Georgia Tech

Mnemo:

Boosting Memory Cost Efficiency in Hybrid Memory Systems

Thaleia Dimitra Doudali

Ada Gavrilovska

Problem Space

Cost of memory dominates in the cloud.

Georgia Tech

Big Data Analytics

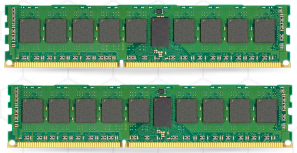
In-Memory Key-Value Stores

aws

Microsoft Azure

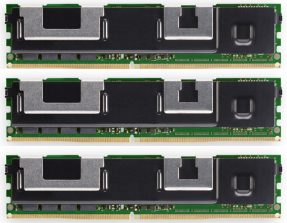
Memory Optimized Virtual Machines with up to **few TBs** of memory.

Memory Technology Options



60% - 85% of the hourly VM cost!

Volatile Memory (DRAM)
e.g. DDR-4



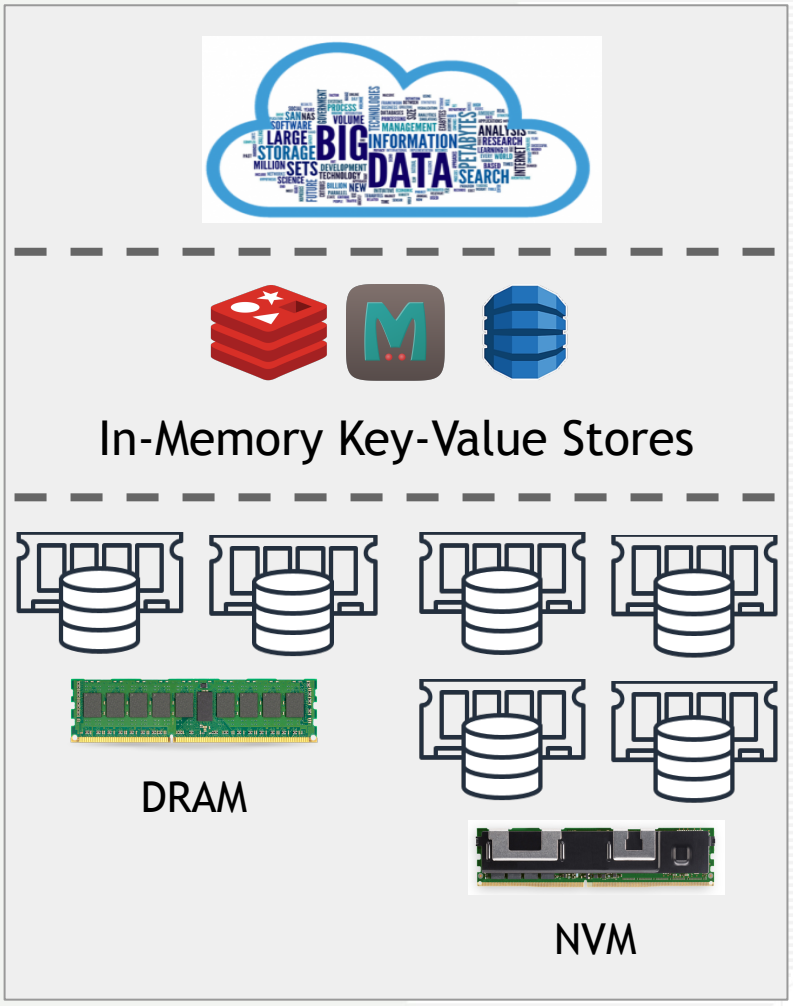
cheaper but slower to access than DRAM

Non Volatile Memory (NVM)
e.g. **Intel Optane DC Persistent Memory**

(Will be available in Google Cloud instances with capacity up to 7 TBs)

Problem Statement

Capacity Sizing of the Hybrid Memory Components.



Facts:

- Future clouds will feature hybrid memory components.
- These components have different cost and access latencies.

Problem:

What is the ideal *capacity ratio* between the hybrid memory components?

Goals:

- Maximize system's cost efficiency.
- Keep performance guarantees.

Existing Solutions

Data Tiering over *fixed* capacities.

In-Memory Key-Value Stores

custom API → modified code → Binary Instrumentation → code execution → Memory Access Trace

high overhead!

representative key access pattern

Profiling Tool

Profiling Tool decides:

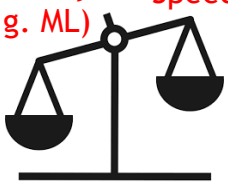
1. Which keys are hot/cold?

Track every single memory access per key.

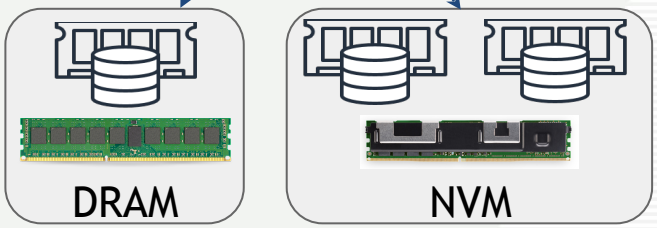
2. Which keys to place/move to DRAM?

Estimate performance benefit from DRAM placement.
Use analytical model with performance baselines.

Accuracy (e.g. ML) — Profiling Speed

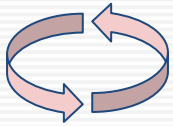


hot keys — cold keys



fixed hardware capacities

Output: Data Tiering across DRAM and NVM.



If different DRAM:NVM capacity ratio:

1. Run profiling again to get a new tiering.
2. Run application and observe change in performance.

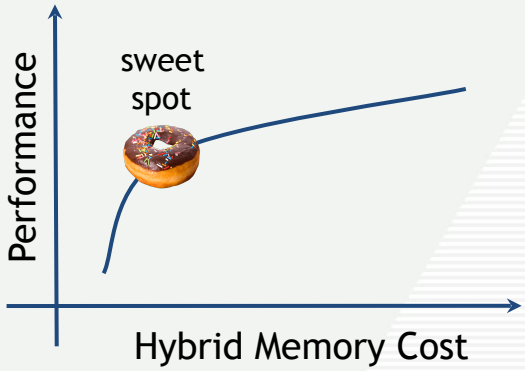
***Problem**: Currently no way to know how much DRAM vs NVM to use.*

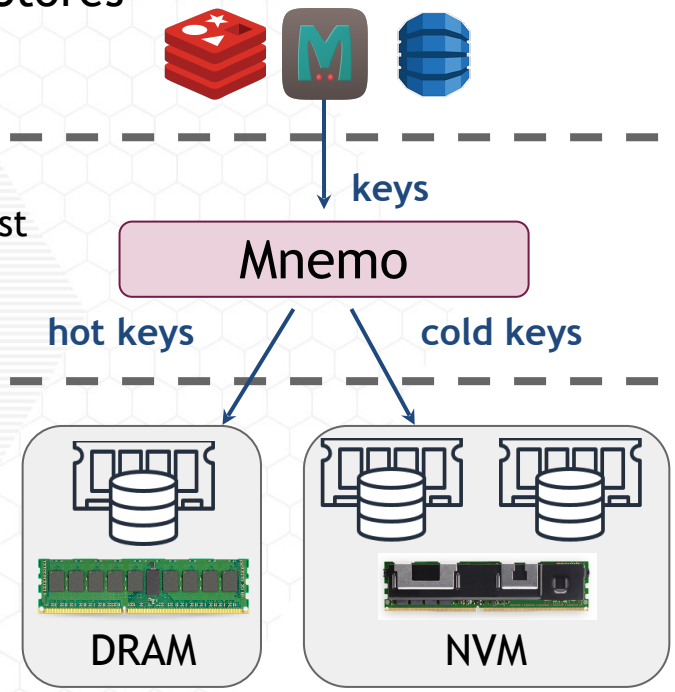
Solution Preview

Mnemo

Offline Profiling Tool for in-memory Key-Value Stores

- **Data Tiering**
  Which keys should be allocated in DRAM vs NVM?
- **Capacity Sizing**
  How many keys should be allocated in DRAM, so that application performance remains high, but memory cost remains low?



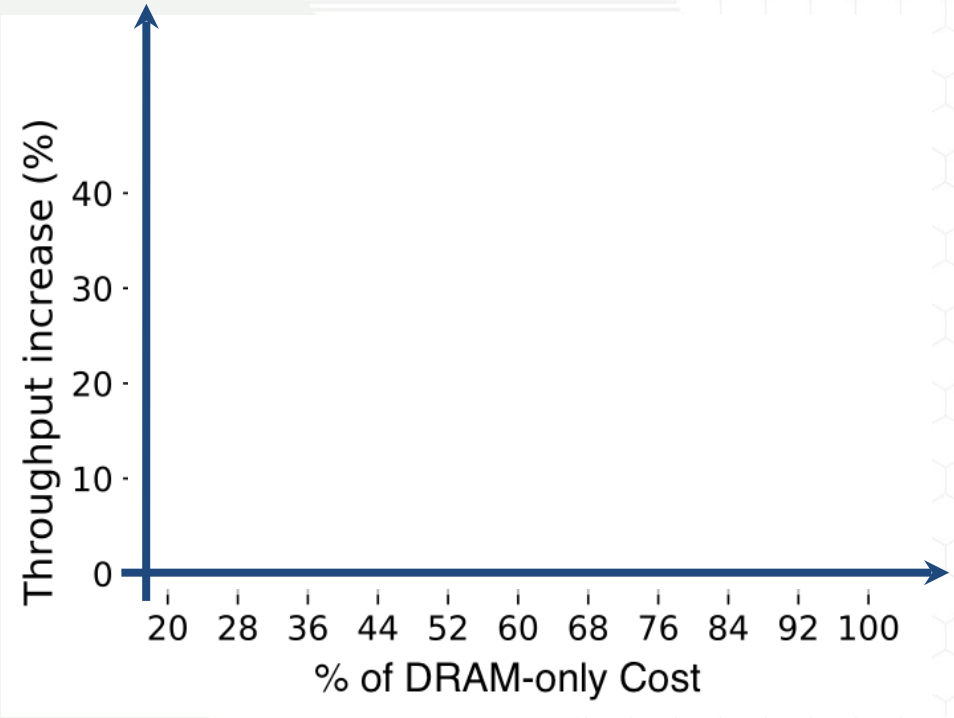


How much capacity of each memory type?

Mnemo *quickly* generates an *accurate* trendline of application performance for incremental DRAM to NVM capacity ratio, thus memory cost.

Motivation

Which parameters affect key-value store performance over hybrid memory systems?



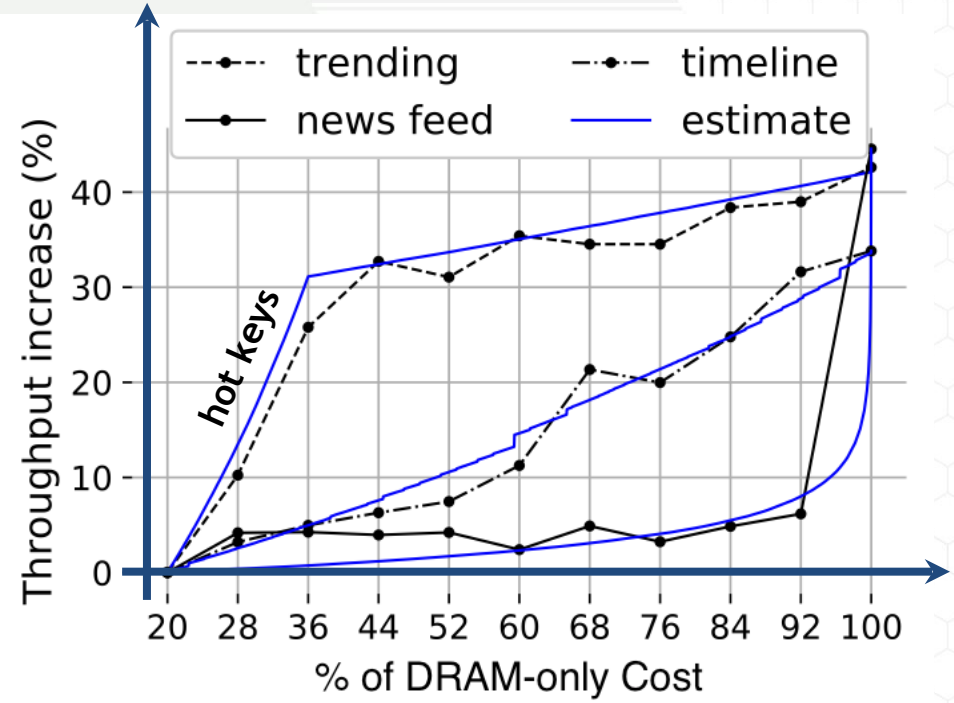
Application Performance Increase

More keys in DRAM
Increasing DRAM capacity
Increasing memory cost

Motivation

Which parameters affect key-value store performance over hybrid memory systems?

Application Performance Increase



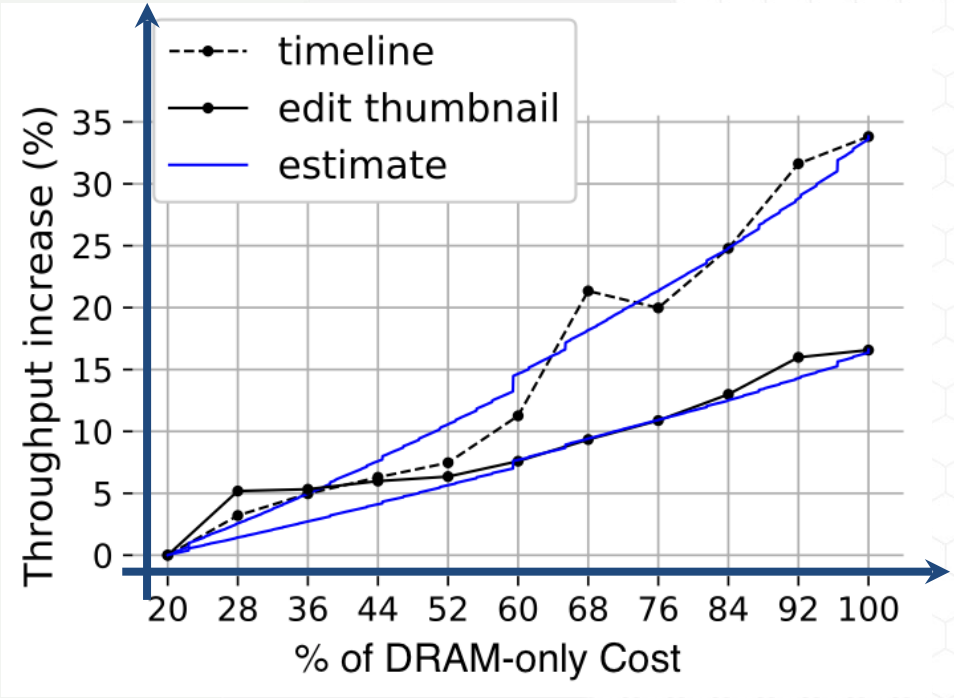
More keys in DRAM
Increasing DRAM capacity
Increasing memory cost

Varying key access pattern

Motivation

Which parameters affect key-value store performance over hybrid memory systems?

Application Performance Increase



more reads

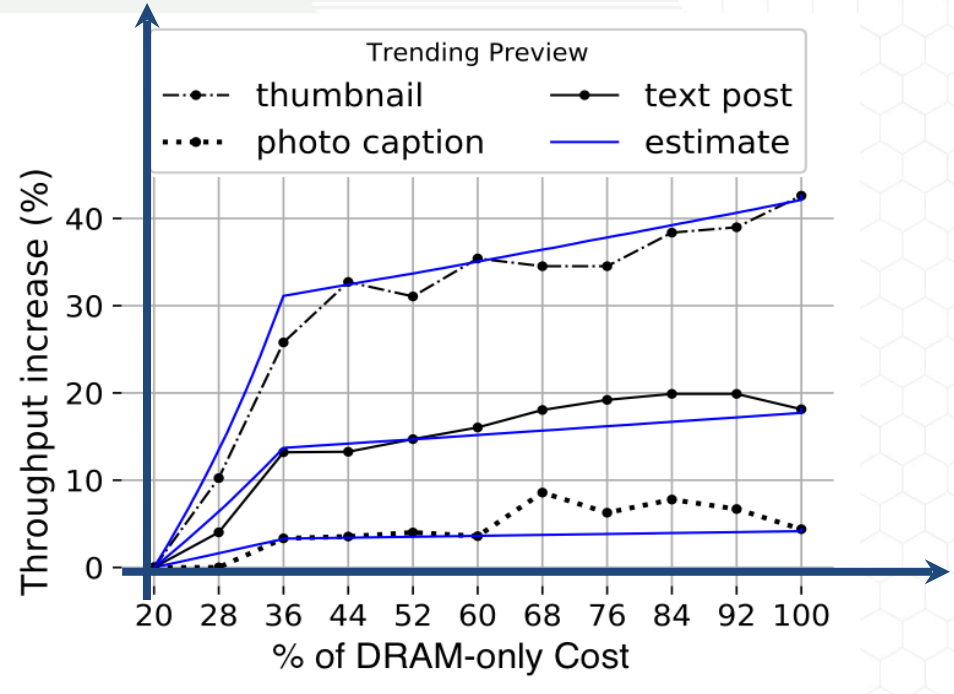
More keys in DRAM
Increasing DRAM capacity
Increasing memory cost

Varying read:write request ratio

Motivation

Which parameters affect key-value store performance over hybrid memory systems?

Application Performance Increase



bigger values

More keys in DRAM
Increasing DRAM capacity
Increasing memory cost

Varying key-value size

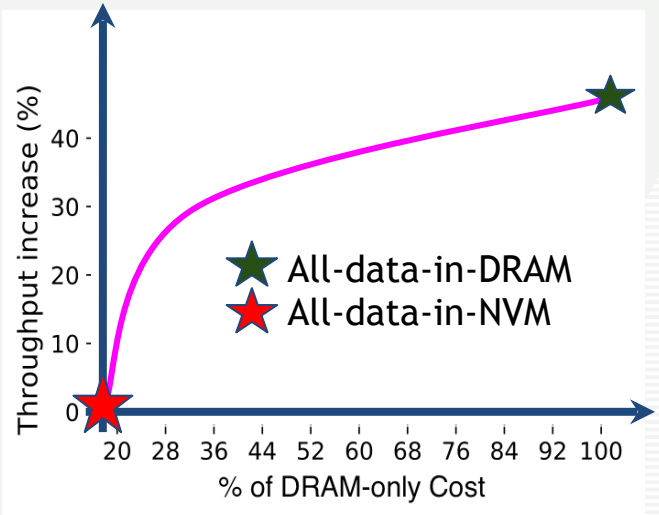
Motivation
Takeaways

Which parameters affect key-value store performance over hybrid memory systems?

- *Key access pattern*
- *Read:Write requests*
- *Key-Value sizes*

These parameters determine the **shape** of the curve.

The **height** of the curve also depends on the latency difference in accessing DRAM vs NVM.



Takeaways:

In order to estimate performance we'll need to capture:

- The workload parameters.
- Performance baselines for DRAM vs NVM.

Mnemo Usage

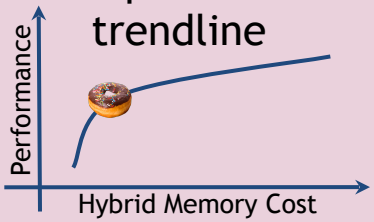
User Input

Provides just a workload description

- ✅ Minimal User Effort
- ✅ No Application Modifications

Mnemo

Generates the performance-to-cost trendline



- ✅ Fast
- ✅ Lightweight
- ✅ Accurate

Output to User

User chooses the sweet spot that brings the desired performance under his cost budget.

- ✅ Maximum Cost Efficiency
- ✅ Desired performance levels

Detailed Design

IP of server with DRAM
IP of server with NVM

DRAM-NVM
price difference

Key Request Pattern
Key-Value Sizes

Sensitivity Engine
Workload Execution over DRAM-only and NVM-only.

Pattern Engine
Key prioritization for DRAM allocation.
weight = # accesses / key-value size

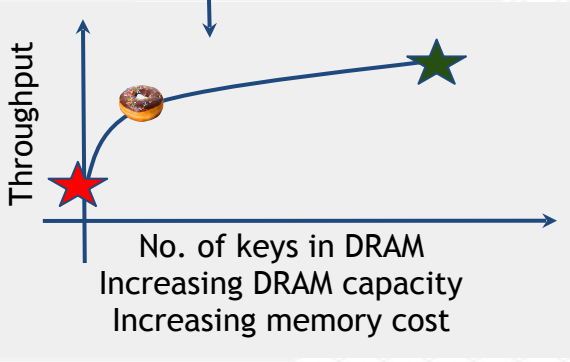
Performance baselines

Keys ordered by weight

Estimate Engine
Generates performance estimate across the key space.

All-data-in-DRAM
All-data-in-NVM

Read Time Diff (NVM-DRAM)
Write Time Diff (NVM-DRAM)



User choice

sweet spot

Throughput Estimate = Reads x ▲ + Writes x △ / Reads + Writes

Placement Engine
Populates the servers with the selected data tiering.

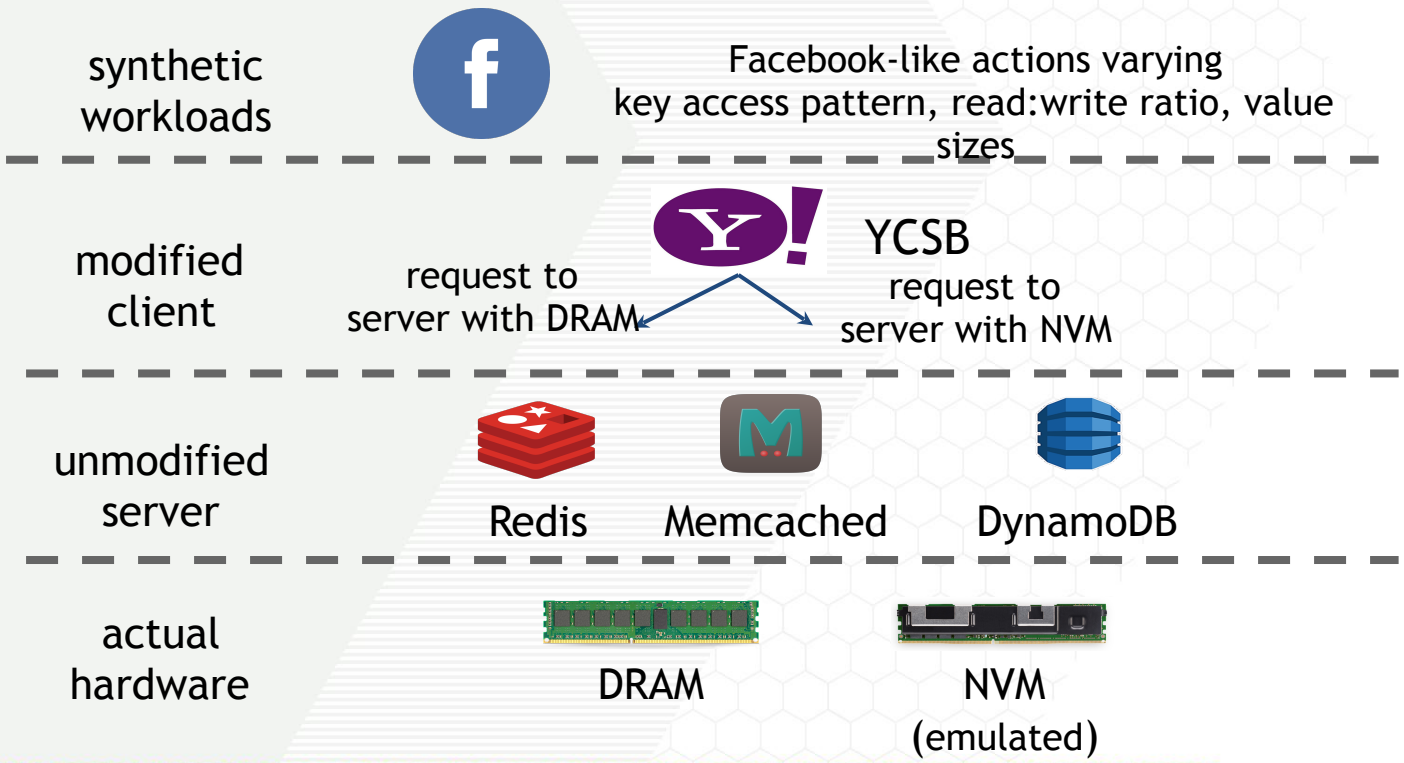
Key-Value pairs

Evaluation Methodology

Metrics:

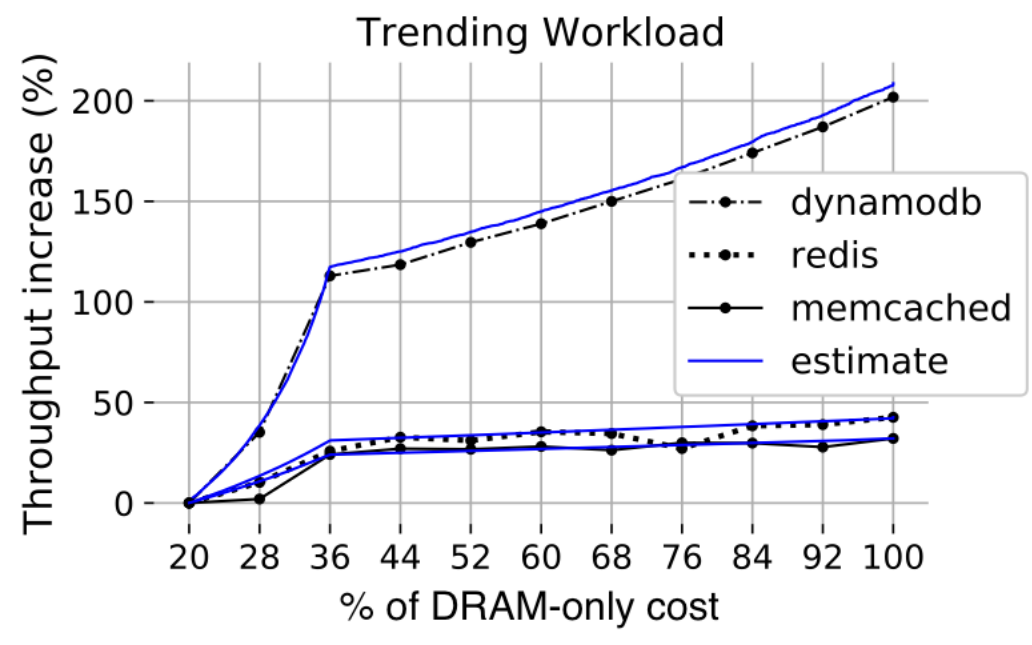
Estimate Accuracy ? Cost Efficiency ? Profiling Overhead ?

Implementation:



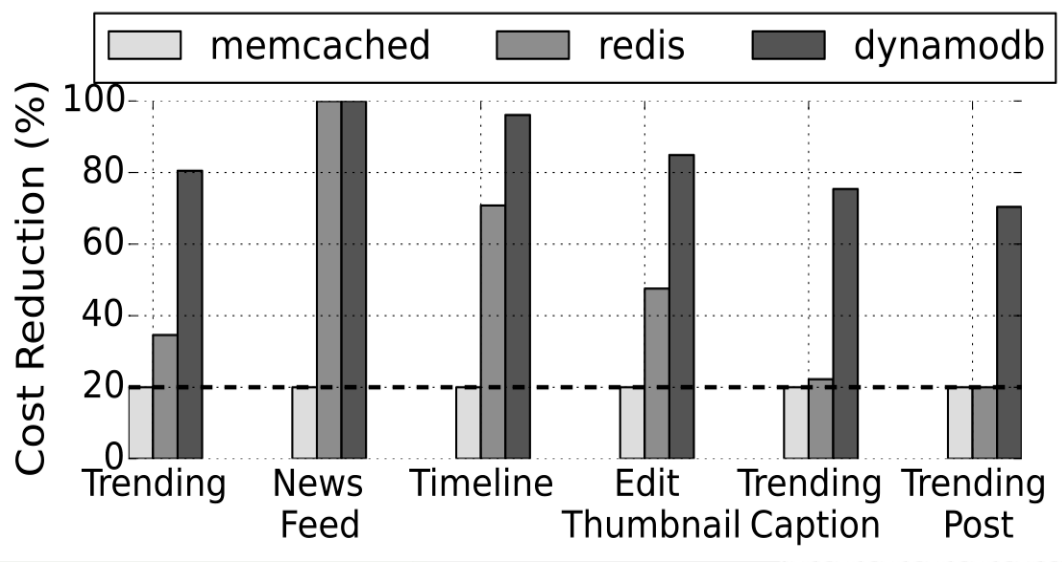
Evaluation Results

Mnemo successfully captures the trade-off between performance and cost



Evaluation Results

Mnemo allows for significant cost reductions

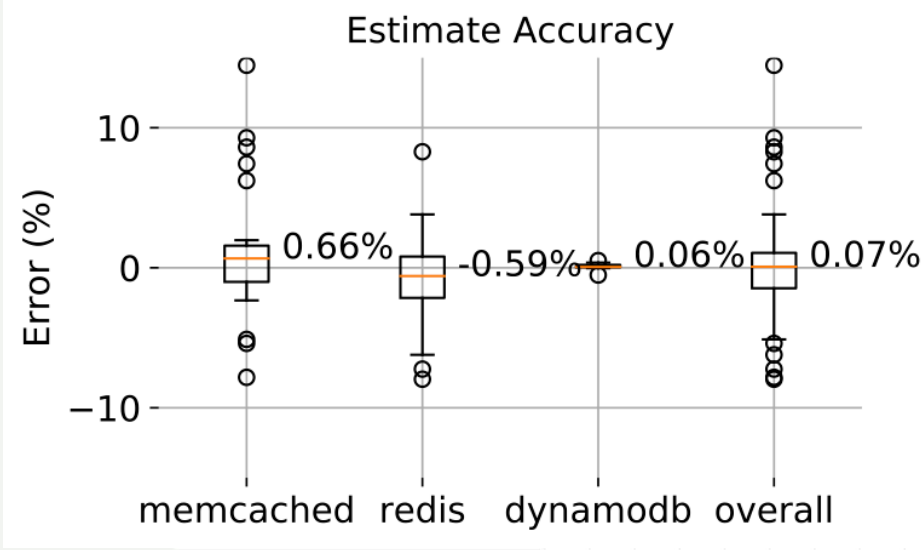


The higher the better ⬆️

For chosen performance sweet spot to be 10% slowdown from all-data-in-DRAM.

Evaluation Results

Mnemo is extremely accurate



Profiling Overhead

In-Memory Key-Value Stores

No code modifications.
Workload description.

✅ Lightweight

key access pattern

Mnemo

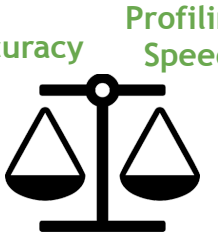
Profiling Tool decides:

1. Which keys are hot/cold?

Extract access information from workload description.

2. Which keys to place/move to DRAM?

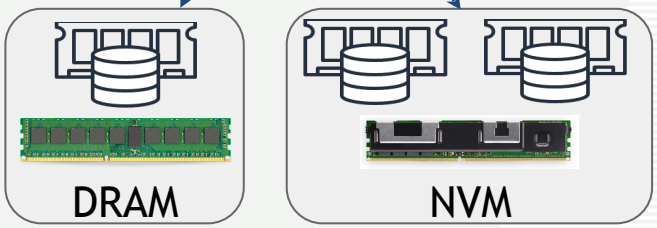
Get performance baselines through workload execution.
Use analytical model with performance baselines.



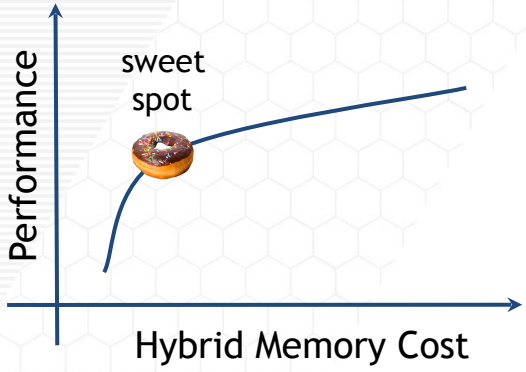
✅ Fast

hot keys

cold keys

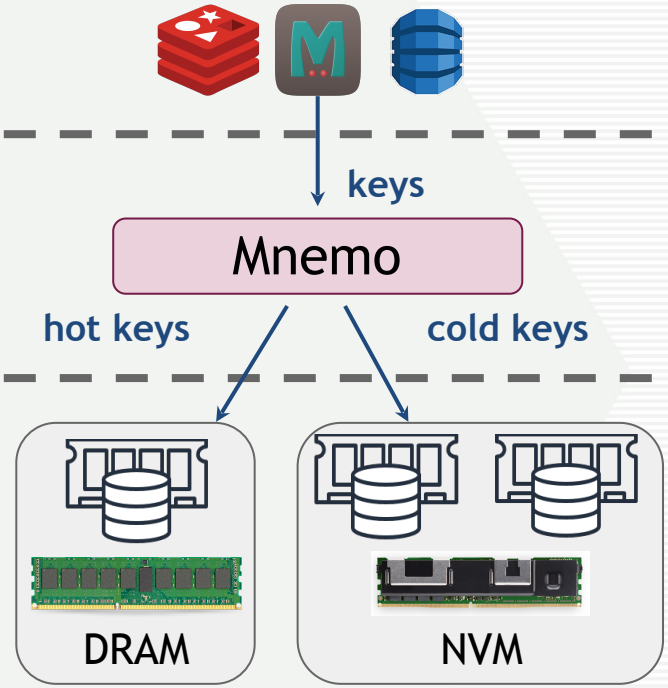


How much capacity of each memory type?

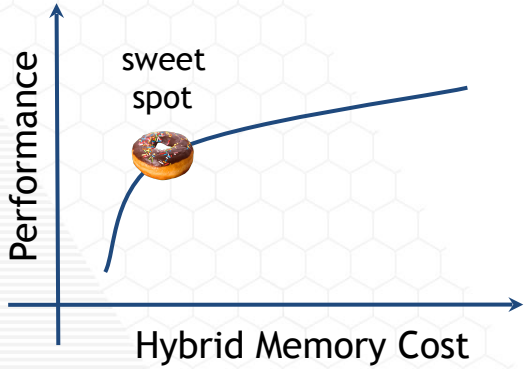


✅ Accurate

Summary



How much capacity of each memory type?



- ✅ Fast
- ✅ Lightweight
- ✅ Accurate
- ✅ Open source

https://github.com/Thaleia-DimitraDoudali/mnemo