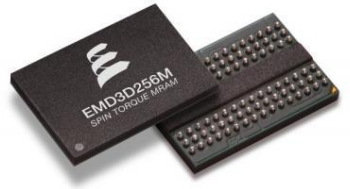
DINOMO: An Elastic, Scalable, High-Performance Key-Value Store for Disaggregated Persistent Memory

Sekwon Lee, Soujanya Ponnapalli, Sharad Singhal,
Marcos K. Aguilera, Kimberly Keeton, Vijay Chidambaram

Persistent Memory (PM)

- Byte-addressable, high-performance
- Non-volatile & high-capacity
  - Retain data across power outage
- Cost per GB >>>> HDD or SSD
  - Need to keep utilization high for cost efficiency

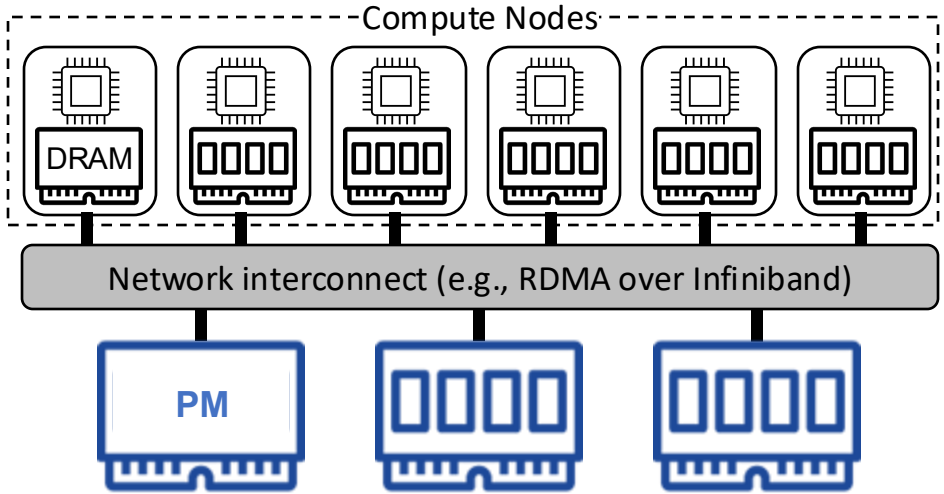
PCM



STT-MRAM

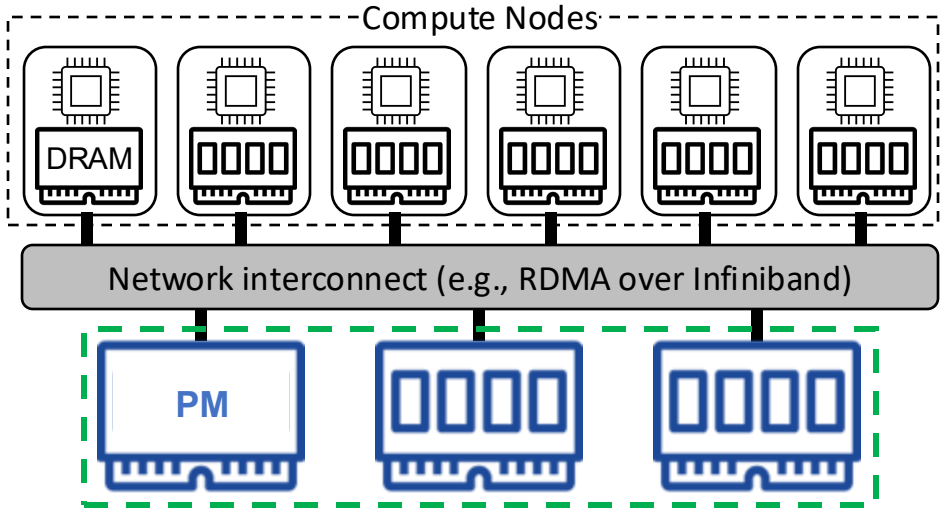
Intel Optane DC PM

Disaggregated Persistent Memory (DPM)



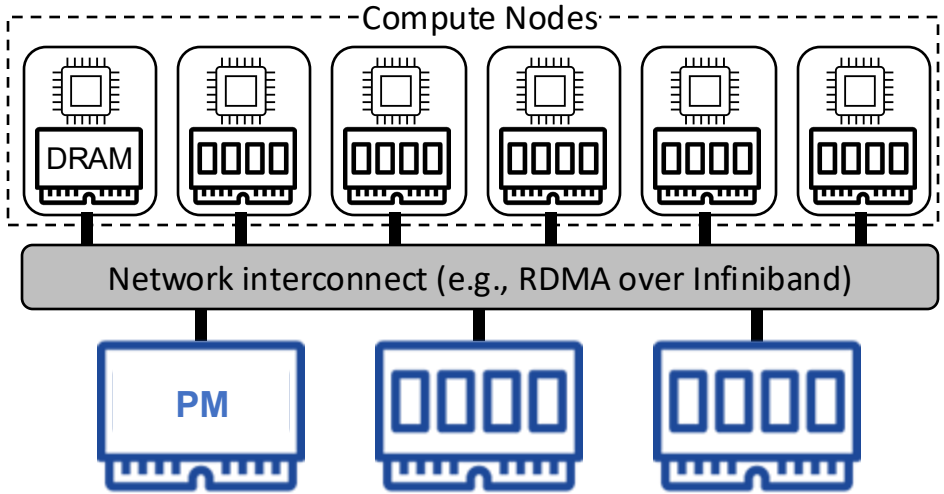
Disaggregated Persistent Memory (DPM)

+ Share PM → Increase utilization, Reduce TCO (Total Cost Ownership)



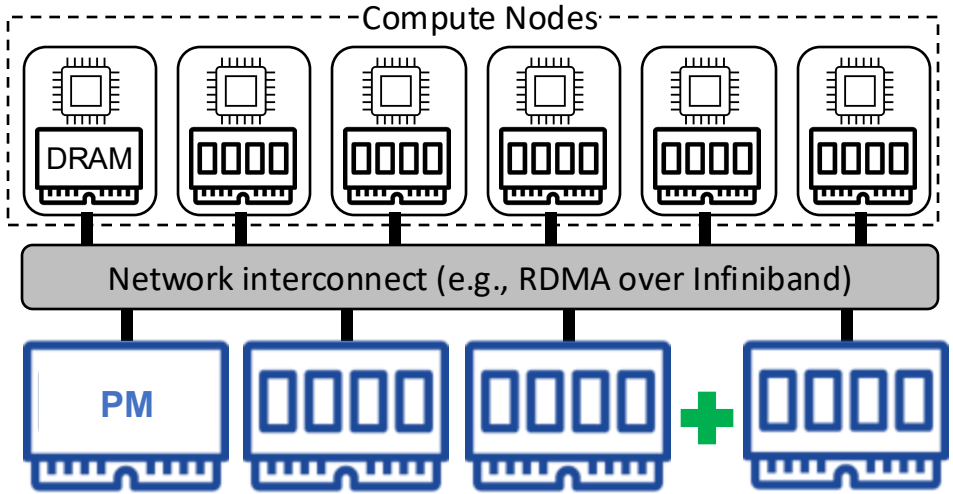
Disaggregated Persistent Memory (DPM)

+ Share PM → Increase utilization, Reduce TCO (Total Cost Ownership)
+ Disaggregate PM → Scale resources independently, Separate failure domains



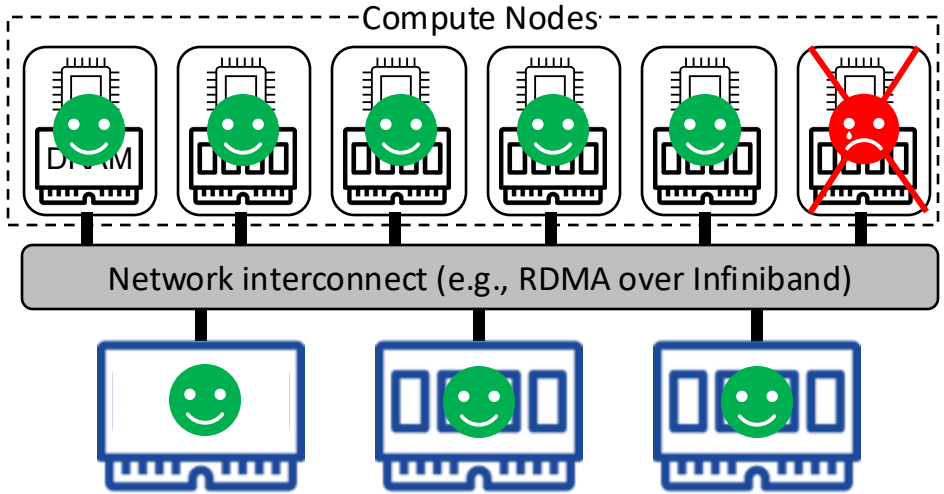
Disaggregated Persistent Memory (DPM)

+ Share PM → Increase utilization, Reduce TCO (Total Cost Ownership)
+ Disaggregate PM → Scale resources independently, Separate failure domains



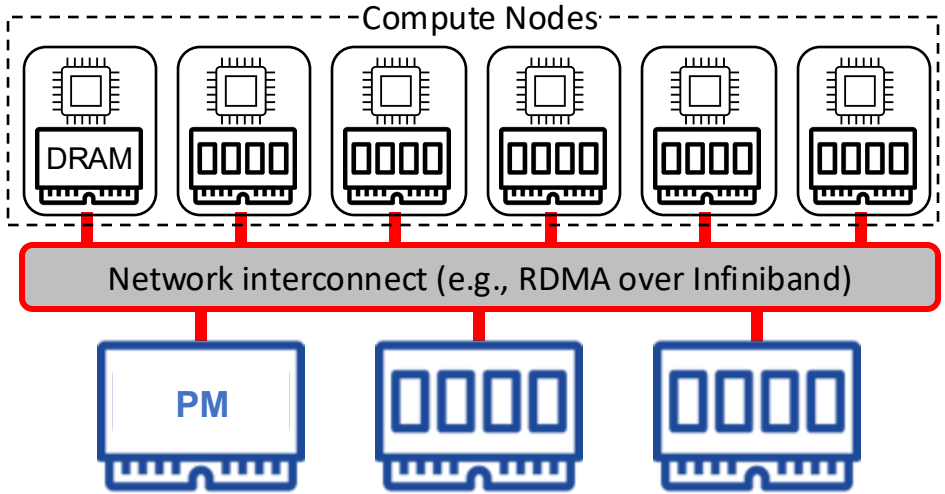
Disaggregated Persistent Memory (DPM)

+ Share PM → Increase utilization, Reduce TCO (Total Cost Ownership)
+ Disaggregate PM → Scale resources independently, Separate failure domains



Disaggregated Persistent Memory (DPM)

+ Share PM → Increase utilization, Reduce TCO (Total Cost Ownership)
+ Disaggregate PM → Scale resources independently, Separate failure domains
– Access PM over network (1 - 4µs) >> local PM latency (300 - 400ns)



Key-Value Store (KVS) for DPM

High common-case performance
despite high-network costs

Scalability
with the increase in provisioned resources

Fast reconfiguration
in response to dynamics (e.g., node addition/failure)

Key-Value Store (KVS) for DPM

Challenge: easy to sacrifice one of the three goals to achieve the others

No DPM KVSs providing all the three goals simultaneously

in response to dynamics (e.g., node addition/failure)

DINOMO

First DPM KVS achieving high performance, scalability, and fast reconfiguration simultaneously

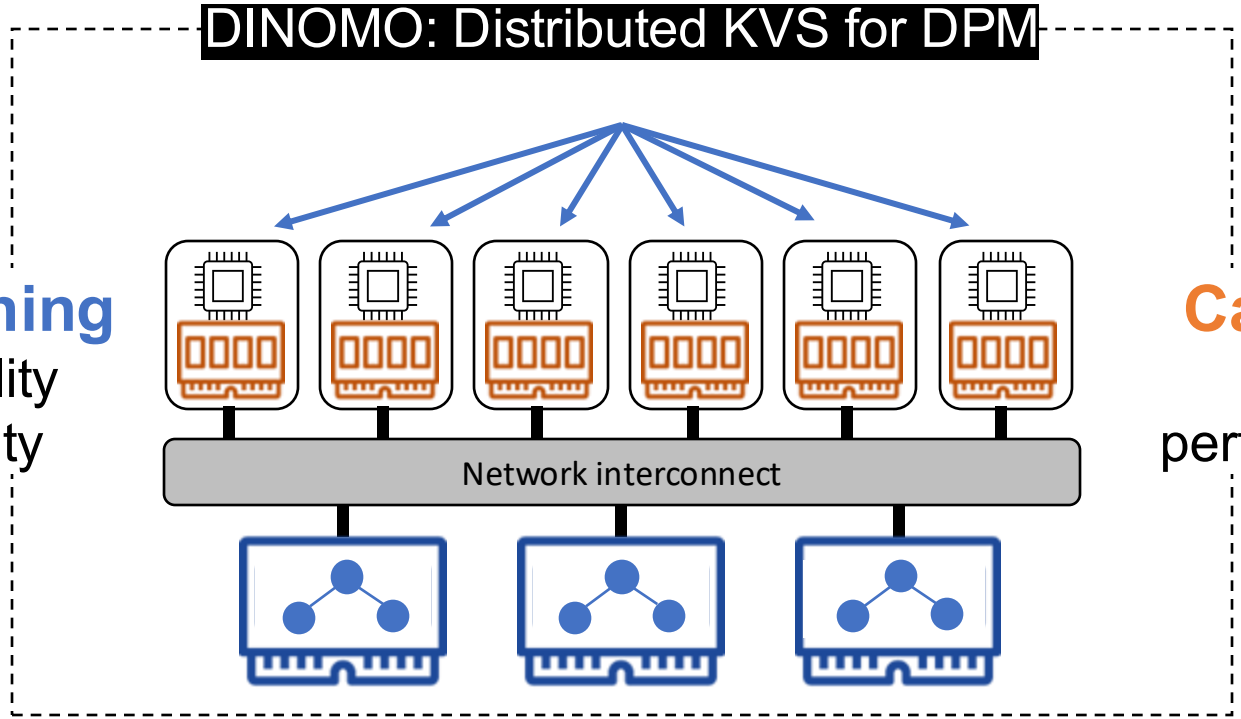
Adapt techniques (e.g., **partitioning**, **caching**, **replication**) from storage research community for DPM

Full end-to-end implementations including KVS data plane, control plane, and client

Better performance up-to **10x** at scale and elasticity

https://github.com/utsaslab/dinomo

Outline



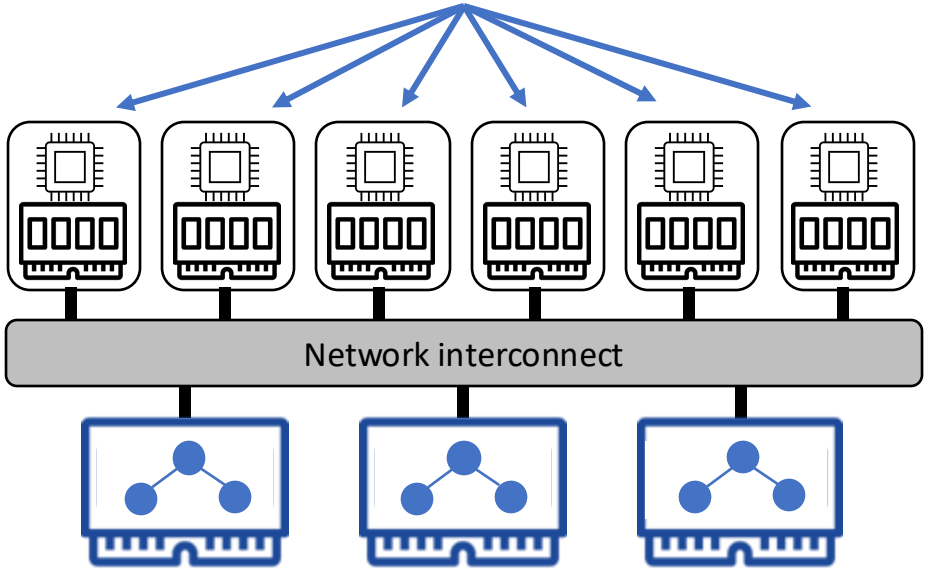
Partitioning
Scalability
Elasticity

Caching
High performance

Outline

DINOMO: Distributed KVS for DPM

Ownership partitioning
Scalability
Elasticity



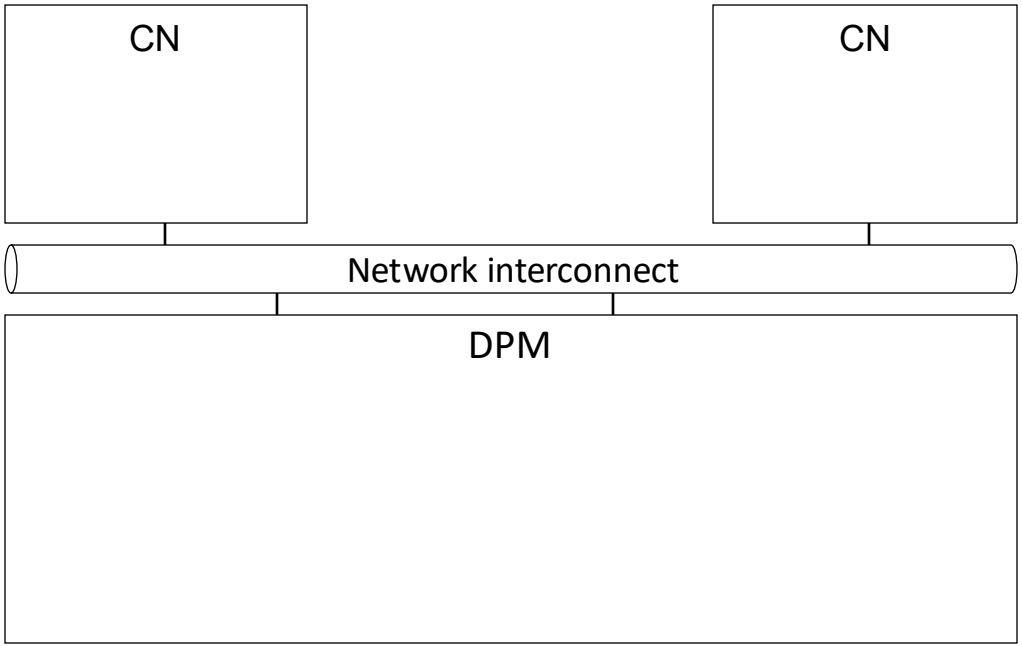
Caching
High performance

System architectures for DPM

- What to share or partition?

Goals \ KVSs	?	Shared everything	Shared nothing
High performance	✓	X	✓
Scalability	✓	X	✓
Lightweight reconfiguration	✓	✓	X

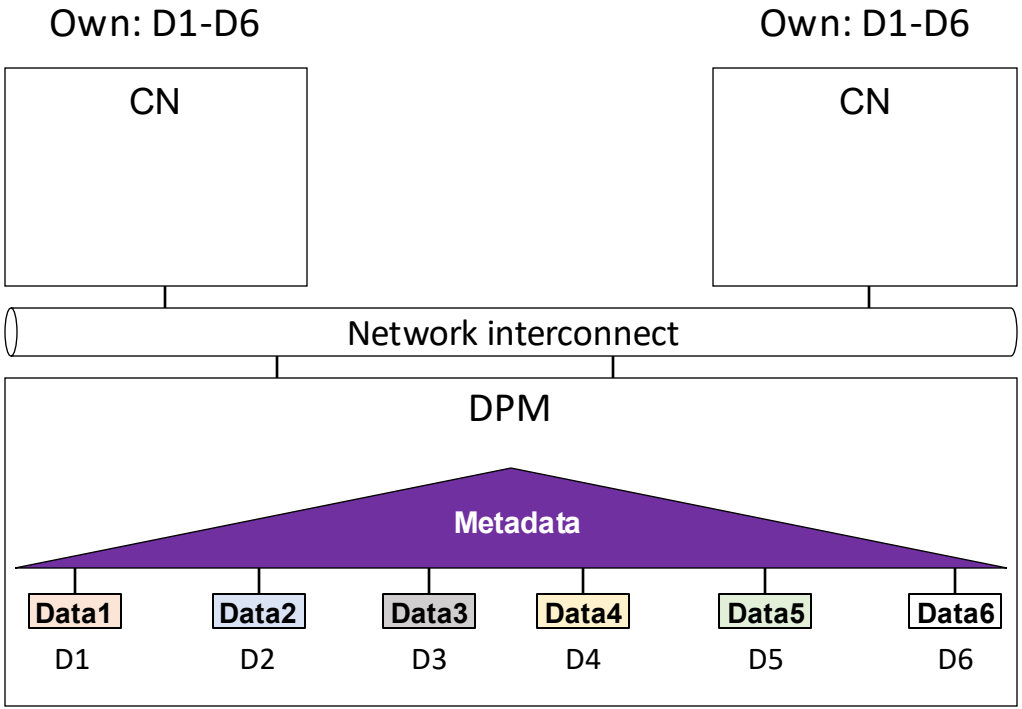
Shared everything



Shared data, metadata, ownership

- Data: key-value pairs
- Metadata: index structures
- Ownership: access permission

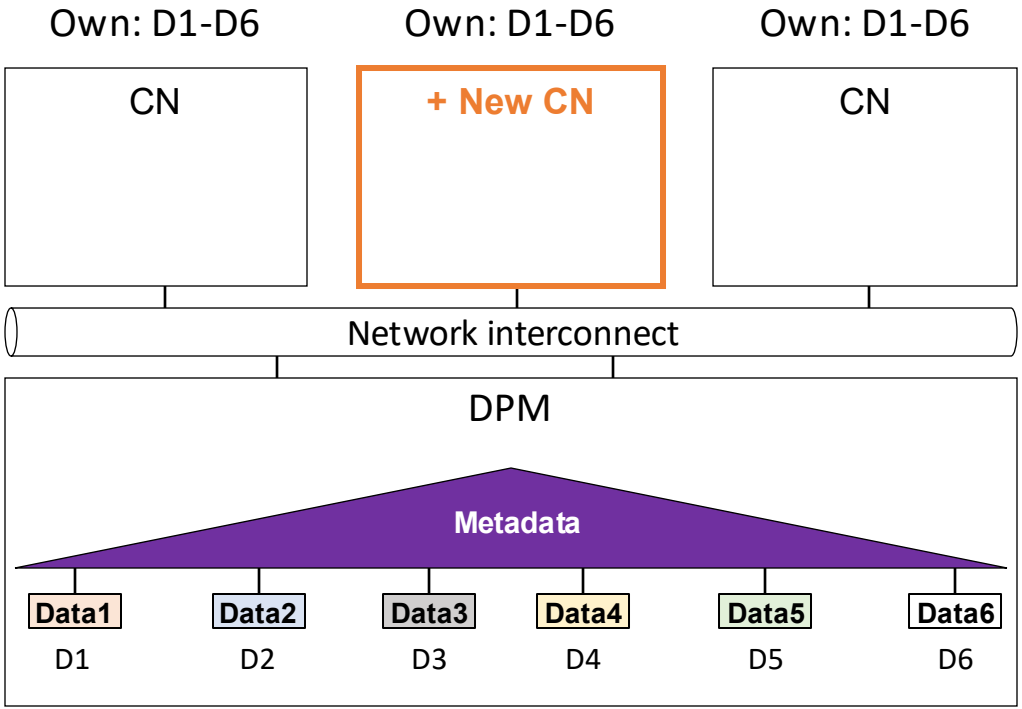
Shared everything



Shared data, metadata, ownership

- Data: key-value pairs
- Metadata: index structures
- Ownership: access permission

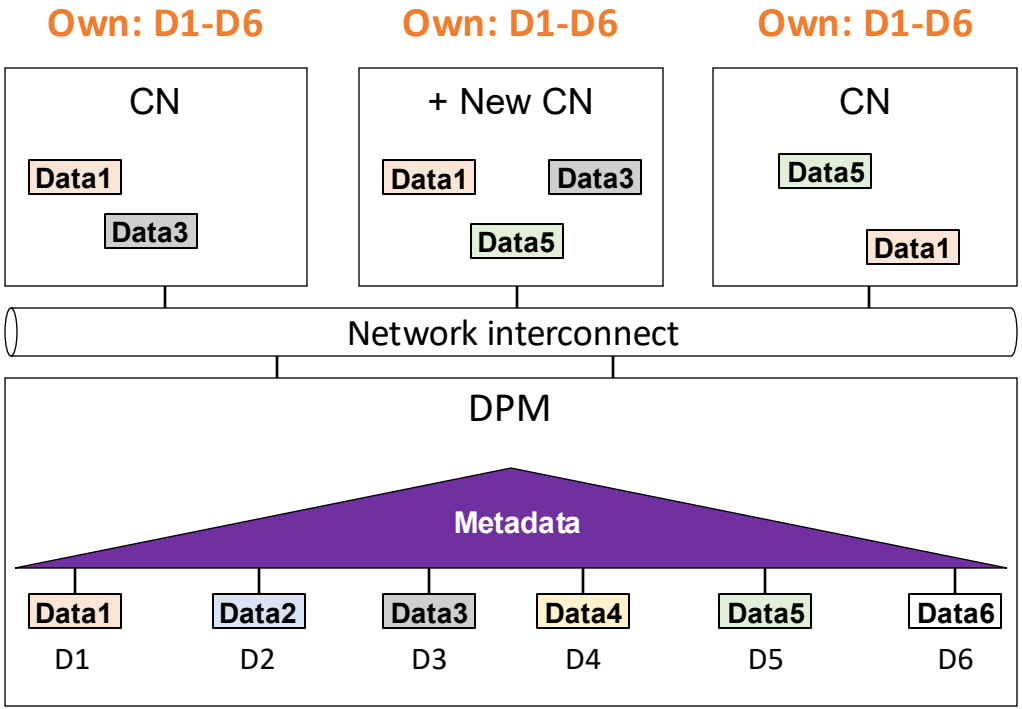
Shared everything



Shared data, metadata, ownership

Fast reconfiguration without data reorganization

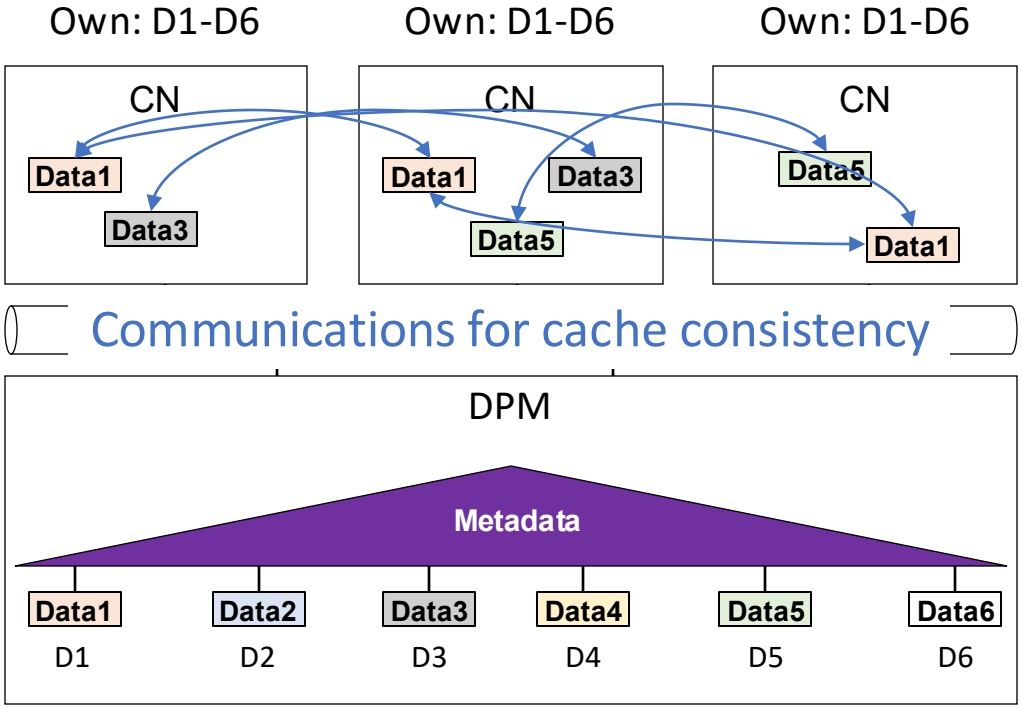
Shared everything



Shared data, metadata, ownership

Fast reconfiguration without data reorganization

Shared everything

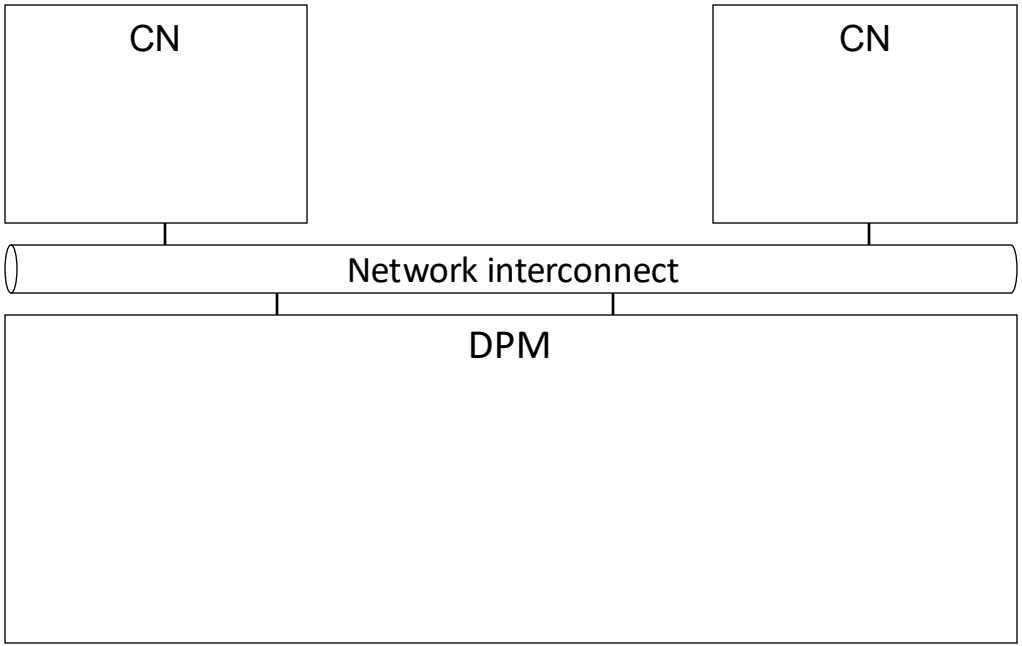


Shared data, metadata, ownership

Fast reconfiguration without data reorganization

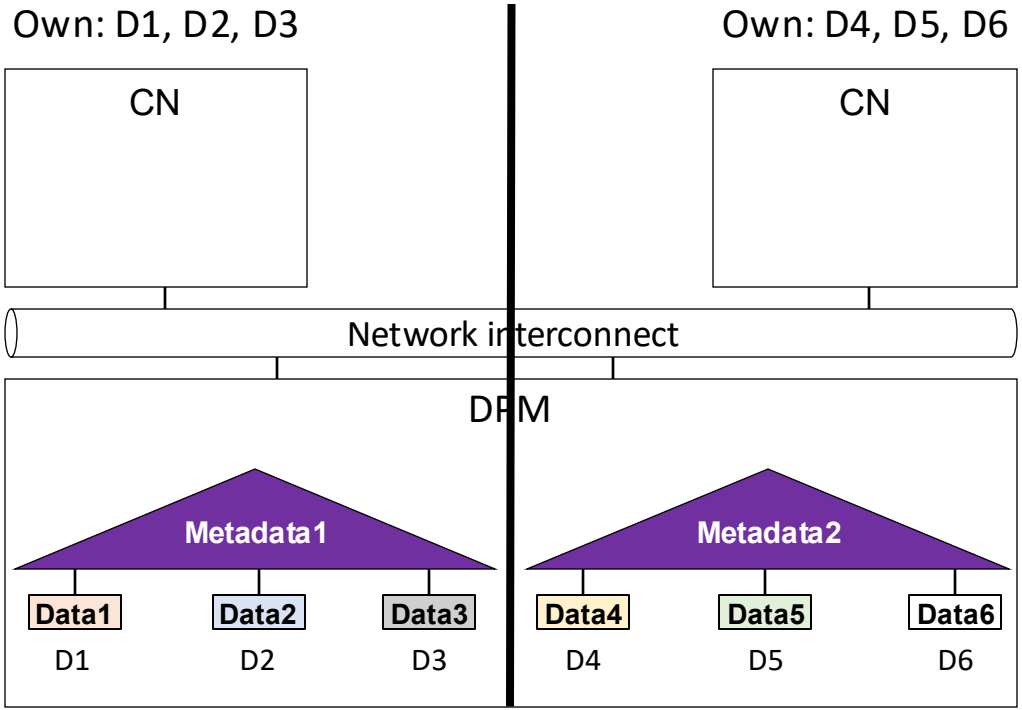
Low performance/scalability due to consistency overheads

Shared nothing



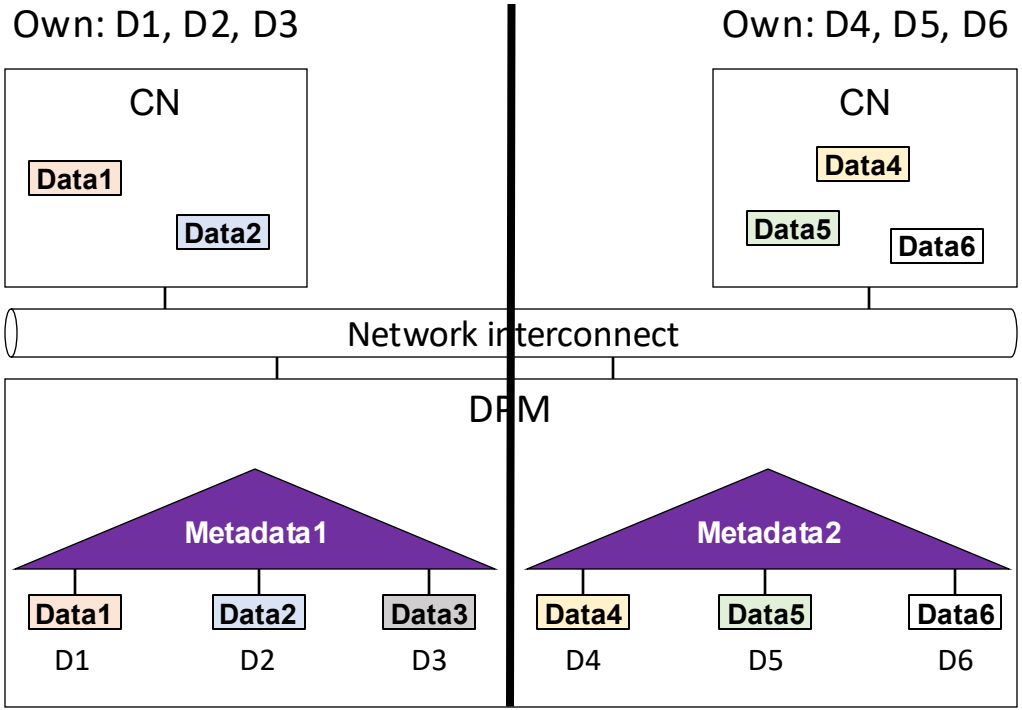
Partitioned data, metadata, ownership

Shared nothing



Partitioned data, metadata, ownership

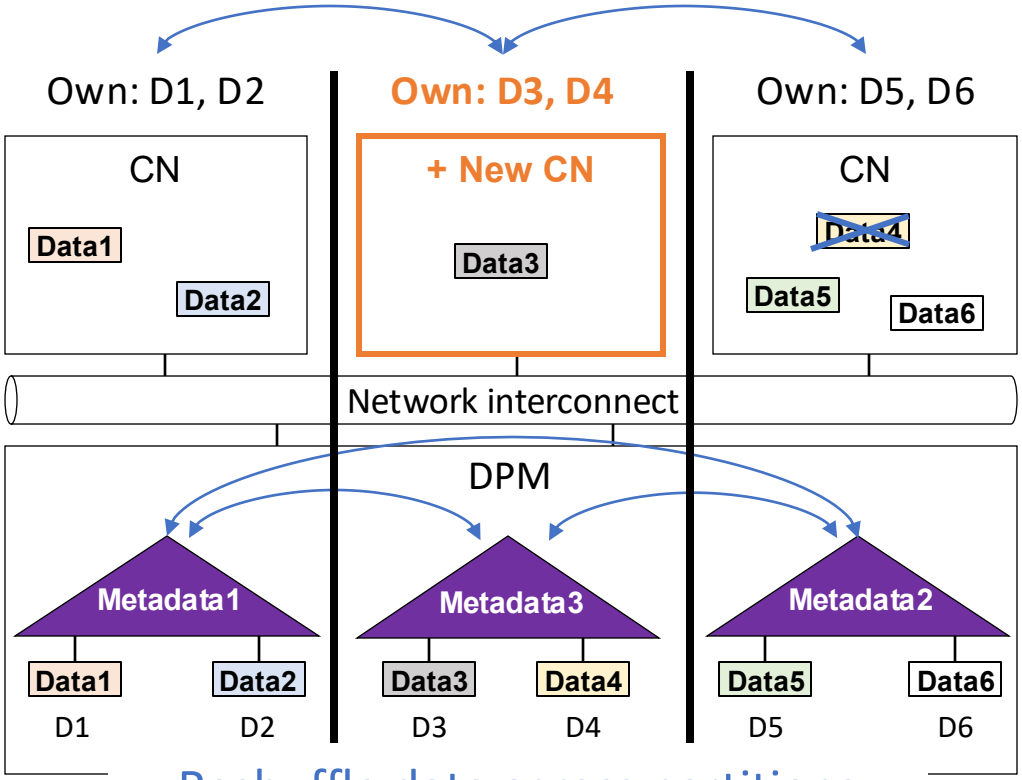
Shared nothing



Partitioned data, metadata, ownership

High performance/scalability without consistency overheads

Shared nothing



Reshuffle data across partitions

Partitioned data, metadata, ownership

High performance/scalability without consistency overheads

Slow reconfiguration due to expensive data reorganization

System architectures for DPM

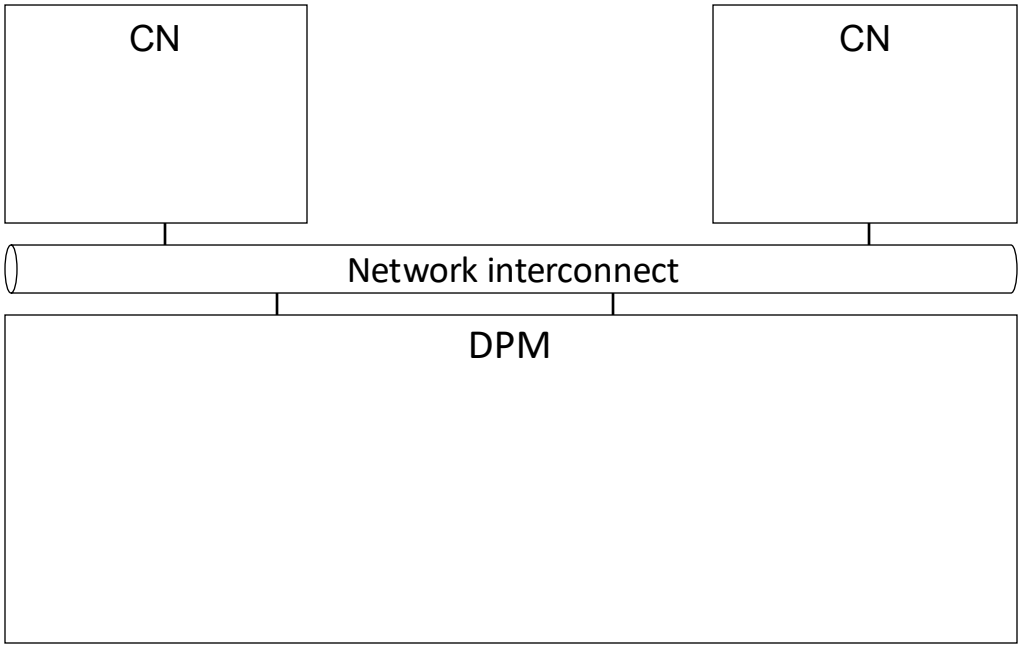
- What to share or partition?

Goals \ KVSs	Ownership partitioning	Shared everything	Shared nothing
High performance	✓	X	✓
Scalability	✓	X	✓
Lightweight reconfiguration	✓	✓	X

Approach: Partition ownership across compute nodes while sharing data through DPM

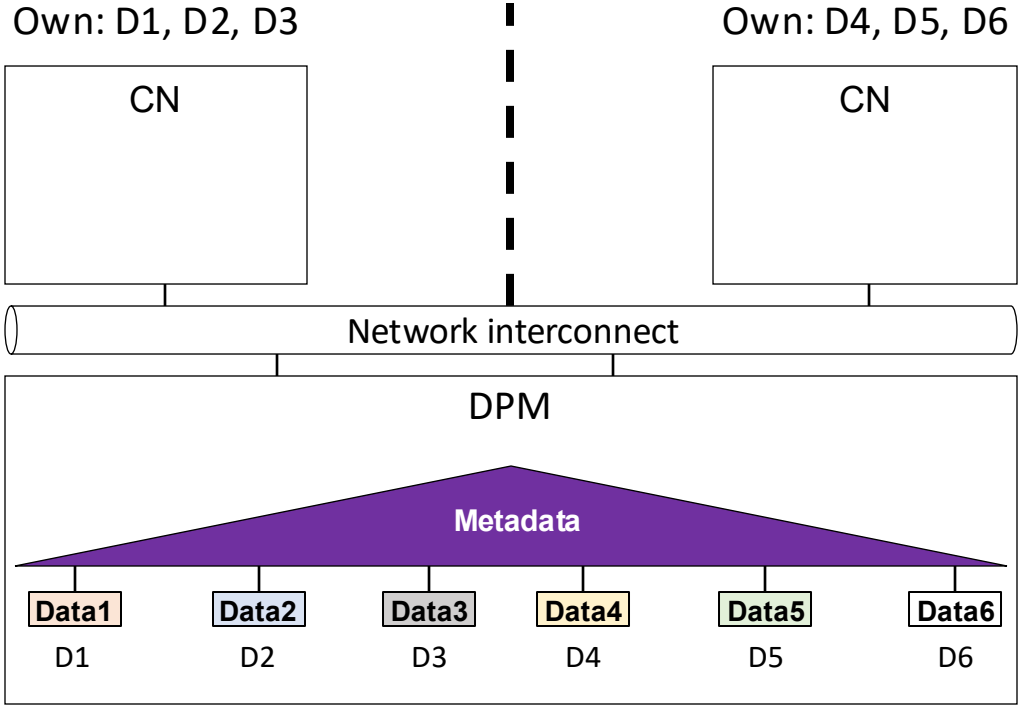
Insight: Data and ownership can be an independent consideration owing to disaggregation

Ownership Partitioning



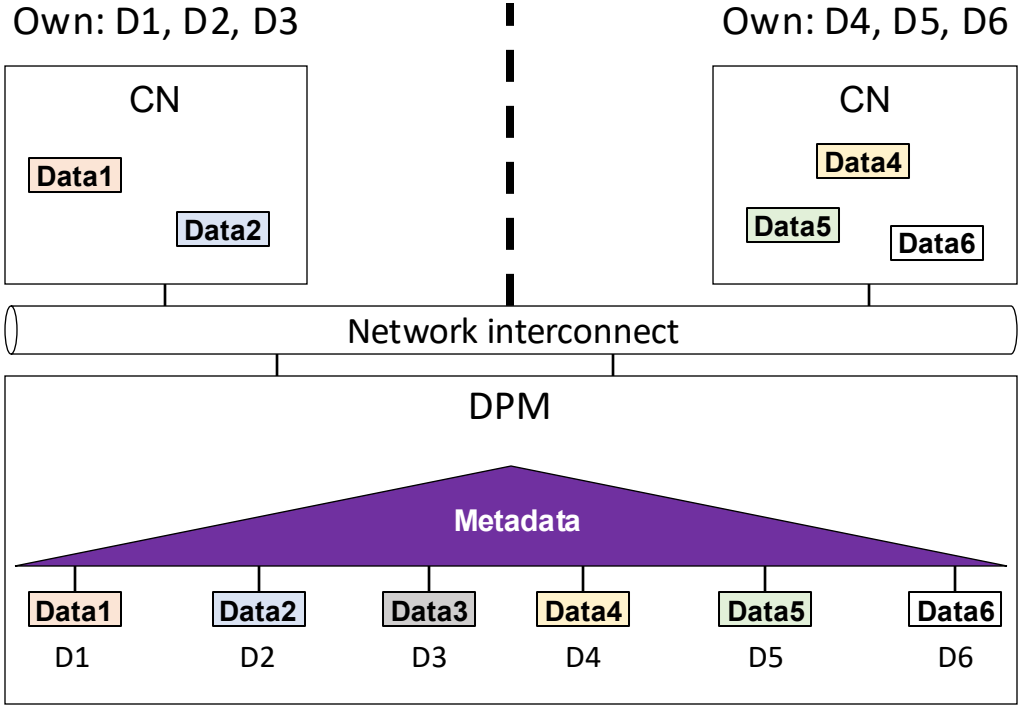
Shared data/metadata, but partitioned ownership

Ownership Partitioning



Shared data/metadata, but partitioned ownership

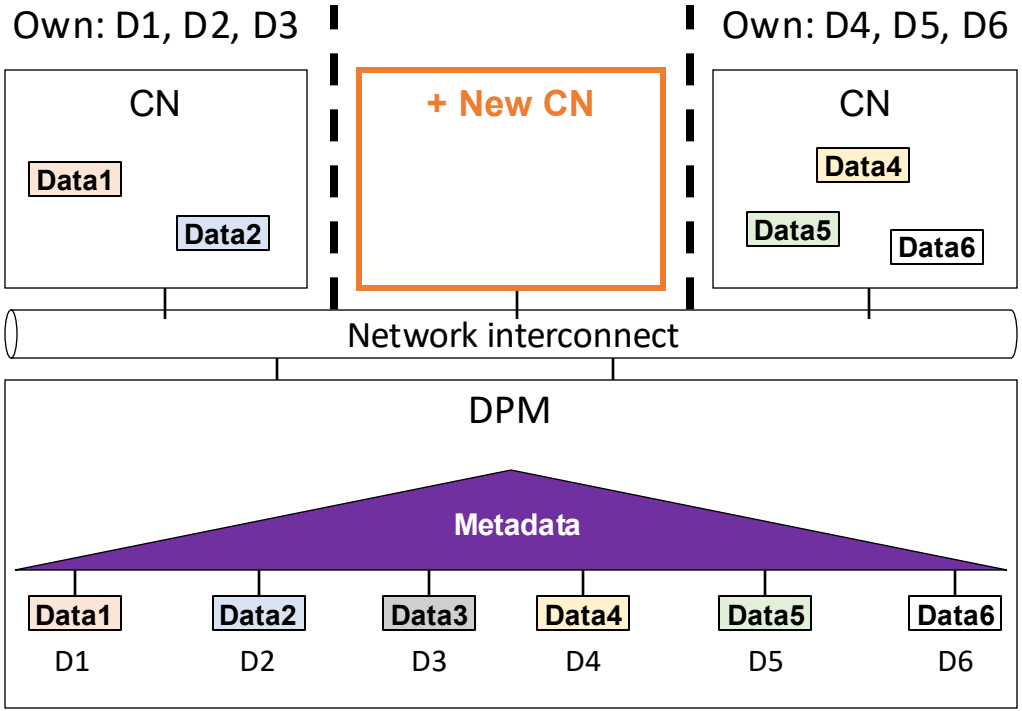
Ownership Partitioning



Shared data/metadata, but partitioned ownership

High performance/scalability without consistency overheads

Ownership Partitioning

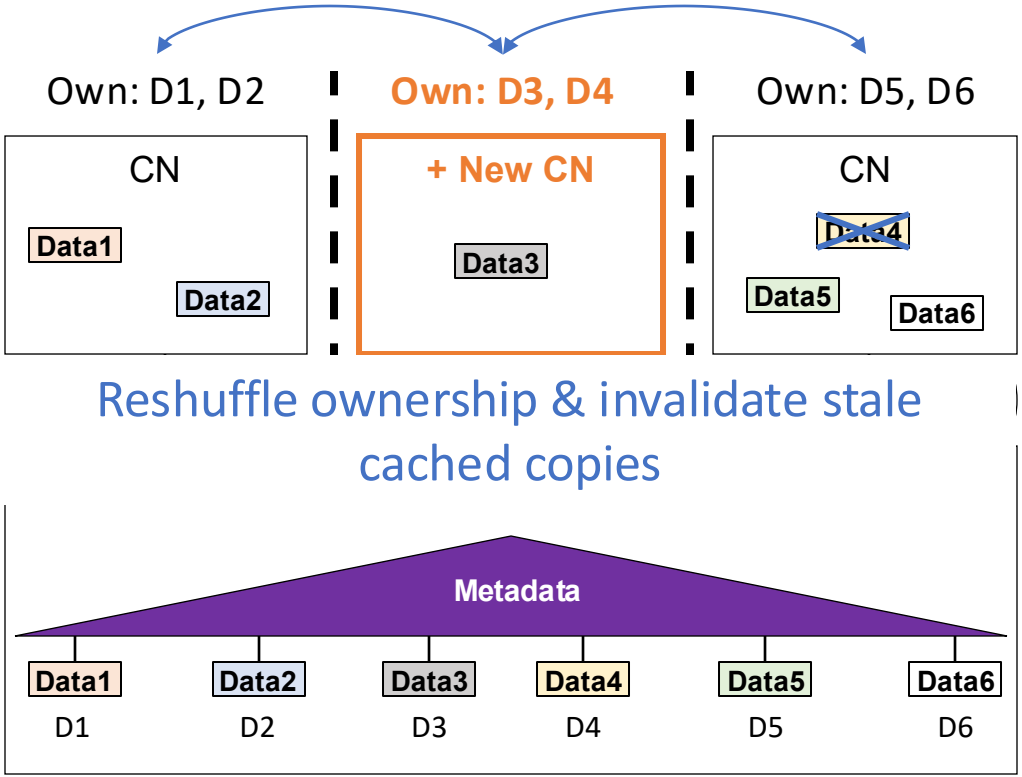


Shared data/metadata, but partitioned ownership

High performance/scalability without consistency overheads

Fast reconfiguration without data reorganization

Ownership Partitioning



Shared data/metadata, but partitioned ownership

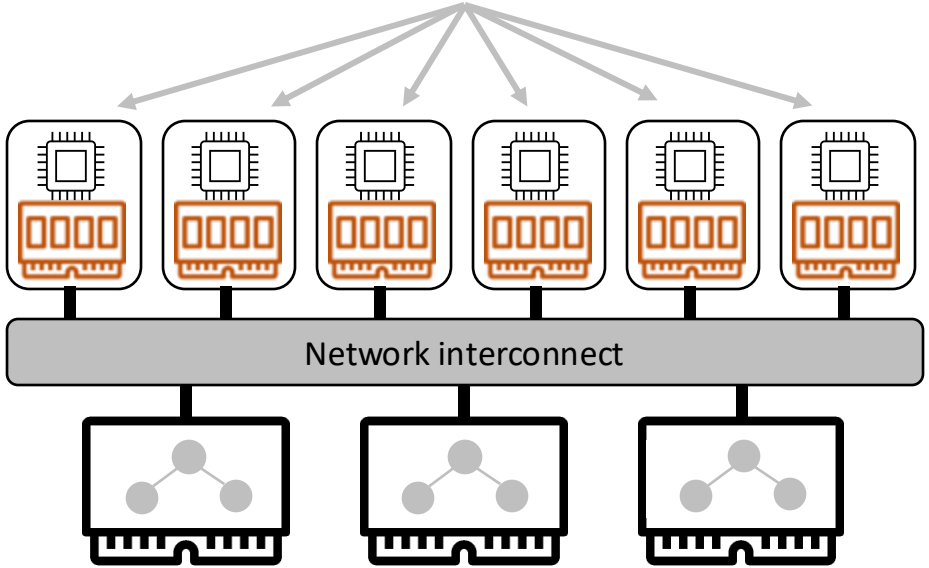
High performance/scalability without consistency overheads

Fast reconfiguration without data reorganization

Outline

DINOMO: Distributed KVS for DPM

Ownership partitioning
Scalability
Elasticity



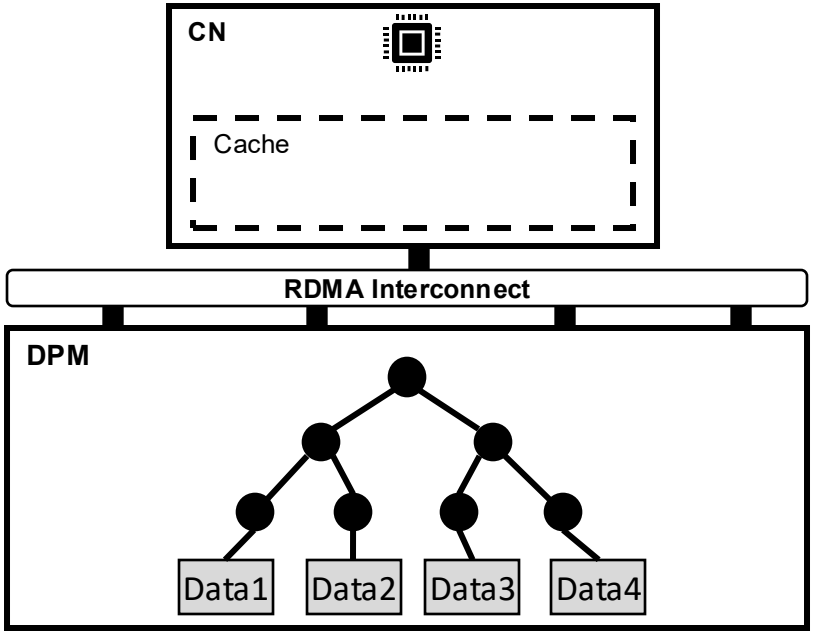
Disaggregated adaptive cache
High performance

Caching for DPM

- Number of network round trips significantly impacts on overall system performance
- Cache data or metadata into the memory of compute nodes to reduce round trips to DPM
  - Important to minimize cache misses

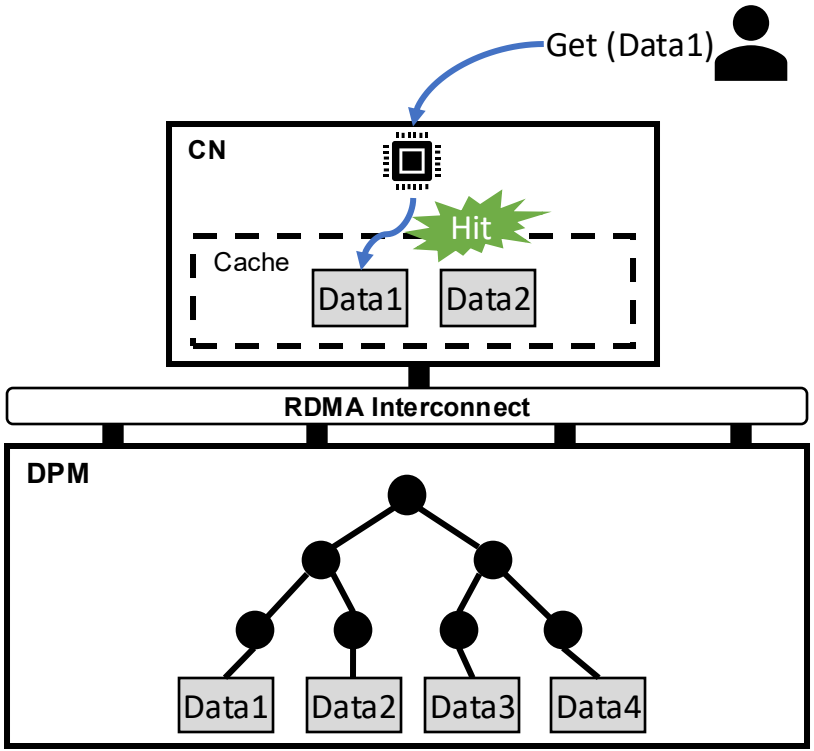
Static caching strategies

- Value
  - Entire copy of data in DPM
- Shortcut
  - Remote pointer to data in DPM



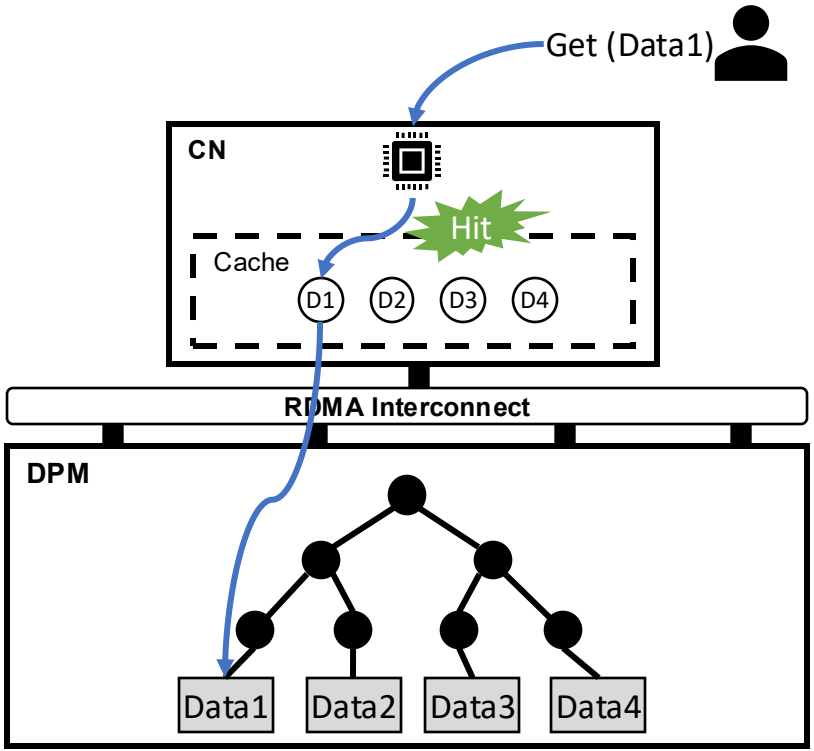
Static caching strategies

- Value
  - Entire copy of data in DPM
  - Zero round trip, but more space
- Shortcut
  - Remote pointer to data in DPM



Static caching strategies

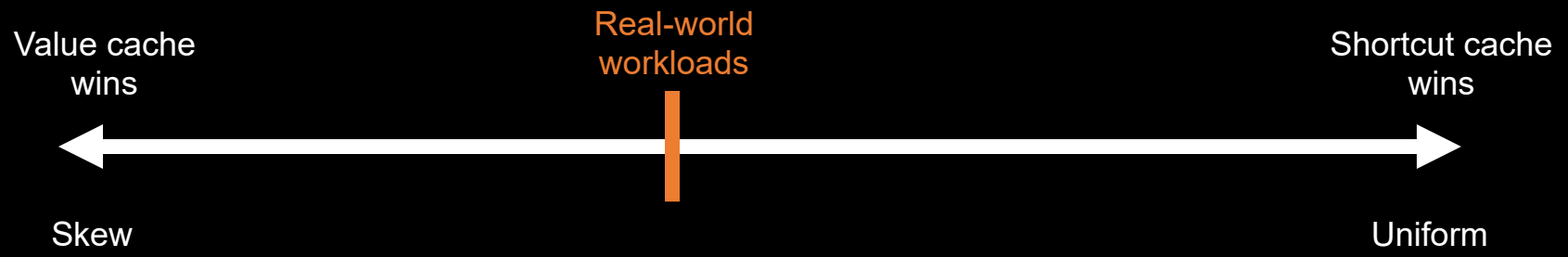
- Value
  - Entire copy of data in DPM
  - Zero round trip, but more space
- Shortcut
  - Remote pointer to data in DPM
  - One round trip, but less space



Is it better to cache a few values without overheads on hits, or a larger number of shortcuts with fixed hit overheads?

Is it better to cache a few values without overheads on hits, or a larger number of shortcuts with fixed hit overheads?

Is it better to cache a few values without overheads on hits, or a larger number of shortcuts with fixed hit overheads?



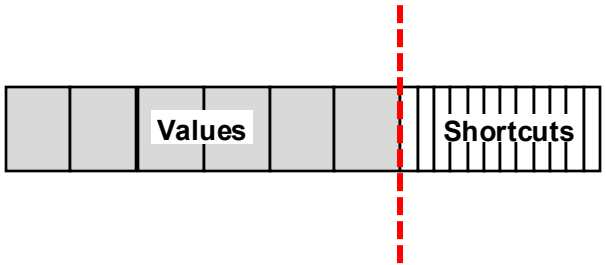
Is it better to cache a few values without overheads on hits, or a larger number of shortcuts with fixed hit overheads?

Answer: Efficient ratio depends on workload patterns and available memory space

We need an **adaptive policy** changing ratio between values and shortcuts!

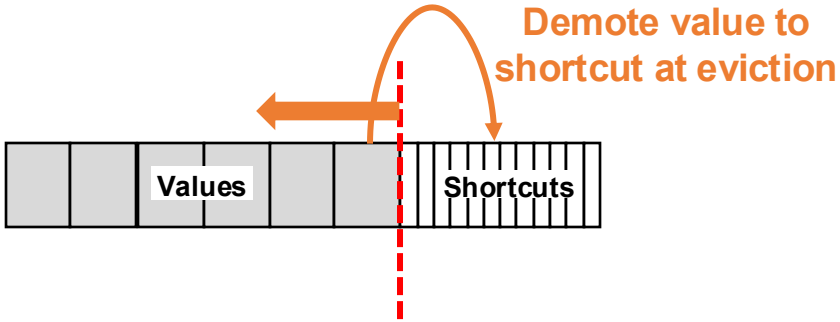
Disaggregated Adaptive Caching

- Adaptive policy
  - Change the boundary via demotion and promotion



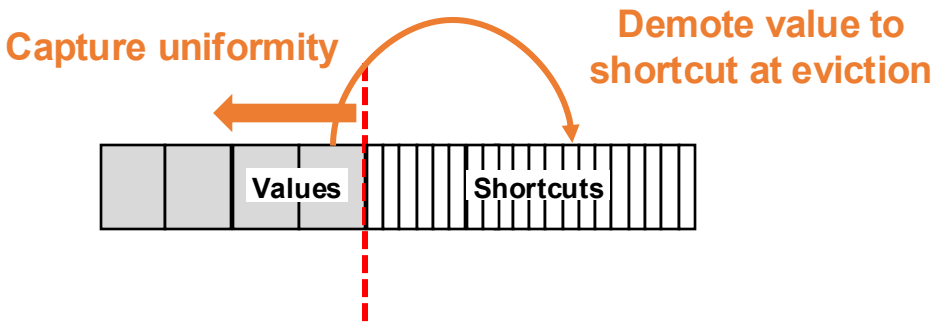
Disaggregated Adaptive Caching

- Adaptive policy
  - Change the boundary via demotion and promotion



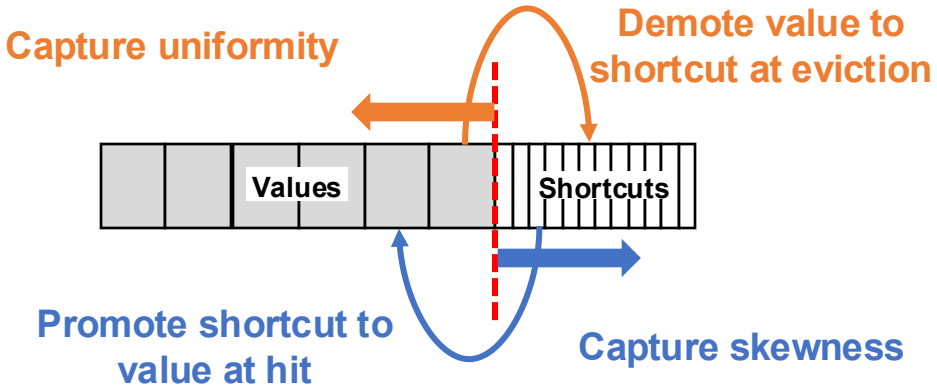
Disaggregated Adaptive Caching

- Adaptive policy
  - Change the boundary via demotion and promotion



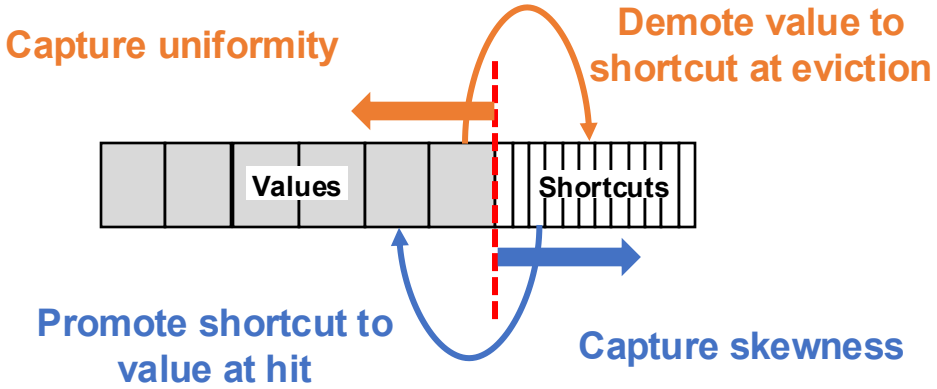
Disaggregated Adaptive Caching

- Adaptive policy
  - Change the boundary via demotion and promotion



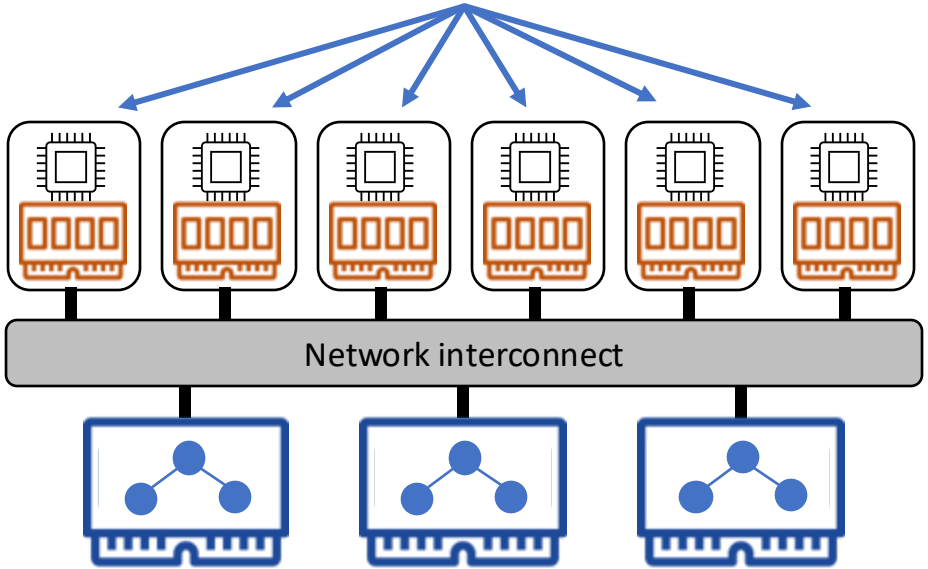
Disaggregated Adaptive Caching

- Adaptive policy
  - Change the boundary via demotion and promotion
  - Promotion policy considering sizes, hit costs, and miss costs
    - *Hit benefit from the promoted shortcut > Miss costs from evicted shortcuts*



Outline

DINOMO: Distributed KVS for DPM



Ownership partitioning
Scalability
Elasticity

Disaggregated adaptive cache
High performance

Evaluation

- How does DINOMO fare against the state-of-the-art in terms of performance and scalability?

- How elastic and responsive is DINOMO while handling workload dynamics, load imbalance, and node failures?

Evaluation

- How does DINOMO fare against the state-of-the-art in terms of performance and scalability?
  - DINOMO scales performance with # of CNs
  - DINOMO performs up-to 10x better than the state of the art
- How elastic and responsive is DINOMO while handling workload dynamics, load imbalance, and node failures?

Evaluation

- How does DINOMO fare against the state-of-the-art in terms of performance and scalability?
  - DINOMO scales performance with # of CNs
  - DINOMO performs up-to 10x better than the state of the art
- How elastic and responsive is DINOMO while handling workload dynamics, load imbalance, and node failures?
  - DINOMO is much more responsive than shared-nothing counterparts, but comparable to shared everything

DINOMO 

- First KVS for DPM achieving high performance, scalability, and elasticity simultaneously
- Use a novel combination of techniques, ownership partitioning and disaggregated adaptive cache
- Experimentally show DINOMO can scale performance and efficiently react to reconfigurations
- Try our KVS: https://github.com/utsaslab/dinomo

Backup

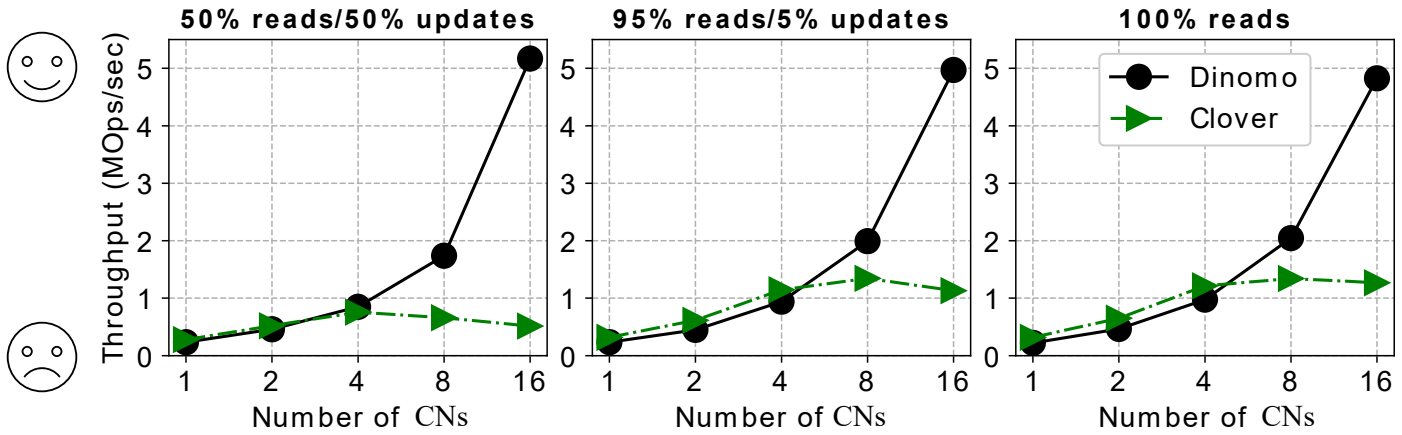
Evaluation setup

- System configuration
  - DPM: 4 threads, 110GB of DRAM to emulate PM
  - 16 CNs: 8 threads, 1GB of DRAM for caching (≈1% of the DPM)
  - Connected via 56Gbps ConnectX-3 RNICs
- Baseline
  - Performance/scalability: Clover (shared everything, shortcut-only cache)
  - Elasticity: DINOMO-N (Disaggregated adaptive caching, but partition data/metadata)
- Workload
  - YCSB workloads with 8B keys and 1KB values

Evaluation

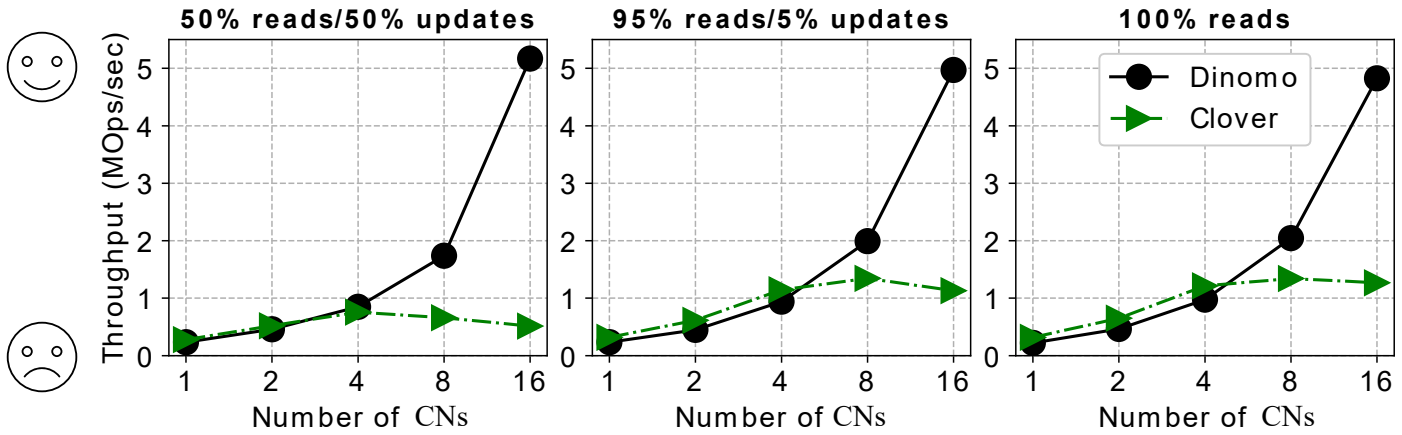
- How does DINOMO fare against the state-of-the-art in terms of performance and scalability?
- How elastic and responsive is DINOMO while handling changes in workloads?

Performance and Scalability



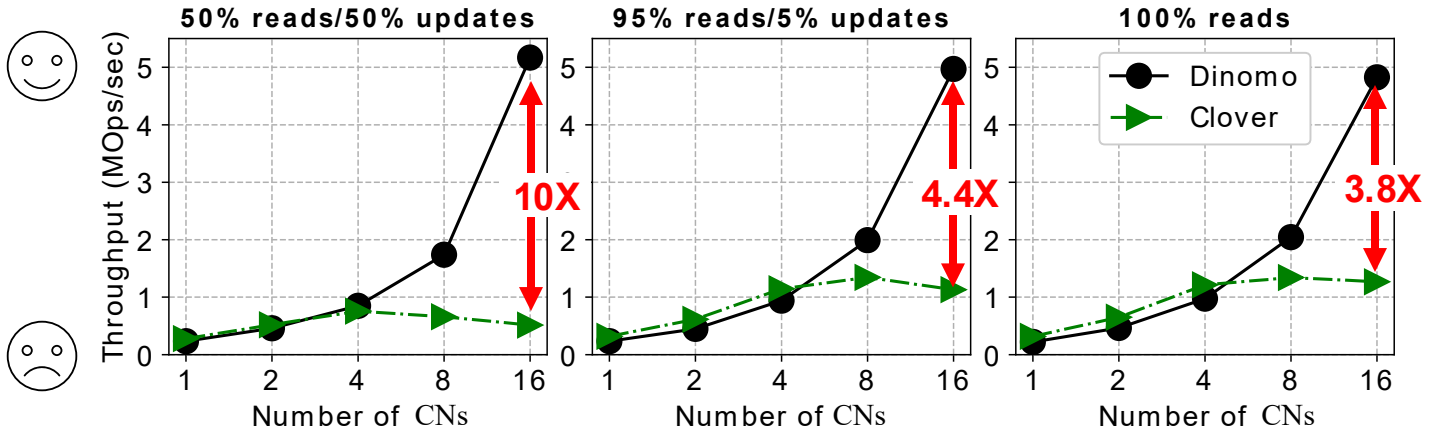
Performance and Scalability

- DINOMO scales to 16 CNs, but Clover does not beyond 4 CNs



Performance and Scalability

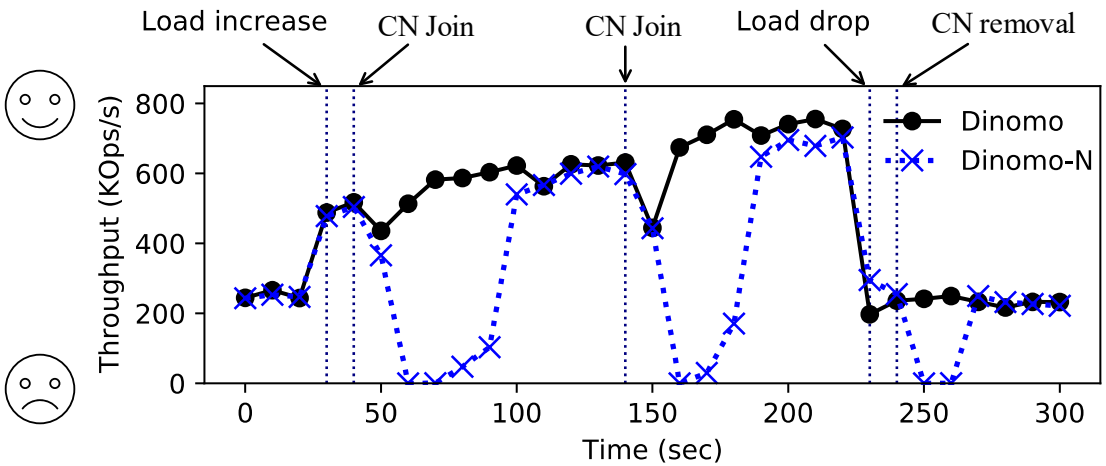
- DINOMO scales to 16 CNs, but Clover does not beyond 4 CNs
- With 16 CNs, DINOMO outperforms Clover upto 10x



Evaluation

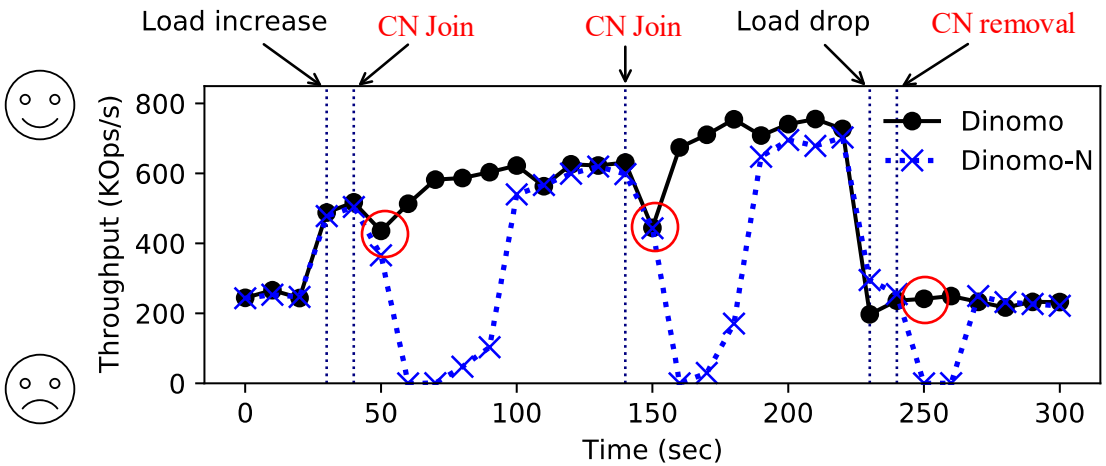
- How does DINOMO fare against the state-of-the-art in terms of performance and scalability?
- How elastic and responsive is DINOMO while handling changes in workloads?

Elasticity


Load increase
CN Join
CN Join
Load drop
CN removal
Throughput (KOps/s)
800
600
400
200
0
0
50
100
150
200
250
300
Time (sec)
Dinomo
Dinomo-N

Elasticity

- DINOMO: Brief throughput dips when adding/removing CNs



Elasticity

- DINOMO: Brief throughput dips when adding/removing CNs
- DINOMO-N: Throughput dips for 20-40 seconds due to expensive data reorganization