Improving Data Access Performance of Applications in IT Infrastructure

Hao Wen
Advisor: David Du

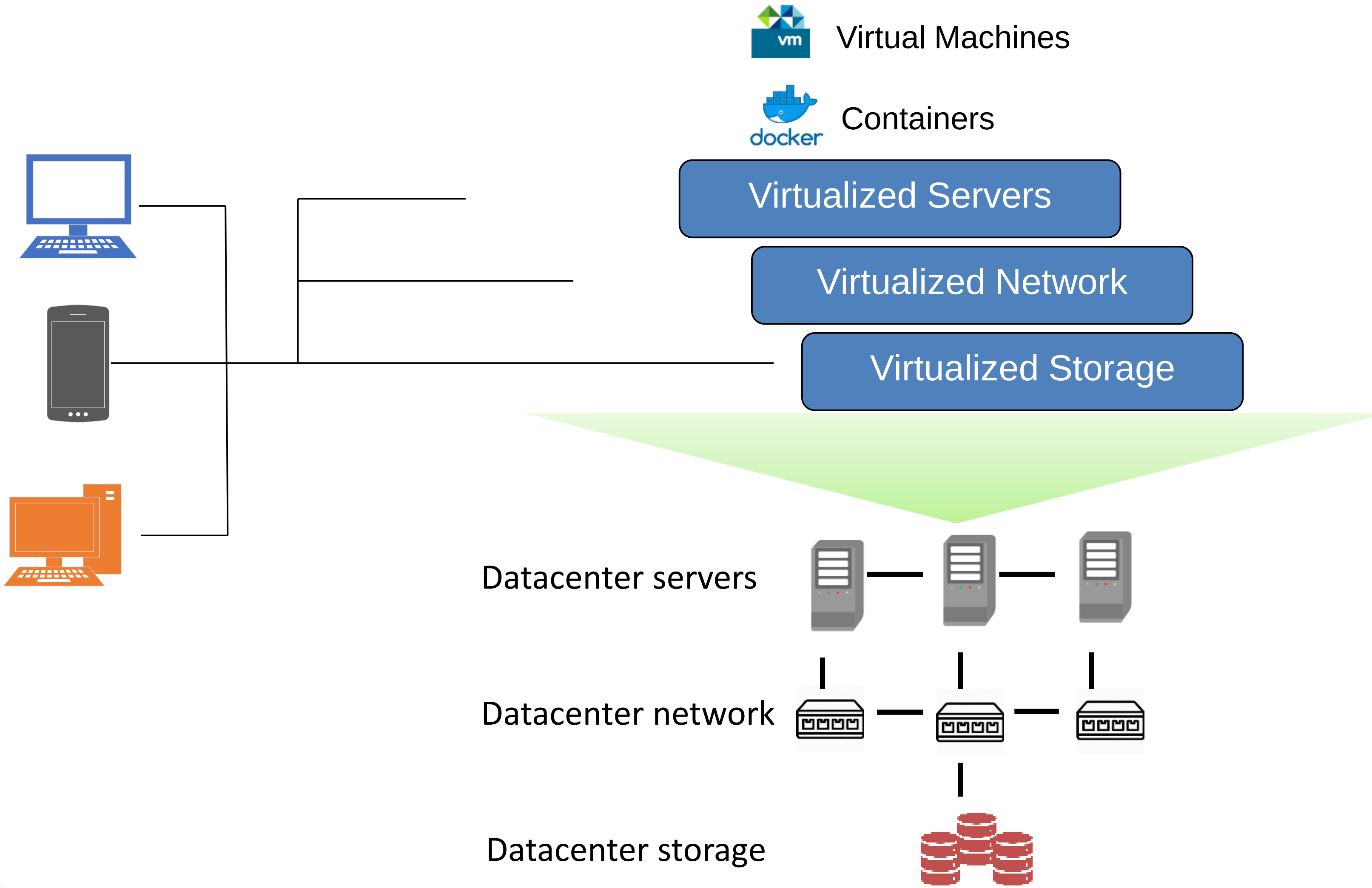
April 24th, 2019

Department of Computer Science and Engineering, University of Minnesota, USA

Center for Research in Intelligent Storage

University of Minnesota

Virtualized and Cloud Infrastructure



Typical Examples

amazon web services

AWS EC2 Instance + VPC + Amazon S3

Google Cloud

Compute Engine + Cloud Networking + Google Cloud Storage

vmware

vmware vSphere + vmware NSX + VMware Virtual SAN

Hyper-converge

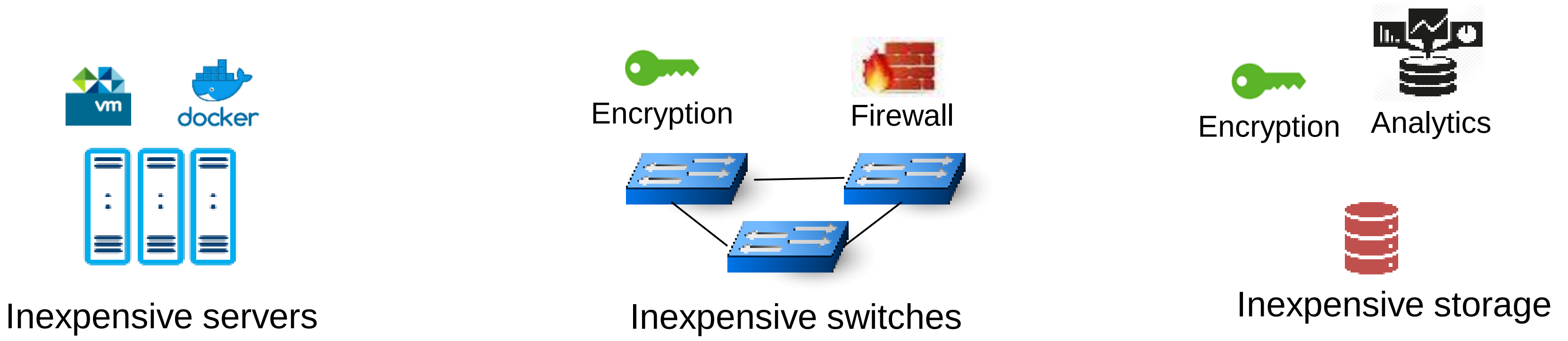
What does virtualization bring?

Mobility (Move applications)

Flexibility (Deploy & Scale applications)

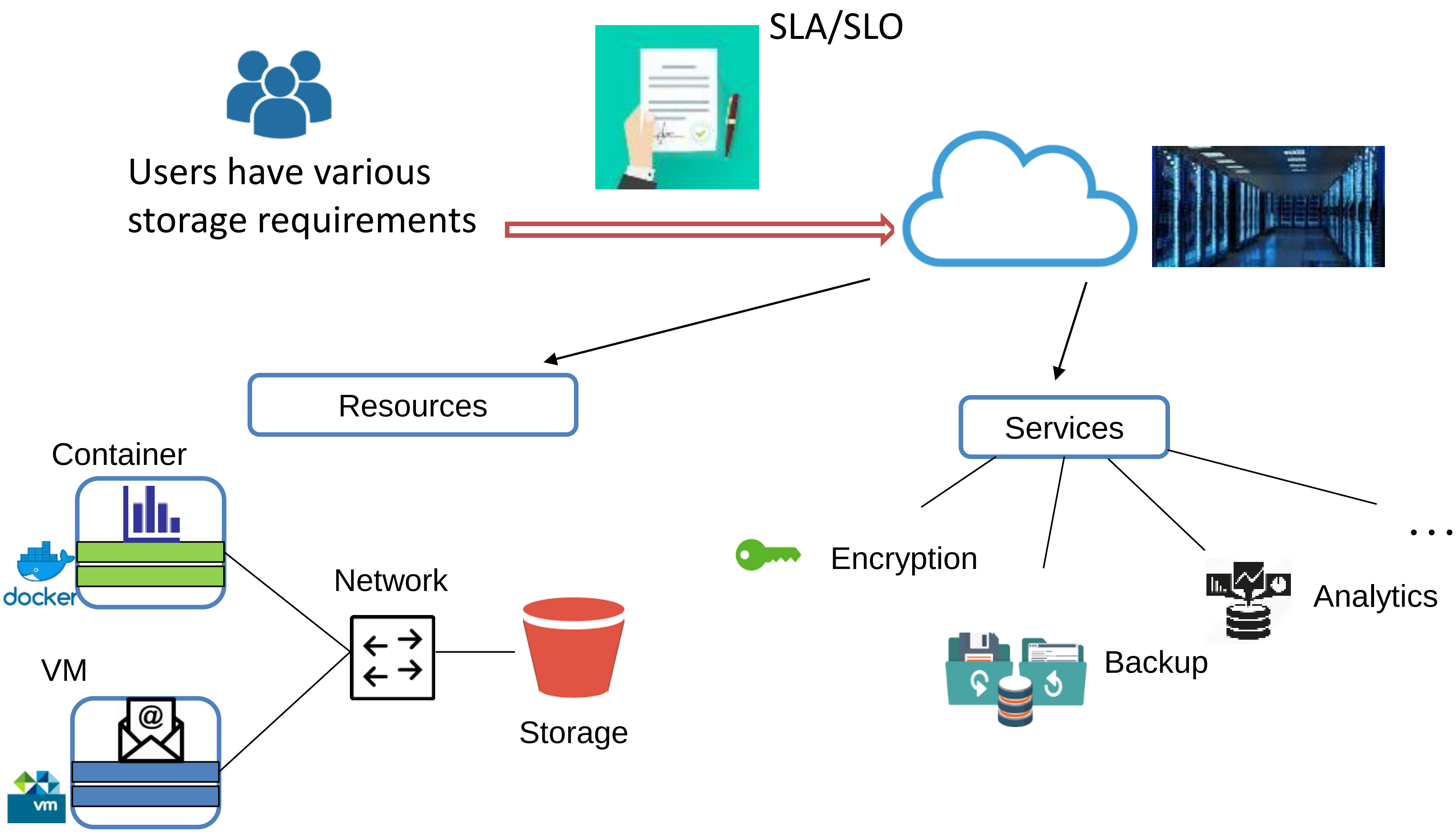
The abilities to customize services and control all resources

- Inexpensive hardware
- Customize services by installing applications in VMs or containers
- Have controls in compute, network and storage

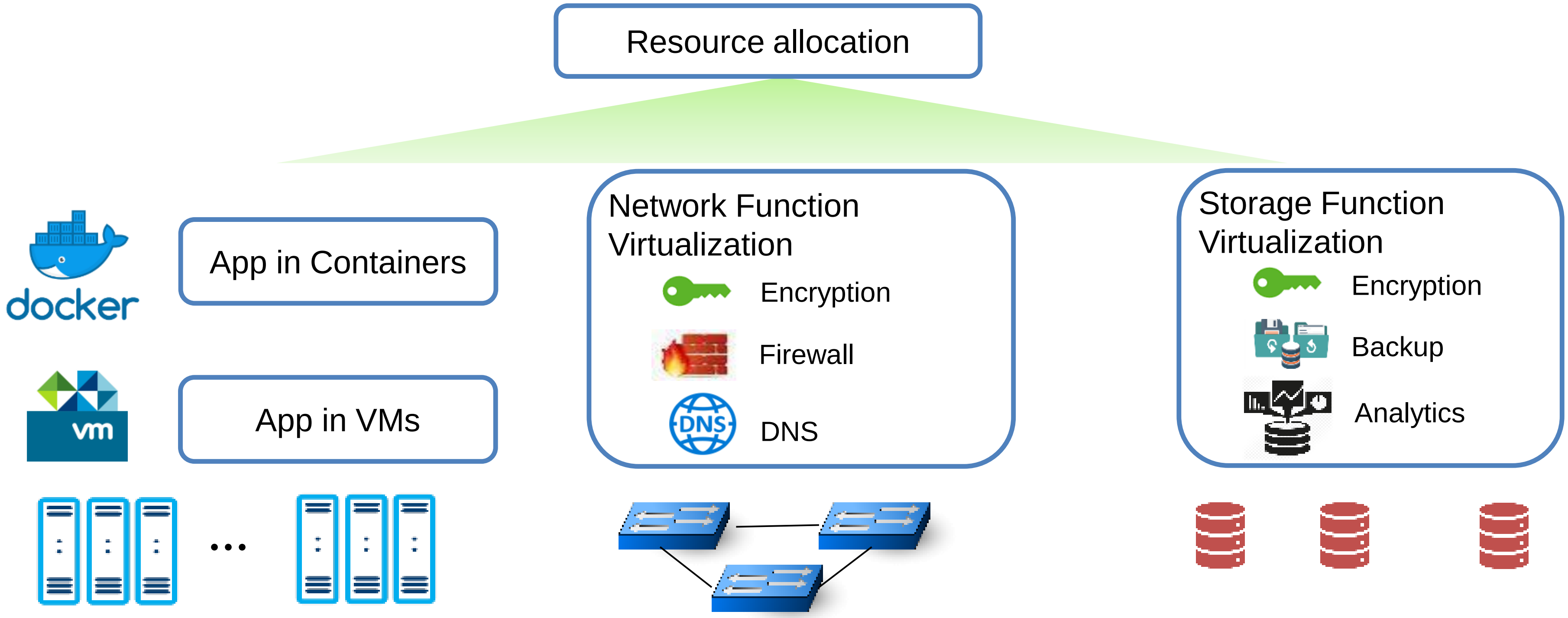


Hyper-converged Infrastructure

Importance of Data Access Performance in Hyper-converged Infrastructure



Improve Data Access Performance in Emerging Hyper-converged Infrastructure

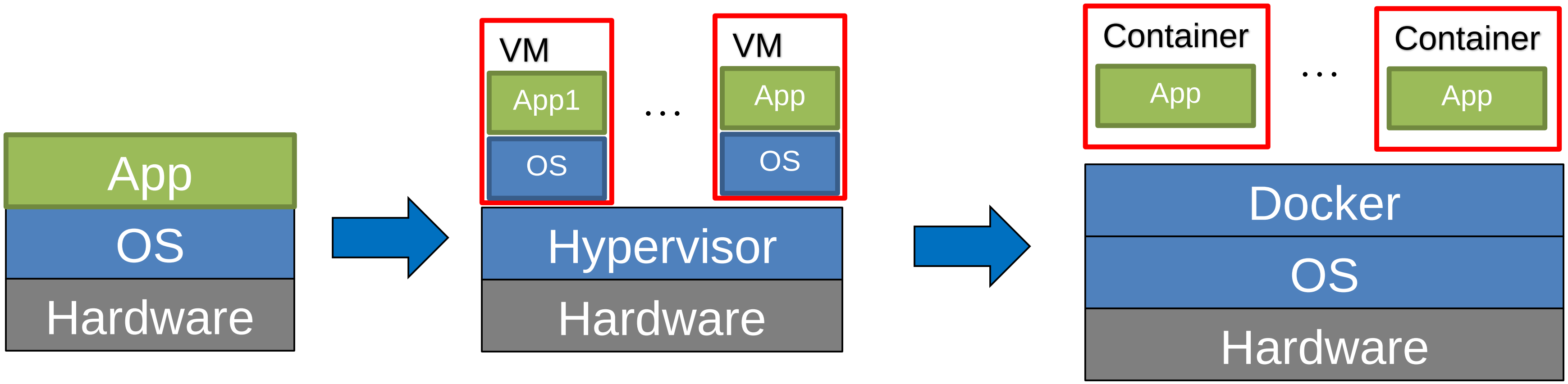


Ability to control all resources ⟹ Systematic control over client, network, storage for app in networked storage

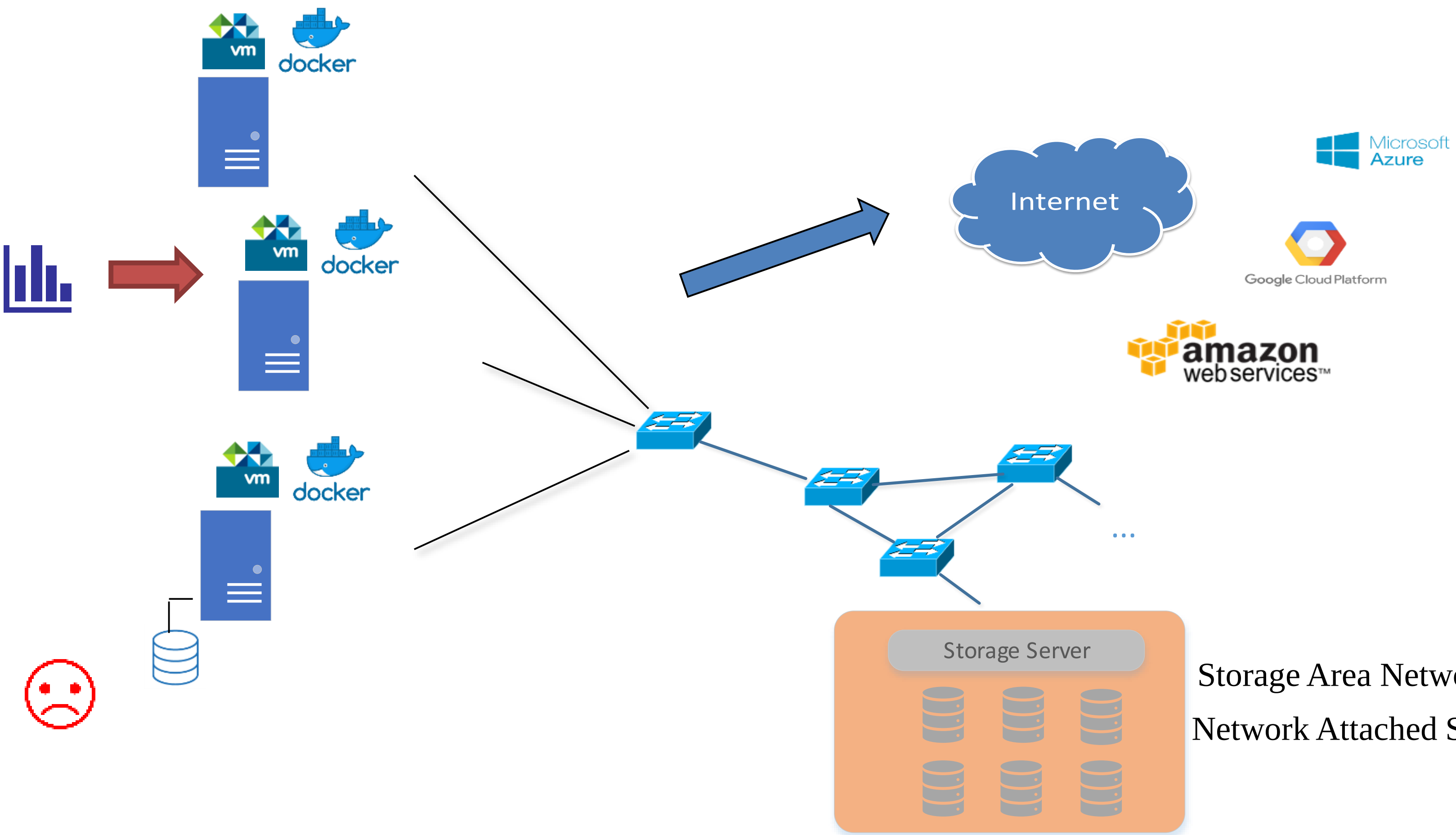
What are Virtual Machines and Containers?

Emulation of a computer system

Unit of software that packages up code and all its dependencies into a single object



What is Networked Storage

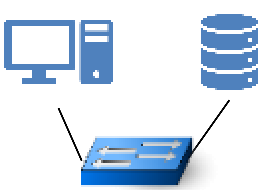


Storage Area Network (SAN) or
Network Attached Storage (NAS)

My Research

- Identify and meet storage requirements in VM
  - Study Virtual Desktop Infrastructure (VDI), to identify and meet storage requirements in VMs. [ICPP2015, IEEE TCC]

- Enhance storage support in container
  - Propose a system that can support applications with various storage requirements deployed in the Kubernetes environment based on Docker containers. [Under submission]



- Improve I/O latency in the networked storage environment
  - Propose a system that coordinates different components along the I/O path to ensure latency SLO for applications in networked storage environment. [MASCOTS 2018]

Meeting Storage Requirements of VDI Applications in the Virtual Machine Environment

Center for Research in Intelligent Storage

University of Minnesota

Virtual Desktop Infrastructure

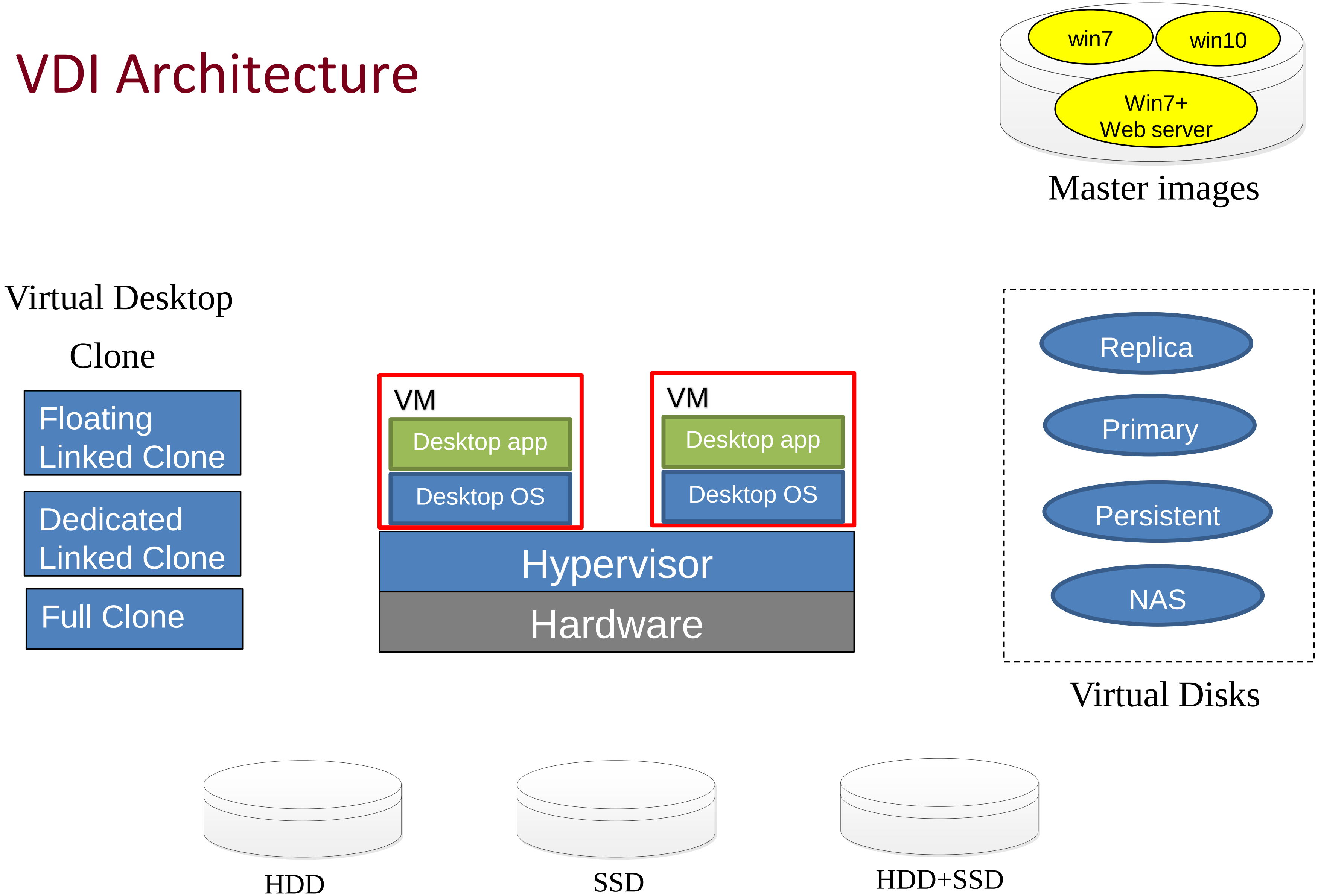
With VMs:

- People run application in VMs in a data center.
- People access applications and data from anywhere at anytime.

- VDI is such a typical and also prevalent VM application
- Virtual Desktop Infrastructure (VDI) [1,2,3] manages Desktops in data center, presents a desktop to users like running locally.

[1] Citrix virtual desktop handbook 7.x. https://support.citrix.com/article/CTX221865.
[2] Desktop virtualisation. https://www.microsoft.com/en-in/cloud-platform/desktop-virtualization.
[3] Horizon 7. https://www.vmware.com/products/horizon.html.

VDI Architecture



Center for Research in Intelligent Storage

University of Minnesota

Problem and Challenges

Users deploy a VDI system in a data center. How can the administrator describe the storage requirements of VDI, and identify what capability a storage appliance needs to satisfy the requirements.

Challenges:

- Different types of virtual desktop accesses different virtual disks.
- The I/O patterns of a virtual desktop are different at different stages during the life cycle.
- On different virtual disks, the I/O patterns of a virtual desktop are different.
- Users may use homogeneous or heterogeneous combination of virtual desktops.

Related Work

- Current VDI sizing work is unable to give a description of accurate storage requirements of virtual desktops.
  - Use rules of thumb to guide storage provisioning[4].
  - Test the performance of their storage array under a given fixed number of VDI instances[5].
  - Using storage capabilities to refer to the VM requirements[6].

- Most of the studies trying to provide methods of meeting VM requirements overlook the characteristics of the VM storage requirements.

- In practice, people always over provision storage resources.

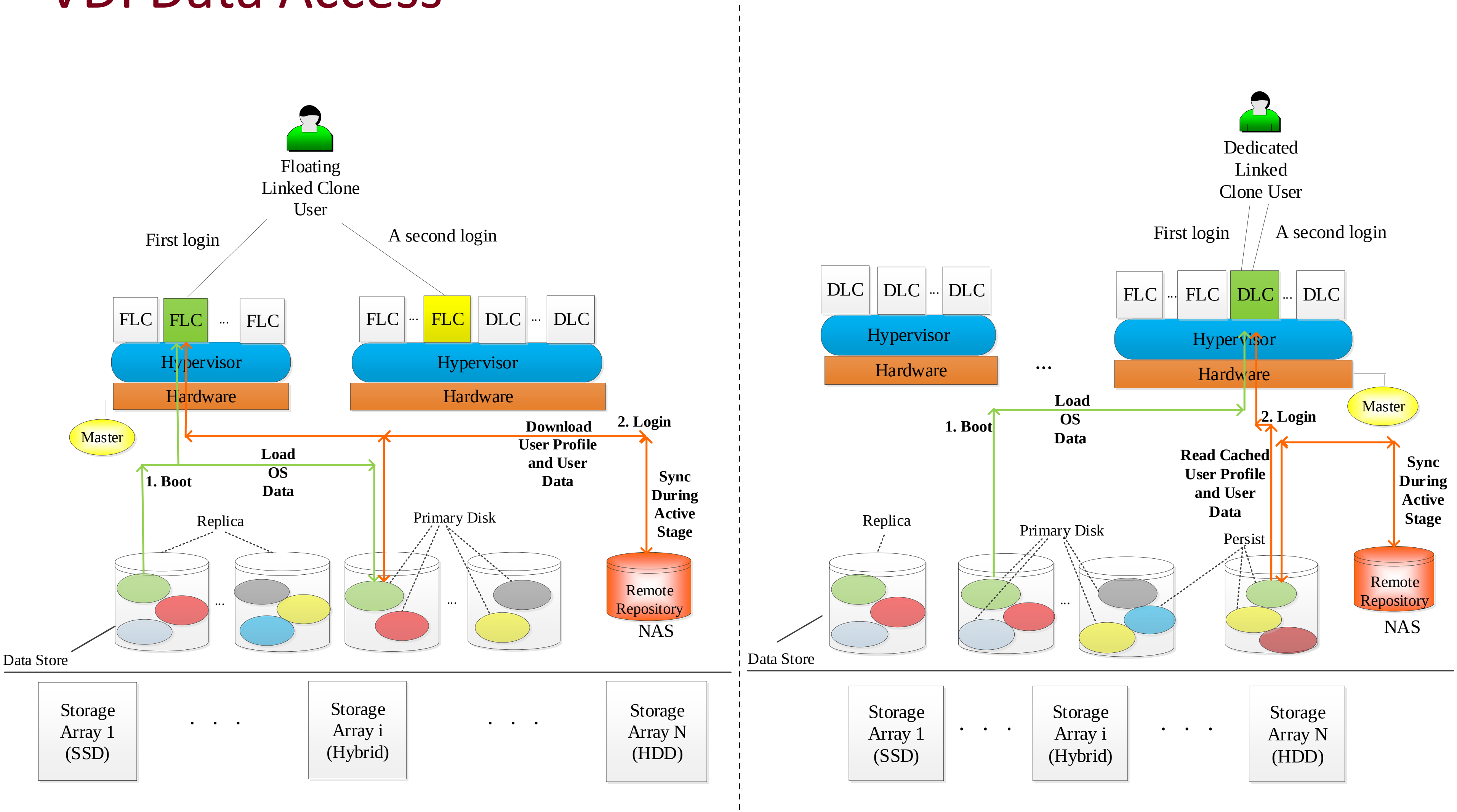
Key: We need a model!

[4]Vmware virtual san design and sizing guide for horizon are.com/content/dam/digitalmarketing/vmware/en/pdf/whitepaper/products/vsan/vmw-tmd-virt-san-dsn-szing-guid-horizon-view-white-paper.pdf.
[5]Sizing and best practices for deploying vmware view 5.1 on vmware vsphere 5.0 u1 with dell equallogic storage. https://downloads.dell.com/manuals/all-products/esuprt_solutions_int/esuprt_solutions_int_solutions_resources/s-solution-resources_white-papers71_en-us.pdf.
[6]vsan. https://www.vmware.com/products/vsan.html.

Our Contributions

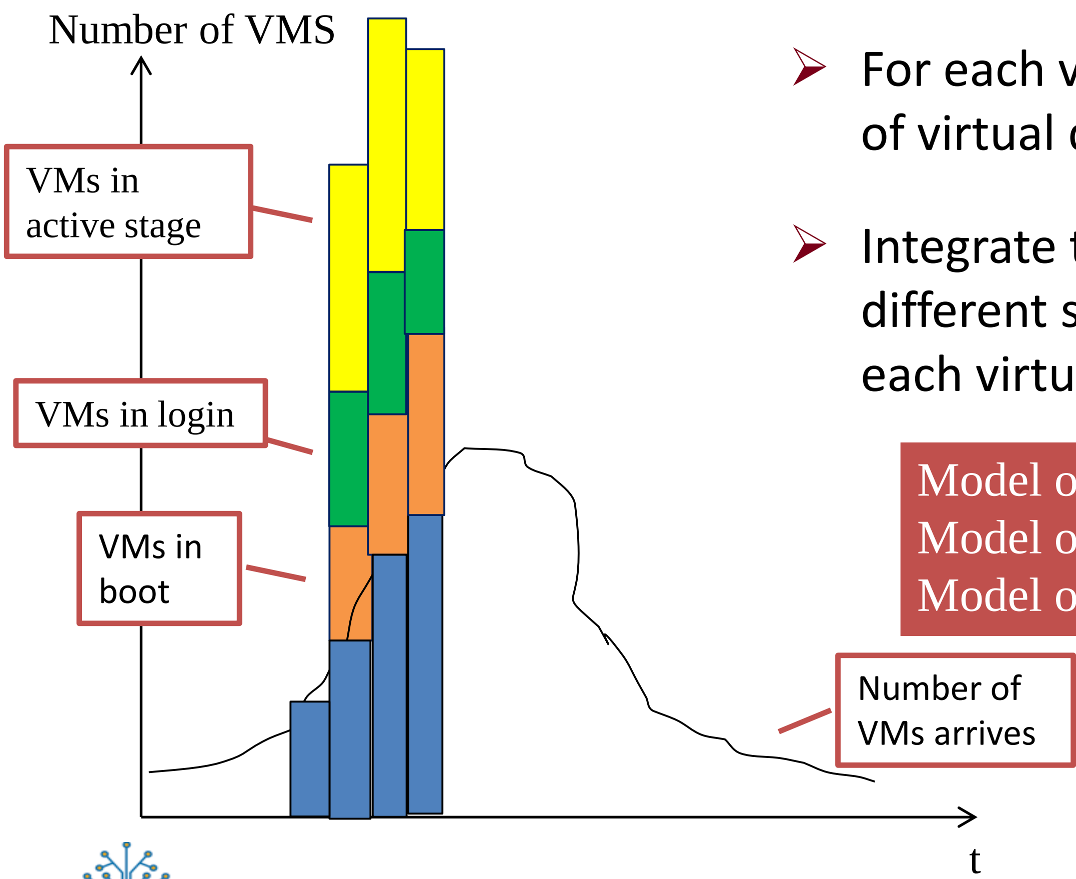
- We describe different types of virtual desktops and discuss their unique storage access patterns.
- We propose a system model to describe the I/O behaviors of both homogeneous and heterogeneous configurations of VDI.
- We identify the storage requirements of VDI and determine the bottlenecks on specific target virtual disks at a specific time.
- With the detailed storage requirements, we showed how to size a minimum storage configuration to satisfy storage requirements of VDI.

VDI Data Access



Model

Answer at time t, how much data will be read from each virtual disk and how much data will be written to each virtual disk.



- For each virtual disk, determine the type of virtual desktops that will access it.
- Integrate the I/Os from those VMs at different stages into the at time t, for each virtual disk.

Model of a single VM
Model of multiple VMs of the same type
Model of multiple VMs of different types

Validation

- Collect boot, login, and active stage traces of different types of virtual desktops in VDI. (VDI cluster + VMware View Planner)
- Analyze the traces and derive those parameters needed in our model
- Plug those parameters into the model and generate storage demands

Comparison between the throughput requirement calculated from the model of a single VM with the direct measurement

Virtual Desktop	Virtual Disk	Average Read KB/s			Average Write KB/s		
		Measured	Calculated	Diff	Measured	Calculated	Diff
Floating Linked Clone	Replica	4813.90	4812.14	0.04%	0	0	0
	Primary	279.91	277.03	1.03%	641.98	649.62	1.19%
	NAS	212.91	209.33	1.68%	15.66	15.05	3.90%
Dedicated Linked Clone	Primary	269.24	266.98	0.84%	397.87	397.62	0.06%
	Persistent	49.68	48.85	1.7%	31.39	30.85	1.7%
	NAS	27.13	25.97	4.3%	12.87	12.74	1.0%
Full Clone	Full Clone Disk	1401.54	1400.66	0.06%	284.09	282.74	0.48%

Application

5.29 IOPS from traces

Table: VDI IOPS Requirements from VMware

User Classification	IOPS Requirements Per User
Light	3-7
Medium	8-16
Standard	17-25
Heavy	25+

Storage for light user!

Storage for heavy user

Table: Requirements of a Floating Linked Clone

	Replica		Primary		NAS	
	R	W	R	W	R	W
Average KB/s	4813.90	0	279.91	641.98	212.91	15.66
IOPS	68.55	0	12.26	29.41	5.00	2.14
Capacity per VDI user if thin provisioned: 4GB. 3 shared replicas: 75GB Shared NAS: 1TB						

More fine-grained QoS requirements of a VDI system

Application

- Storage sizing tool

Storage requirements of a company with 5000 FLCs

	Replica	Primary Disk	NAS
Throughput	Read: 3 GB/s – 3.3 GB/s	Read: 350 MB/s Write: 600MB/s	Read: 70 MB/s
Capacity	66 TB		
IOPS	105,000		

Table: Specifications of 4 HP 3PAR Storage Systems

HP 3PAR Storage	F200	F400	T400	T800
Max Throughput	1300 MB/s	2600 MB/s	2800 MB/s	5600 MB/s
Max IOPS	46,800	93,600	128,000	256,000
Max Capacity	128 TB	384 TB	400 TB	800 TB
Drive Types	50GB SSD 300 & 600 GB FC 2TB NL	50GB SSD 300 & 600 GB FC 2TB NL	50GB SSD 300 & 600 GB FC 2TB NL	50GB SSD 300 & 600 GB FC 2TB NL

Improve Storage Services of Docker Container and Kubernetes

Kubernetes - Distributed OS of Containers

An orchestrator is essential to deploy and manage applications in containers across multiple hosts.

- Application scheduling
- Resource management
- Mainstream: Docker swarm, Mesos, and Kubernetes (k8s)[7] [Verma et al. EuroSys '15, Burns et al. Queue 14, 1]

Kubernetes is the most popular container orchestration platform according to surveys from Cloud Native Computing Foundation (CNCF) [8,9]

In this research, we focus on Kubernetes environment based on Docker.

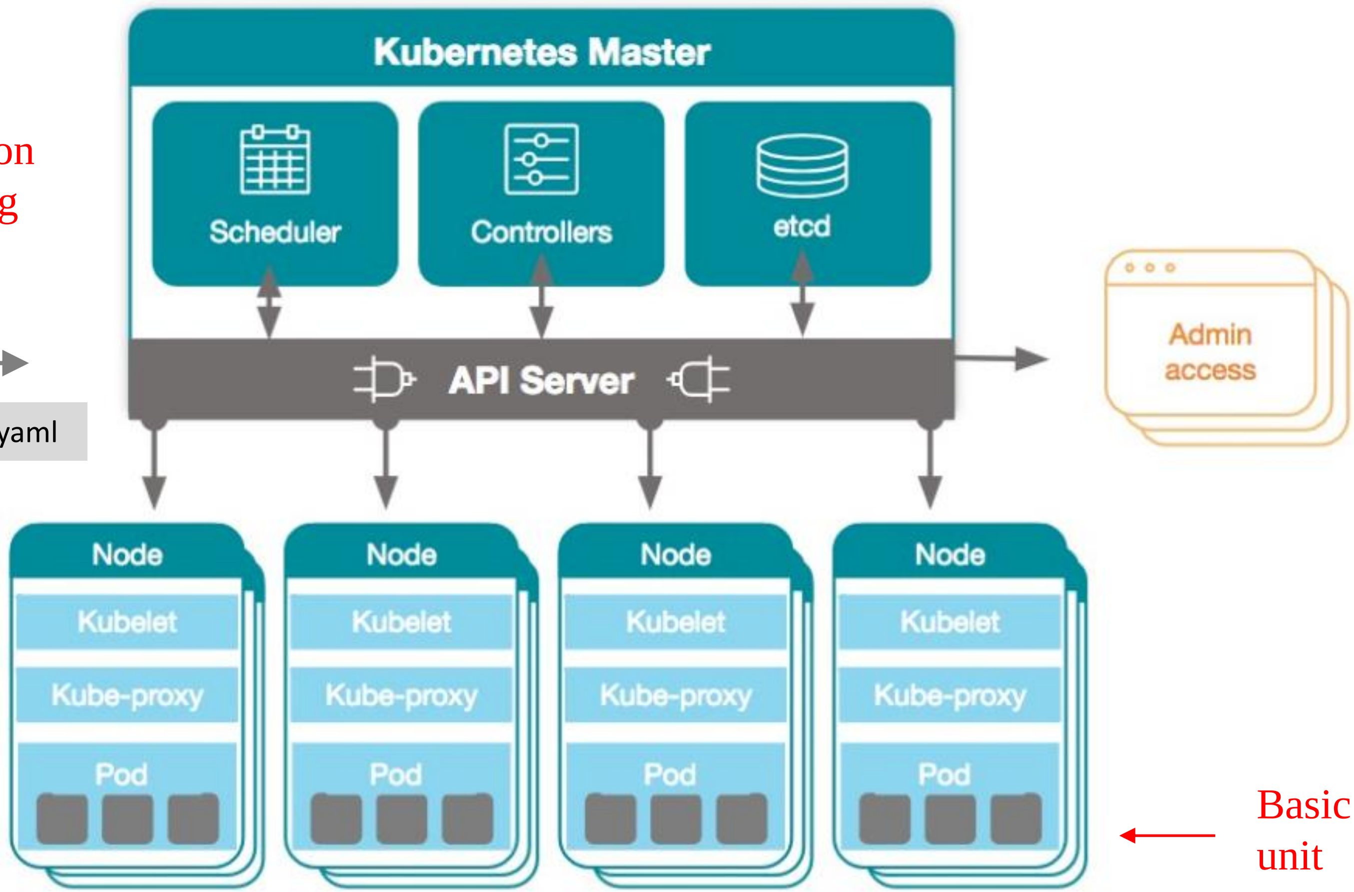
[7] Kubernetes concepts. https://kubernetes.io/docs/concepts/overview/what-is-kubernetes/.
[8] Survey Shows Kubernetes Leading as Orchestration Platform. https://www.cncf.io/blog/2017/06/28/survey-shows-kubernetes-leading-orchestration-platform/.
[9] CNCF Survey: Use of Cloud Native Technologies in Production Has Grown Over 200%. https://www.cncf.io/blog/2018/08/29/cncf-survey-use-of-cloud-native-technologies-in-production-has-grown-over-200-percent.

Kubernetes - Distributed OS of Containers

Application scheduling

kubectl create –f app.yaml

Create pods and allocate storage



Basic unit

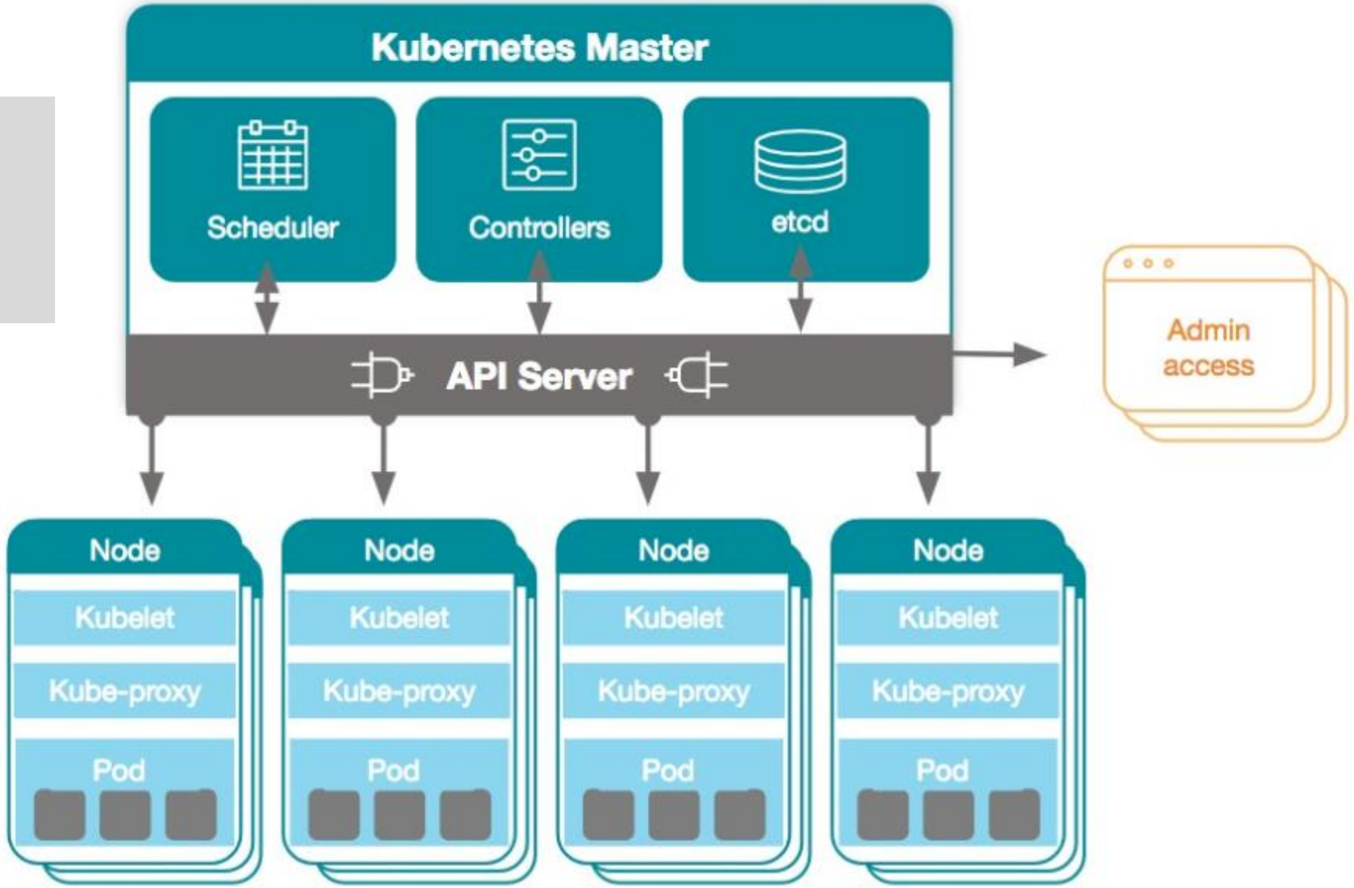
Issues of Kubernetes in Storage Allocation

Storage allocation is static

CPU, Mem, Affinities to apps/nodes ✓

Storage resources ✗

Error-prone, not resource efficient storage allocation



Static Storage Allocation in K8s

- K8s allocates storage based on *StorageClass* (SC)

Admins create SCs

Users choose SCs

Limitations:
- SC is static. Storage performance is changing
- Few SCs -> Over provisioning
  Lots of SCs -> Hard to maintain
- Advanced storage requirements, e.g., rate limiting, caching, etc.
- Not user friendly and error-prone

How can we make k8s better meet users’ storage requirements & all other requirements, and at the same time save resources?

Related Work

- Storage management in Kubernetes is still underexplored.
  - REX-Ray[10] requires a volume that is to be used by a k8s must be **previously created and discoverable**. (manual provisioning of PV)
  - NetApp's Trident[11] can work as a storage provisioner in k8s. **Based on SC.**

- Wisdom from storage management in VM
  - Pesto [Gulati et al. SOCC '11] is implemented as part of VMware's Storage DRS component of vSphere (VMware hypervisor). Automatically model, estimate storage performance, recommend VM disk placement and migration.
  - **Not directly applicable** to container orchestrator.
  - Regardless of the storage backend, it **must follow the interface of SC**.

[10] REX-Ray. https://rexray.readthedocs.io/en/stable/
[11] Trident. https://netapp-trident.readthedocs.io/en/stable-v19.01/

Our Contributions

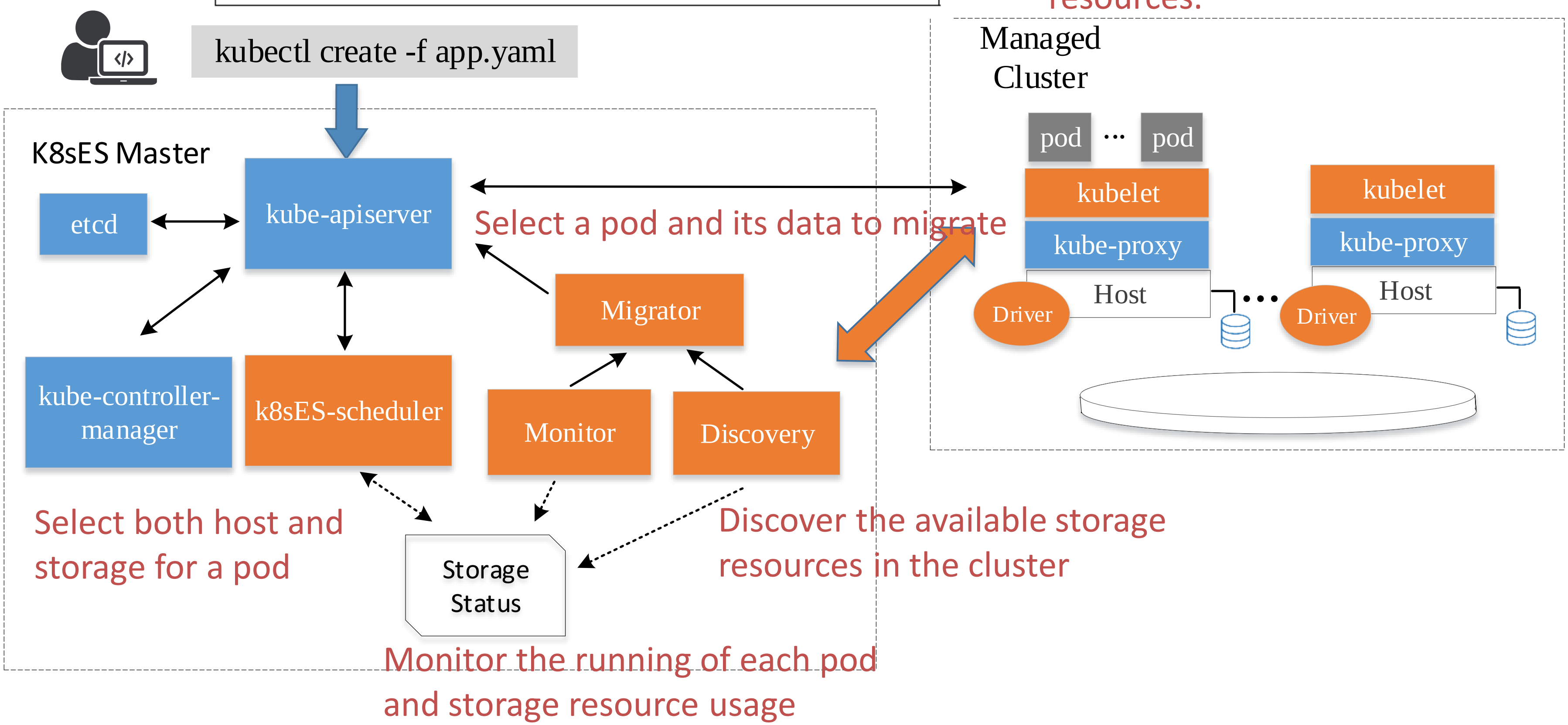
We propose *K8sES* (k8s Enhanced Storage), a system that can dynamically allocate storage to applications in Kubernetes based on users' storage requirements.

- Initial storage allocation
  - **Storage monitoring capabilities**: performance of storage devices
  - **User friendly.** Allow users to specify storage requirements directly in config.
  - **No limitations of SC.** Admins don't create SC.
  - **Strengthened scheduling.** Select storage with other k8s related requirements
  - **Automatic storage provisioning** based on users' requirements
- Storage adjustment at runtime
  - **Storage monitoring capabilities:** storage SLOs of a pod
  - **Migration**
- **Improves storage utilization efficiency** in k8s: thin provisioning, multiplexing, balancing utilization between storage and non-storage

Pod Creation in K8sES

```
<k8sES volume>
  size: x GB
  sustained bw: y MB/s
  sharing: False
  reclaim: Retain
  policy: WHEN GETS/s > z, SET CACHING
```

The kubelet receives the storage decision from k8es-scheduler and call the ***Driver*** to carve out storage resources.



Evaluation Setup

k8s workers	CPU	MEM	Local Size	Local Bandwidth	Shared Storage
Worker 1	2	2GB	70GB	20MB/s	Share same 100GB at 50MB/s
Worker 2	2	2GB	50GB	50MB/s	
Worker 3	2	2GB	20GB	100MB/s	
Worker 4	6	4GB	50GB	50MB/s	Share same 100GB at 50MB/s
Worker 5	3	3GB	70GB	20MB/s	
Worker 6	3	2GB	50GB	50MB/s	
Worker 7	2	2GB	10GB	100MB/s	

- Applications:

  ssbench[12] + OpenStack Swift

  HTTP + Nginx[13]

  Synthetic workloads with uniform distributed I/O throughput

  Synthetic applications with various requests

[12] Swiftstack benchmark suite (ssbench). https://github.com/swiftstack/ssbench.
[13] Nginx. https://www.nginx.com/.

Evaluation Result (initial allocation)

We deploy pods E1, E2, and F in sequence, each requiring *10GB, 20MB/s, non-sharing storage + 1 CPU core + 1 GB Mem*

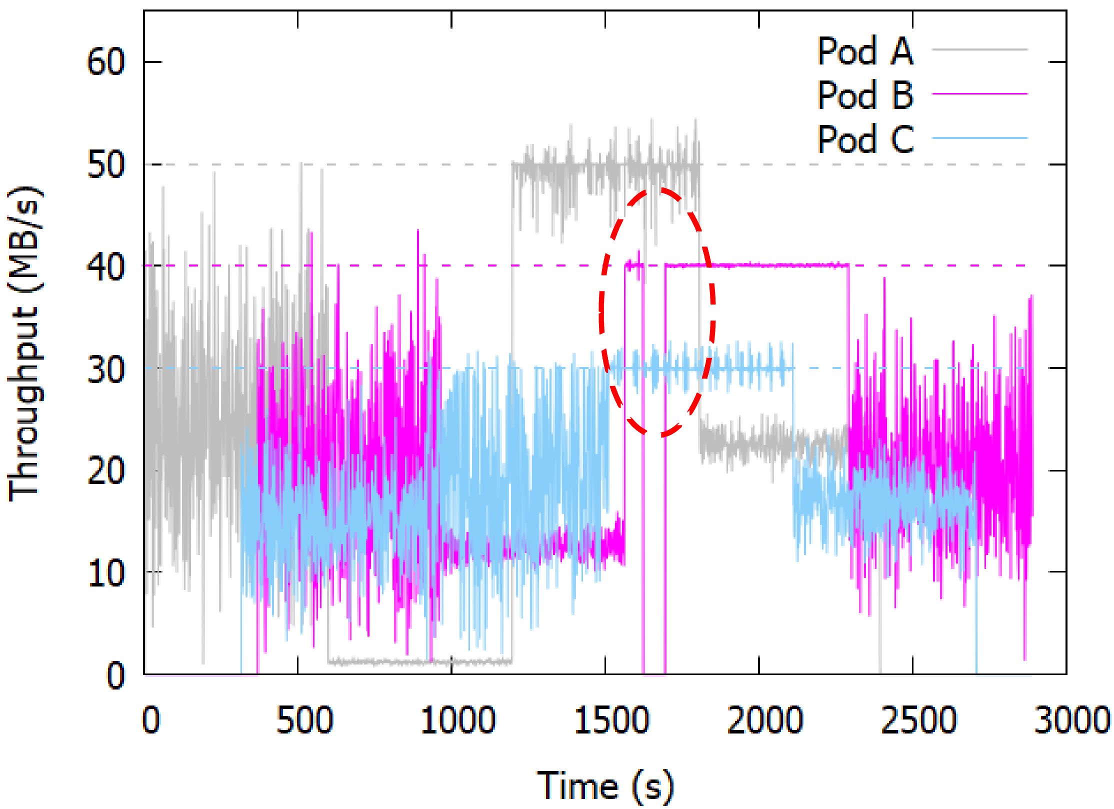
Workers	CPU (before F)	Memory (before F)	Remaining Capacity (before F)	Remaining Bandwidth (before F)	k8s Scheduling Results
Worker 5	2	2	60GB	0	E1,F (✗)
Worker 6	2	1	40GB	30MB/s	E2

(a) k8s scheduling result

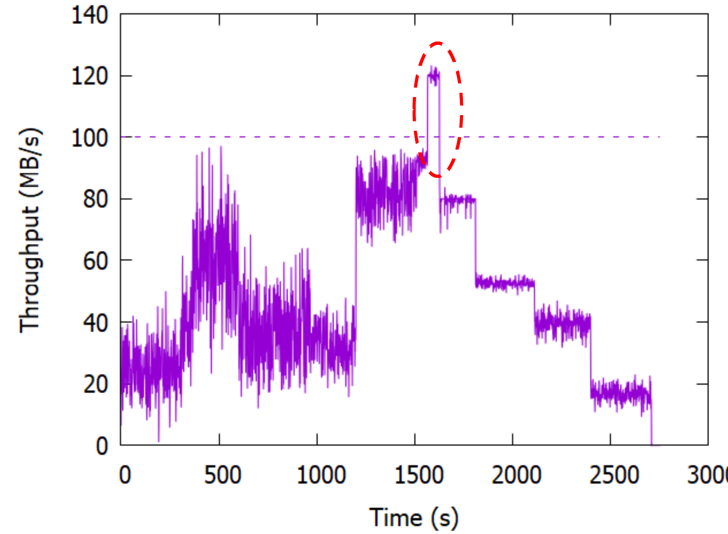
Workers	CPU (before F)	Memory (before F)	Remaining Capacity (before F)	Remaining Bandwidth (before F)	k8s Scheduling Results
Worker 5	2	2	60GB	0	E1
Worker 6	2	1	40GB	30MB/s	E2,F

(b) k8sES scheduling result

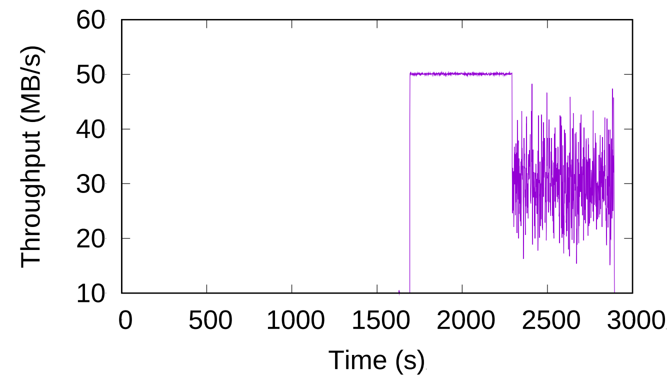
Evaluation Result (at runtime)



Throughput of applications over their lifetime



I/O throughput of pod B on Worker3



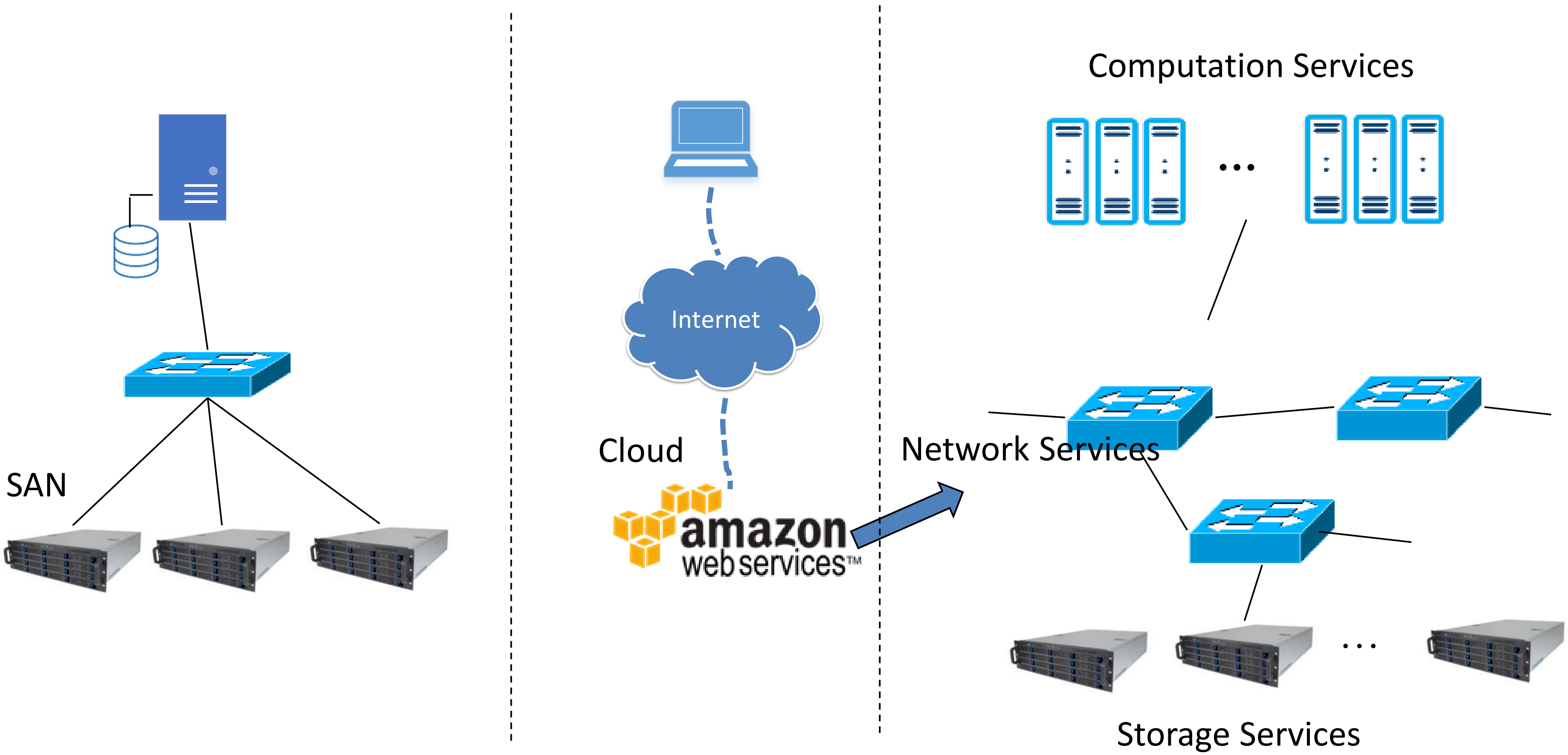
I/O throughput of pod B on Worker4

Improve Latency SLO with Integrated Control for Networked Storage

Center for Research in Intelligent Storage

University of Minnesota

Network is Important in Storage



E.g., OpenStack (VM), Kubernetes (containers)

Problem and Challenges

In the networked storage environment, how can we **coordinate different components** in network and storage to improve latency SLOs for applications?

Challenges:

- Different components involved, e.g., clients, network switches, storage servers, disks, etc.
- Status of the components are dynamically changing
- Each component performs different functions on I/Os

Related Work

- Most existing research focuses on a single component
  - Redundancy-based approach may alleviate the congestion on network, but congest storage [Zhe et al. NSDI ’15].
  - RED [Sally et al. TON 1 ,4] and ECN [Sally Floyd. SIGCOMM 24, 5] throttle client in network congestions, but may waste resources in underloaded storage.

- Studies that try to involve both network and storage
  - IOFlow [Eno et al. SOSP ‘13] enforces policies on I/O stacks, but does not have control on network between clients and servers.
  - sRoute [Ioan et al. FAST ‘16] extends IOFlow’s routing functions and is able to forward I/Os from over loaded servers onto less loaded servers. But does not consider the status of network.
  - PriorityMeister [Zhu et al. SOCC ‘14] automatically and proactively configure workload priorities and rate limits. It assumes the system has **full visibility and control over all workloads**. The priorities are **static** and **at the granularity of workload**.

Our Contributions

- We identify the need to consider all the components along the I/O path from client to storage to ensure latency SLO.
- We design controller-based mechanisms to coordinate the control on different components based on the status of each component.
- We design an approach to control I/O packets with little overhead based on the asymmetry property in read and write.
- We build a real system called JoiNS, to coordinate clients, network, and storage, and demonstrate the effectiveness of JoiNS in ensuring latency SLO.

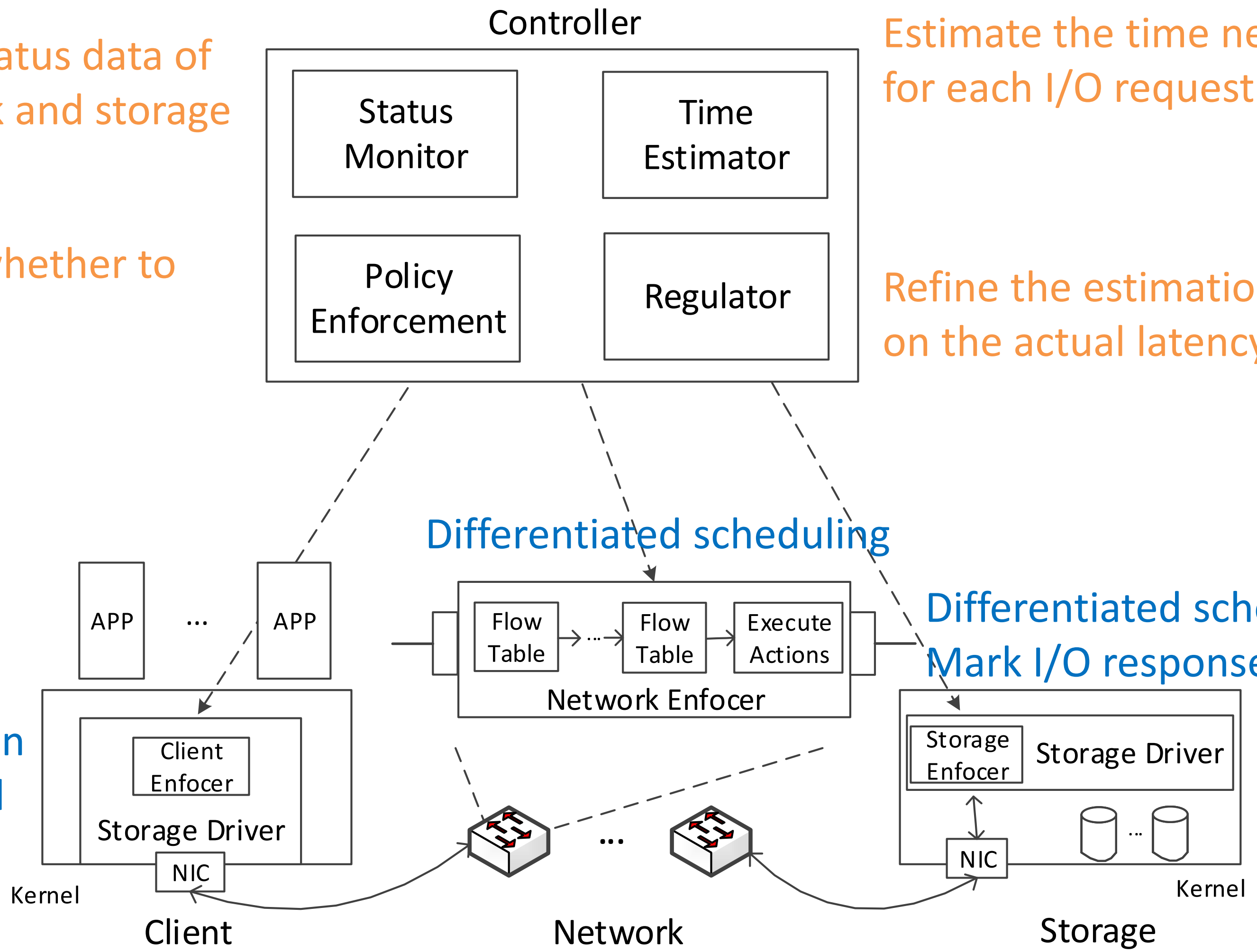
JoiNS Architecture

Collect the status data of each network and storage node

Determine whether to control I/Os

Estimate the time needed for each I/O request

Refine the estimation based on the actual latency



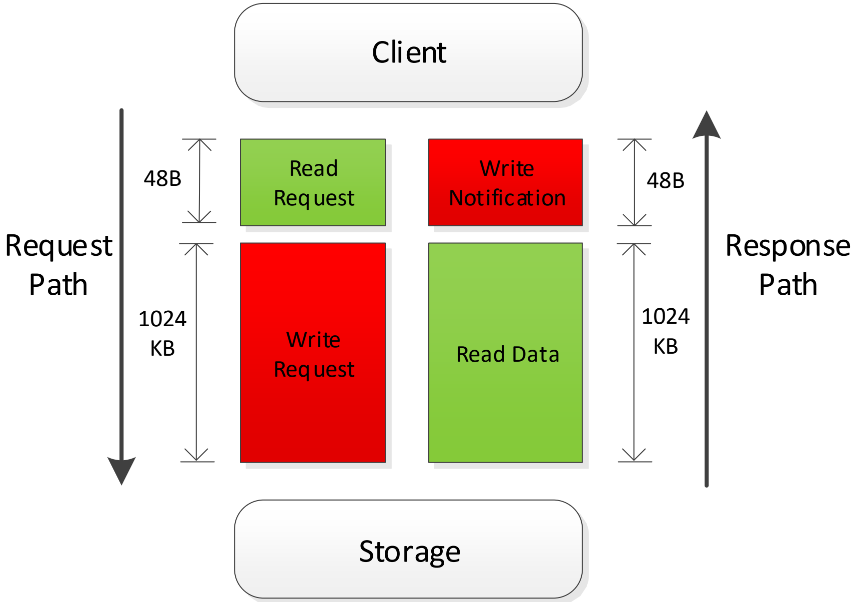
Admit I/Os
Mark I/O requests in packet headers and storage commands

Coordination

- Probe and Test
  - The status monitor send out probes periodically to collect the status data of network and storage.
  - Estimate the latency for each I/O request
  - Test the congestion level for each I/O request.
- Admission control at client enforcer
  - Marking I/O in the corresponding network packet header
- Interactions
  - Control information sent to network incorporated in network headers. Control information sent to storage incorporated in SCSI commands.
- Network and storage enforcers prioritize I/O

Cost-effective Control

- Distinguish Read from Write
  - Based on the asymmetry property in read and write along its I/O delivering path.
  - Read requests can be prioritized on request path with little penalty.
  - Write responses can be prioritized on return path with little penalty.



Evaluation Setup

- System setup

  1 client. 2 network nodes serve as SDN switches with the link speed of 1Gb/s.

  1 storage proxy. 1 storage server with one HDD backend.

- Datasets

  MSR block traces and synthetic traces

- Policies

  JoiNS: Our primary mechanism

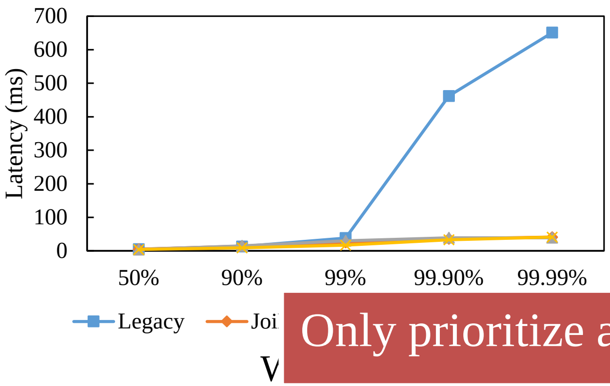
  Legacy: FIFO in network and storage.

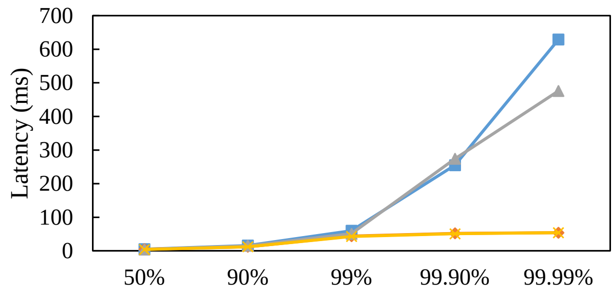
  Pri_all: prioritize all read requests and write responses regardless of congestion level

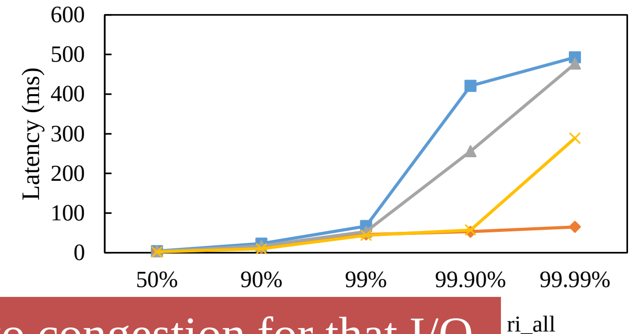
  PM: PriorityMeister (rate limiters + static priorities to workloads)[25]

Evaluation Result

Request latency of workloads running at the same time at different percentiles

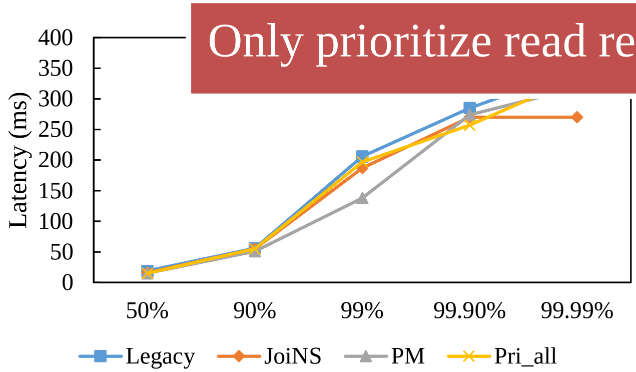




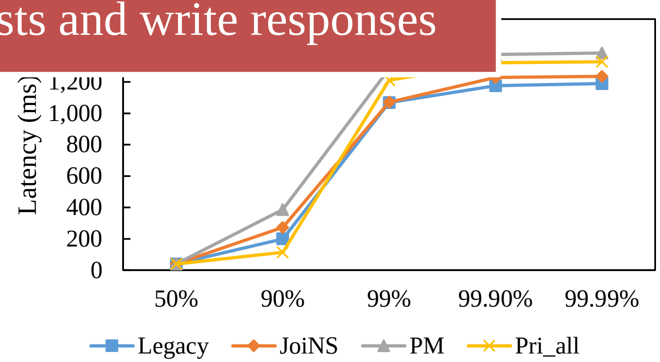


Only prioritize an I/O when the system is close to congestion for that I/O

Only prioritize read requests and write responses



Workload D



Workload E

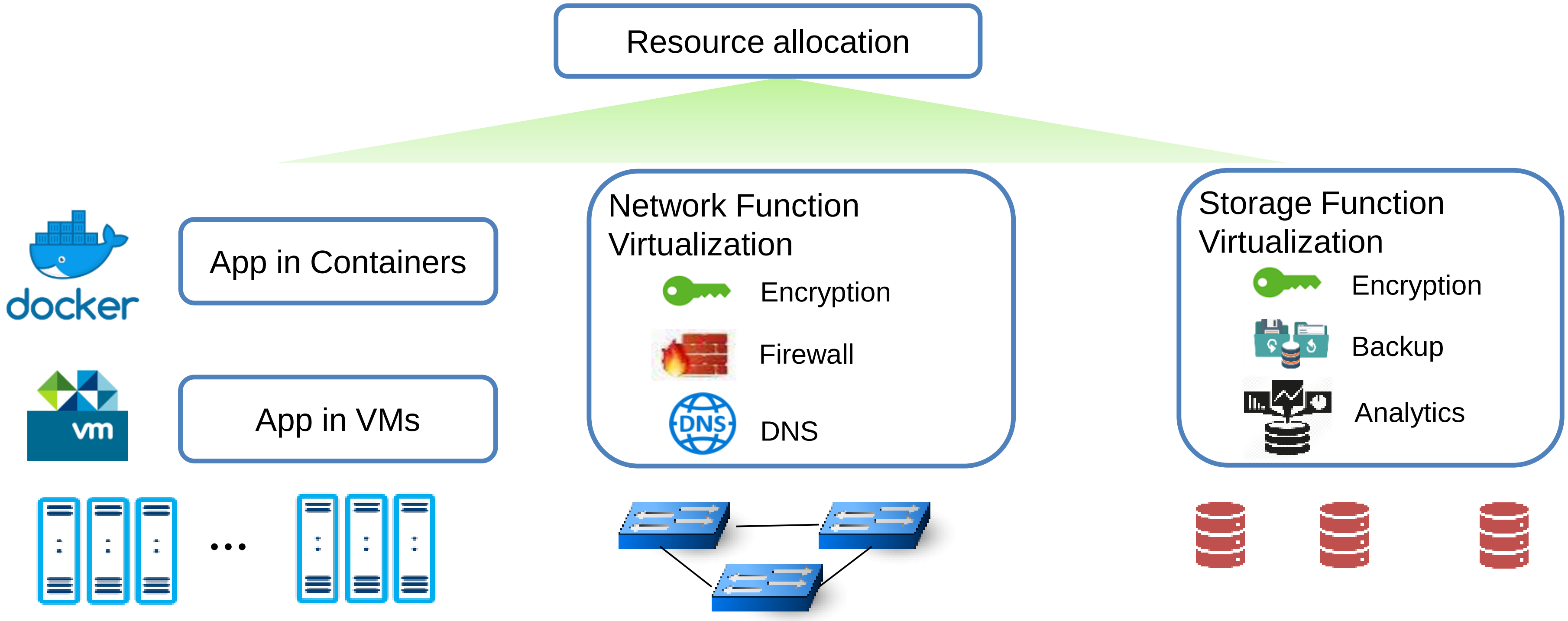
Conclusion

- Meeting Storage Requirements of VDI Applications in the Virtual Machine Environment
  - Propose a system model to describe Virtual Desktop Infrastructure (VDI), identify and meet the storage requirements.

- Improve Storage Services of Docker container and Kubernetes
  - Propose *K8sES* (k8s Enhanced Storage) to meet users' storage requirements as well as other requirements in k8s, and improve the storage utilization efficiency.

- Improve Latency SLO with Integrated Control for Networked Storage
  - Propose JoiNS, a system that coordinates different components along the I/O path to meet latency SLO in a networked storage environment

Future Work



Ability to control all resources ⇒ Systematic control over client, network, storage for app in networked storage

Selected Publications

Published

- **Hao Wen**, David HC Du, Milan Shetti, Doug Voigt, and Shanshan Li. Guaranteed bang for the buck: Modeling vdi applications with guaranteed quality of service. In Parallel Processing (ICPP), 2016 45th International Conference on, pages 426-431. IEEE, 2016.
- Zhichao Cao, **Hao Wen**, Fenggang Wu, and David H.C. Du. ALACC: Accelerating restore performance of data deduplication systems using adaptive look-ahead window assisted chunk caching. In 16th USENIX Conference on File and Storage Technologies (FAST 18), pages 309-324, Oakland, CA, 2018. USENIX Association.
- Li, B., **Wen, H.**, Toussi, F., Anderson, C., King-Smith, B. A., Lilja, D. J., & Du, D. H. (2019). NetStorage: A synchronized trace-driven replayer for network-storage system evaluation. Performance Evaluation, 130, 86-100.
- Fenggang Wu, Baoquan Zhang, Zhichao Cao, **Hao Wen**, Bingzhe Li, Jim Diehl, Guohua Wang, David H.C. Du, "Data Management Design for Interlaced Magnetic Recording", HotStorage'18.
- **Hao Wen**, Zhizhao Cao, Yang Zhang, Ziqi Fan, Doug Voigt, David H.C. Du, "JoiNS: Meeting latency SLO with Integrated Control for Networked Storage", In Proceedings of the 26th IEEE International Symposium on the Modeling, Analysis, and Simulation of Computer and Telecommunication Systems MASCOTS'18.
- Zhichao Cao, **Hao Wen**, Xiongzi Ge, Jingwei Ma, Jim Diehl, and David H. C. Du. Tddfs: A tier-aware data deduplication-based file system. ACM Trans. Storage,15(1):4:1-4:26, February 2019.
- **Wen, H.**, Du, D. H., Shetti, M., Voigt, D., & Li, S. (2018). Guaranteed Bang for the Buck: Modeling VDI Applications to Identify Storage Requirements. IEEE Transactions on Cloud Computing.

Selected Publications

On-going

- **Hao Wen**, David H.C. Du, Zhichao Cao, Bingzhe Li, Doug Voigt, Ayman Abouelwafa, Shiyong Liu, Fenggang Wu, Jim Diehl. K8sES: Kubernetes with Enhanced Storage Service-Level Objectives. [Under Revision]
- Fenggang Wu, Bingzhe Li, Zhichao Cao, Baoquan Zhang, Ming-Hong Yang, **Hao Wen**, David H.C. Du. ZoneAlloy: Elastic Data and Space Management for Hybrid SMR Drives. [Under Submission]

Reference

[1] Lxc. https://help.ubuntu.com/lts/serverguide/lxc.html.

[2] BERNSTEIN, D. Containers and cloud: From lxc to docker to kubernetes. IEEE Cloud Computing 1, 3 (2014), 81–84.

[3] VERMA, A., PEDROSA, L., KORUPOLU, M., OPPENHEIMER, D., TUNE, E., AND WILKES, J. Large-scale cluster management at google with borg. In Proceedings of the Tenth European Conference on Computer Systems (2015), ACM, p. 18.

[4] BURNS, B., GRANT, B., OPPENHEIMER, D., BREWER, E., AND WILKES, J. Borg, omega, and kubernetes. Queue 14, 1 (2016), 10.

Gulati A, Shanmuganathan G, Ahmad I, et al. Pesto: online storage performance management in virtualized datacenters[C]//Proceedings of the 2nd ACM Symposium on Cloud Computing. ACM, 2011: 19.

[5] Joe Beda. Containers at scale. https://speakerdeck.com/jbeda/containers-at-scale?slide=2.

[6] Zhe Wu, Curtis Yu, and Harsha V. Madhyastha. Costlo: Cost-effective redundancy for lower latency variance on cloud storage services. In Proceedings of the 12th USENIX Conference on Networked Systems Design and Implementation, pages 543-557, 2015.

[7] Sally Floyd and Van Jacobson. Random early detection gateways for congestion avoidance. IEEE/ACM Transactions on Networking (ToN), 1(4):397-413, 1993.

[8] Sally Floyd. Tcp and explicit congestion notification. ACM SIGCOMM Computer Communication Review, 24(5):8-23, 1994.

[9] Eno Thereska, Hitesh Ballani, Greg O'Shea, Thomas Karagiannis, Antony Rowstron,Tom Talpey, Richard Black, and Timothy Zhu. Ioflow: A software-defined storage architecture. In Proceedings of the Twenty-Fourth ACM Symposium on Operating Systems Principles, pages 182-196, 2013.

[10] Ioan Stefanovici, Bianca Schroeder, Greg O'Shea, and Eno Thereska. sroute: Treating the storage stack like a network. In 14th USENIX Conference on File and Storage Technologies (FAST 16), pages 197-212, 2016.

[11] Zhu T, Tumanov A, Kozuch M A, et al. Prioritymeister: Tail latency qos for shared networked storage[C]//Proceedings of the ACM Symposium on Cloud Computing. ACM, 2014: 1-14.

Thanks!

Questions/Comments

Backup – VDI Model

- Answer at time t, how much data will be read from each virtual disk and how much data will be written to each virtual disk.

(1) Model of a single VM

$$\sum_{i} E_{stage,target}(t) \times dt \times RWper_{stage,target} \times S^{i}_{stage,target} \times Psize^{i}_{stage,target}$$

Target: the virtual disk that IOs will reach

Stage: the stage in VM life cycle

$RWper_{stage,target}$: read ratio or write ratio during different stages on different targets

$S^{i}_{stage,target}$: Significant IO sizes

$Psize^{i}_{stage,target}$: Percentage of each significant IO size

$E_{stage,target}(t)$: the expected number of IOs at time t

Backup – VDI Model

- Model of multiple VMs of the same type

$$\sum_{x=t_1}^{t_2} [N(x) \times \sum_i E_{stage,target}(\mathrm{t}) \times dt \times RWper_{stage,target} \times S_{stage,target}^{i} \times Psize_{stage,target}^{i}]$$

N(x): indicates the number of VMs arriving at time x(x<t) (VM arrival rate)

$E_{stage,target}$（t）: For each group of N(x) VMs that arrive at time x, it describes the expected number of IOs at time t for that particular group of VMs.
$[t_1, t_2]$: For all those VMs that are now at stage, they arrive during time interval [t1, t2]

Backup – VDI Model

- Model of multiple VMs of different types
  - Define VM type:

$$OS = \{OS_1, OS_2, \dots, OS_n\}$$
$$VD = \{FLC, DLC, FC\}$$
$$APP = \{app_1, app_2, \dots, app_n\}$$
$$VM = OS \times VD \times APP$$

  - For each element in VM set, apply the model of multiple VMs of the same type

  - Plug in the weight of proportion of each type and calculate the weighted average of all VM types to get the overall size of data accessed

Backup – VDI Measured Throughput and IOPS

TABLE 2: Requirements of a Floating Linked Clone

TABLE 3: Requirements of a Dedicated Linked Clone

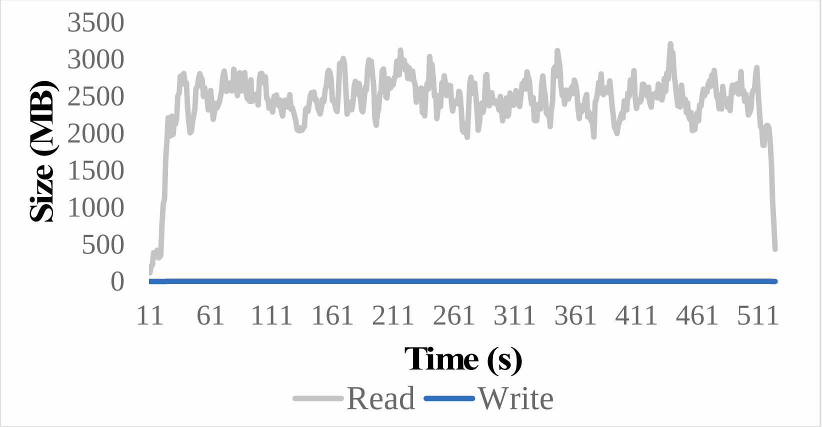
	Primary	Persistent	NAS
			W
Aver			12.87
IOPS			1.78
Capa			
3 sha			
Shar			

TABLE 4: Requirements of a Full Clone

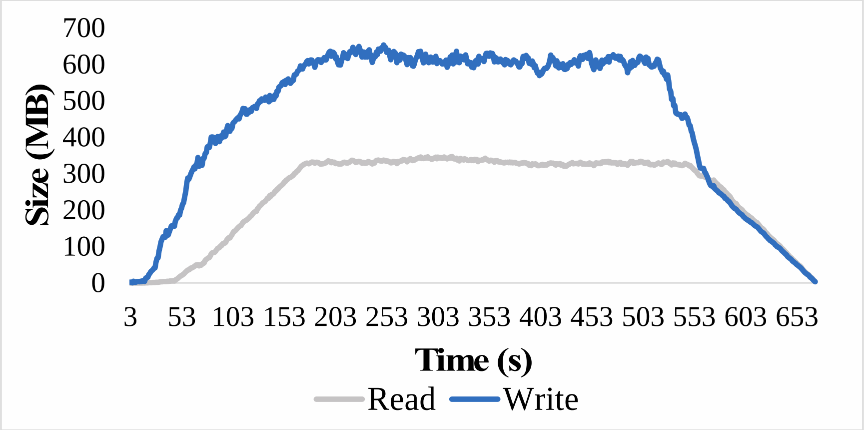
	Full Clone Disk	
	R	W
Average KB/s	1401.54	284.09
Average IOPS	26.34	22.18
Capacity per VDI user if thin provisioned: 12GB.		

Backup – VDI Simulation on Multiple Virtual Desktops

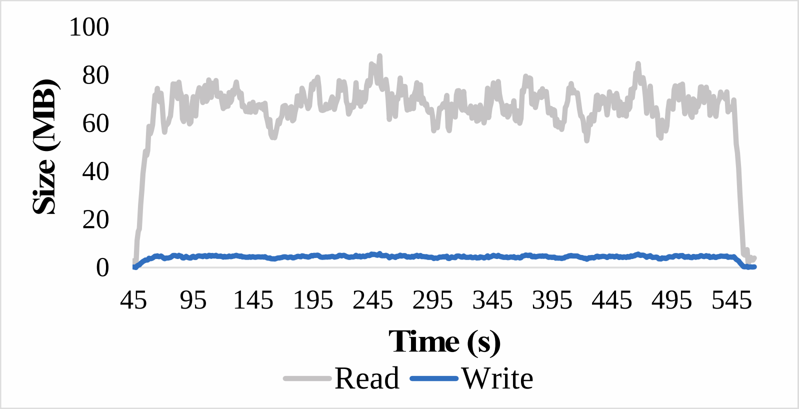
- E.g., Floating linked clone



Replica



Primary Disk



NAS

Backup – VDI Validation

- Comparison between the parameters calculated from the simulation and the experimental results from two Hewlett Packard Enterprise (HPE) systems

In HPE systems:

- IOmark-VDI [9] is used as the benchmark tool to generate VDI workloads.

Result:
Peak IOPS per virtual desktop: 139
Peak IOPS happens at boot stage.
The peak read IOPS 9x of the peak write IOPS.
Avg IOPS during active stage: 6.26.

- We simulate multiple floating linked clones arriving at the same time.
- We set the arrival rate of virtual desktops to match the total number of virtual desktops from the HPE.

Result:

✓Peak IOPS 141

✓ Peak read IOPS 8.8x peak write IOPS
✓ Peak IOPS happens during the boot stage.
✓ Avg IOPS during active stage: 5.29.

Backup – Easy and direct interface

kubectl create –f <manifest>

```yaml
kind: Pod
apiVersion: v1
metadata:
  name: mypod
spec:
  containers:
    - name: myfrontend
      image: nginx
      volumeMounts:
      - mountPath: "/var/www/html"
        name: mypd
  volumes:
    - name: mypd
      persistentVolumeClaim:
        claimName: myclaim
```

Goal: Keep the interface but also enable users specify various storage requirements?

Current: Ask the admin to create a suitable SC first!

Why don't we enable users to put their requests directly in the manifest?

```
<k8sES volume>
  size: x GB
  sustained bw: y MB/s
  sharing: False
  reclaim: Retain
  policy: WHEN GETS/s > z, SET CACHING
```

Backup – PV and SC in K8s

- Currently, k8s allocates storage based on *PersistentVolume* (PV) and *StorageClass* (SC), which have limited storage support.

```
kind: StorageClass
apiVersion: storage.k8s.io/v1
metadata:
  name: slow
provisioner: kubernetes.io/aws-ebs
parameters:
  type: io1
  iopsPerGB: "10"
  fsType: ext4
```

By Admin

```
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
  name: myclaim
spec:
  accessModes:
    - ReadWriteOnce
  volumeMode: Filesystem
  resources:
    requests:
      storage: 8Gi
  storageClassName: slow
```

By User

```
kind: Pod
apiVersion: v1
metadata:
  name: mypod
spec:
  containers:
    - name: myfrontend
      image: nginx
      volumeMounts:
      - mountPath: "/var/www/html"
        name: mypd
  volumes:
    - name: mypd
      persistentVolumeClaim:
        claimName: myclaim
```

By User

Backup – Current Storage Support of K8s

- Summary of limitations

(1) SC is static and cannot be used to efficiently schedule storage resources.
   - The actual performance of the storage changes with the utilization of the storage resources.

(2) Hard to decide a proper number of SCs that just satisfies users' requirements without wasting resources.
   - Few SCs -> Has to pick one that provide more resources.
   - More SCs -> Higher resource utilization but harder to maintain.

(3) Do not support advanced storage requirements, e.g., rate limiting, caching, etc.

(4) Not user friendly and error prone.

Backup – Problem and Challenges

Users deploy their stateful applications in containers in k8s. They have service-level objectives (SLOs) on storage. How can we make k8s **better meet users' various storage requirements along with all other requirements**, and at the same time **save resources**?

Challenges:

- Allocate appropriate amount of resources to meet users' storage SLOs, and save resources at the same time in k8s. (issues of fewer SCs or more SCs)
- Unchanged API and more intelligent storage allocation?
- In addition to storage requirements, applications in k8s have CPU, memory, and other k8s specific requirements, e.g., node affinity, pod affinity, etc. How can we integrate the intelligent storage allocation into the current pod scheduling process?
- Ensuring SLOs at runtime.

Backup – Scheduling Algorithm

Predicate

- filter out hosts that cannot meet all the predefined predicates, e.g. Mem > 1 GB → {host list}
- {host list} + <storage accessibilities> →{storage list}
- Check storage predicates → {host:{storage list}}

Priority

- Score hosts and storage based on a list of rules

for storage

$$least\_storage\_usage\text{: } (10 \times \frac{Size_{total} - Size_{req}}{Size_{total}} + 10 \times \frac{BW_{total} - BW_{req}}{BW_{total}})/2$$

$$usage\_leveling\text{: } 10 - 10 \times |CPU_{usage} + Mem_{usage} - Size_{usage} - BW_{usage}|$$

Select

- fair consideration: pick the storage with highest score, and add it to the score of host.

Backup – More Resource Efficient

Compared with VM, servers and storage see a higher application consolidation in containers.

- Usage leveling in priority rules

  Balance the usage between storage and other resources.

- Thin provisioning

  Allocate a portion (ρ) of the request capacity initially and increase (µ) when the utilization reaches a threshold (θ).

- Multiplexing

  Monitor the average throughput over a time interval τ (e.g., six hours) for device j: $\overline{TP^j}$

  Literal bandwidth: B^j_{total}. Requested bandwidth: B^j_{req}

  Amplification factor α^j: $\frac{1}{\alpha^j} = \frac{\overline{TP^j}}{B^j_{req}}$ (cap to e.g., 120%)

  K8sES-scheduler schedules pods as if storage j has BW of $\alpha^j \cdot B^j_{total}$

Backup – Continuously Ensuring SLOs

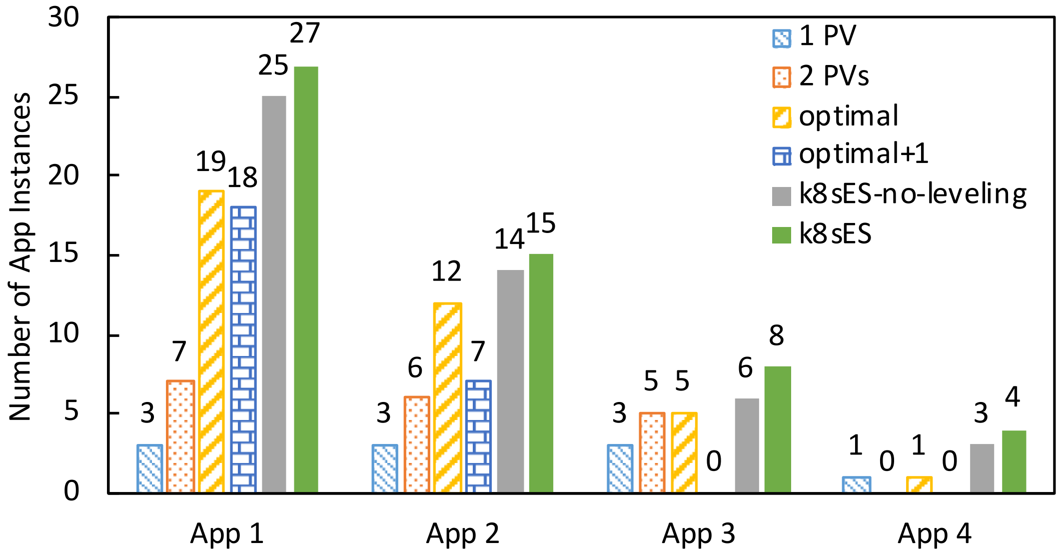
- We develop a Monitor in k8sES that monitors the I/O activities at granularities of pods and devices.
- We develop a Migrator that can migrate an application along with storage in case of SLO violation, or software failure on nodes and storage.
- The migration is triggered by the Monitor.

Backup – Evaluation Result (saving resources)

n PVs: divide each SC into n PVs evenly

optimal: the maximum number of instances that can be deployed if we evenly divide the SC.

k8sES-no-leveling: if we do not balance the usage between storage and other resources.



Backup – Design Challenges

- Global visibility of I/O stacks
  - Network and storage nodes are often remotely located and geographically distributed.
- Coordination between components
  - Interactions between components. Network knows I/O. Storage knows network stack.
  - Interpret data from network and storage for coordinated control.
- SLO aware
- Cost-effective control

Backup – Probe and Test

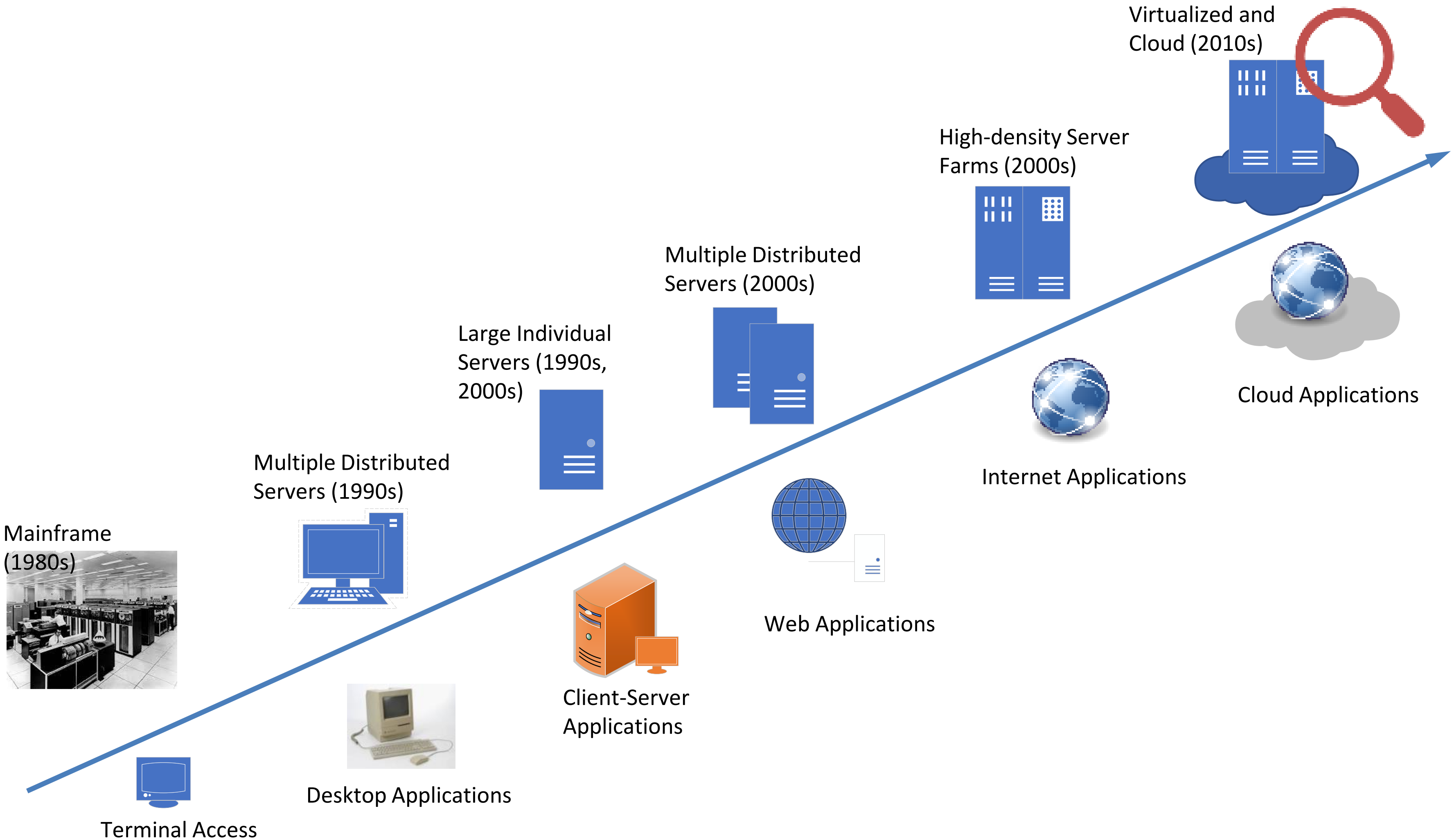
- Probe
  - Send out *Read, Write* and *Storage probes* periodically Eg., The read probe will collect T_{rq}^{r}, T_{rt}^{r}
- Estimate
  - Eg. For a *m* KB read request

$$t_{est}^{r} = t_{rq}^{r} + t_{rt}^{r} + t_{s}^{r} + \delta$$

- Test

$t_{est} < \beta D$	Not congested. Issue.
$\beta D < t_{est} < D$	Close to congested. Control.
$D < t_{est}$	Fully congested. Throttle.

Backup – Evolving Applications and Infrastructures



Backup – A Look at Virtualized and Cloud Infrastructure

Client Architecture

Internet

Cloud

Applications

Compute SVC

Network SVC

Storage SVC

What's the impact on data access performance?

Computation:

- Powerful Units
- Large Scale
- Virtualized (VM)
- Containerized

Network:

- Large (10K-100K switches)
- Software Defined
- On I/O path

Storage:

- Heterogeneous (HDD,SSD,SMR)
- High capacity
- Distributed

Backup – Impacts to Data Access Performance

- Data access in VM
  - Applications run in VMs. Data are stored in storage servers.
  - People can access data from anywhere at anytime.
  - How are storage allocated to support such access pattern?
  - What are the storage requirements for such applications?
- Data access in container
  - What is the current storage support for containerized applications?
  - How to allocate storage & manage storage based on users' requirements?
- Data access over network
  - The dynamic network results in long I/O path and increased end-to-end management complexity.
  - A systematic view of client, network and storage is essential to improve data access performance.