Optimizing Distributed Systems using Machine Learning

Ignacio (Nacho) Cano

Ph.D. Defense

Paul G. Allen School of Computer Science & Engineering
University of Washington

December 2018

Distributed Systems

- Multiple interconnected components
- Cooperate with each other to perform certain task(s)
- Components have well-defined interfaces
- Interested in their efficiency and performance

Example 1: Storage Tiering Services

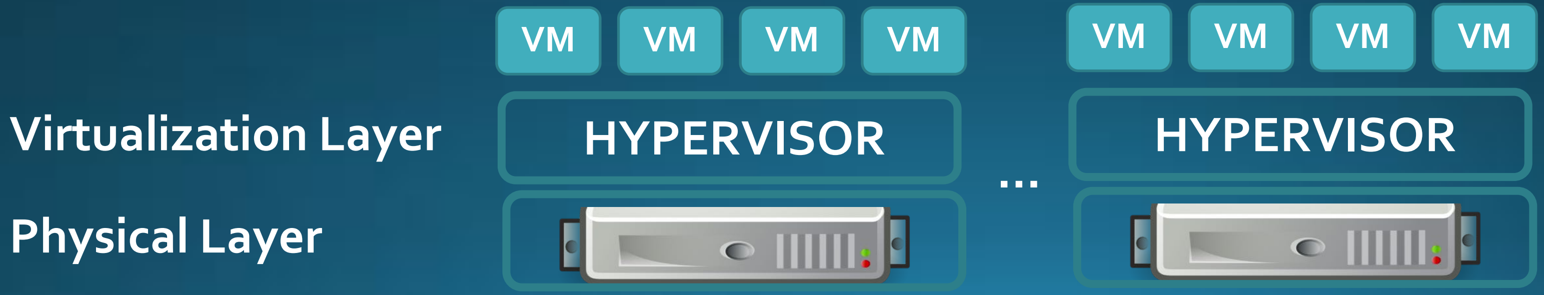
- Migrate data from SSDs to HDDs

Problems with Storage Tiering Services

- Rely on operator-defined thresholds for migrations
- Disregard workload characteristics

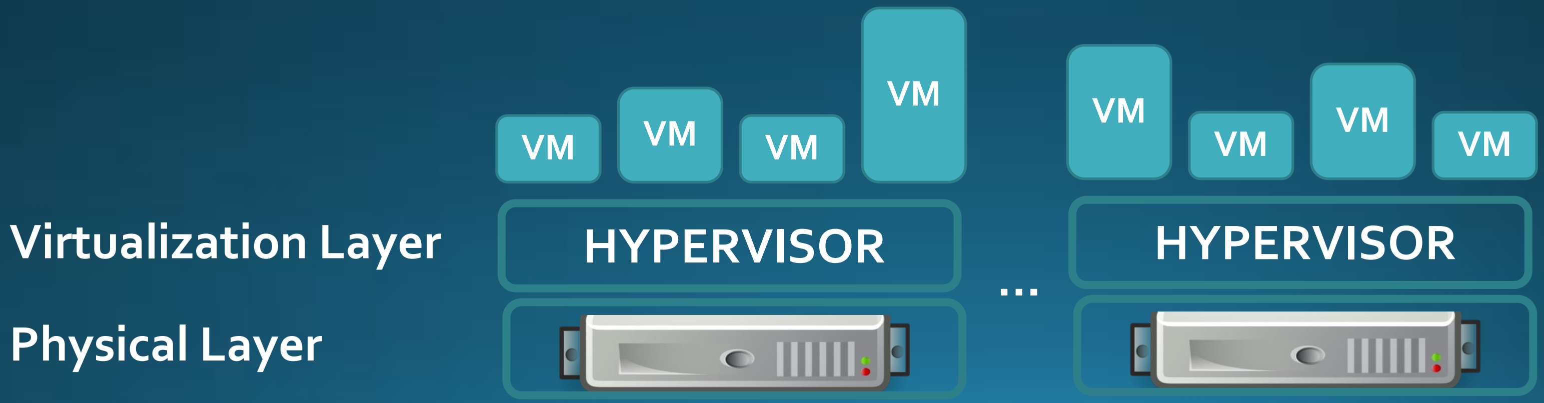
Example 2: Virtualization Software

- Package applications on VMs
- Execute them together with other workloads on same hardware



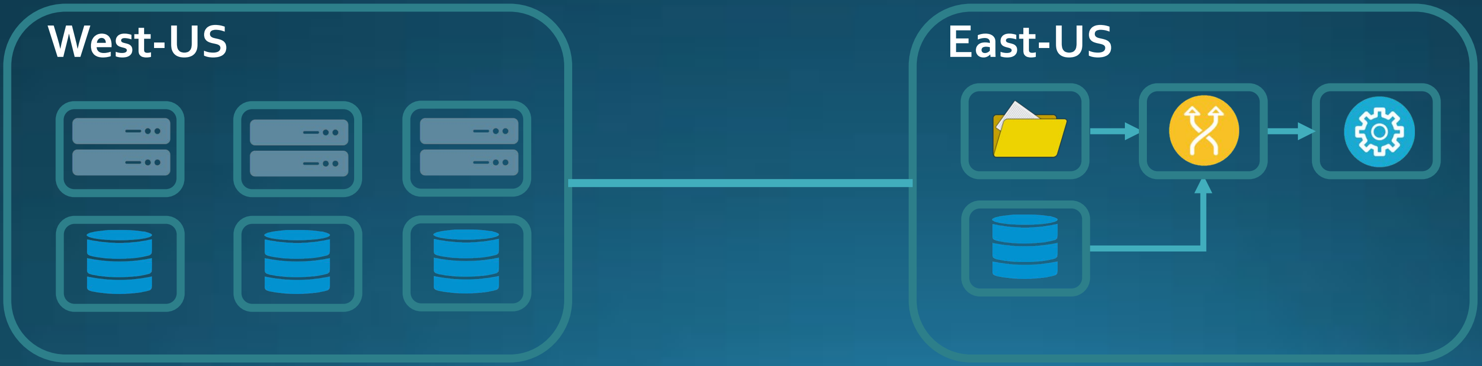
Problems with Virtualization Software

- Rely on static user-defined allocations (vCPUs, memory)
- Disregard workload temporal patterns



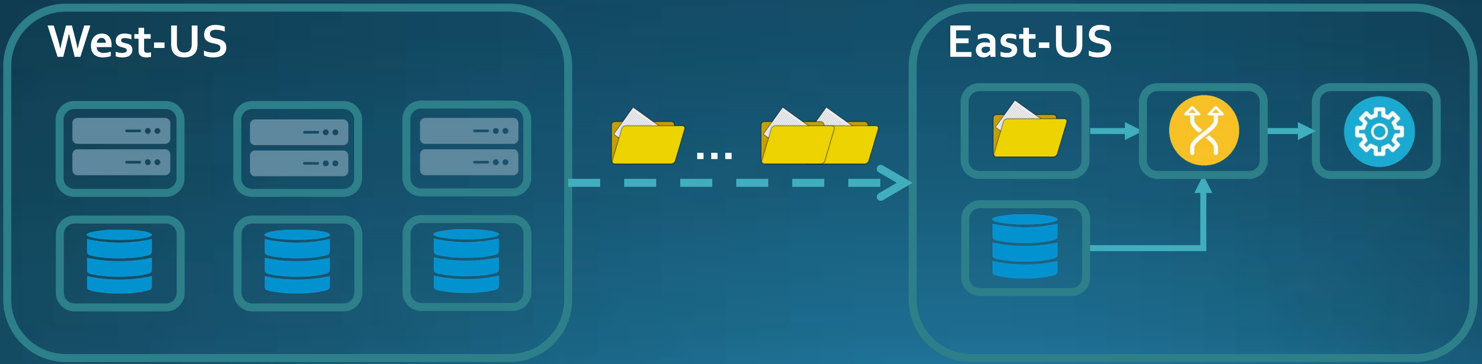
Example 3: Data Processing Systems

- Support pipelines to join and analyze disperse datasets



Problems with Data Processing Systems

- Transfer huge amounts of data
- Disregard that transferring data summaries may suffice



Distributed Systems typically rely on …

- Fixed configurations
- One-size-fits-all thresholds
- Hardcoded rules

Sub-optimal systems
efficiency and performance

What we actually want …

- ~~Fixed configurations~~
- ~~One-size-fits-all thresholds~~
- ~~Hardcoded rules~~

- Dynamic configurations
- Custom thresholds
- Learn rules

Improve systems efficiency and performance

Distributed Systems need to …

- Adapt to different runtime conditions
- Be tuned on a case-by-case basis at running time
- Leverage data and problem structure

Machine Learning to the rescue!

Optimizing Distributed Systems using Machine Learning

1. **CURATOR: ML-based policy** to schedule storage tasks
2. **ADARES: ML-based mechanism** to adjust VM resources
3. **PULPO: ML-System co-design** to train models from geo-distributed datasets

Outline

- Motivation
- Challenges
- CURATOR
- ADARES
- PULPO
- Conclusions

Outline

- Motivation
- **Challenges**
- CURATOR
- ADARES
- PULPO
- Conclusions

ML Challenges for Distributed Systems

1. Cold start
   - Speed up training
   - Minimize interactions with environment
2. Model setup
   - Collection of features
   - Quantify performance
3. Exploration and Interpretability
   - Maintain normal functioning
   - Insight to operators

ML Challenge 1: Cold Start

- Leverage **historical traces**
  - Pre-train models to accelerate training and reduce sample complexity
- Use **transfer learning** from simulations to real environments
  - Expose agents to relevant situations in advance

ML Challenge 2: Model Setup

- Create efficient **sensing** mechanisms
  - Cluster-level metrics
  - Node-level metrics
  - VM-level metrics
- Propose intuitive **reward** functions
  - High performance (e.g., low latency)
  - High efficiency (e.g., high CPU usage)

ML Challenge 3: Exploration and Interpretability

- Promote **safe online exploration**
  - Do unsafe exploration offline using simulators
  - Revise ML-based decisions with business constraints

- Leverage models that provide **uncertainty** in predictions
  - Better understanding of the decision-making process

Outline

- Motivation
- Challenges
- **CURATOR**
- ADARES
- PULPO
- Conclusions

Cluster Storage Systems

- Significant functionality
  - Automatic replication and recovery
  - Seamless integration of SSDs and HDDs
  - Snapshotting and reclamation of unnecessary data
- Much of functionality can be done in the background

Scheduling of these tasks is key to overall cluster performance

Curator

Framework and systems support for building background tasks

CHALLENGE

- Heterogeneity across and within clusters over time
  - Use **reinforcement learning** to schedule the background tasks

Curator

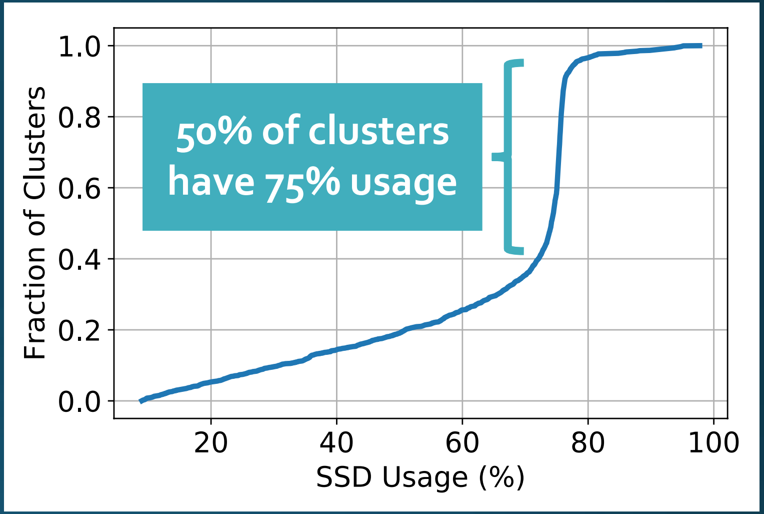
Framework and systems support for building background tasks

CHALLENGE

- **Heterogeneity across and within clusters over time**
  - Use **reinforcement learning** to schedule the background tasks

Tiering Task

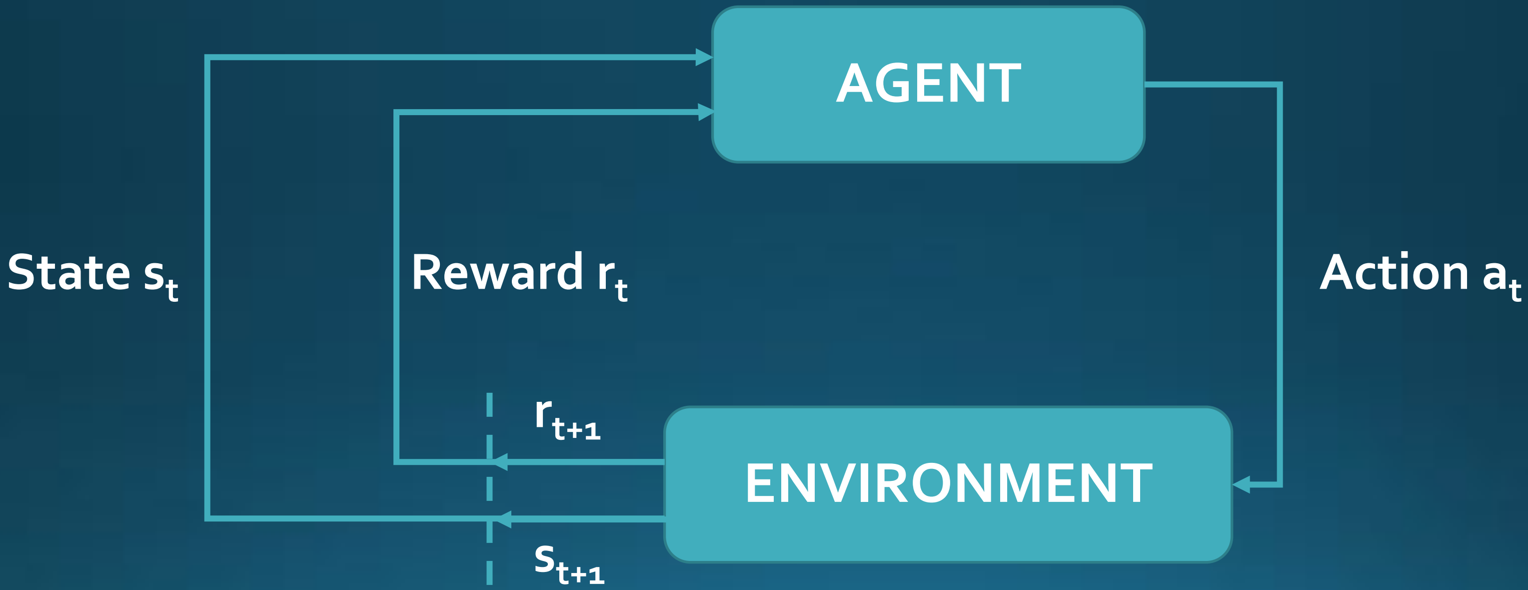
- Move cold data from a faster storage tier to a slower tier
- Maximize SSD effectiveness for both reads and writes in order to reduce latency
- Threshold-based policy to trigger the task



Many clusters waste 25% of fast storage

Need smarter scheduling policies

Reinforcement Learning (RL)



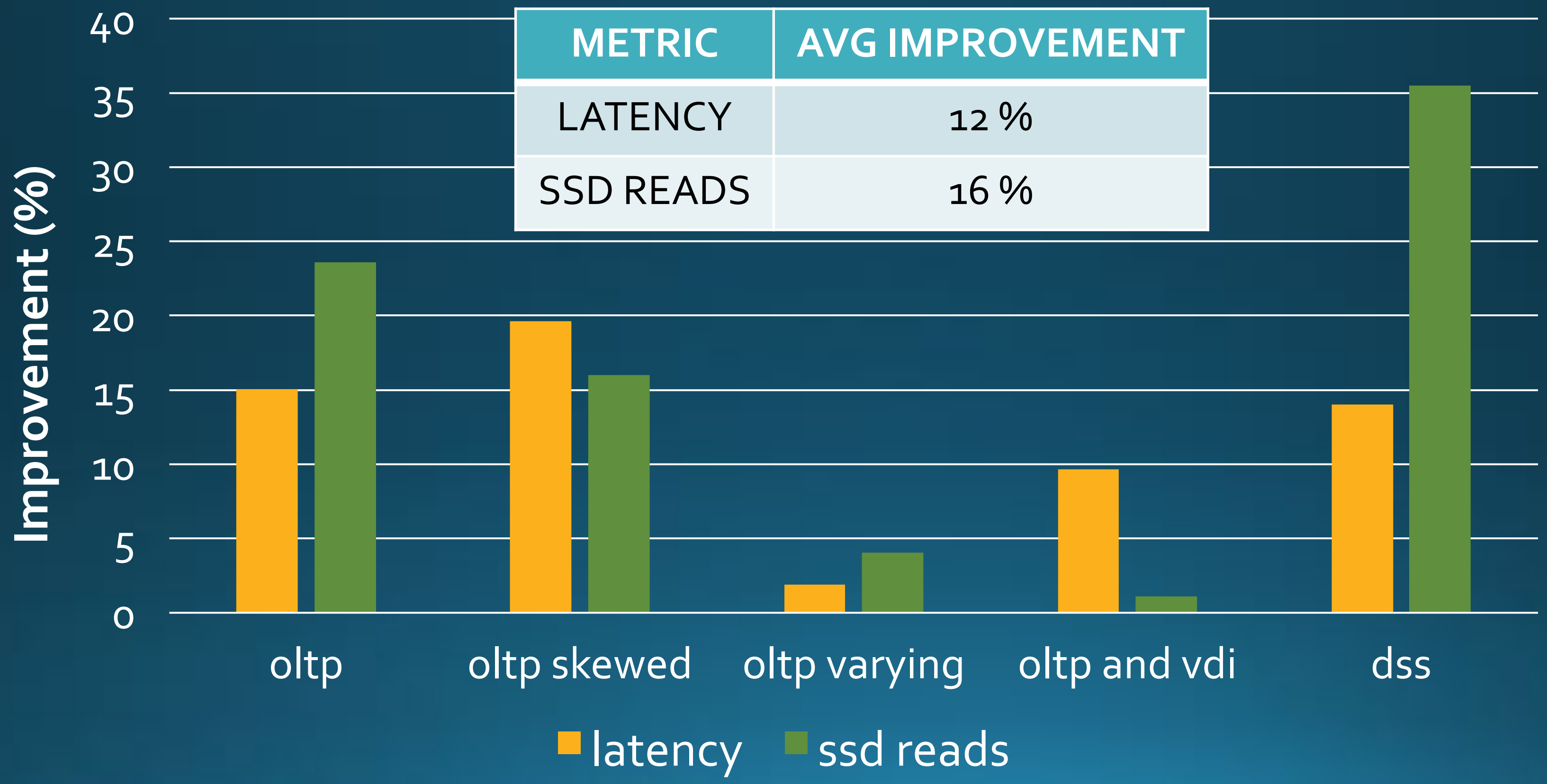
Reinforcement Learning for Tiering

- State: cluster-level features
  - Utilization: CPU, memory, SSD
  - Performance: read / write IOPS
- Actions: run, not run
- Reward: -1 * latency

- Pre-trained our agents with real traces from other clusters

Evaluation Results

METRIC	AVG IMPROVEMENT
LATENCY	12 %
SSD READS	16 %



Curator Summary

Framework and systems support for building background tasks

- Used **reinforcement learning** to schedule the tasks
  - Bootstrapped our agents with historical traces from real clusters
  - Results on Tiering showed up to ~20% latency improvements

Outline

- Motivation
- Challenges
- CURATOR
- **ADARES**
- PULPO
- Conclusions

Large-Scale Measurement Study

- 1-month trace from Nutanix clusters
  - 253k VMs
  - 17k nodes
  - 3.6k clusters

Measurement Findings

- Most VMs in enterprise clusters not sized appropriately
- Many clusters with both under and overprovisioned VMs
- Significant variation of utilization for VMs across time

Need a system that adaptively changes resources allocated to VMs in a cluster

AdaRes

Framework and systems support for adjusting VM resources on-the-fly, namely vCPUs and memory

CHALLENGES

- Adaptive and improve over time
  - Use **contextual bandits** to perform the adaptations
- Extensible, flexible, and scalable framework
  - Decompose architecture into **decoupled** and **highly configurable** components

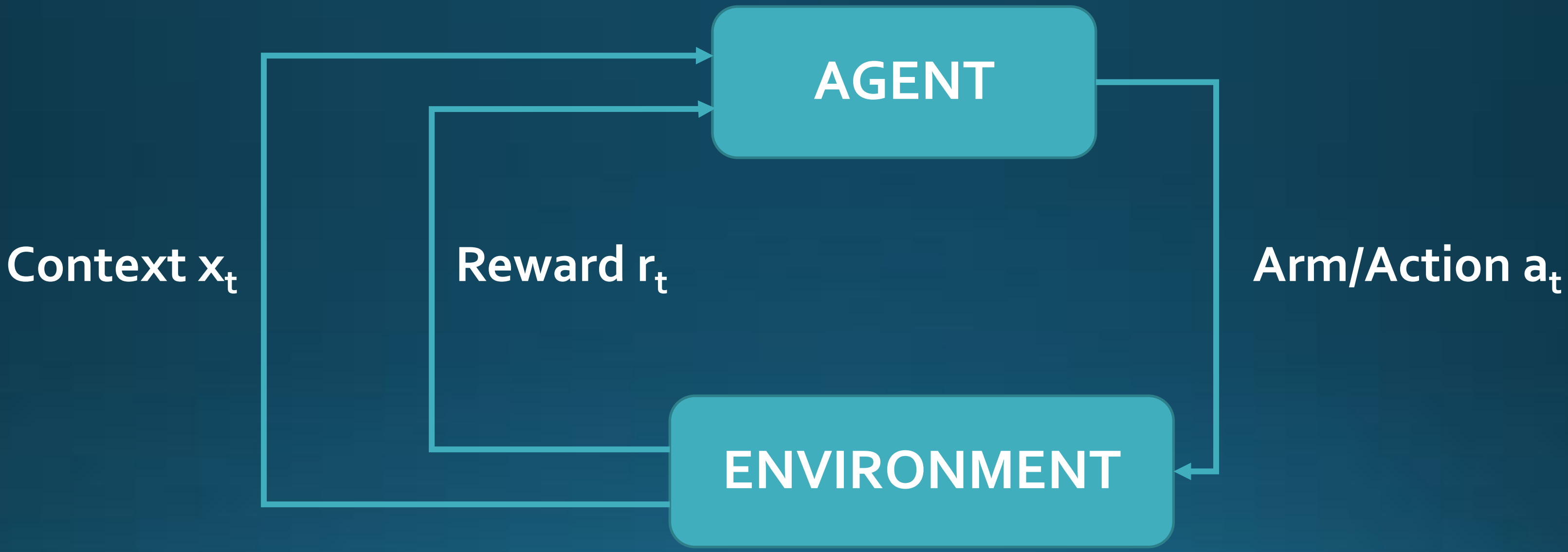
AdaRes

Framework and systems support for adjusting VM resources on-the-fly, namely vCPUs and memory

CHALLENGES

- **Adaptive and improve over time**
  - Use **contextual bandits** to perform the adaptations
- Extensible, flexible, and scalable framework
  - Decompose architecture into **decoupled** and **highly configurable** components

Contextual Bandits



Contextual Bandits for VM Resource Management

- Context: cluster, node, and VM-level features
  - Utilization: CPU / memory
  - Performance: latency, IOPS, swap rates, CPU ready times
- Arms/Actions: per resource type
  - Up / Down / Noop
- Reward: {0, 1} per resource type
  - 1: Move away from “bad” states, increase utilization
  - 0: Lead to “bad” or “worse” states, decrease utilization

- Pre-train our agents offline using simulators

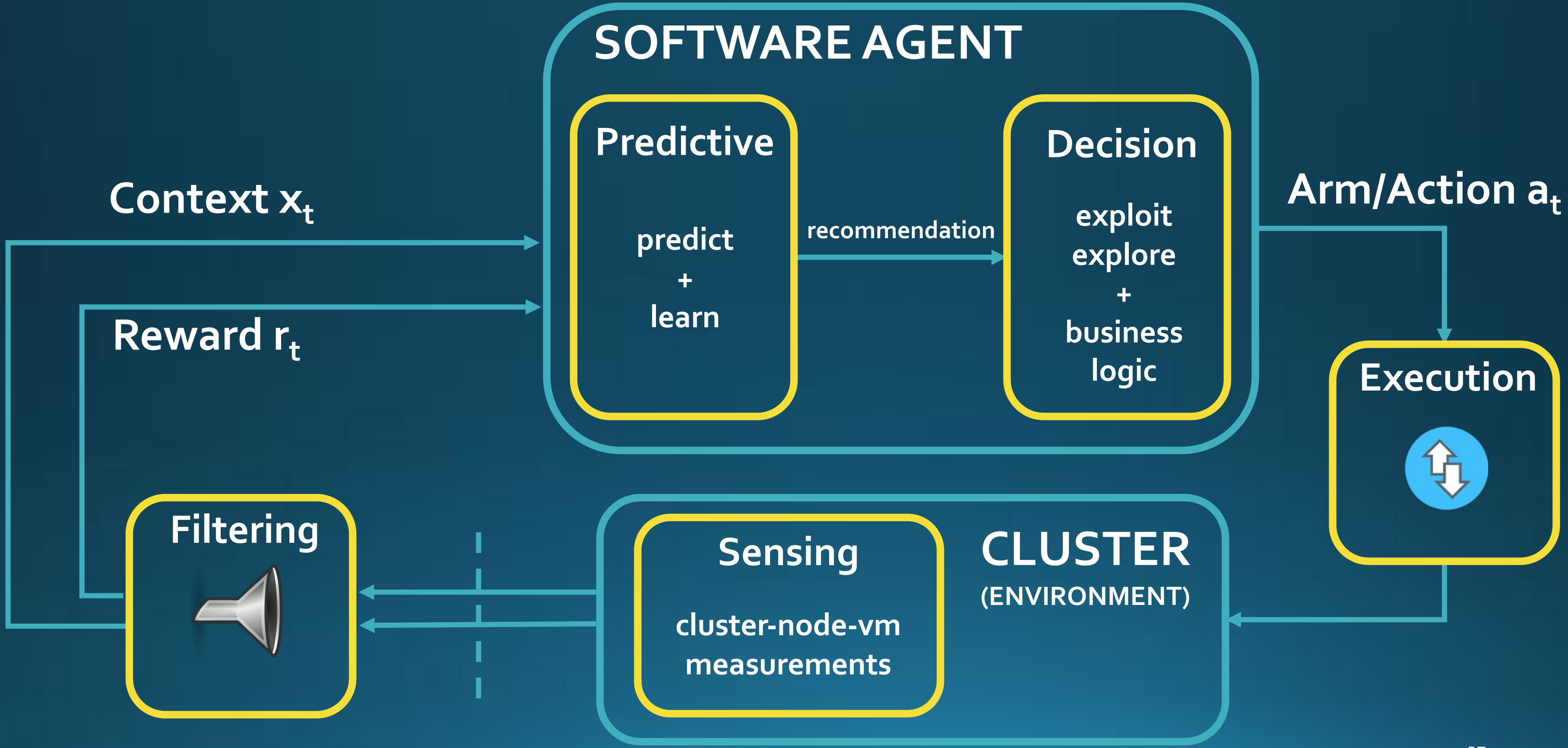
Challenges

- Adaptive and improve over time ✓
- Extensible, flexible, and scalable framework

Challenges

- Adaptive and improve over time ✓
- **Extensible, flexible, and scalable framework**

ADARES Services

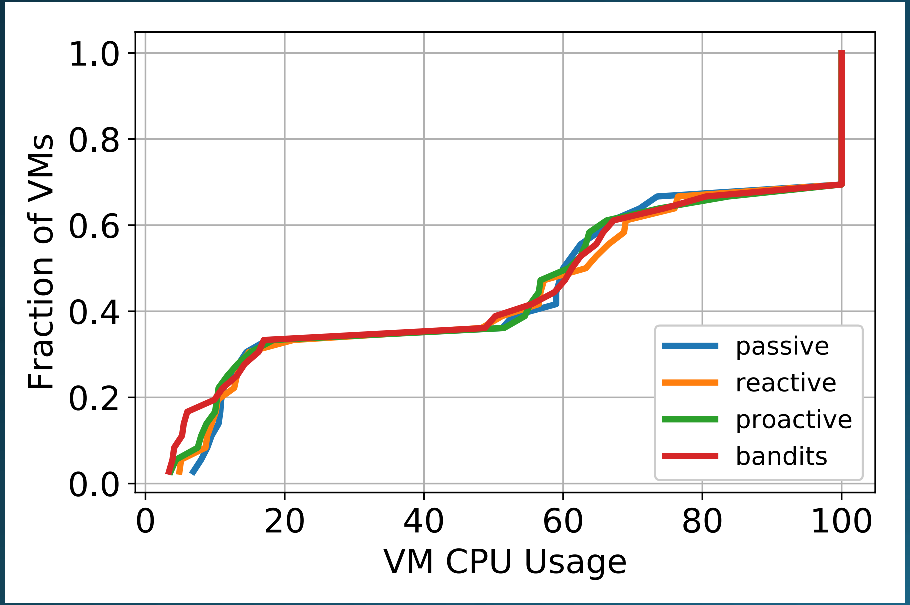


Methods

- **Passive:** no configuration changes
- **Reactive:** changes based on target thresholds using current resource utilization
- **Proactive:** changes based on target thresholds using predicted max resource utilization
- **Bandits:** adjusts resources using contextual bandits with model that provides uncertainty in predictions

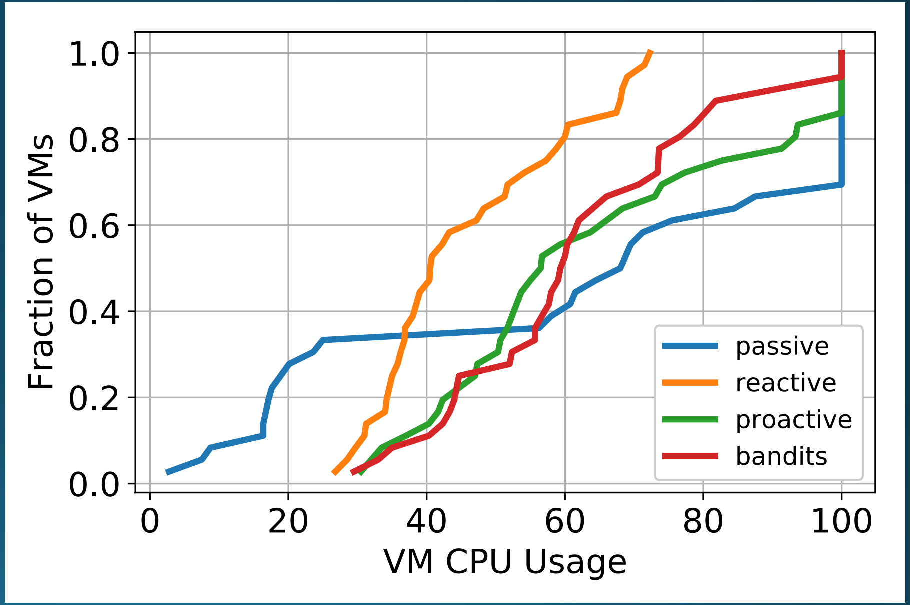
Resource Balancing

Start



Under and overprovisioning

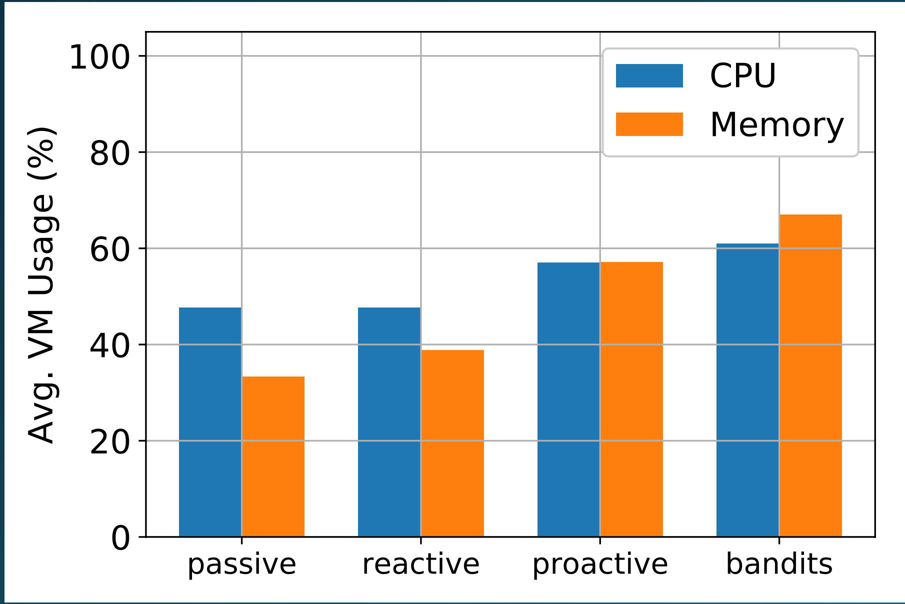
End



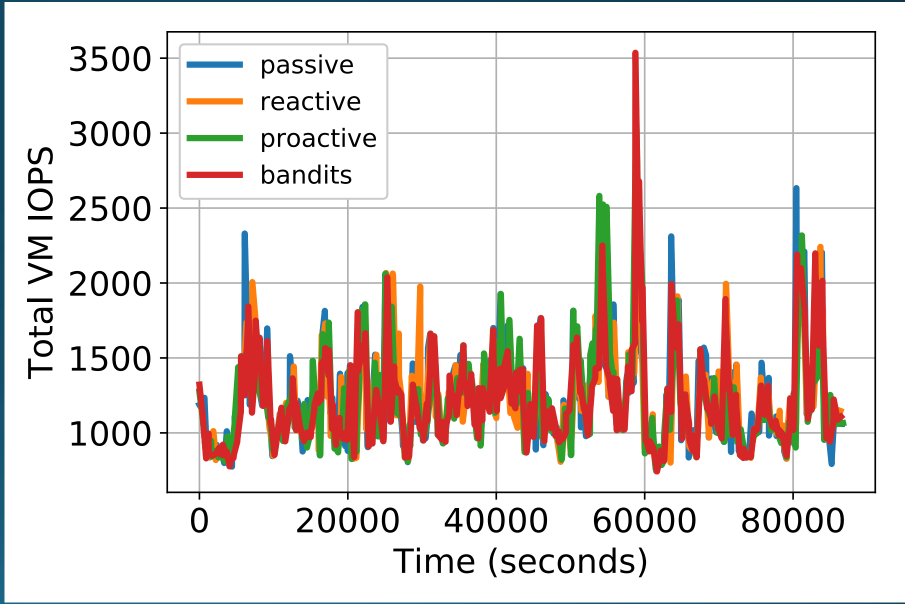
Less under and overprovisioning

Resource Utilization

Increases utilization



Keeps up with IOPS



AdaRes Summary

Framework and systems support for adjusting VM resources on-the-fly, namely vCPUs and memory

- Used **contextual bandits** to perform the VM adaptations
  - Leveraged transfer learning from simulations to real environment
  - Results showed allocation and utilization improvements over other baselines
- Decomposed architecture into **decoupled** and **highly configurable** components
  - Easily extensible and scalable, and agnostic to ML model

Outline

- Motivation
- Challenges
- CURATOR
- ADARES
- PULPO
- Conclusions

Geo-Distributed Machine Learning (GDML)

- Data generated and stored in data centers around the world
  - Minimize latency between serving infra and end-users
  - Respect regulatory constraints
- Machine learning apps require global view
  - Fraud Prevention
  - Recommender Systems

Previous Solutions: Centralized

1. Copy all the data partitions into one data center
2. Training takes places intra-data center (in-DC)

Problems with Centralized

- High cross-data center (X-DC) bandwidth consumption
- Privacy and data sovereignty regulations

Need a system to efficiently train ML models from geo-distributed datasets

PulPo

Framework and systems support for geo-distributed training

CHALLENGES

- Reduce communication between data centers
  - **Trade-off in-DC computation and communication** with **X-DC communication**
- Extensible, flexible, and scalable framework
  - Use/extend **Apache Hadoop YARN** and **Apache REEF**

PuLPo

Framework and systems support for geo-distributed training

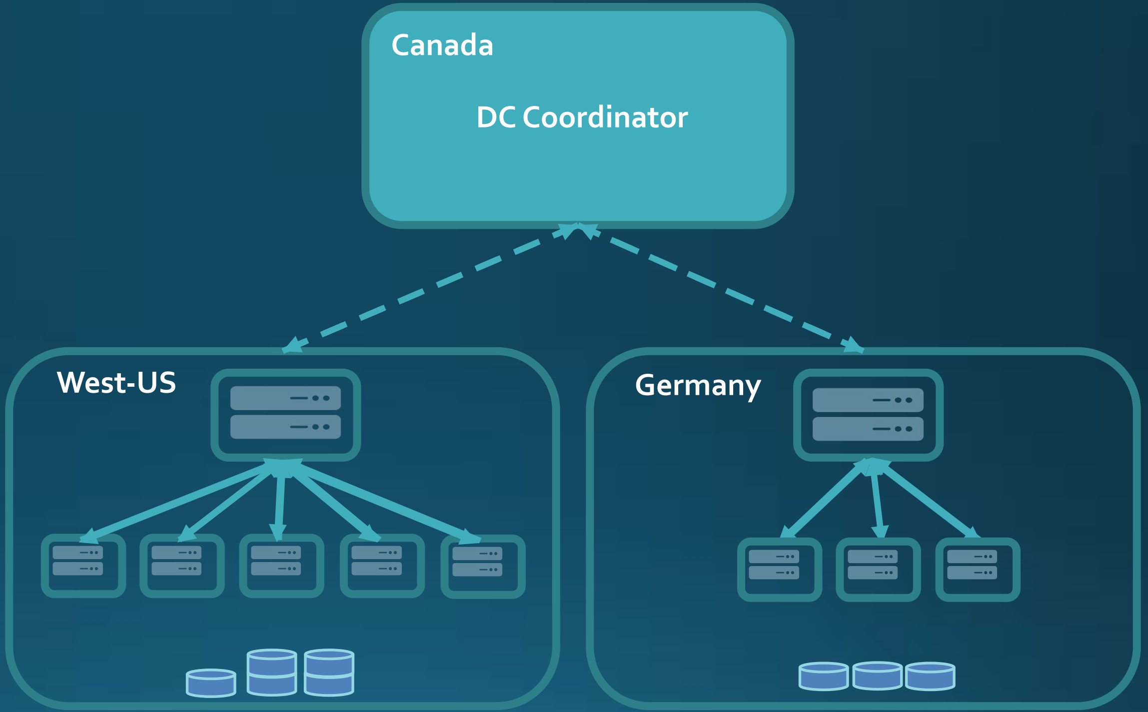
CHALLENGES

- **Reduce communication between data centers**
  - **Trade-off in-DC computation and communication** with **X-DC communication**
- Extensible, flexible, and scalable framework
  - Use/extend **Apache Hadoop YARN** and **Apache REEF**

Communication-Efficient Algorithm

FADL [Mahajan et al., JMLR '15]

- Initialize *global state*
- Send *global state*
- DCs compute a *local state*
- Send *local state*
- Aggregate *local states*
- Negligible DC computation
- Update *global state*



More in-DC cmp. and comm.
Less X-DC communication

Challenges

- Reduce communication between data centers ✓
- Extensible, flexible, and scalable framework

Challenges

- Reduce communication between data centers ✓
- **Extensible, flexible, and scalable framework**

PuLPo Architecture

Application Layer (DML/GDML)	• Multi-level communication trees across DCs • Learning algorithms in terms of B/R primitives
Apache REEF	• Generalized control plane • Data aggregation, communication, etc.
Apache Hadoop YARN	• Federated version • Single massive YARN cluster • Network-aware resource requests

Evaluation Setup

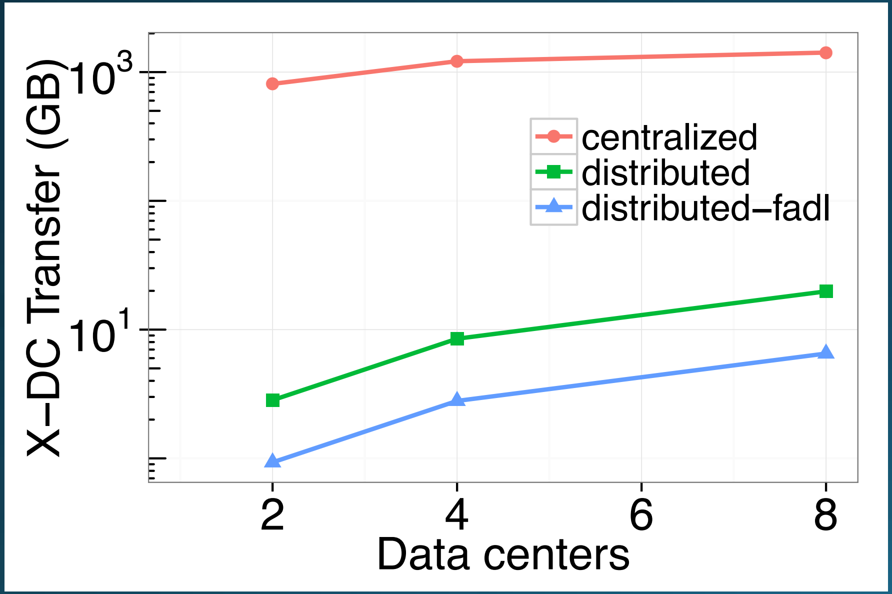
- Logistic Regression with L2 regularization
- Data randomly distributed
- Click-through rate datasets (CRITEO and KAGGLE)
- Simulation (2, 4, 8 data centers)
- Real Setup (West US and West Europe)

Methods

- **Centralized**: trains in-DC
  - **Bulk**: batch replication scheme (copy time included)
  - **Stream**: streaming copy model (copy time not included)
- **Distributed**: trains X-DC w/o comm-efficient algorithm
- **Distributed-Fadl**: trains X-DC with comm-efficient algo

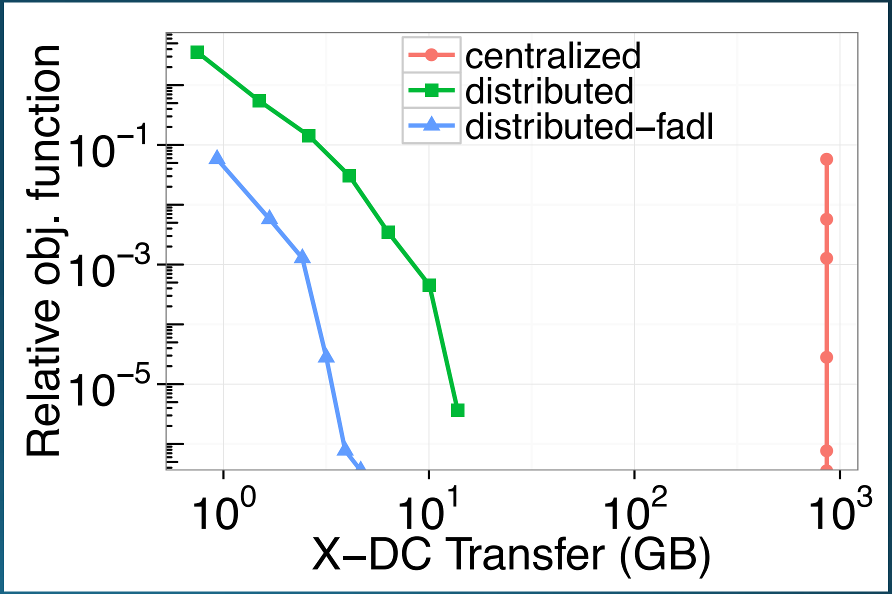
X-DC Transfer (Simulation)

CRITEO 10M



Orders of magnitude reduction

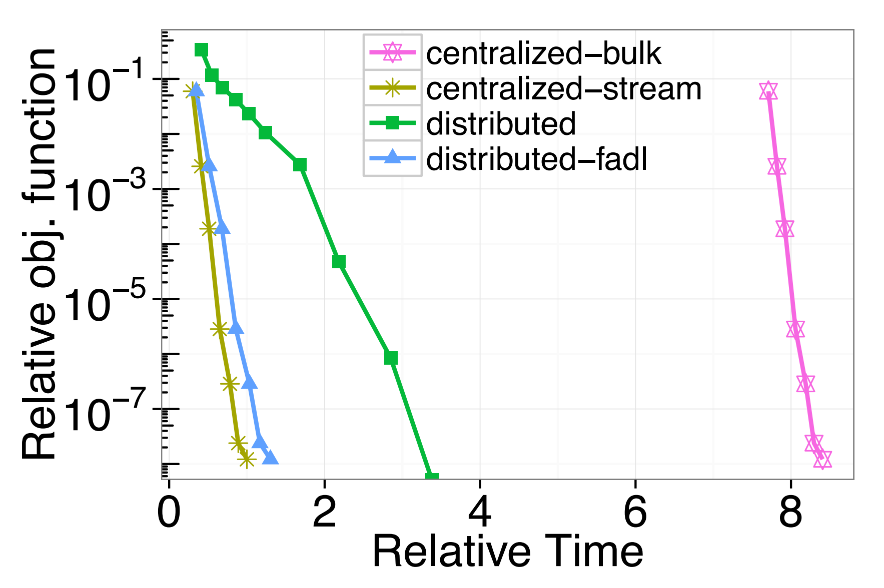
CRITEO 50M – 2DC



Better models sooner

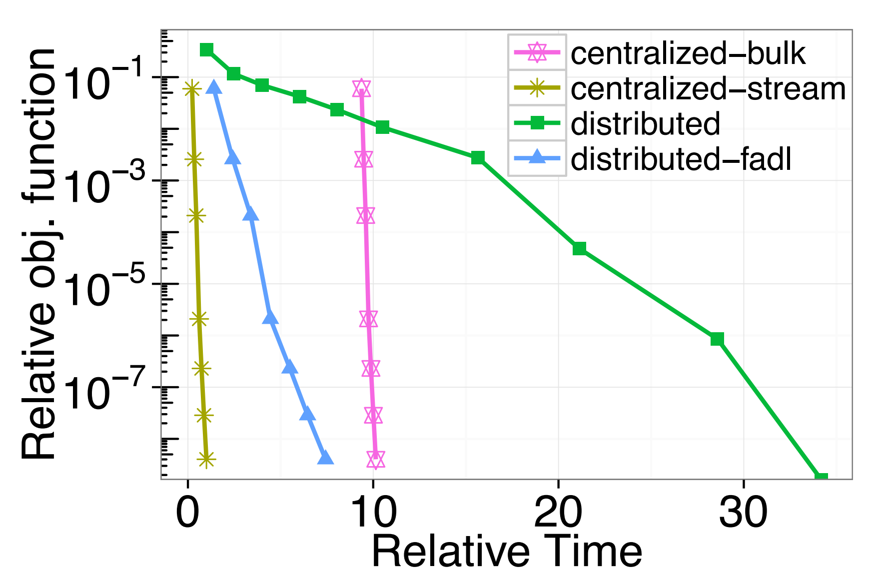
Running Time (Real Azure Setup)

KAGGLE 500K



Close to optimal in low dimensional models

KAGGLE 5M



Deteriorates with model size

PuLPo

Framework and systems support for geo-distributed training

- Traded-off **in-DC comp. and comm.** with **X-DC communication**
  - Reduced WAN bandwidth consumption while achieving same accuracy results
- Used/extended **Apache Hadoop YARN** and **Apache REEF**
  - Single job across data centers
  - Network-aware placement of tasks
  - Requires algorithm to be expressed in terms of B/R primitives

Outline

- Motivation
- Challenges
- CURATOR
- ADARES
- PULPO
- **Conclusions**

Conclusions

We can use Machine Learning to optimize Distributed Systems

- ML-based Policy
  - **CURATOR**, framework and systems support for building background maintenance tasks
  - **RL-based scheduling** showed performance improvements over a threshold-based approach
- ML-based Mechanism
  - **ADARES**, framework and systems support for adjusting VM resources on-the-fly
  - **Contextual bandits-based adjustments** showed more efficient resource allocations compared to other baselines
- ML-System Co-Design
  - **PULPO**, framework and systems support for efficiently training geo-distributed ML models
  - **Co-designed ML-System** solution showed orders of magnitude savings in terms of X-DC bandwidth utilization compared to other approaches

Thanks!

Special Thanks!

Conclusions

We can use Machine Learning to optimize Distributed Systems

- ML-based Policy
  - **CURATOR**, framework and systems support for building background maintenance tasks
  - **RL-based scheduling** showed performance improvements over a threshold-based approach
- ML-based Mechanism
  - **ADARES**, framework and systems support for adjusting VM resources on-the-fly
  - **Contextual bandits-based adjustments** showed more efficient resource allocations compared to other baselines
- ML-System Co-Design
  - **PULPO**, framework and systems support for efficiently training geo-distributed ML models
  - **Co-designed ML-System** solution showed orders of magnitude savings in terms of X-DC bandwidth utilization compared to other approaches

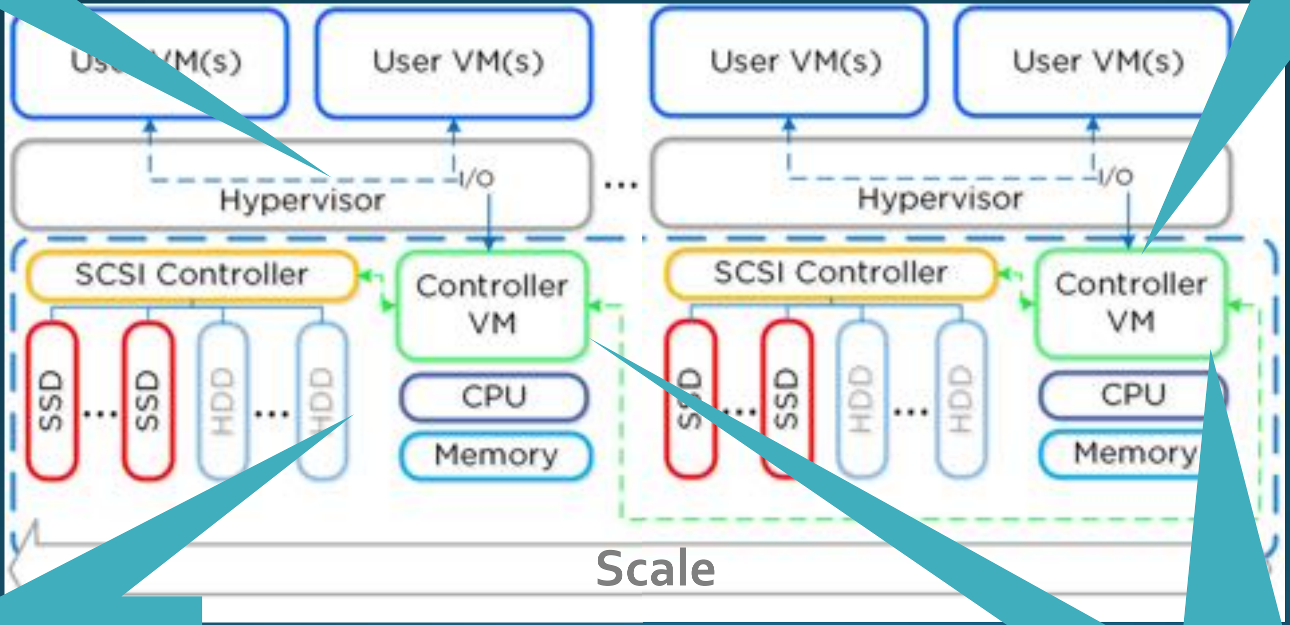
Backup Slides

CURATOR Backup

Nutanix Clusters

Operations interposed at the hypervisor level and redirected to CVMs

Data replication
Disk balancing
VM Migration

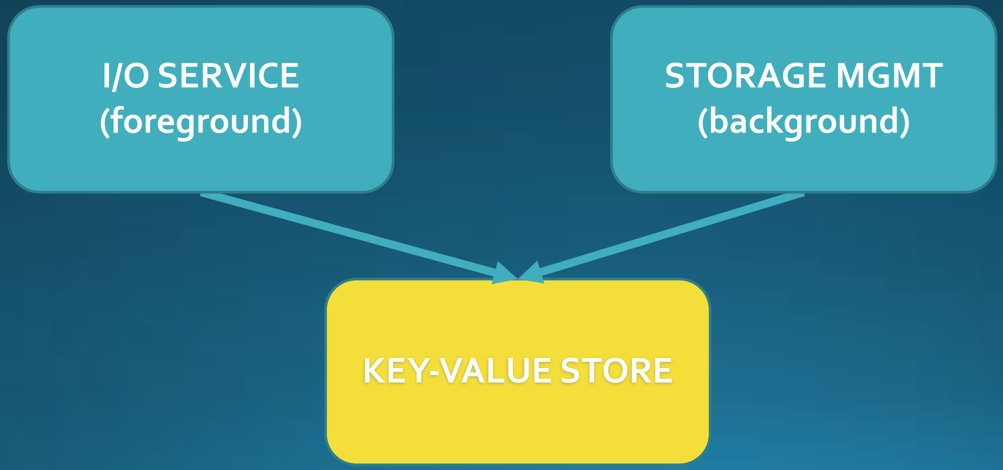


Integrated Compute-Storage

Global view of cluster state

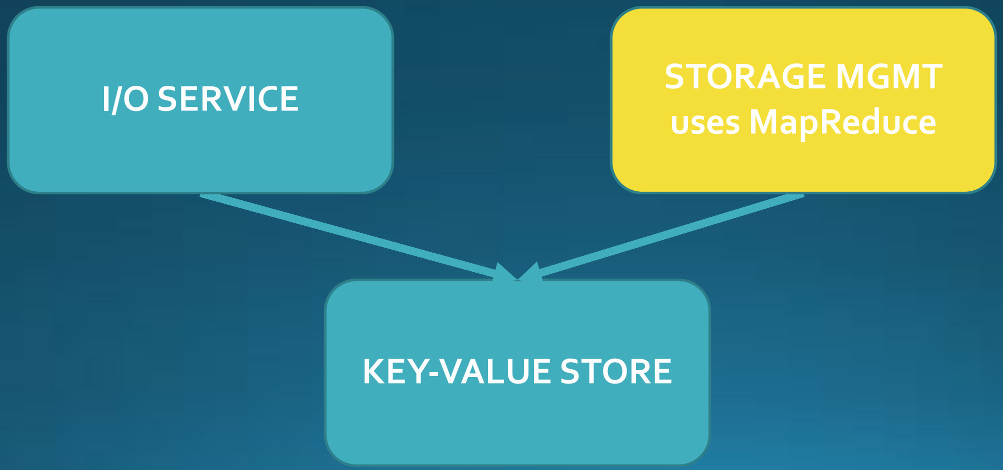
Distributed Key-Value Store

- Metadata for the entire storage system stored in k-v store
- Foreground I/O and background tasks coordinate using the k-v store
- Key-value store supports replication and consistency using Paxos



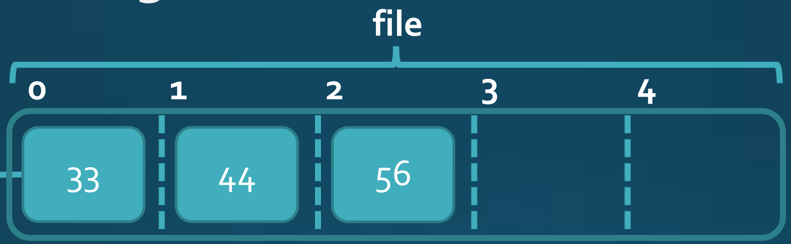
MapReduce Framework

- Globally distributed maps processed using MapReduce
- System-wide knowledge of metadata used to perform various self-managing tasks



Data Structures and Metadata Maps

- Data stored in units called **extents**
- Extents are grouped together and stored as **extent groups** on physical devices



Extent Id Map

eid	egid
33	1984
44	1984
56	2010

Extent Group Id Map

egid	physical location
1984	disk1,disk2
...	...

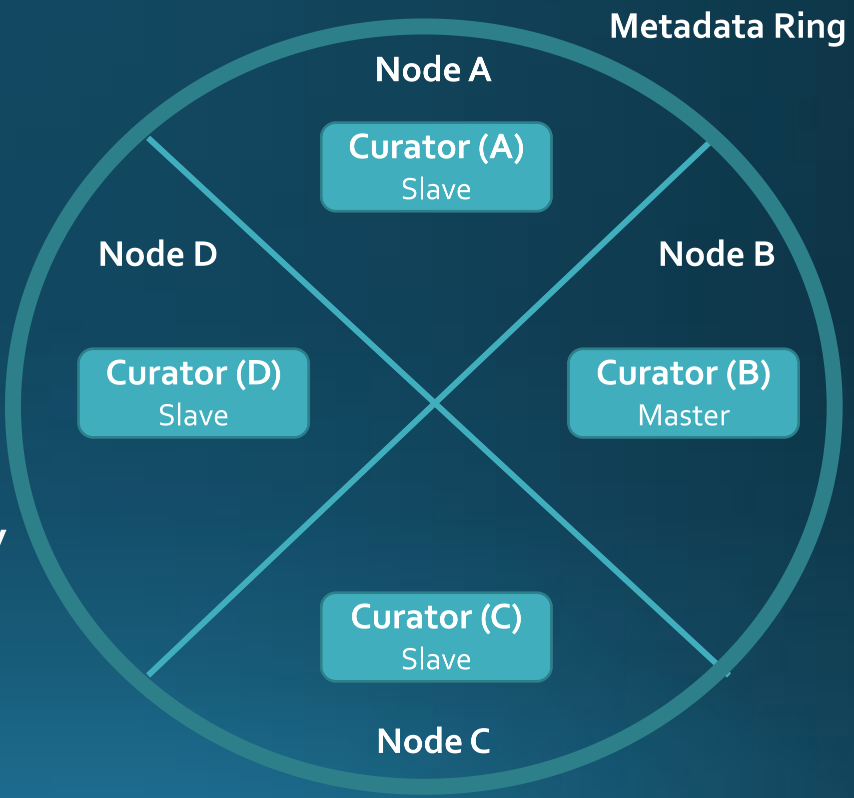
- Multiple levels of redirection simplifies data sharing across files and helps with minimizing map updates

Example: Tiering

- Move cold data from fast (SSD) to slow storage (HDD, Cloud)
- Identify cold data using a MapReduce job
  - Modified Time (mtime): Extent Group Id map
  - Access Time (atime): Extent Group Id Access Map

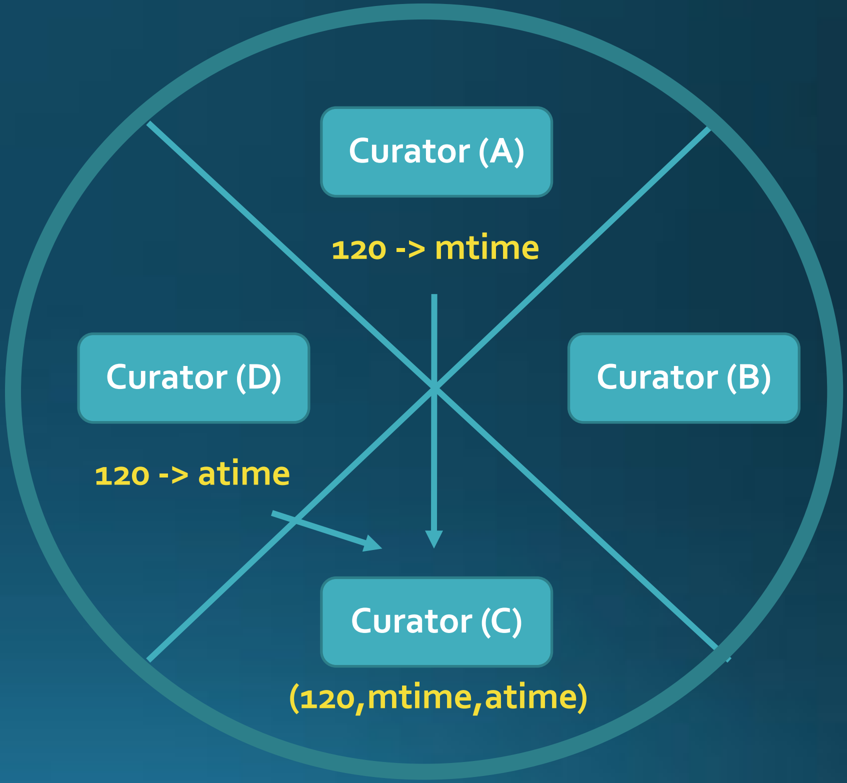
Example: Tiering

- egid 120
  - mtime owned by Node A
  - atime owned by Node D
- egid 120 == "cold" ?
  - Maps globally distributed
    → not a local decision
- Use MapReduce to perform a "join"



Example: Tiering

- Map phase
  - Scan both metadata maps
  - Emit egid -> mtime or atime
  - Partition using egid
- Reduce phase
  - Reduce based on egid
    - Generate tuples (egid,mtime,atime)
  - Sort locally and identify the cold egroups



Tiering Modeling Constraints

- Wide heterogeneity of clusters and workloads
- Variability of resource demands over time
- Don't know *what would have happened* had we made a different decision, need to try things out
- Decisions may impact performance over a long horizon
- Delayed feedback

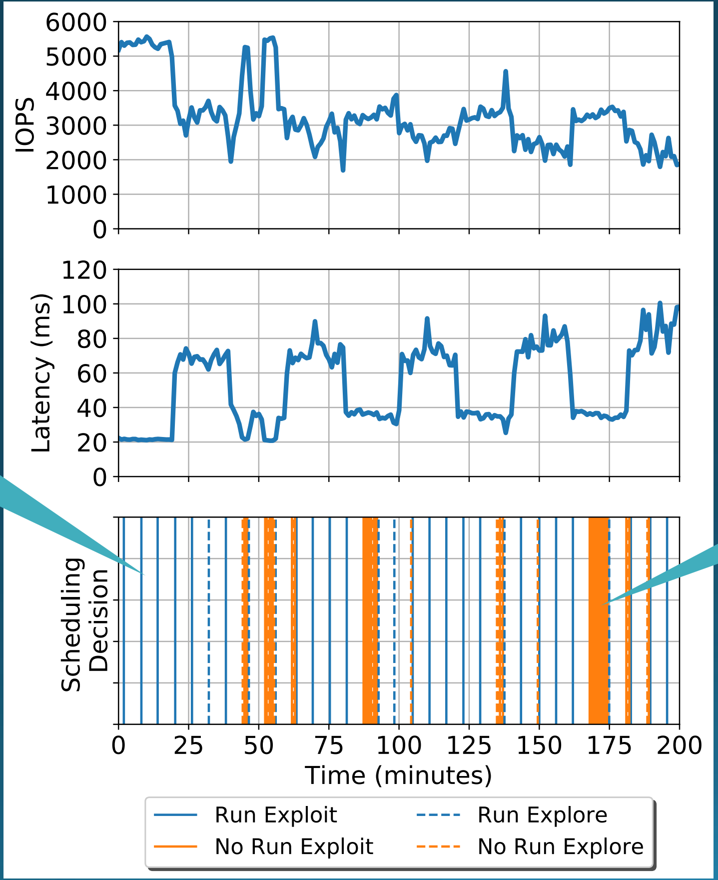
Q-Learning: RL algorithm

$$Q(s_t, a_t) = \underbrace{(1-\alpha)\underbrace{Q(s_t, a_t)}_{\text{Old Value}} + \overbrace{\alpha(r_{t+1} + \underbrace{\gamma max_a Q(s_{t+1}, a)}_{\text{Estimate of optimal future value}})}^{\text{Learned Value}}}$$

$0 \leq \alpha \leq 1$ **Learning Rate**

$0 \leq \gamma \leq 1$ **Discount Factor**

Scheduling Decisions with RL



run although cluster highly utilized and low latency

LEARNED not to run in those cases

Related Work (non-exhaustive)

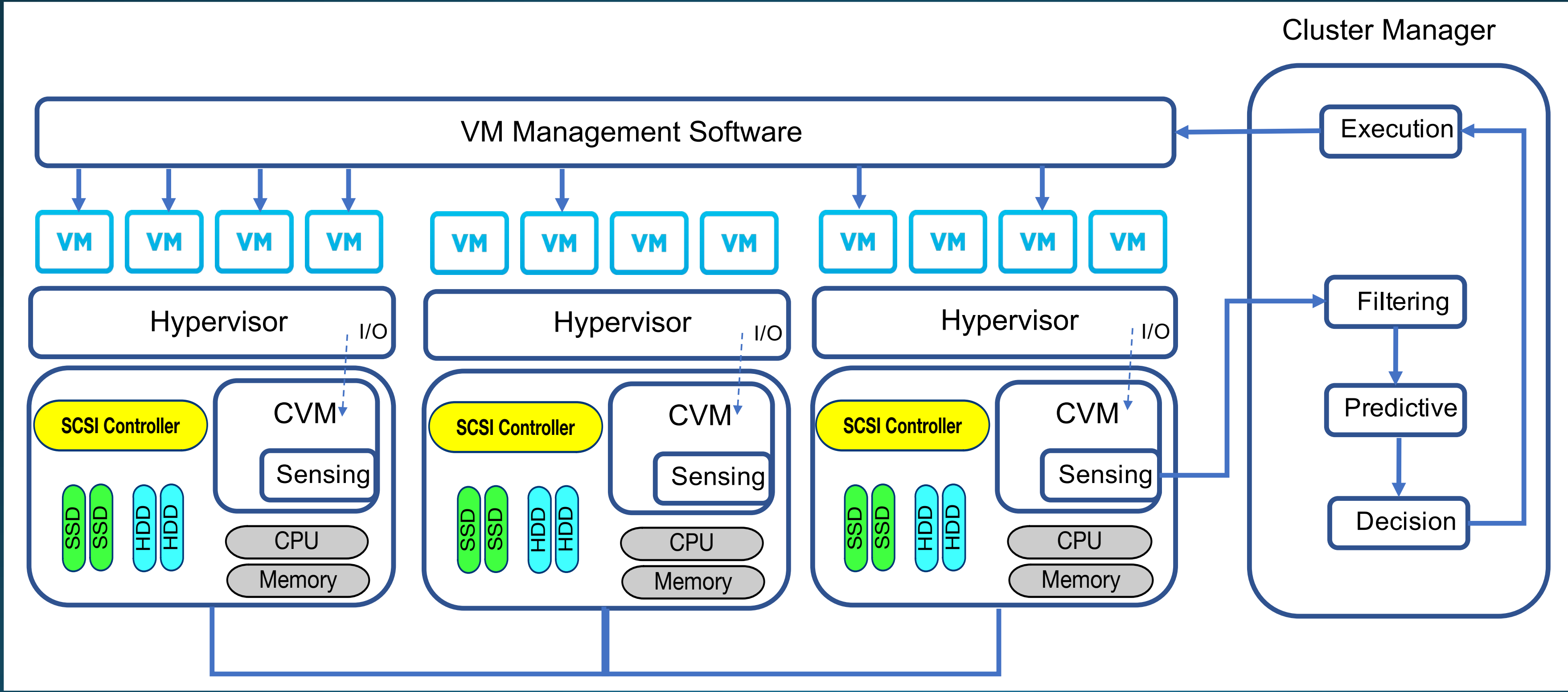
- Ipek et al. [ISCA '08]: RL-based memory scheduler to decide which DRAM command to perform in the next cycle (precharge, activate, read, write)
- Eastep et al. [ICAC '10] - SmartLocks: uses RL to decide which waiter process will get the lock next for the best long-term effect
- Prashanth et al. [IEEE TITS ' 11]: RL-based controller for scheduling traffic control signals
- Mao et al. [HotNets '16] - DeepRM: RL-based scheduler of jobs in a cluster
- Chinchali et al. [AAAI '18] - RL-based scheduler to determine the traffic rate for IoT data in mobile networks

AdaRes Backup

Why Contextual Bandits?

- VM workloads change frequently
- Incoming VMs don’t have records at all
- Learning task should estimate the result of making a resource adjustment
- Don’t know *what would have happened* had we done a different change, need to try things out
- Immediate feedback

System Architecture



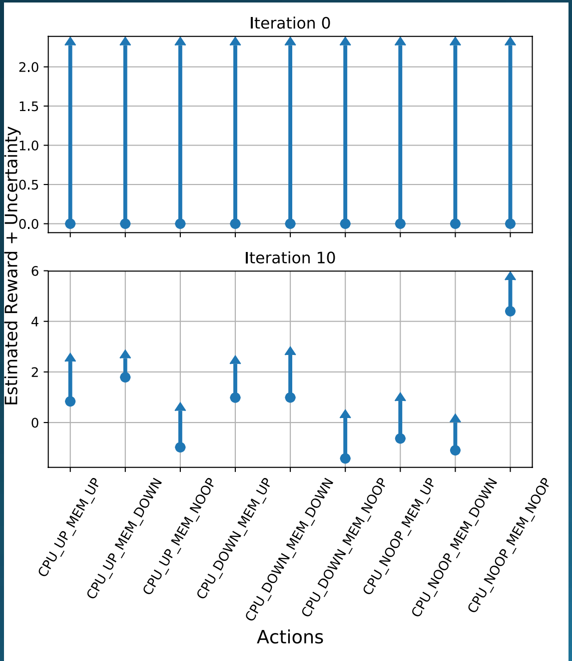
Transfer Learning: Simulation to Real

- Requirements
  - Reasonable emulation of the dynamics of the cluster
  - Simplistic analytical models to obtain $\mathbf{x}_t$ and $\mathbf{r}_t$
- Challenges
  - Large # of components and connections
  - Complex dependencies, irregular interactions
- Data-driven approach
  - Controlled experiments in real clusters where we perform VM configuration changes and record their impact
  - I/O benchmarks (rr, rw, rrw, sequential) to profile IOPS and latencies

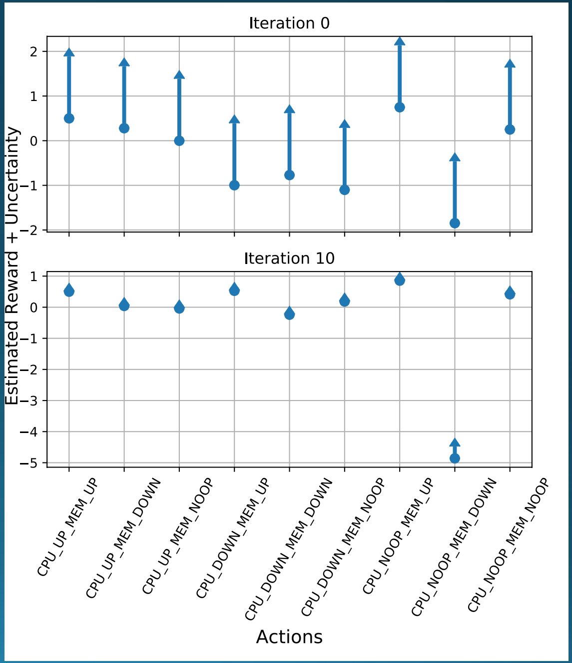
Transfer Learning Results

Low memory context

w/o transfer learning
mem noop



with transfer learning
mem up



Related Work (non-exhaustive)

- Auto-scaling systems (AWS, GCP)
  - Scale out/in based on target utilization metrics, i.e., thresholds
  - No vertical scaling but they do sizing recommendations
- Vasic et al. [ASPLOS '12] - DejaVu: predictable workloads, clustering to identify workload categories
- Bu et al. [IEEE TPDS '12]: CoTuner: RL to change VM limits in the hypervisor
- Delimitrou et al. [ASPLOS '13] - Paragon: online workload profiling and classification using collaborative filtering
- Venkataraman et al. [NSDI '16] - Ernest: Predictable structure of jobs to predict runtime and assign right hardware configuration
- Yadwadkar et al. [SoCC '17] - RF to identify best VM across cloud providers
- Cortez et al. [SOSP '17] - Resource Central: assignment of VMs to servers

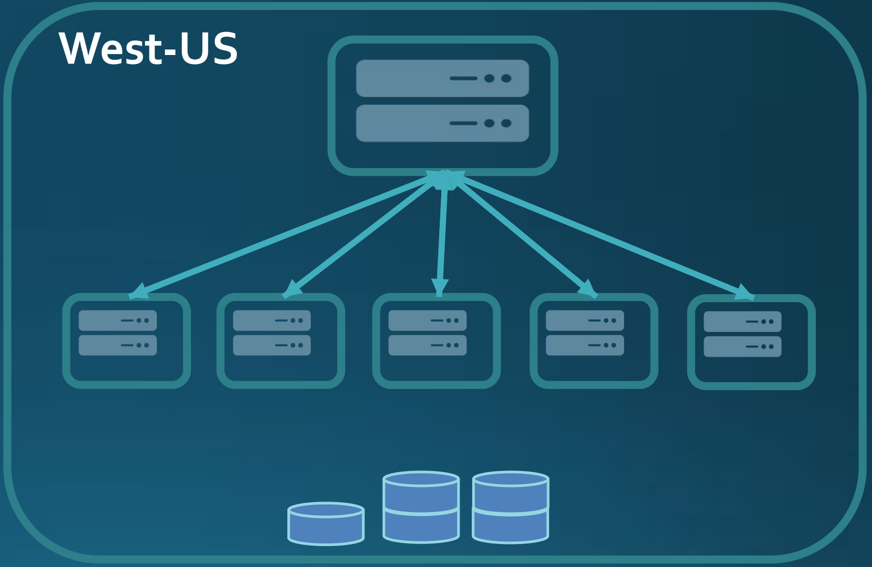
AdaRes Extensions

- More comprehensive evaluations (e.g., real workloads, sensitivity analyzes of thresholds)
- More measurement sensors (e.g., application-level metrics)
- Control other type of resources (e.g., storage, networking)
- Manage containers
- …

PULPO Backup

Distributed Machine Learning (DML)

- Dataset partitioned among workers
- Training proceeds in comm. rounds
- Server node sends algorithm "state"
- Workers perform computations based on the received "state" and their shard of the dataset
- Workers send update back to server
- Server applies the updates to the "state" and process repeats



More computation
Less communication

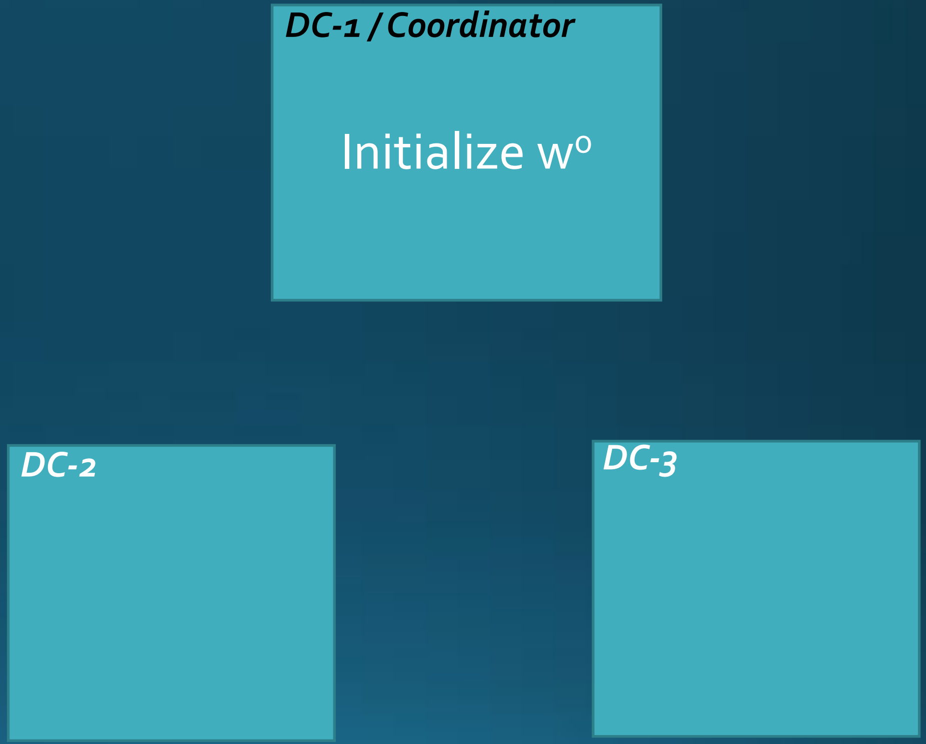
FADL [Mahajan et al., JMLR '15]

Algorithm

Choose w^0
for $r = 0, 1...$ **do**
  Compute g^r (X-DC communication)
  Exit if $||g^r|| \leq \epsilon_g ||g^0||$
  for $p = 1, ..., P$ (in parallel) **do**
    Construct $\hat{f}_p(w)$
    $w_p \leftarrow$ Optimize $\hat{f}_p(w)$ (in-DC communication)
  end for
  $d^r \leftarrow \frac{1}{P} \sum_p w_p - w^r$ (X-DC communication)
  Line Search to find t (negligible X-DC communication)
  $w^{r+1} \leftarrow w^r + t\, d^r$
end for

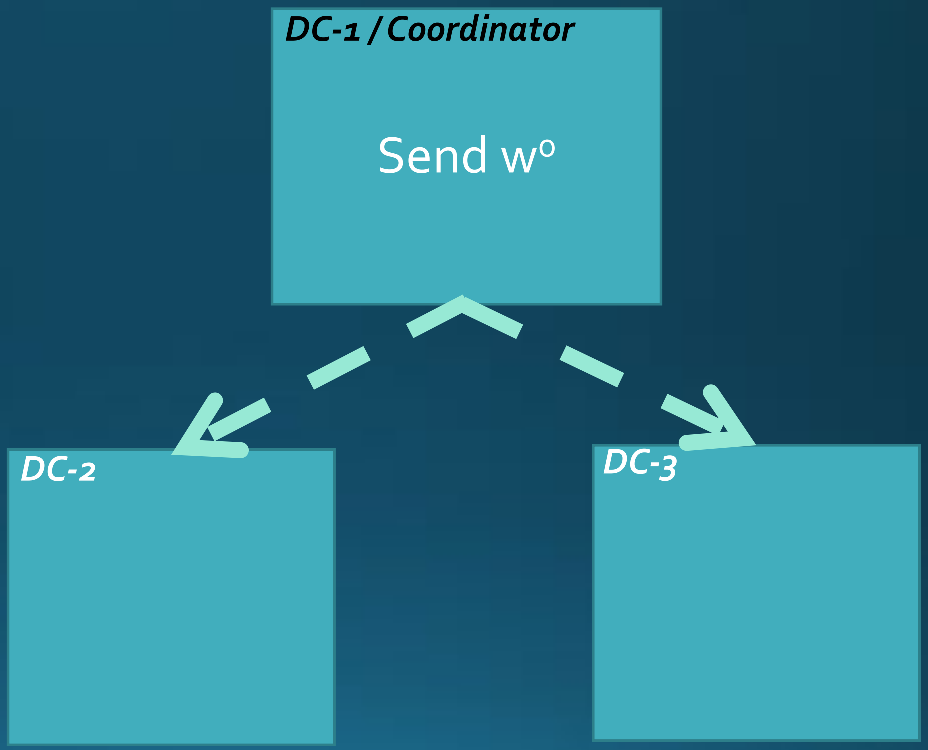
Algorithm

1. Initialize w^0



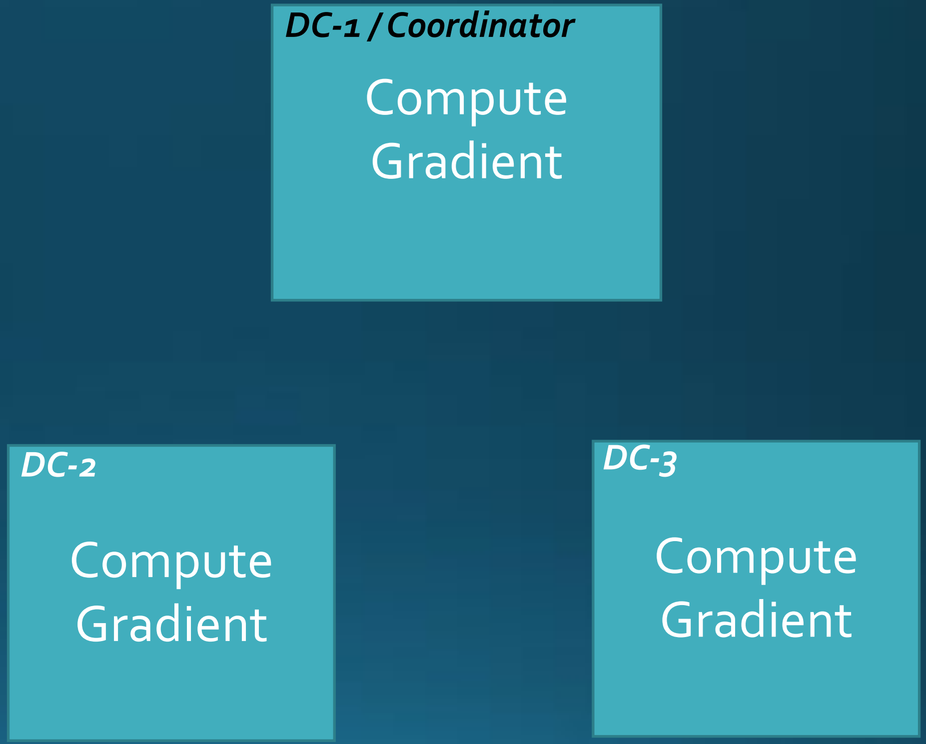
Algorithm

1. Initialize w^0



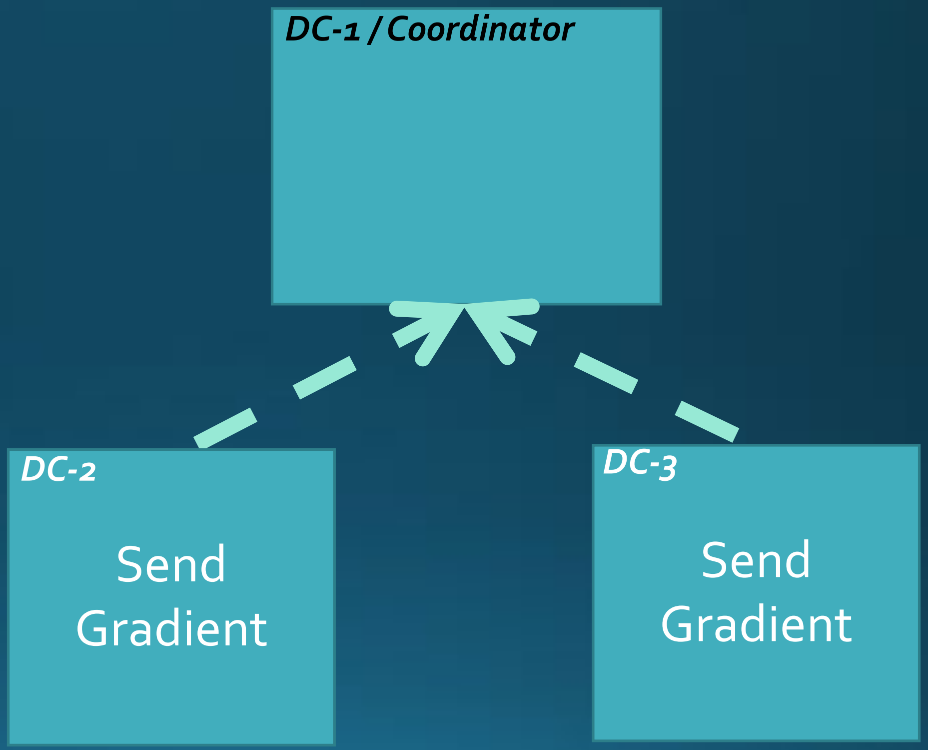
Algorithm

1. Initialize w^0
2. **DCs compute gradient in parallel**



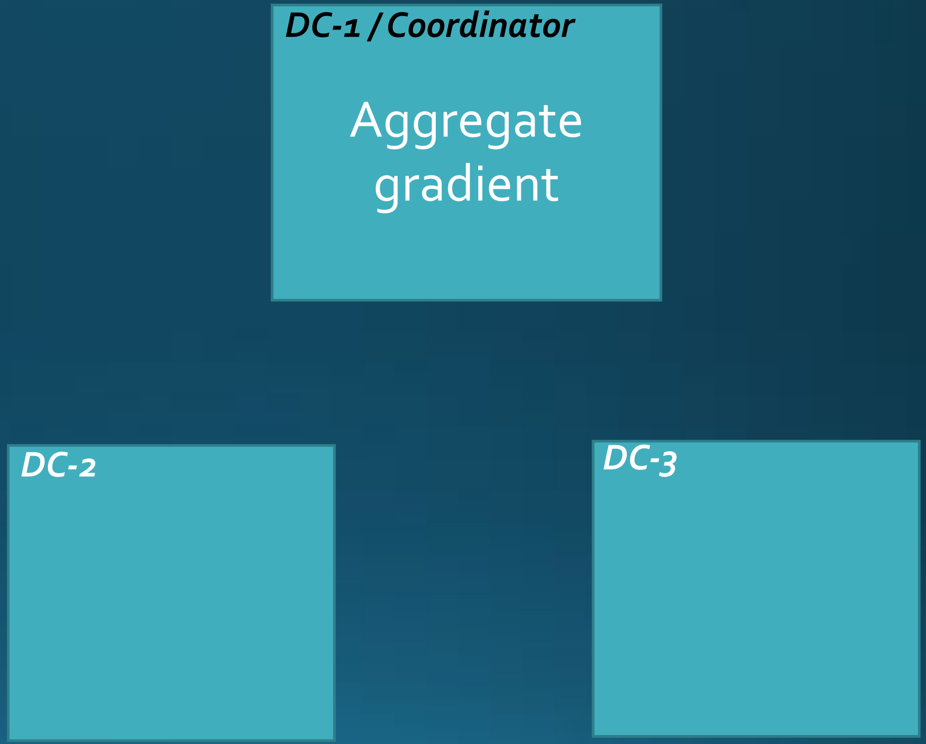
Algorithm

1. Initialize w^0
2. **DCs compute gradient in parallel**



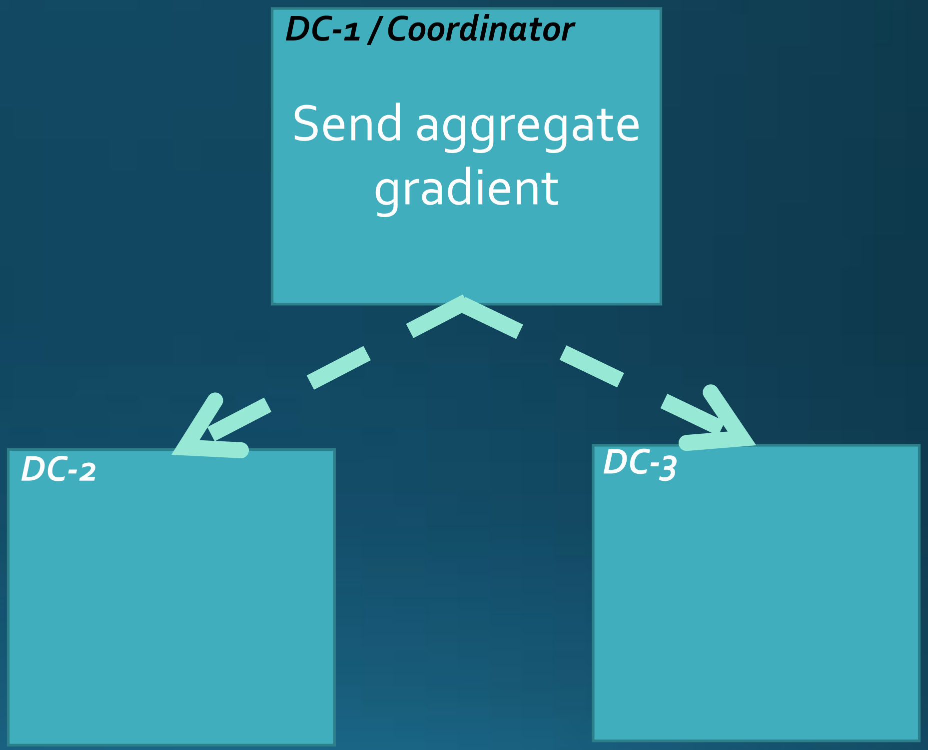
Algorithm

1. Initialize w^0
2. DCs compute gradient in parallel
3. **Aggregate gradient**



Algorithm

1. Initialize w^0
2. DCs compute gradient in parallel
3. **Aggregate gradient**



Algorithm

1. Initialize w^0
2. DCs compute gradient in parallel
3. Aggregate gradient
4. **DCs local optimization in parallel**

DC-1 / Coordinator

Optimize $\hat{f}_1$

no X-DC communication

DC-2

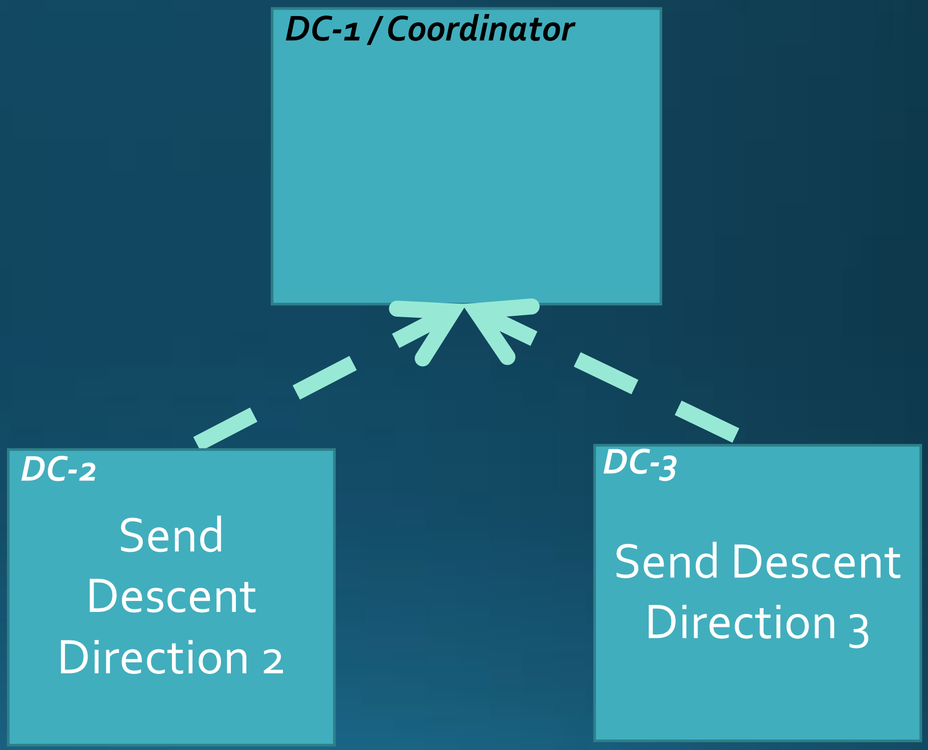
Optimize $\hat{f}_2$

DC-3

Optimize $\hat{f}_3$

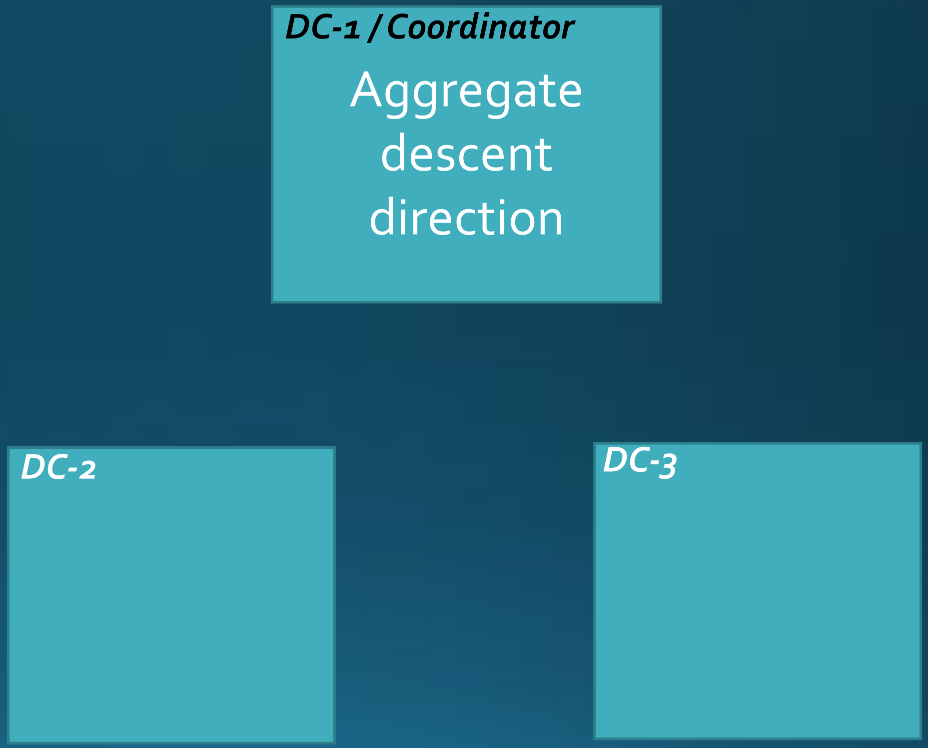
Algorithm

1. Initialize w^0
2. DCs compute gradient in parallel
3. Aggregate gradient
4. **DCs local optimization in parallel**



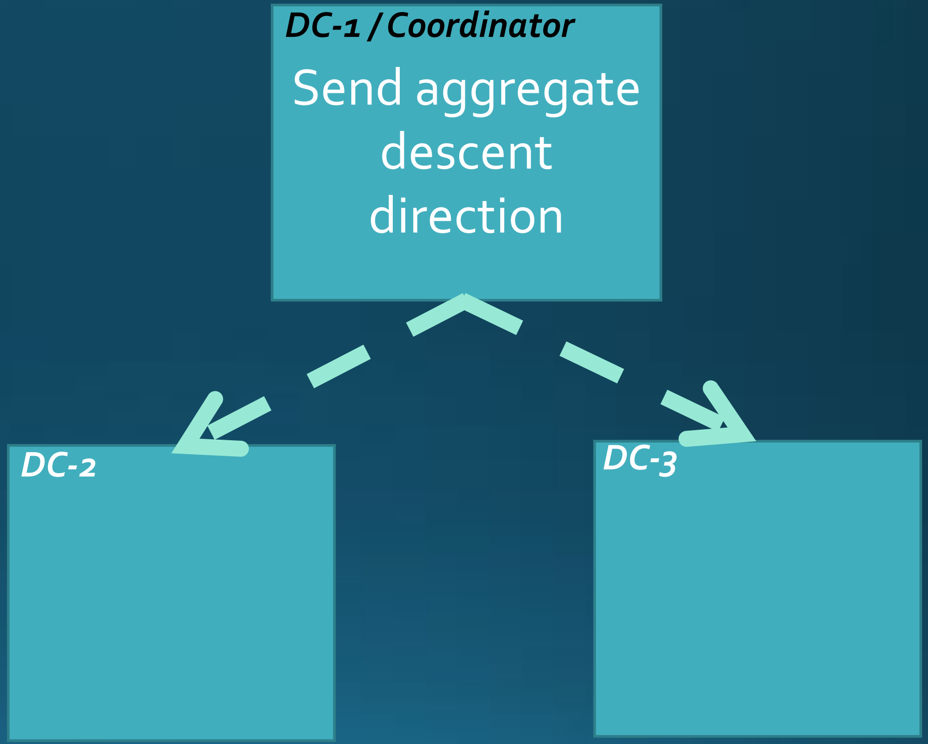
Algorithm

1. Initialize w^0
2. DCs compute gradient in parallel
3. Aggregate gradient
4. DCs local optimization in parallel
5. **Aggregate descent direction**



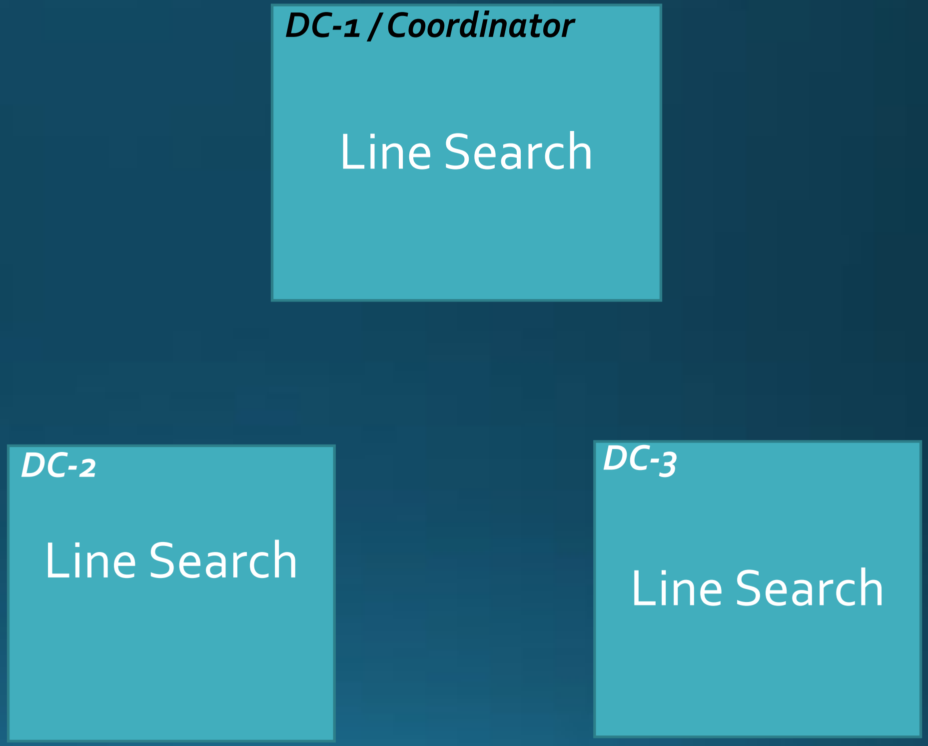
Algorithm

1. Initialize w^0
2. DCs compute gradient in parallel
3. Aggregate gradient
4. DCs local optimization in parallel
5. **Aggregate descent direction**



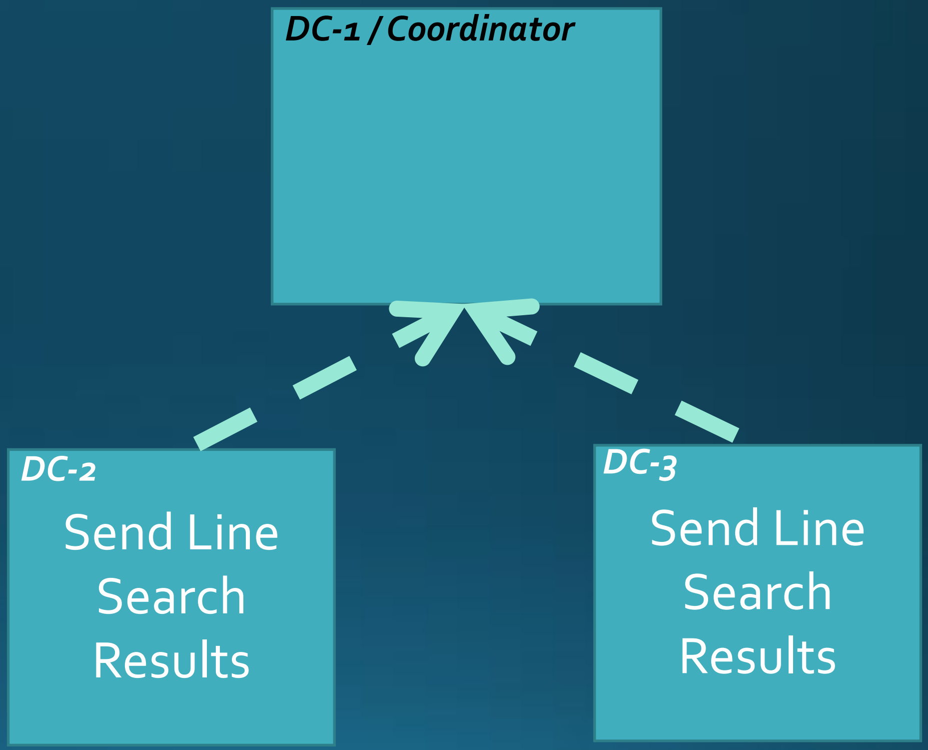
Algorithm

1. Initialize w^0
2. DCs compute gradient in parallel
3. Aggregate gradient
4. DCs local optimization in parallel
5. Aggregate descent direction
6. **DCs do line search in parallel**



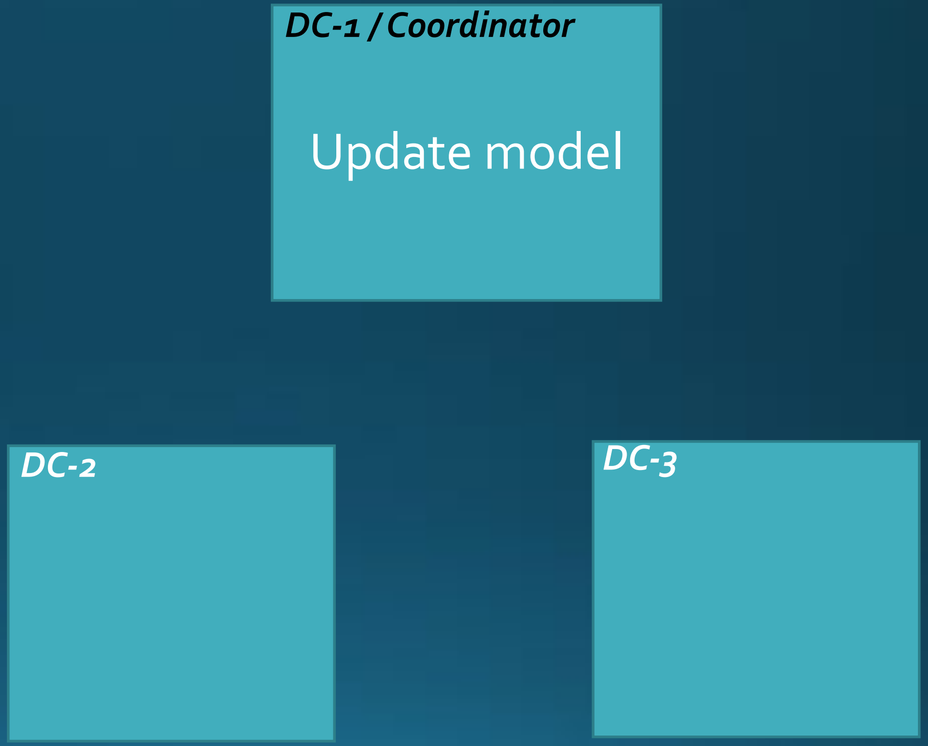
Algorithm

1. Initialize w^0
2. DCs compute gradient in parallel
3. Aggregate gradient
4. DCs local optimization in parallel
5. Aggregate descent direction
6. **DCs do line search in parallel**



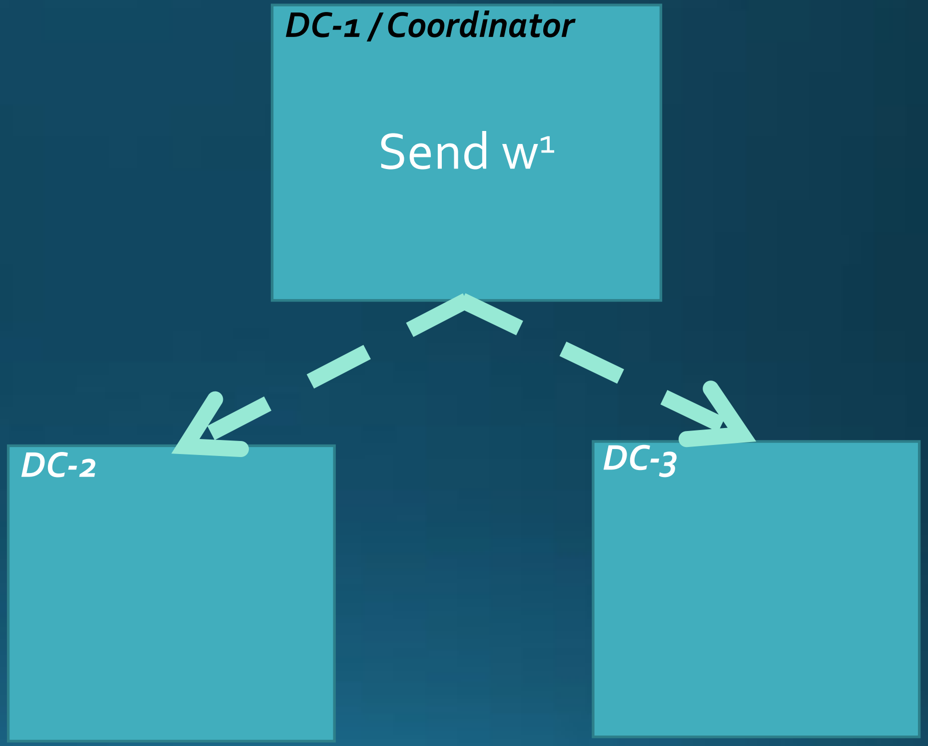
Algorithm

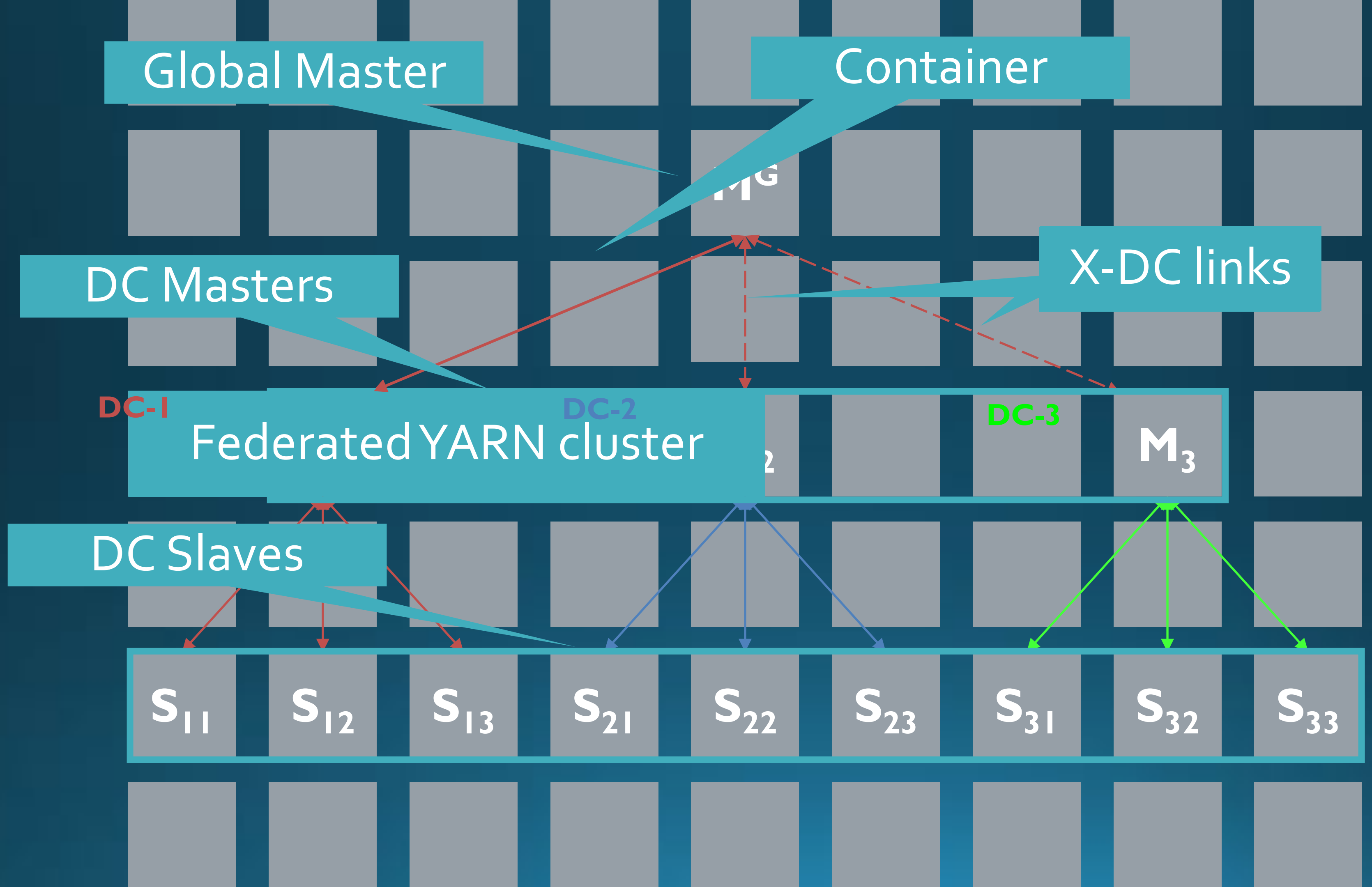
1. Initialize w^0
2. DCs compute gradient in parallel
3. Aggregate gradient
4. DCs local optimization in parallel
5. Aggregate descent direction
6. DCs do line search in parallel
7. **Update model with best step size**



Algorithm

1. Initialize w^0
2. DCs compute gradient in parallel
3. Aggregate gradient
4. DCs local optimization in parallel
5. Aggregate descent direction
6. DCs do line search in parallel
7. Update model with best step size
8. **Repeat**





Global Master
Container
M^G
X-DC links
DC Masters
DC-1
DC-2
DC-3
Federated YARN cluster
M₃
DC Slaves
S11
S12
S13
S21
S22
S23
S31
S32
S33

Related Work (non-exhaustive)

- Analytic workloads
  - Vulimiri et al. [NSDI '15]: reduce WAN bandwidth
  - Pu et al. [SIGCOMM '15] - Iridium: optimize task and data placement to minimize query response time
- Streaming setting
  - Rabkin et al. [NSDI '14]: compute near the edge and only send "important" data
  - Lazerson et al. [VLDB '15]: distributed monitoring
- Information retrieval
  - Baeza-Yates et al. [CIKM '09]: reduce end-user latency in multi-site search engines
- Machine learning
  - Hsieh et al. [NSDI '17] - Gaia: emphasis on reducing training time. Different consistency models to do asynchronous updates
  - McMahan et al. [AISTATS '17]: federated learning using mobile devices