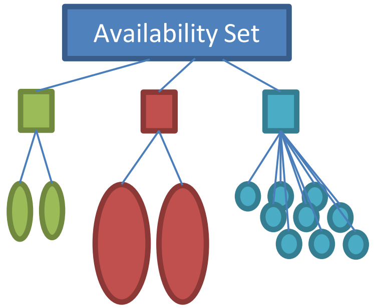
Fabric Clusters

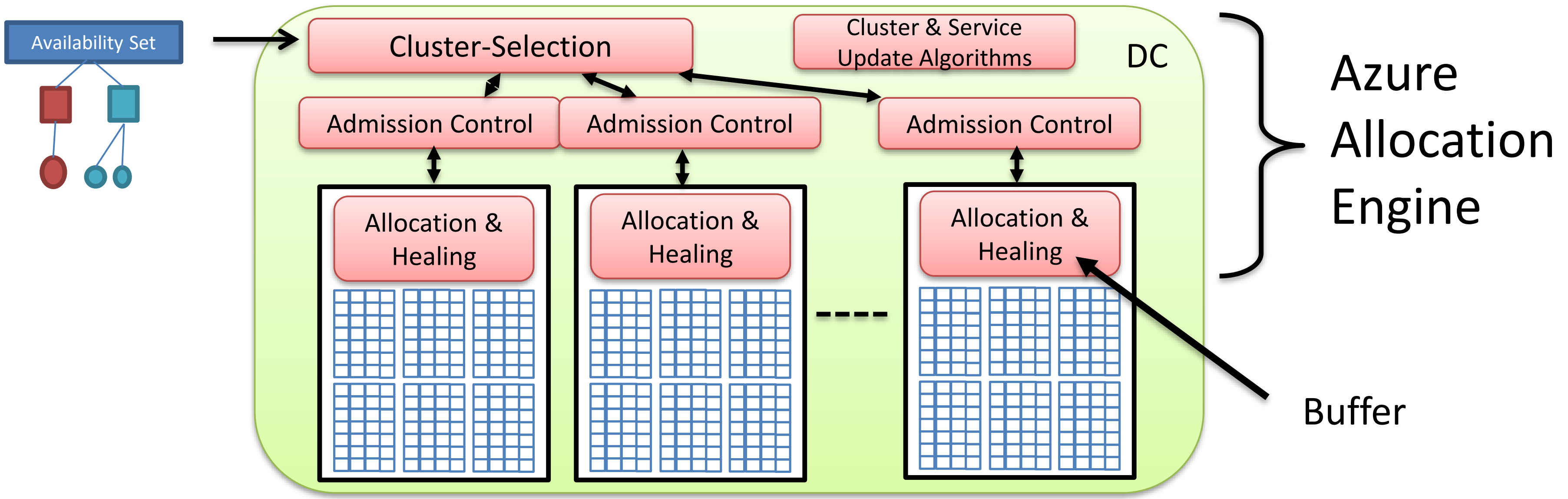
- Fabric Controller: Hardware and VM manager for a “cluster” of servers
  - Uses 5-server Paxos-type replication for high availability
  - Exposes API for deploying, deleting and updating VMs
  - Keeps track of server and VM health
- Fabric Controller can autonomously “heal” a VM
  - Detects server has failed and restarts VM on a healthy server



VM Allocator

- Composed of cluster-selection, admission-control, and intra-cluster allocation algorithms
- Multi-level:
  - First, select FC cluster
  - Then, FC cluster allocator places VMs on servers

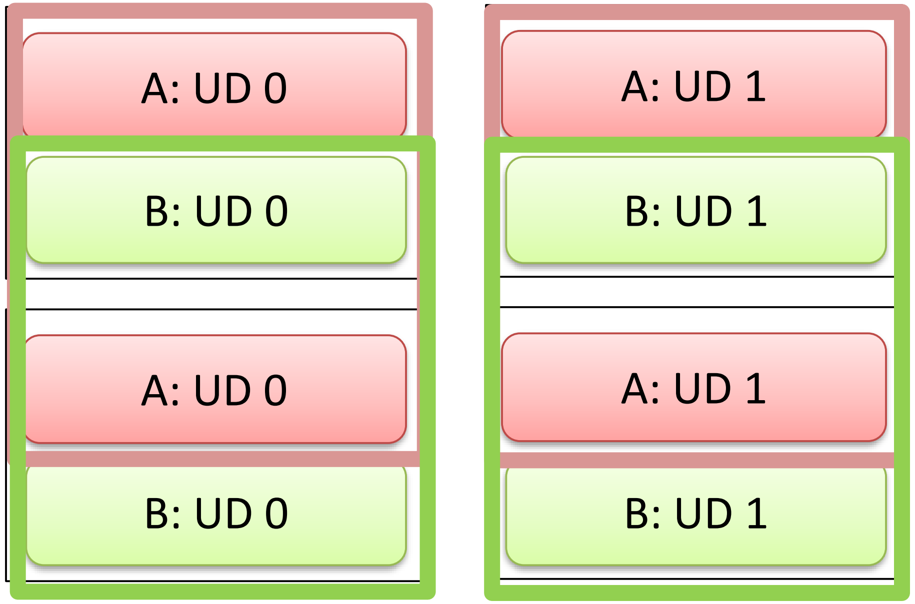




ALLOCATION BASICS

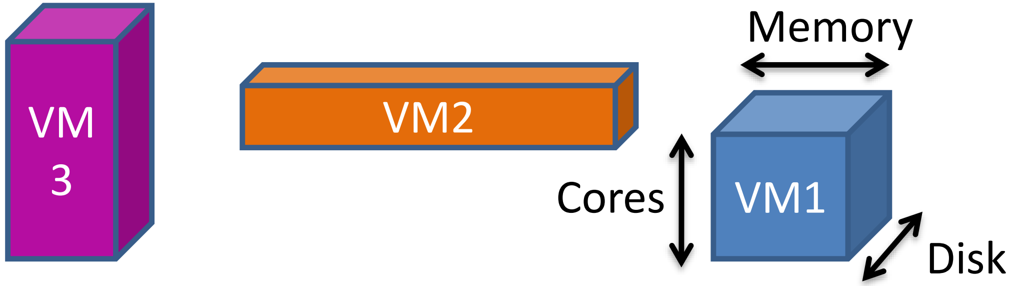
Allocation Scenarios

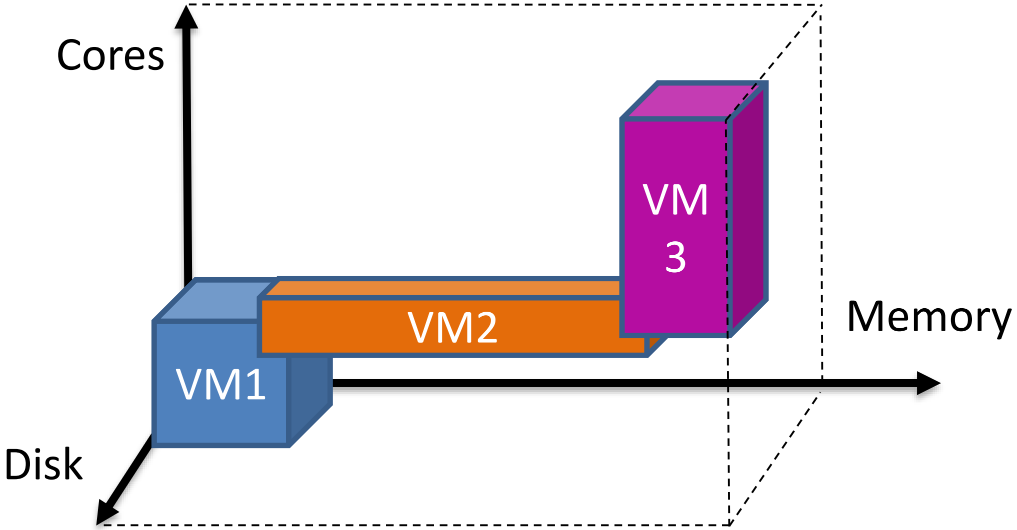
- Newly deployed services, service evictions, pre-emptions, …
- Scale-out of existing services
- Service healing after failures
- Optimizing for host OS updates



Constraints

- Placement constraints
  - Resource constraints: Sum of resources of all VMs on a node cannot exceed server resources (CPU, memory, disk, SSD, network IO,…)
    → Bin-Packing
  - Failure domain constraint: VMs of the same tenant must be spread across many failure domains
  - Co-location constraints: Certain types of VMs cannot be co-located together



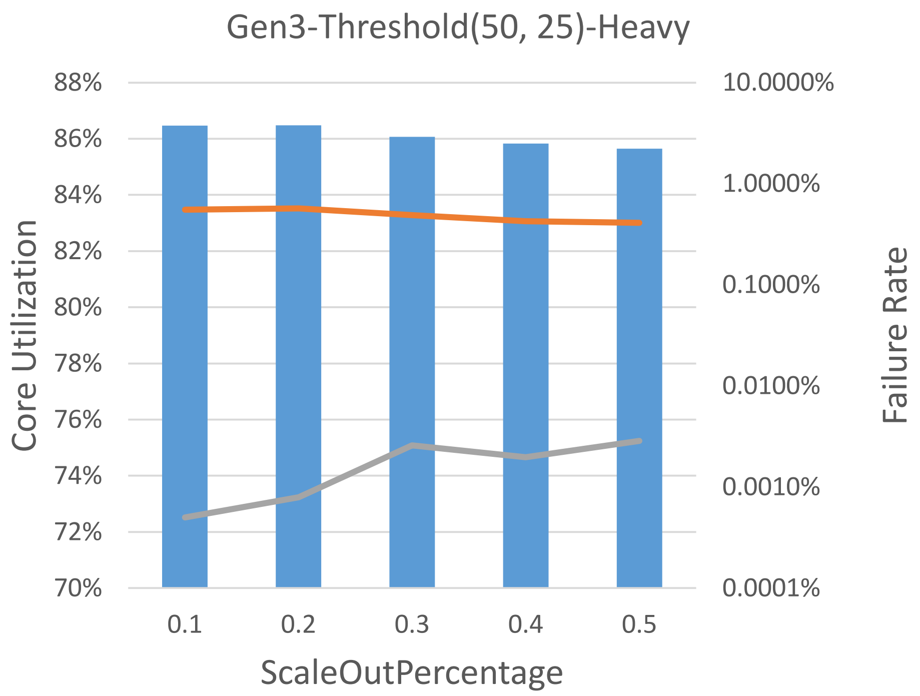


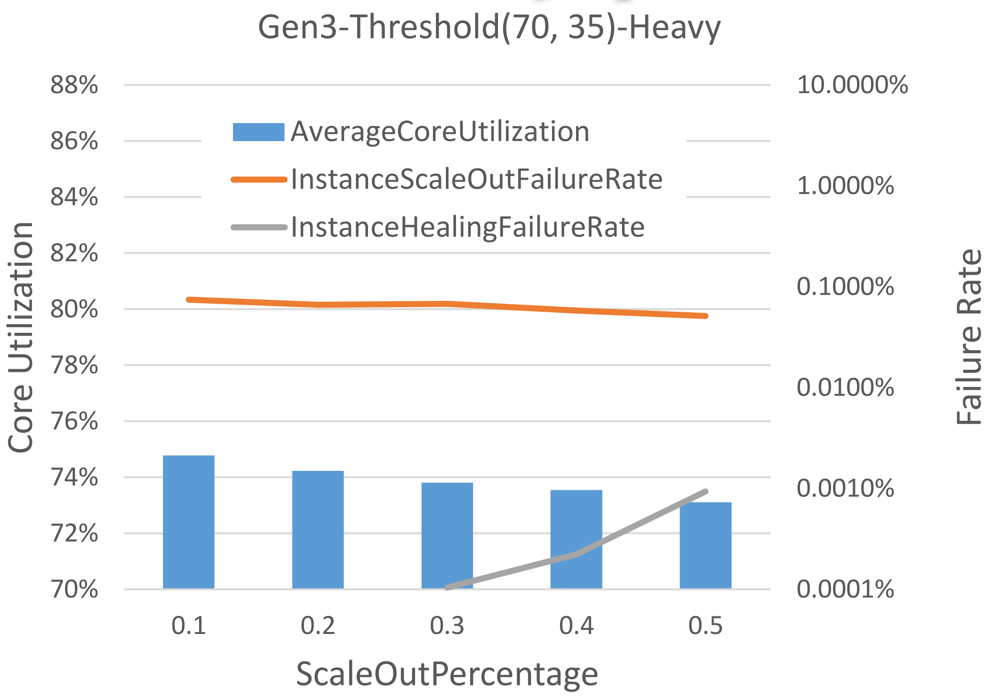
Buffers

- Unit of allocation is a cluster
  - If a node or rack fails, VMs must be healed to empty capacity within the cluster.
  - If a service wants to scale-out, extra VMs are placed within the cluster
  - → We keep sufficient empty-resource **buffers** in each cluster (healing, scale-outs, turn-space).

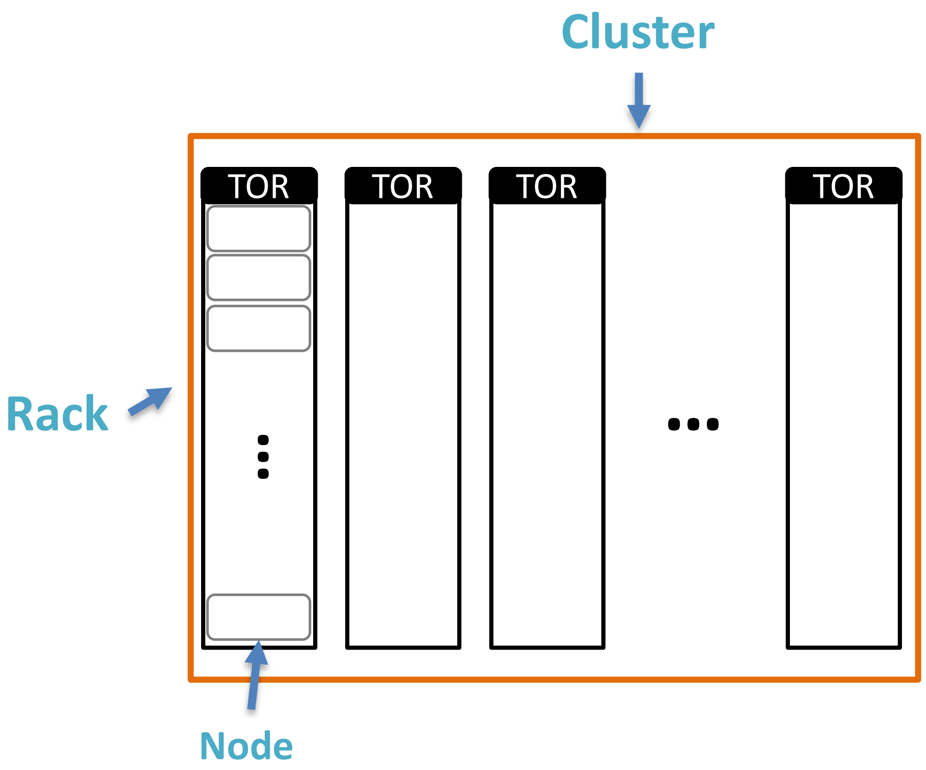
Healing threshold (#empty nodes)

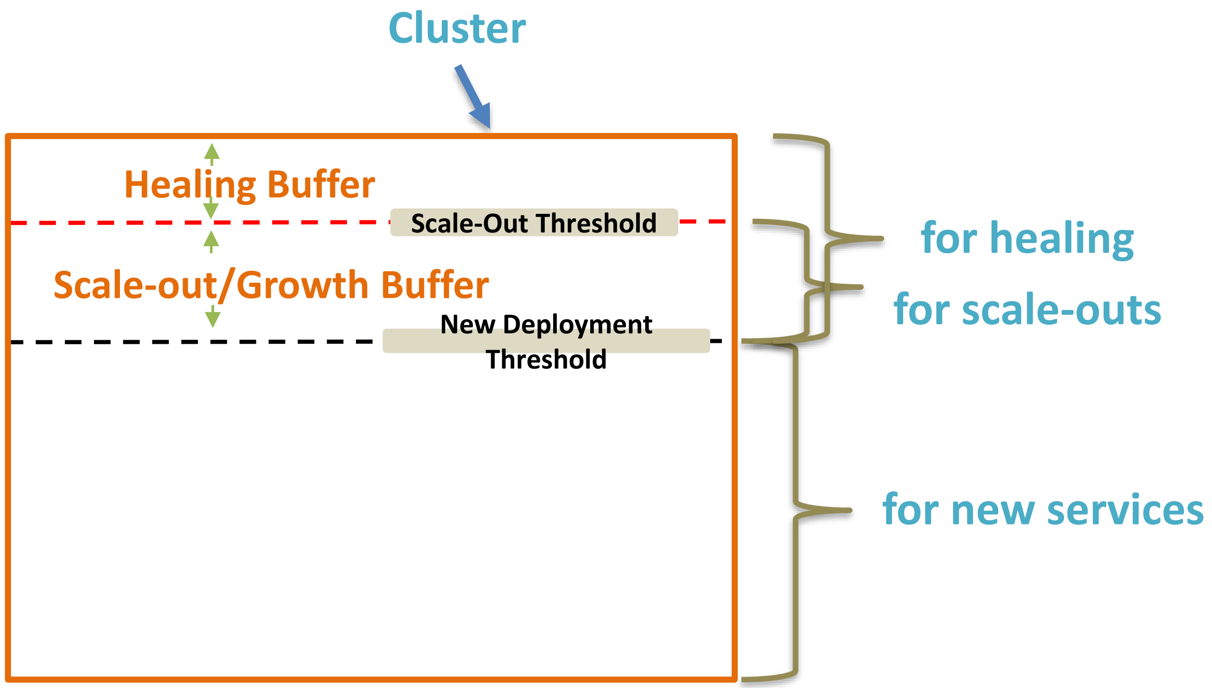
Scale-out threshold (#empty nodes)





Simplified View of Cluster Buffers



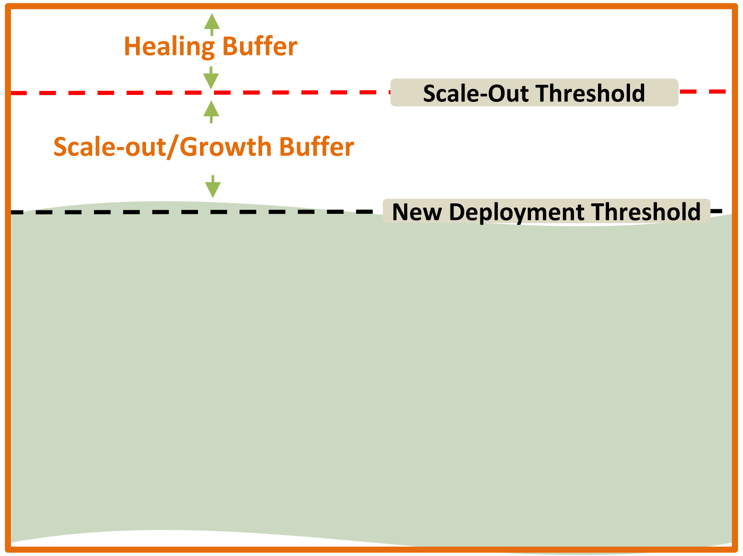


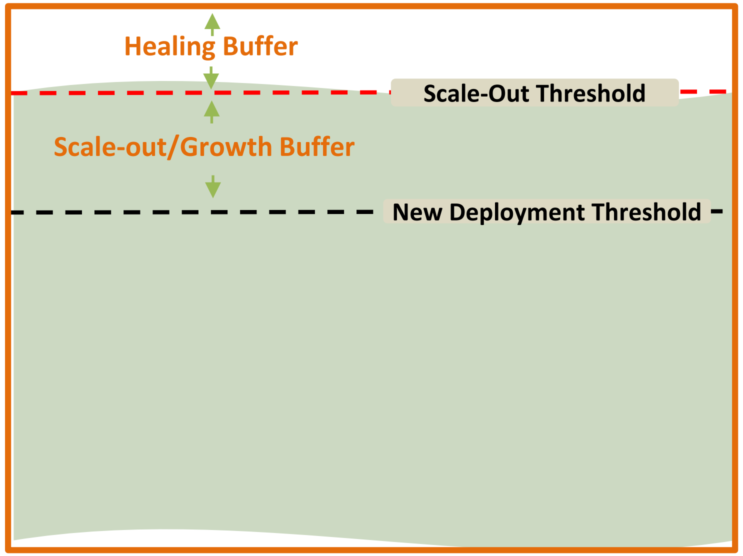
Simplified View of Cluster Buffers

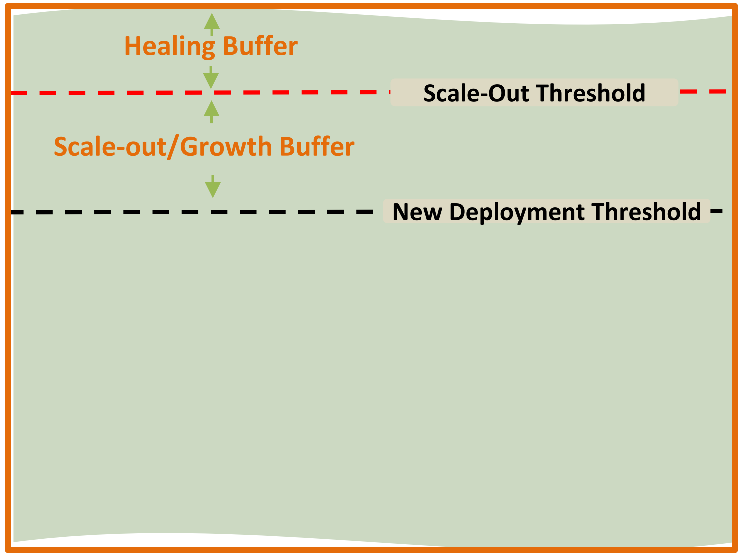
When new deployment threshold is reached, **no new deployments into this cluster**.

When scale-out threshold is reached, existing tenants cannot grow anymore. **Scale-Out Failures** occur!

When healing buffer is exhausted, node/rack failures cannot be healed. **Healing Failures** occur!

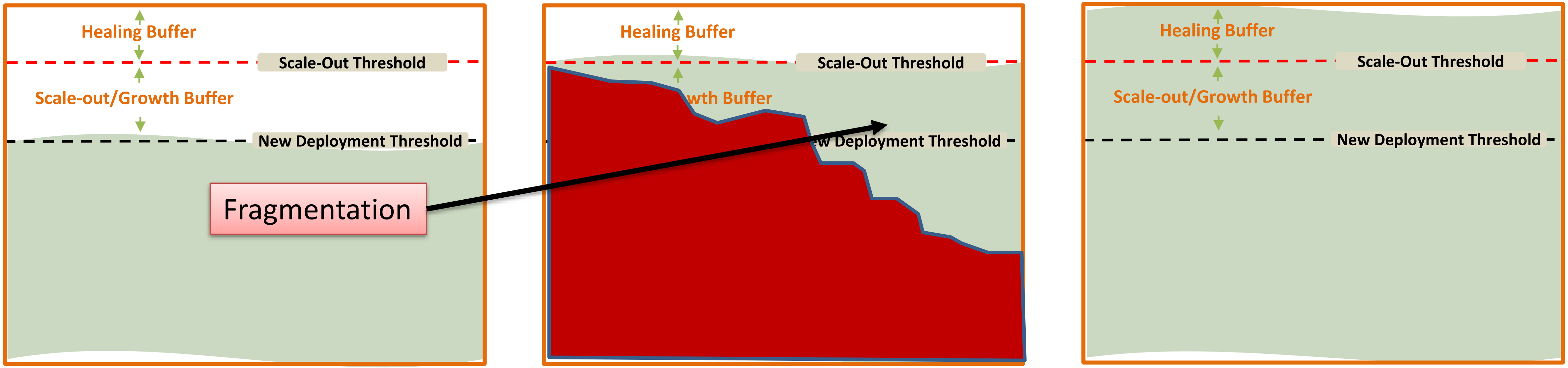




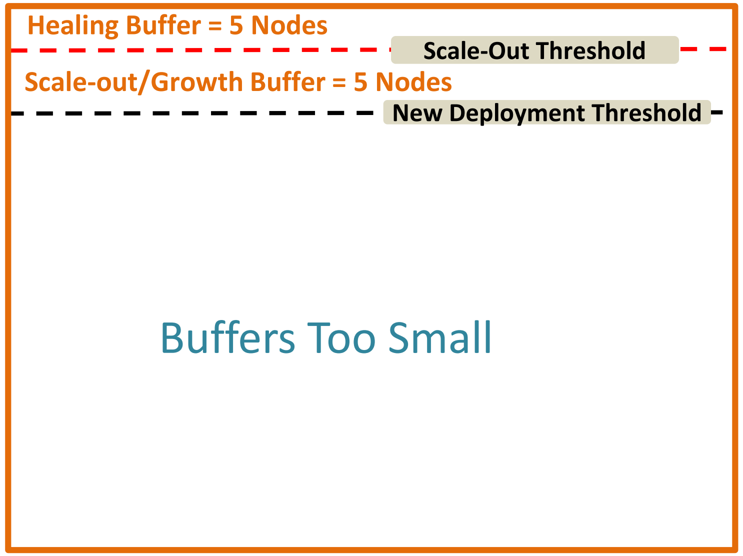


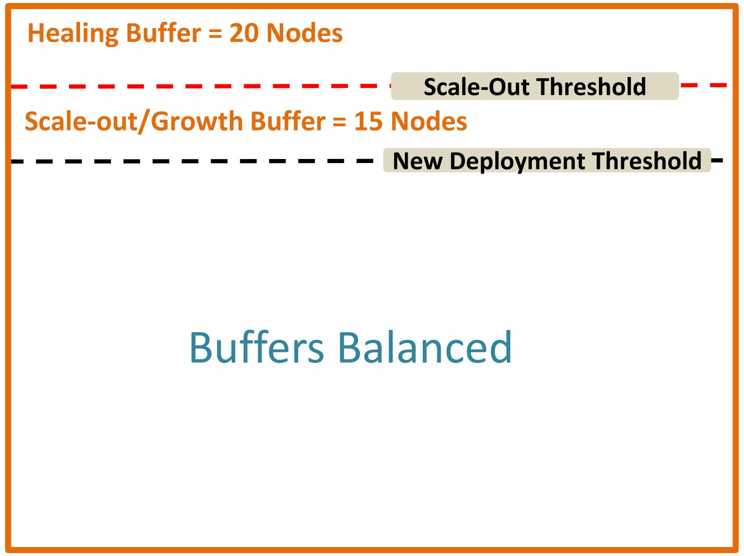
Fragmentation

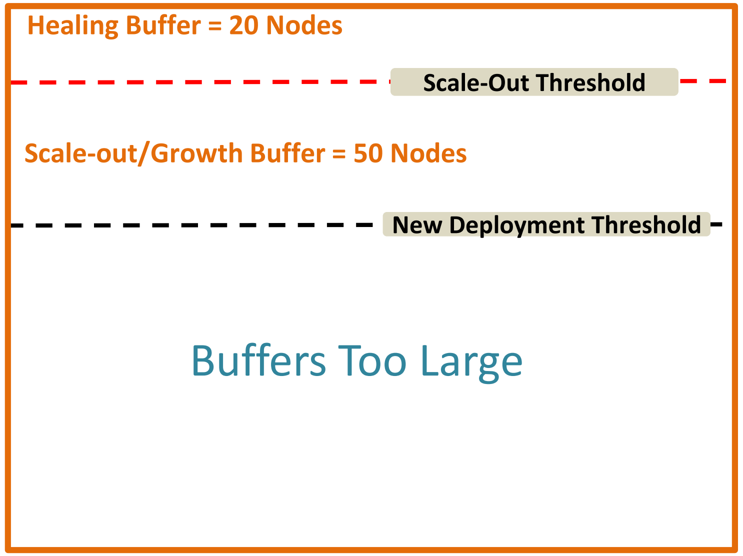
- The actual utilization in a cluster is lower than New Deployment Threshold
- **Fragmentation** → spatial fragmentation + temporal fragmentation (church)
- Amount of fragmentation depends on workload, cluster generation, policy settings, features, etc.

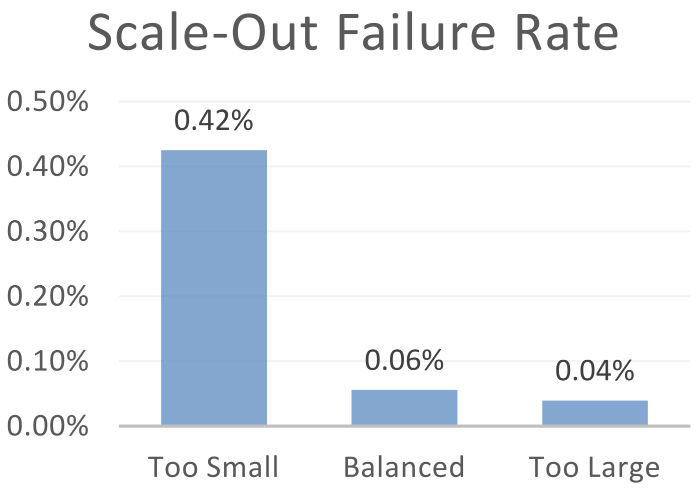


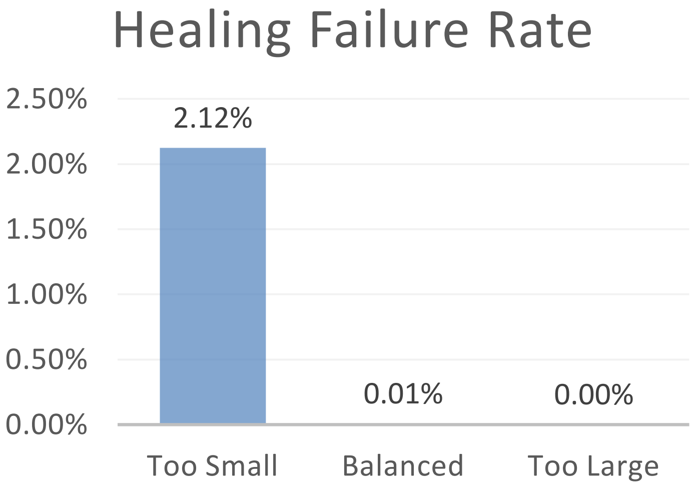
Setting the Thresholds / Limits

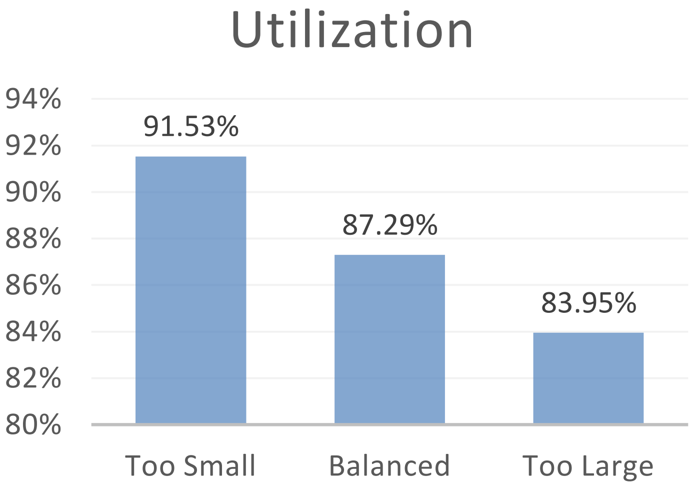




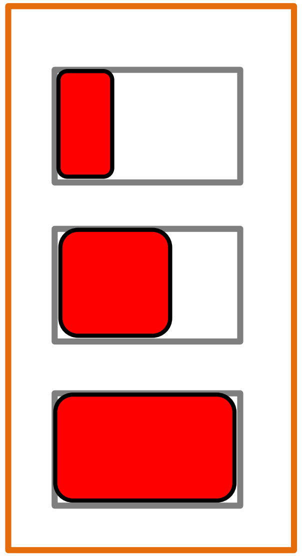








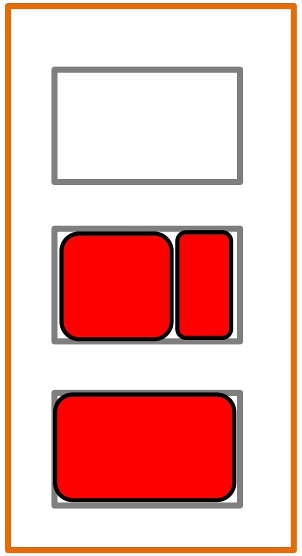
Utilization vs. Empty Nodes



Utilization: ~ 66%
Empty nodes: 0

Cannot heal 1/3 possible single-node failures

Cannot host one more full-size instance

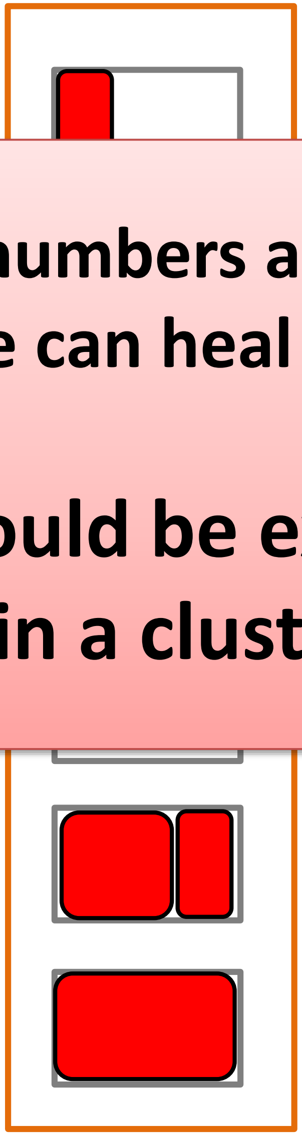


Utilization: ~ 66%
Empty nodes: 1

Can heal all possible one-node failures

Can host one more any possible instances

Utilization vs. Empty Nodes



Utilization: ~ 66%

Utilization numbers are not well correlated to whether we can heal or scale-out in a cluster.

Limits should be expressed as "#empty nodes" in a cluster – not utilization.

Empty nodes: 1

Can heal all possible one-node failures

Can host one more any possible instances

Optimizing

- The more tightly we can pack VMs,
  - … the less buffer we need.
  - … the less fragmentation we have.
  - … the easier for healing & scale-outs.

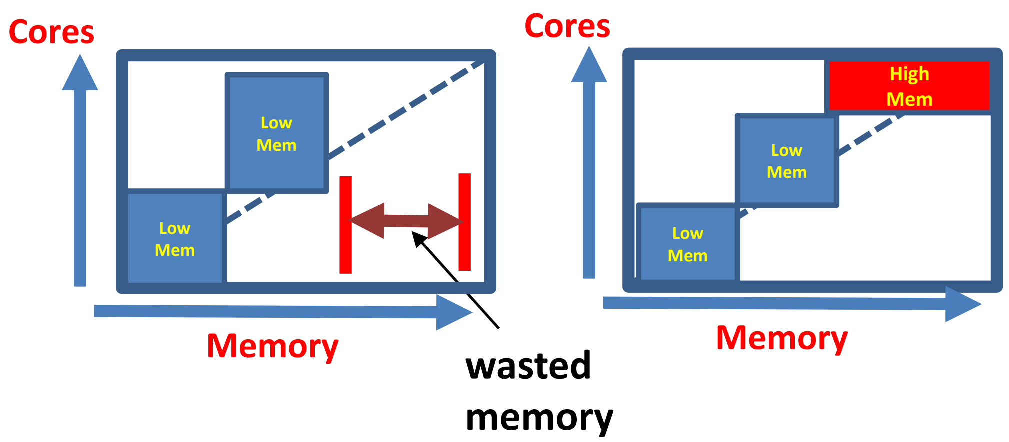
High utilization → Lower COGS
Each 1% of utilization gain results in millions of $ savings.

- Allocation decision is in critical path of deployment. We want relatively simple and very fast algorithms
- Algorithms must take decision based on little knowledge
  - Algorithms are online → need to take decision for each VM immediately
  - We do not know how long each VM will remain deployed before it leaves
- We want to avoid VM migrations as much as possible
- Algorithms should be adaptive to adjust to changes in workloads, hardware, policies, constraints, platform characteristics, etc.

Resource Utilization

- VM Packing should achieve high utilization across all resource dimensions
  1. Multi-dimensional resource packing

VM Allocator should be aware of Multiple Resource Dimensions:

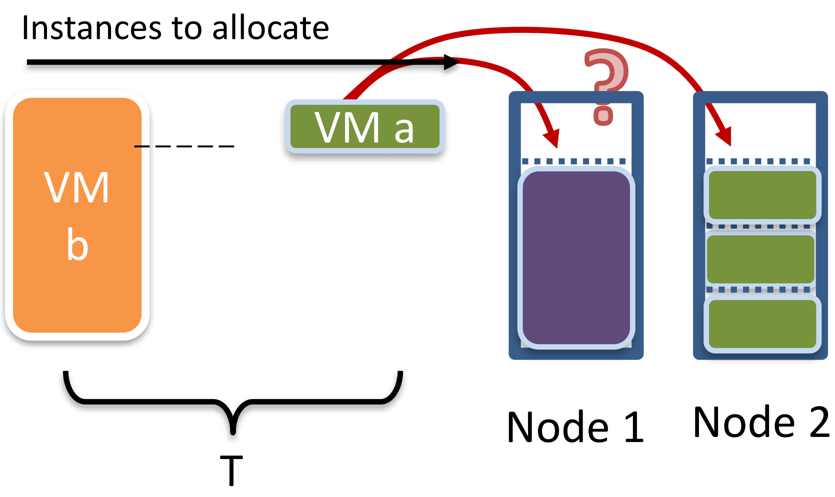


- We use **multi-dimensional best-fit**. [*Heuristics for Vector Bin Packing*, Panigrahy et al., MSR Tech Report 2011]
- Each resource dimension d is assigned a weight w_d → scarcity of the resource.
- r_d is the residual resource of a node
- Allocate the VM to the node that minimizes $\sum_d w_d * r_d$

Multi-Dimension Optimization

- VM Packing should achieve high utilization across all resource dimensions
  1. Multi-dimensional resource packing
  2. Take into account online nature of service allocation

VM Allocator should be aware of online nature of allocation



- Simple example: Assume every VM has probability of ½ of leaving until time T.
- Probability that we can deploy VM_b ?
  - If new VM is placed on Node 1:

$$\left(\frac{1}{2}\right)^2 + \left(\frac{1}{2}\right)^3 = \frac{6}{16}$$

  - If new VM is placed on Node 2:

$$\left(\frac{1}{2}\right) + \left(\frac{1}{2}\right)^4 = \frac{9}{16}$$

→ Placing new VM on Node 2 is better !

Azure Multi-Dimensional, Adaptive VM Packing

- Azure allocation algorithm achieves higher utilization across all resource dimensions
  - Multi-dimensional resource packing
  - Take into account online nature of service allocation
- Achieves near-optimal properties in terms of healing & availability

Reduces resource waste by ~40% compared to simple baseline algorithms

- Allocation engine is adaptable
  - Easy to evaluate impact of changes (new service or VM types, hardware, policy configuration, features, etc…)
- Adjusts to workload, hardware, etc…

MULTI-PRIORITY ALLOCATION

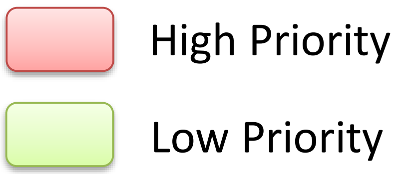
Multi-Priority Allocation

- So far, we assume all VMs are of equal priority
- What if we want to run workload of different priorities?
- For example, run low-priority VMs in unused resource slots (fill in fragmentation) or in safety buffers. Evict these VMs if higher-priority VMs arrive.

→ "**Multi-priority bin-packing problem**"

- Objective: Pack as much as possible from highest-priority. Given that, pack as much as possible from next highest-priority, etc…

Multi-Priority Allocation – Metrics

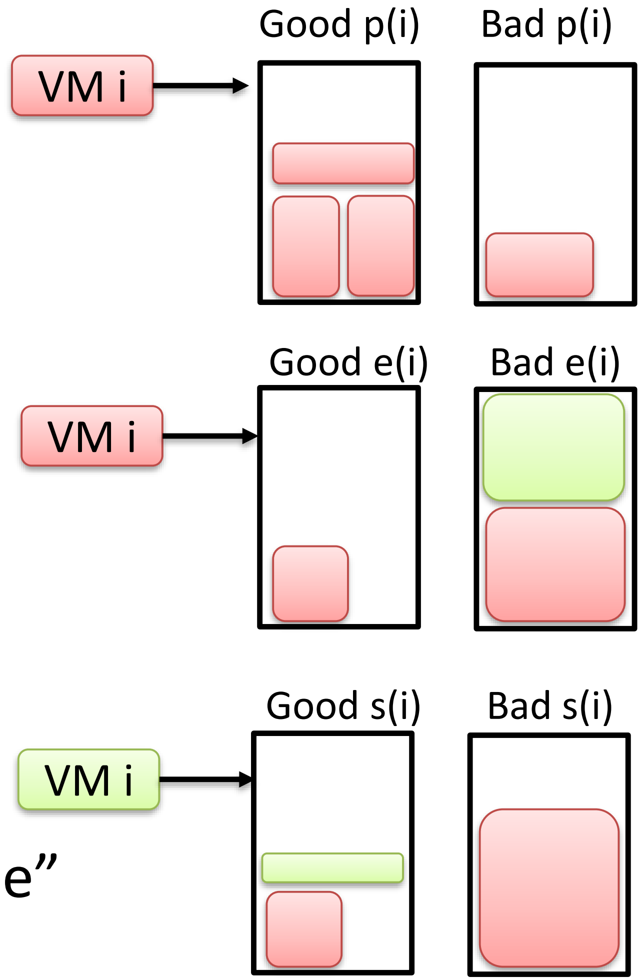


- Three metrics determine allocation decision for a new VM i

1. **Packing-Quality p(i):** Same as in single-priority case. High packing-quality means a VM “fits” well.

2. **Eviction Cost e(i):** Cost of evicting lower-priority VMs when deploying the VM to a node

3. **Safety-Score s(i):** We should deploy a low-priority VM to a node on which the VM is likely going to “survive” for a long time. Safety-score is high if the expected “survival-time” is high → less impact on future high-priority VM allocation.



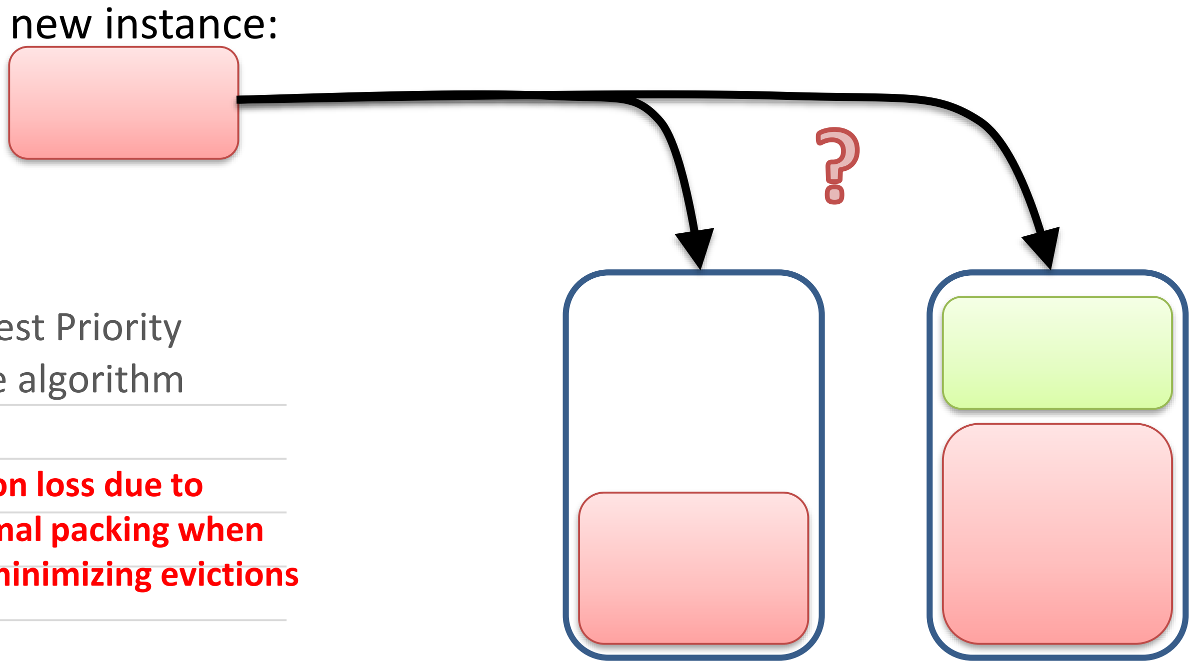
Next time a high-priority VM is allocated, it will likely be placed on Node 2

High Priority

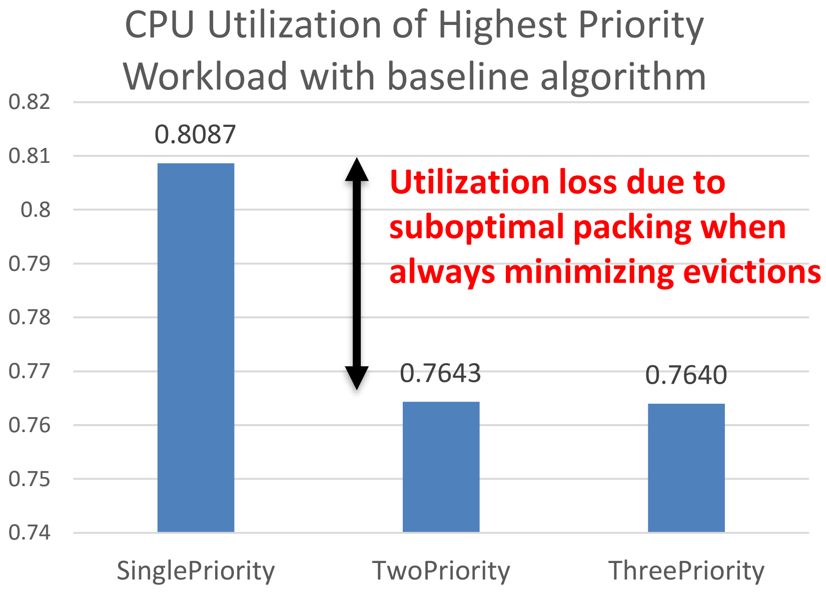
Low Priority

Multi-Priority Allocation – Trade-Offs

- The three metrics are often at odds with each other. Which node to place the new VM?



If we minimize eviction cost, packing quality decreases.



Packing-quality vs. eviction-cost trade-off

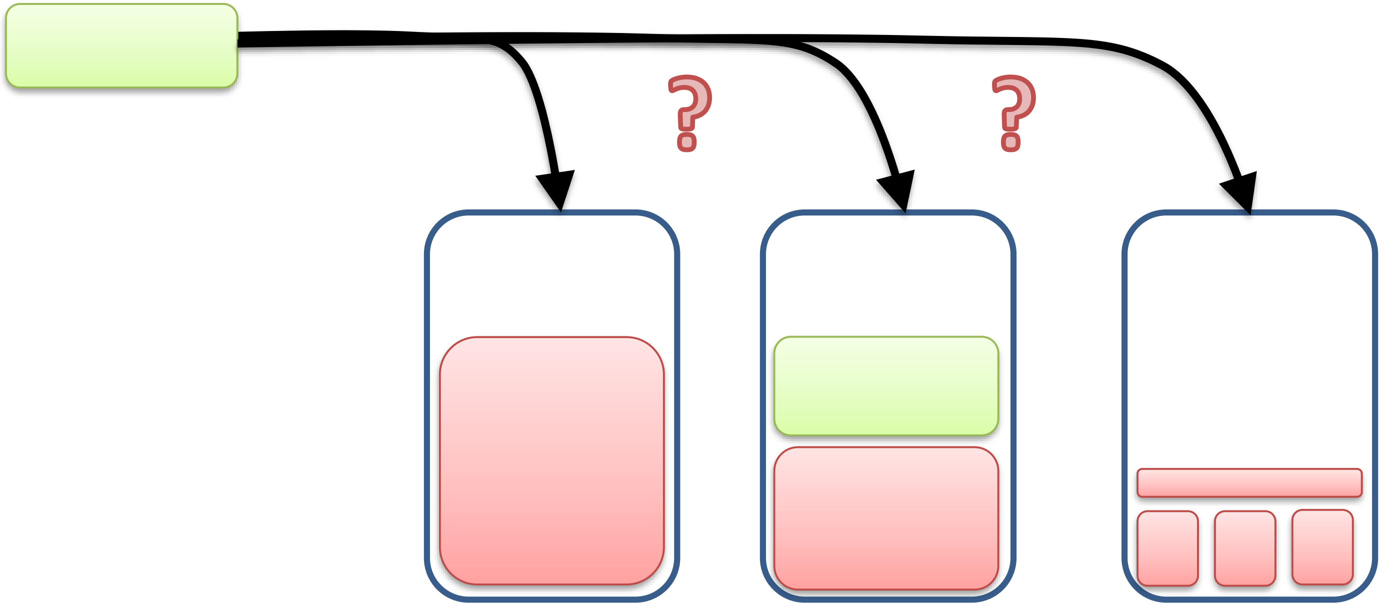
High Priority

Low Priority

Multi-Priority Allocation – Trade-Offs

- The three metrics are often at odds with each other. Which node to place the new VM?

new instance:



Packing score is optimized for Nodes 1 or 2.

Survival-time is best in Node 3.

Packing-quality vs. “survival-time” trade-off

Computing the Safety Score

1. Compute arrival rate of each VM type

We use statistical information of workloads (Data-driven)

2. For each node v, and each VM type t:

   Compute **safety-distance(v,t).**

   → Expected time until some VM will be evicted due to subsequent VM of type t, if new VM is deployed on node v.

   danger-probability(v,t) = 1 / safety-distance(v,t).

We use a clever algorithm that can compute these values very quickly.

3. For each node v:

   danger-probability(v) = Σ_t (danger-probability(v,t))

   Safety-score(v) = 1/danger-probability(v)

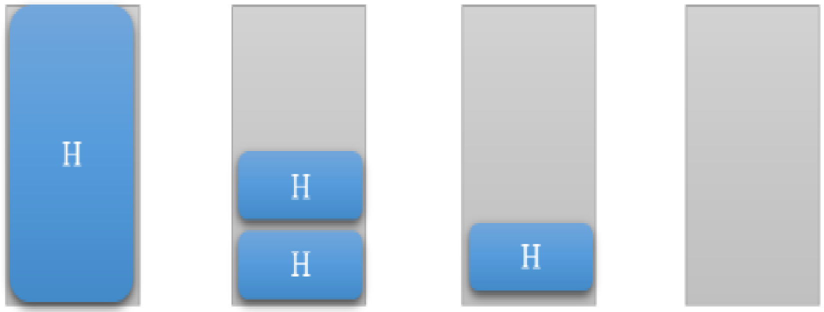
The approx. probability that some VM will be evicted within the next time interval, if the new VM is placed on Node v.

Computing the Safety Score

<u>Example 1:</u> Algorithm is effective at **capturing true safety of different nodes:**

- Assume two VM types Large and Small (Arrival intervals: Large=3, Small=1)

L Instance to Deploy



H(small)	**2**	**5**	**9**
H(large)	-	-	3

safety-distance(v,t)

danger-probability(v,t)

H(small)	**1/2**	**1/5**	**1/9**
H(large)	0	0	1/3
Sum	1/2	1/5	4/9

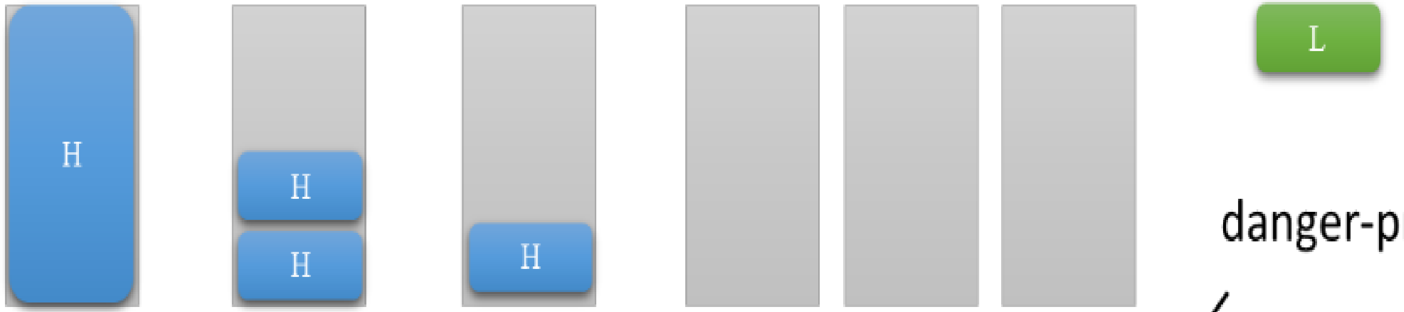
danger-probability(v)

Safest

Computing the Safety Score

Example 2: Algorithm is effective at **automatically adjusting to cluster state:**

- Same example as before, except we add two additional empty nodes



Instance to Deploy

safety-distance(v,t)

H(small)	**2**	**5**	**17**
H(large)	-	-	9

The existence of additional empty nodes has made this node much more safe!

danger-probability(v,t)

danger-probability(v)

H(small)	**1/2**	**1/5**	**1/17**
H(large)	0	0	1/9
Sum	1/2	1/5	26/153

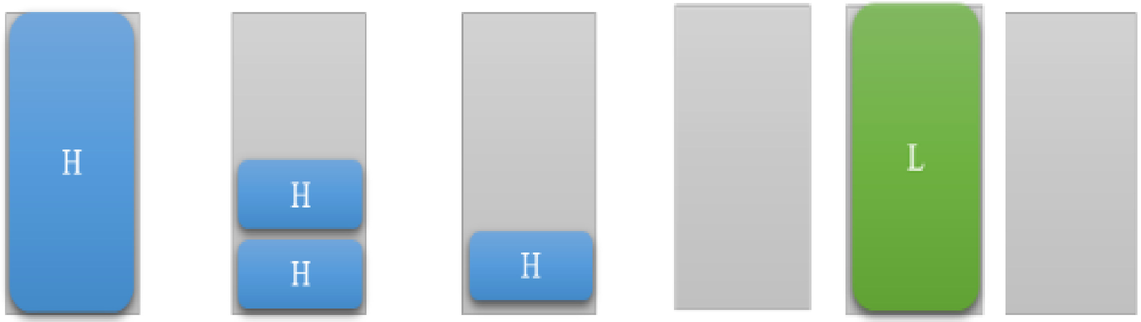
Safest node has changed!

Safest

Computing the Safety Score

Example 3: Algorithm is **aware of existing low-priority VMs:**

- Same example as before, except one empty node now contains a low-priority VM



safety-distance(v,t)

H(small)	**2**	**5**	**13**
H(large)	-	-	6

The existence of low-priority VM in this node has made the empty nodes less safe! Smaller "empty-node buffer"

danger-probability(v,t)

H(small)	**1/2**	**1/5**	**1/13**
H(large)	0	0	1/6
Sum	1/2	1/5	19/48

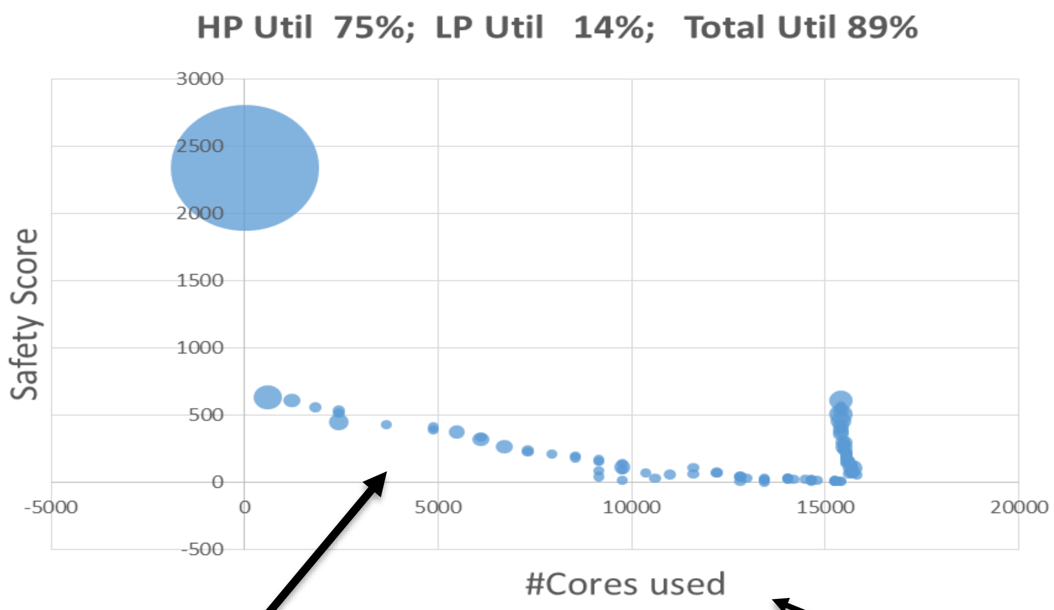
danger-probability(v)

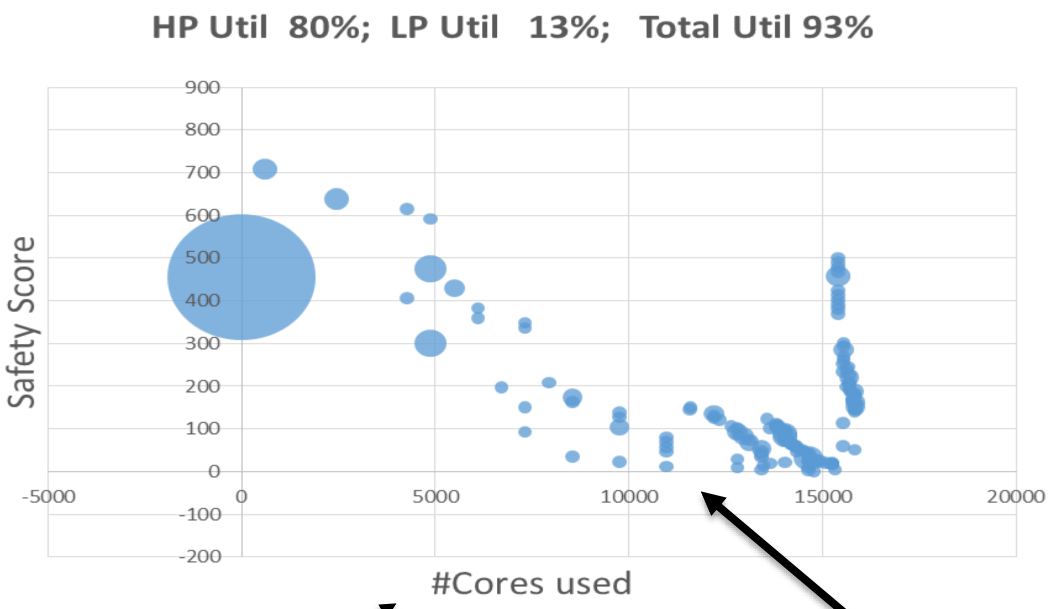
Safest node has changed!

Safest

Adaptivity of Safety Scores

- Safety scores automatically adapts to changes in Azure clusters (due to workload changes, policy changes, hardware changes, etc…)





Low-utilization cluster. Many empty nodes

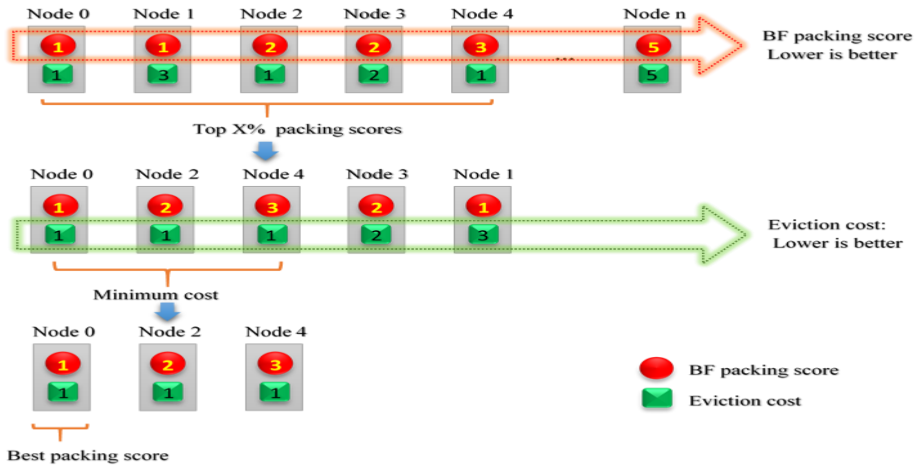
X-axis: All possible node-states, ordered according to #cores used in this state.

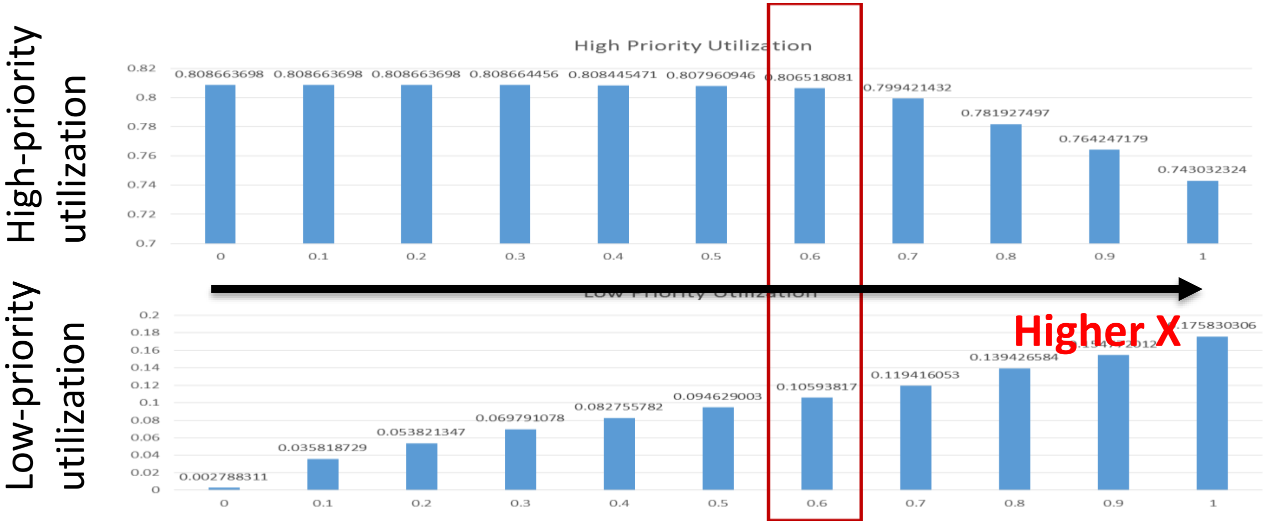
High-utilization cluster. Few empty nodes

Balancing the Metrics

- <u>Example:</u> Balancing Packing-Awareness and Eviction Cost

1. Order nodes according to packing scores
2. Pick top X% of nodes
3. From among these, pick nodes with least eviction cost



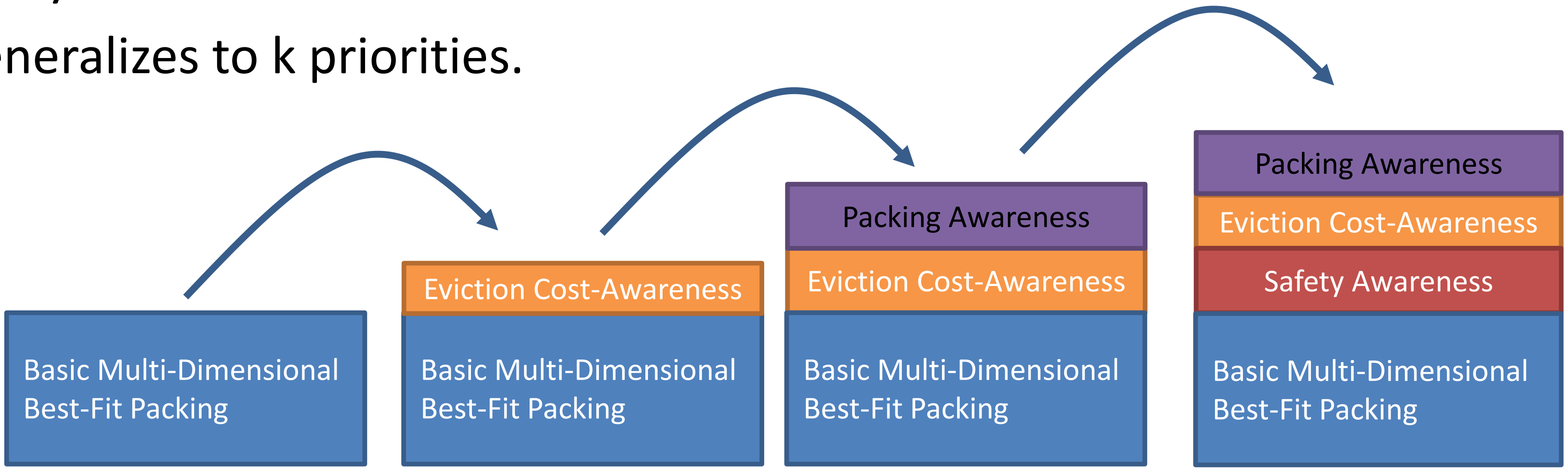


Choice of parameter X is based on workload and hardware characteristics. (data-driven)

Putting it all together

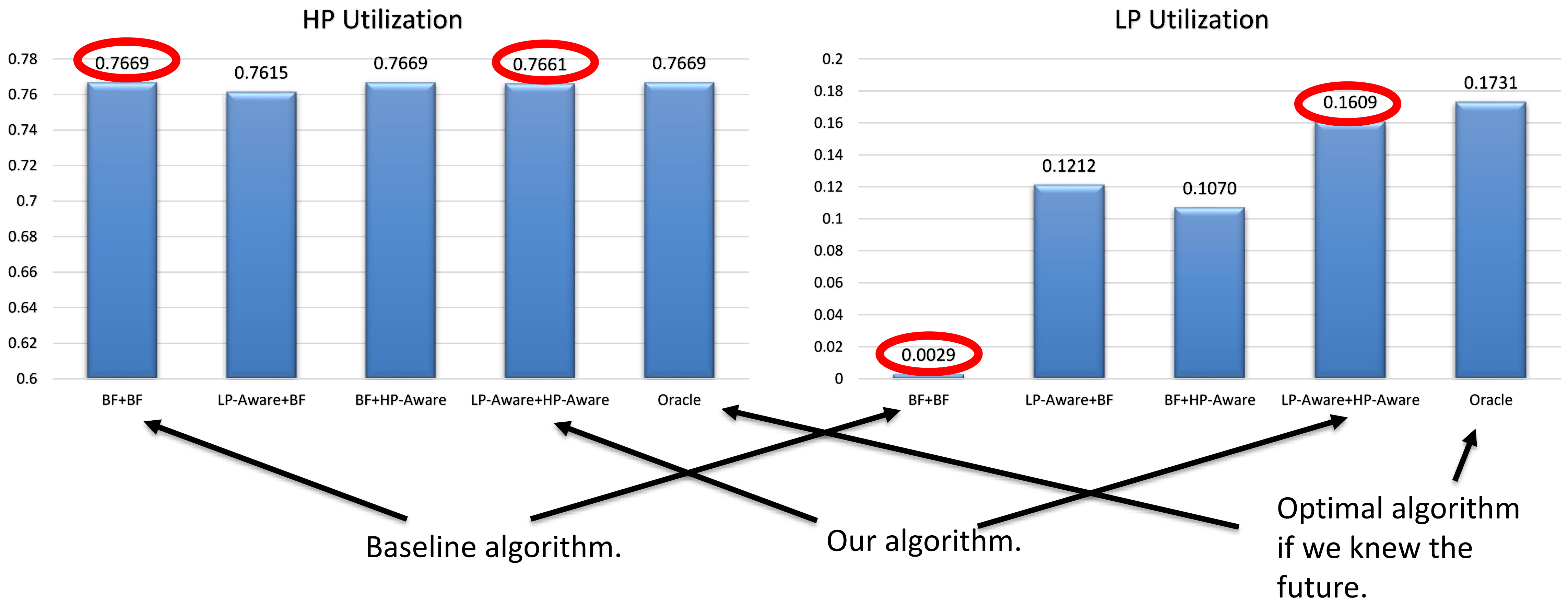
- Highly-efficient, state-of-art **Multi-Priority Resource Allocation** in Azure
- For each allocation and eviction, we have to balance
  - Cost of evicted instances → Eviction-Cost
  - Packing Quality → Packing Score
  - Survival time of newly deployed instances → Safety-Score
- Algorithm is priority-rule based.
- Our algorithm generalizes to k priorities.

We are not aware of any similar multi-priority allocation work in academia



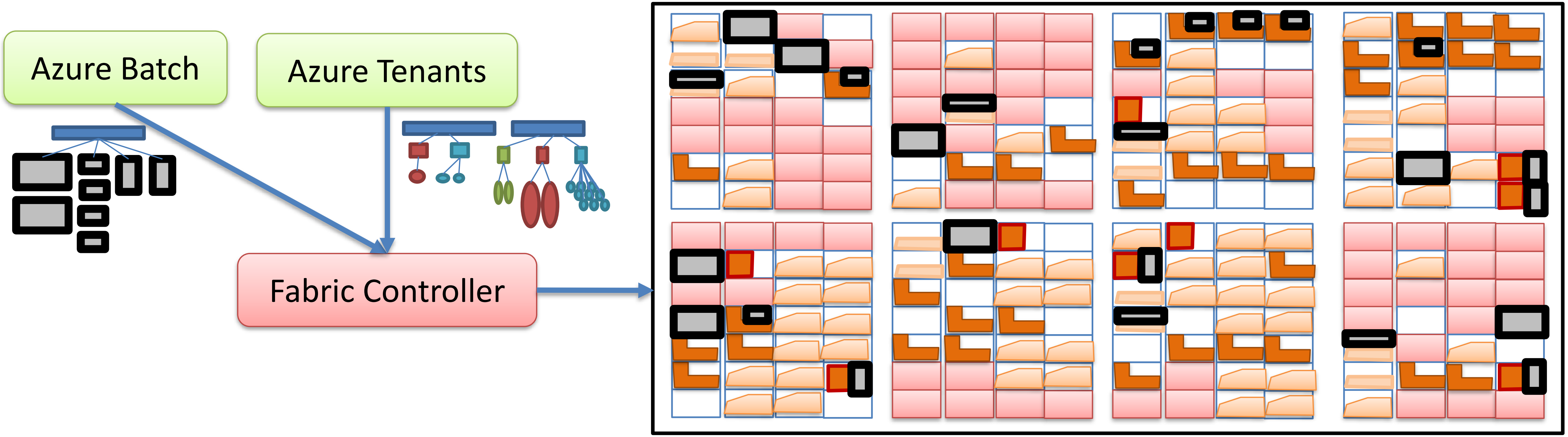
Multi-Priority – Allocation Engine

- Multi-priority allocation algorithm significantly improves low-priority utilization, without decreasing high-priority utilization.



Towards 100% Utilization – Azure Batch

- Service built on top of **Multi-Priority Resource Allocation** Fabric
- Idea: Run batch jobs in free resource slots of Azure Compute clusters
  → Azure Batch manages low-priority Azure jobs
- Initial results: Batch jobs can be completed quickly in spite of evictions.



Towards 100% Utilization – Azure Batch

- **Transient Resource (TR) Computing**: New computing paradigm.
  - Schedule tasks on transient resources
  - Good predictors for resource availability and task durations are crucial.
  - Designing new state-of-art TR-scheduling algorithms

- Challenge: **Available resource slots have vastly different "survival-times"**
  - Different VM sizes have different survival-times and deployment probabilities
  - Different Azure Batch Jobs have different processing times

- Building **Azure Fabric Resource Prediction Engine** for higher-level service

With Azure Multi-Priority Allocation Engines + Azure Batch, **we can run clusters at near 100% utilization…**

Hyper-Scale Creates Lots of Hard and Ground-Breaking Problems!

Microsoft

marcusfo@microsoft.com

http://www.fontoura.com