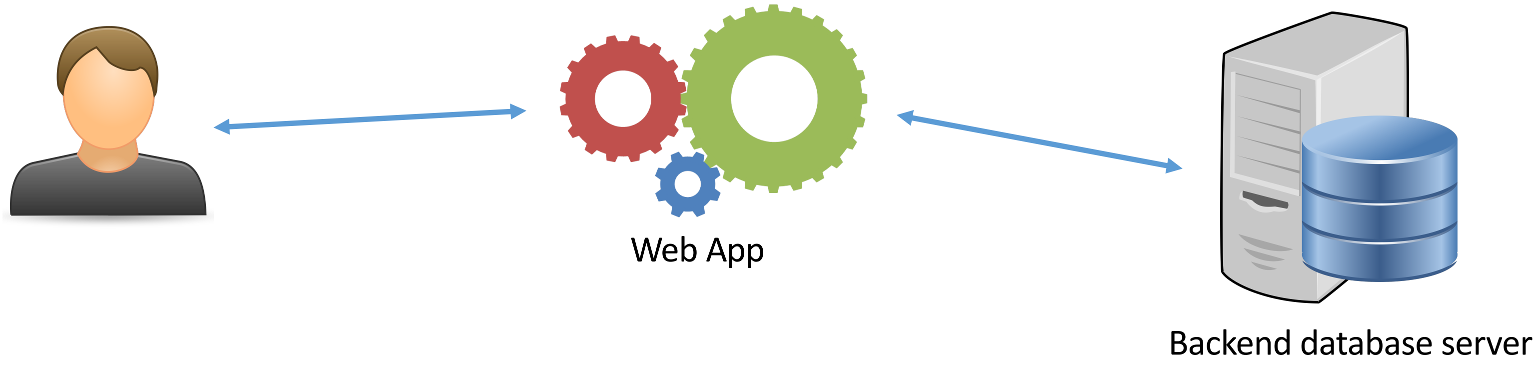
ElCached: Elastic Multi-Level Key-Value Cache

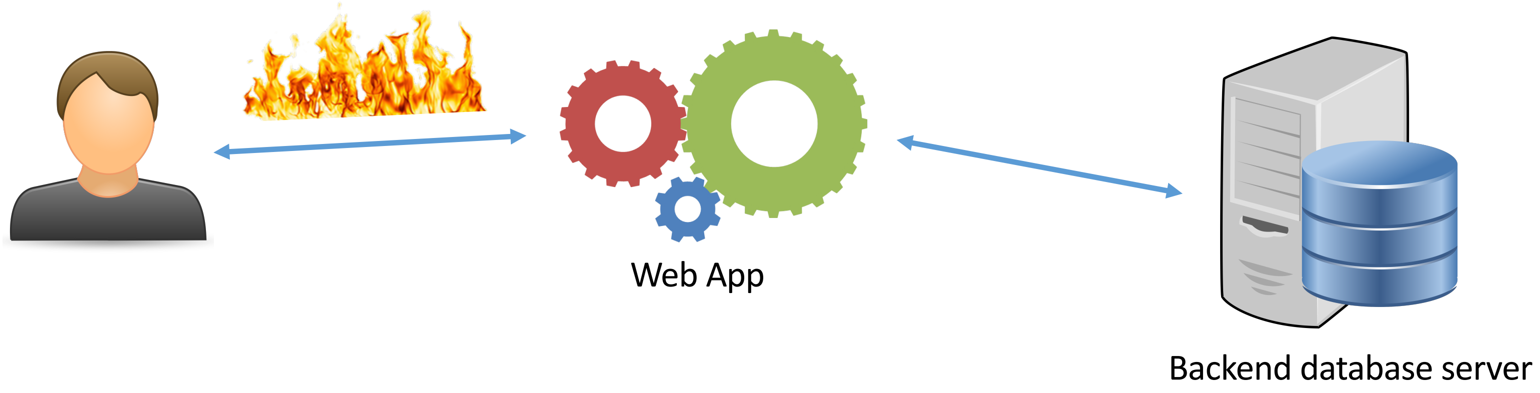
Rahman Lavaee,

Stephen Choi, Yang-Suk Kee,

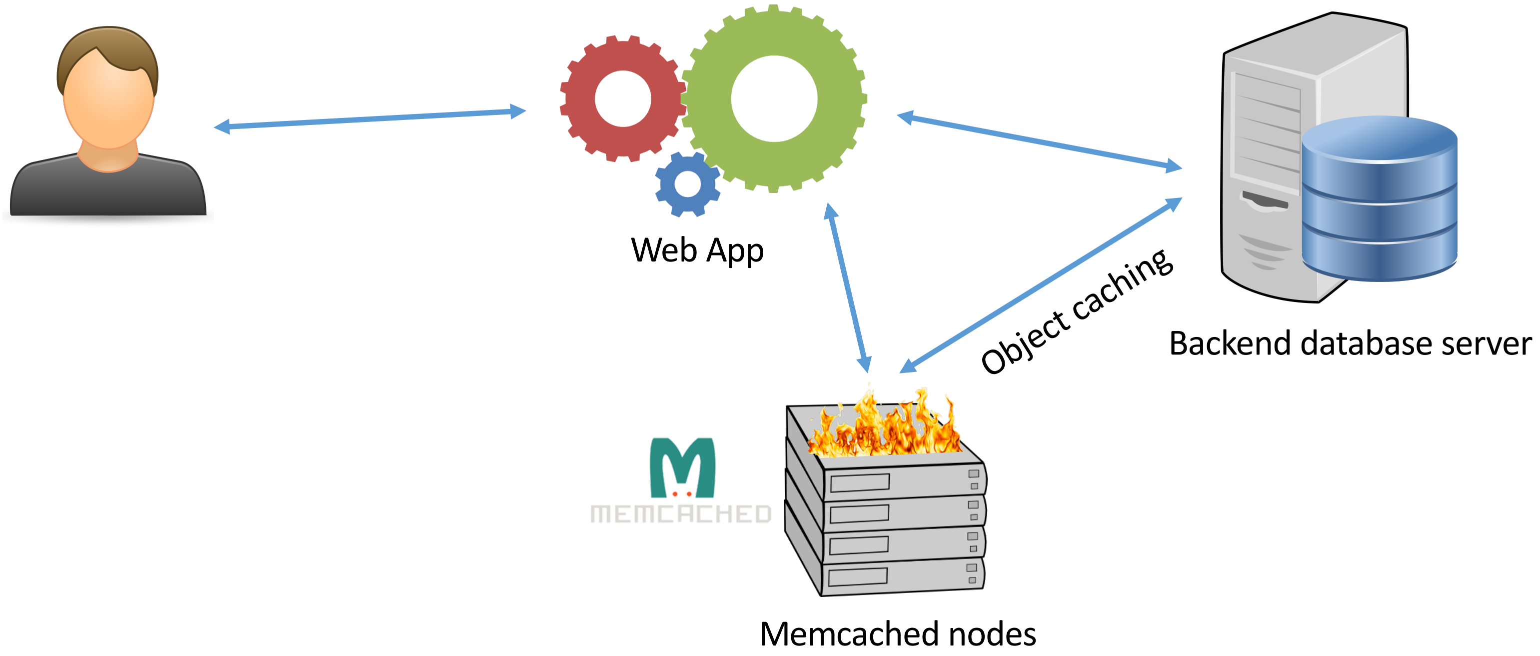
Chen Ding

November 1st, Savannah, GA

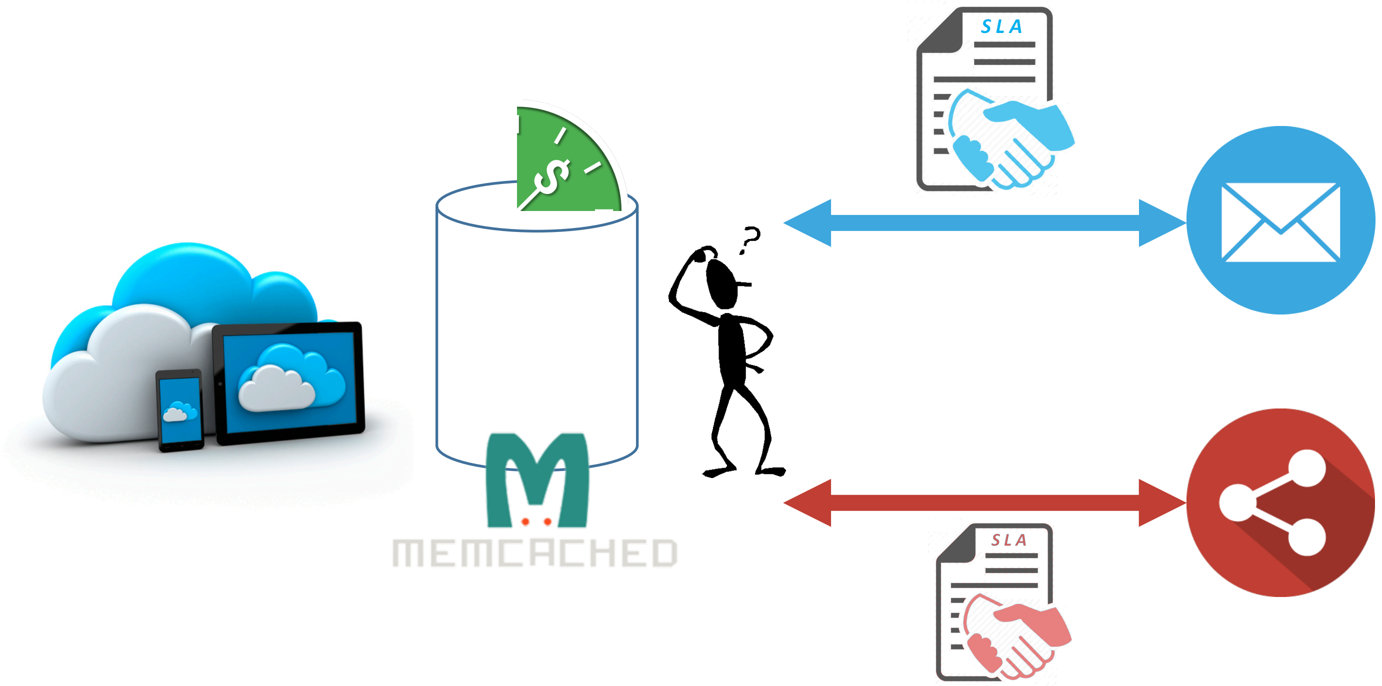

Web App
Backend database server


Web App
Backend database server

Web Caching

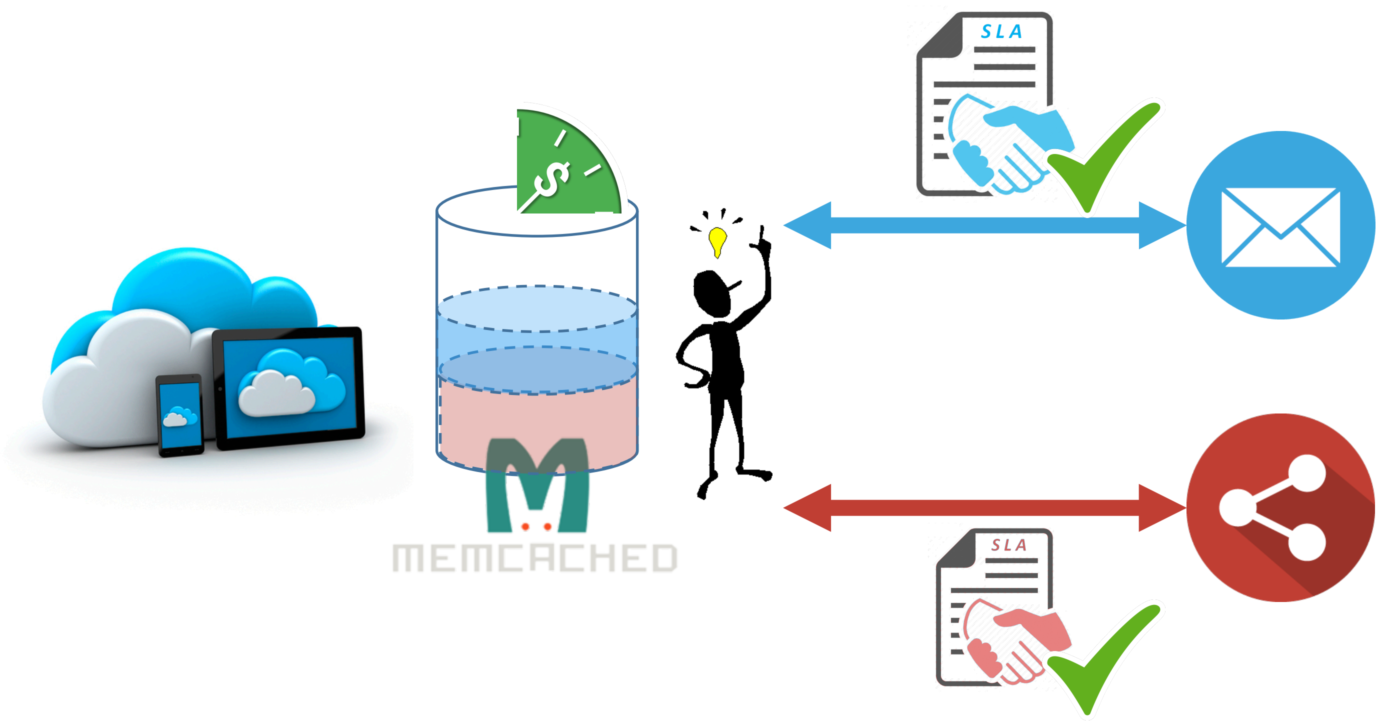

Web App
Object caching
Backend database server
MEMCACHED
Memcached nodes

Resource Provisioning for Memcached



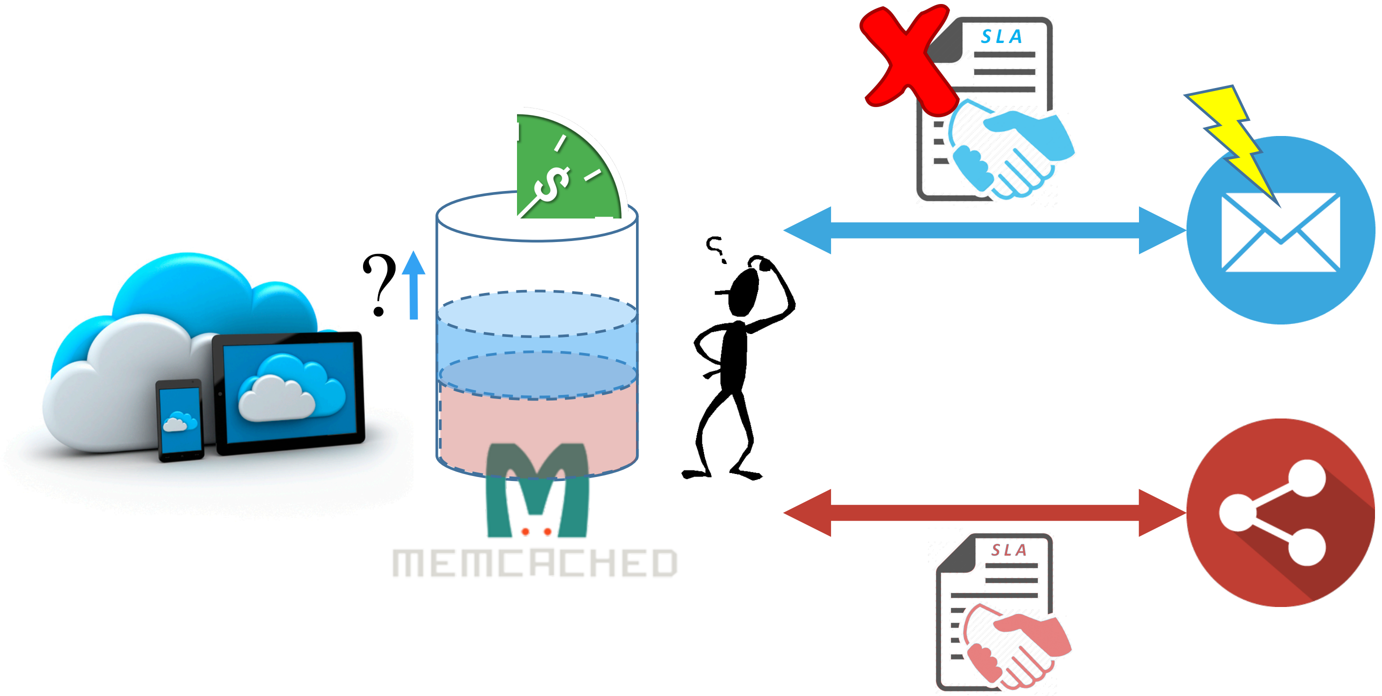
Resource Provisioning for Memcached

- The service provider must wisely allocate the resource to guarantee each tenant's SLA, while minimizing TCO.



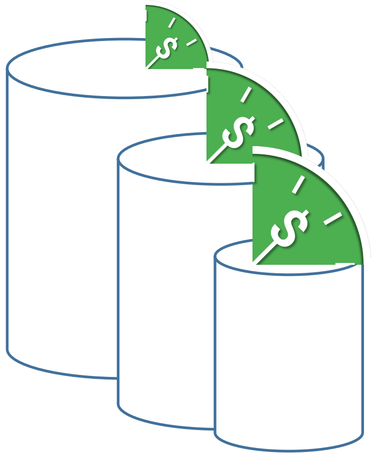
Elastic Resource Provisioning

- Optimal resource provisioning requires **elasticity**
  - capability to adapt to workload changes by dynamic resource provisioning.



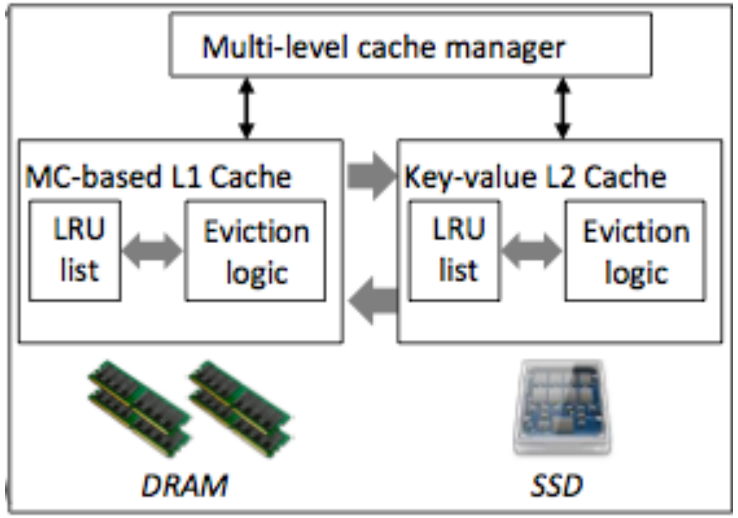
Multitenant Resource Provisioning

- The problem becomes more complex as
  - more tenants are added to the system.
  - more web caching layers are used.



MlCached [HotCloud'16]

- A multi-level **key-value** caching system.
  - L1: DRAM-based Memcached
  - L2: exclusive NAND-flash-based key-value cache (SSD).

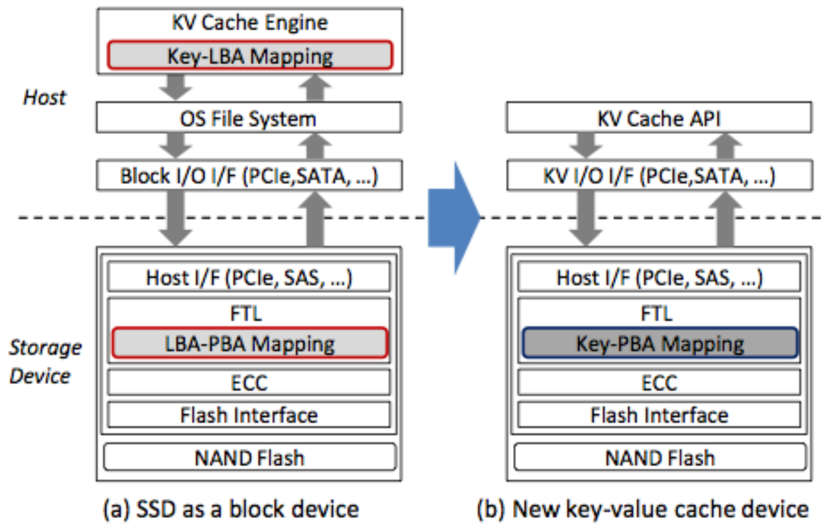


Mlcached

- MlCached implements direct key-to-PBA mappings on SSD.
  - Independent resource provisioning
- In this work, we extend MlCached by adding the elasticity feature.

Independent Resource Provisioning in MlCached

- MlCached implements direct key-to-PBA mappings on SSD.
- This removes the need for storing redundant key-to-LBA mapping tables in memory



Performance Model

Latency Model

$$Lat = l_m + l_s . \mathrm{M_m} + l_{db} . M_s$$

l_m	Latency of Memcached server
l_s	Latency of SSD
l_{db}	Latency of backend DB server

$\mathrm{M_m}$	Miss rate of Memcached
M_s	Miss rate of SSD

Cost Model

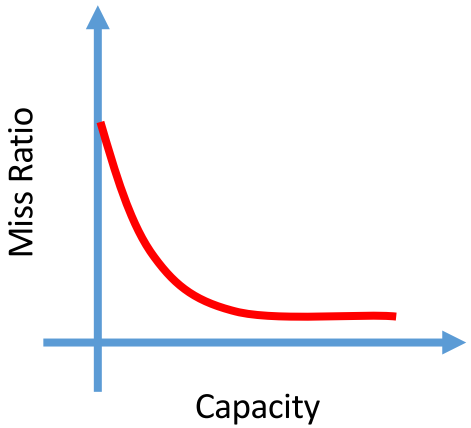
$$Cost = c_m . p_m + c_s . p_s$$

c_m	Size of DRAM
c_s	Size of SSD

p_m	Price per unit of DRAM
p_s	Price per unit of SSD

Latency Based on Miss Ratio Curve

- The key to optimal resource provisioning is to find the miss ratio curve (MRC).



$$Lat = l_m + l_s.mr(\mathrm{c_m}) + l_{db}.mr(c_s)$$

How to compute the MRC?

Reuse Distance

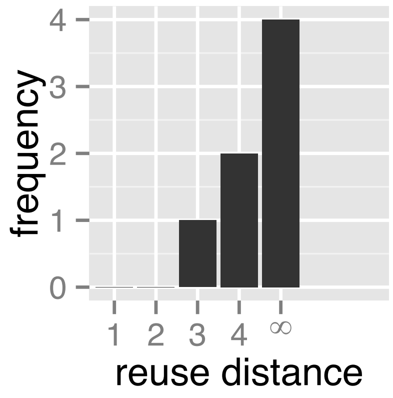
- We use the reuse distance theory to compute the miss ratio curve.
- Reuse distance is the number of distinct memory locations accessed between two consecutive uses of the same memory location.

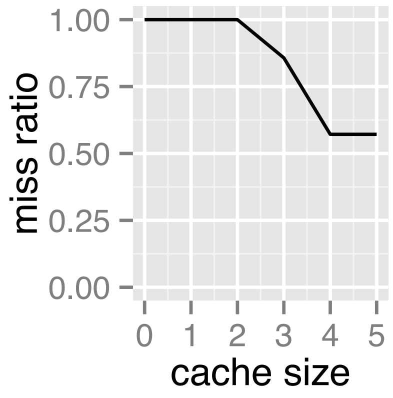
a	b	c	d	b	a	c
∞	∞	∞	∞	3	4	4

Reuse Distance Histogram

- The reuse distance information is best represented by the reuse distance histogram, which shows the frequency for every reuse distance.
- The MRC can be computed from this histogram.

a	b	c	d	b	a	c
∞	∞	∞	∞	3	4	4



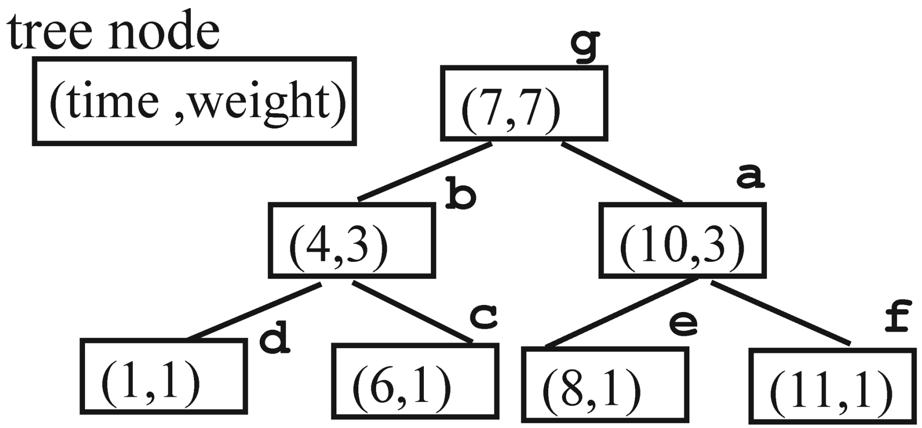


Reuse Distance Computation

- Olken tree [Olken 1981]
- Approximate reuse distance [Ding+ 2001]
- Footprint estimation [Xiang+ 2011]
- Stack counters [Wires+ 2014]

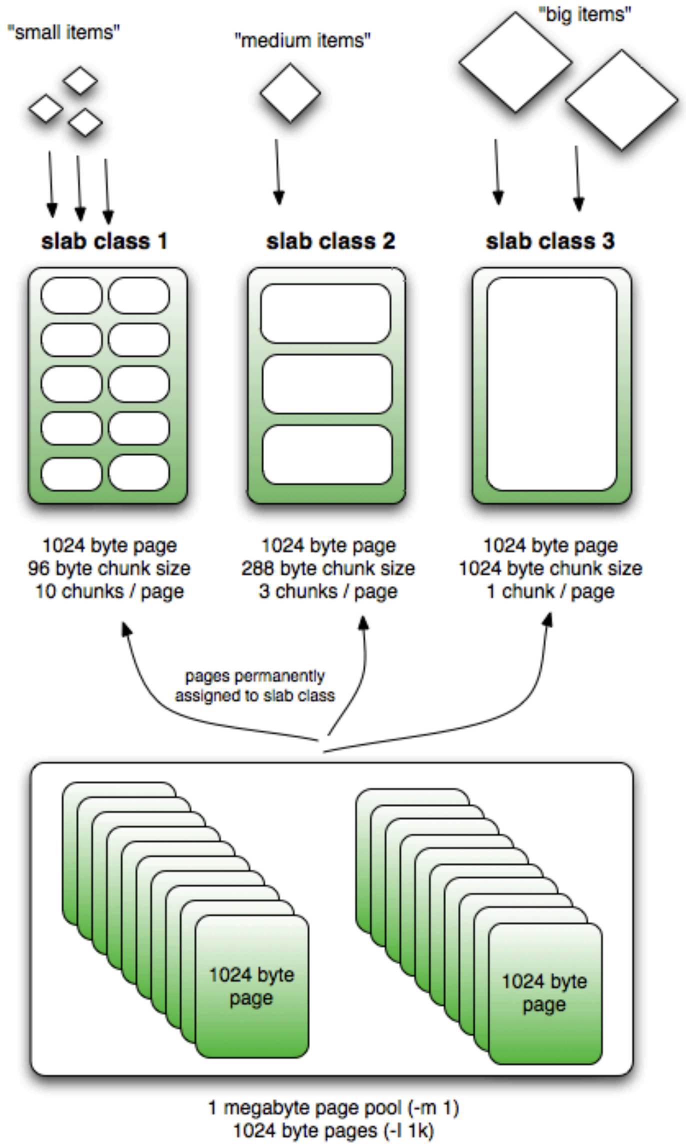
Reuse Distance Computation

- Olken tree [Olken 1981] ✓
- Approximate reuse distance [Ding+ 2001]
- Footprint estimation [Xiang+ 2011]
- Stack counters [Wires+ 2014]



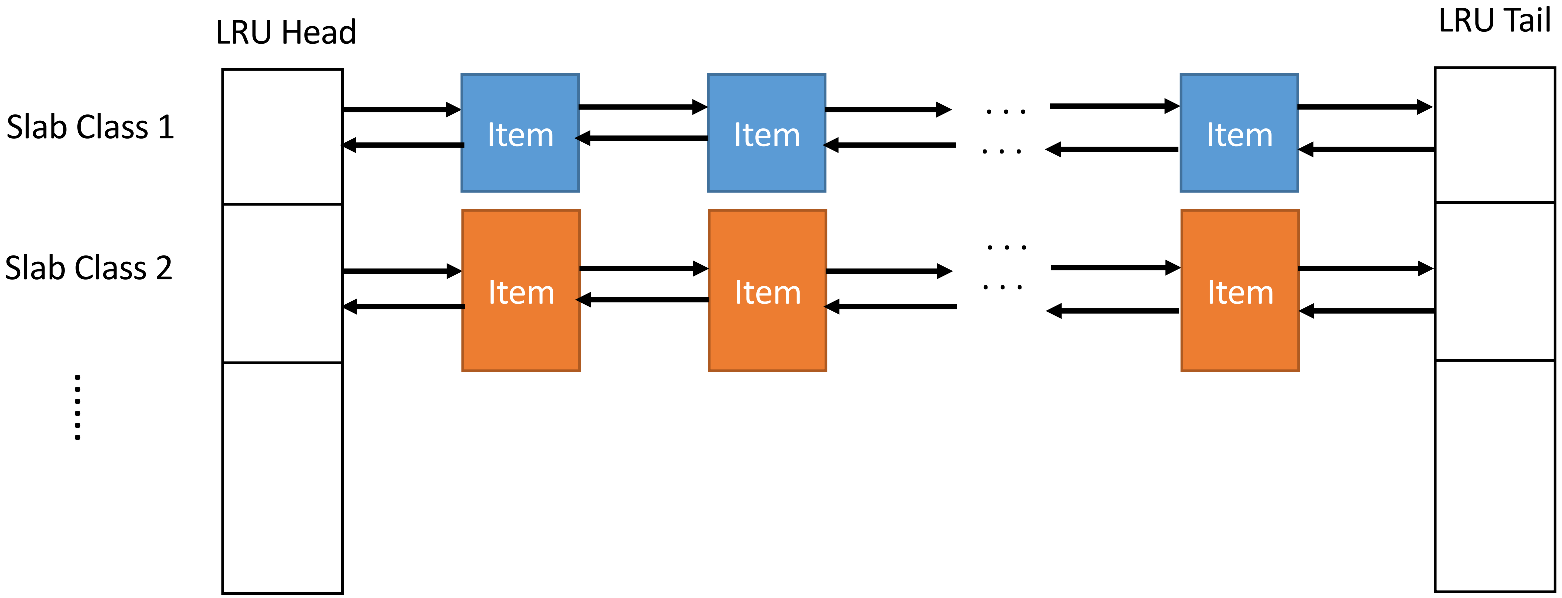
Reuse Distance for Memcached

- Memcached distributes items among different slab classes, according to their sizes.
- Slab allocation is done during the cold start.



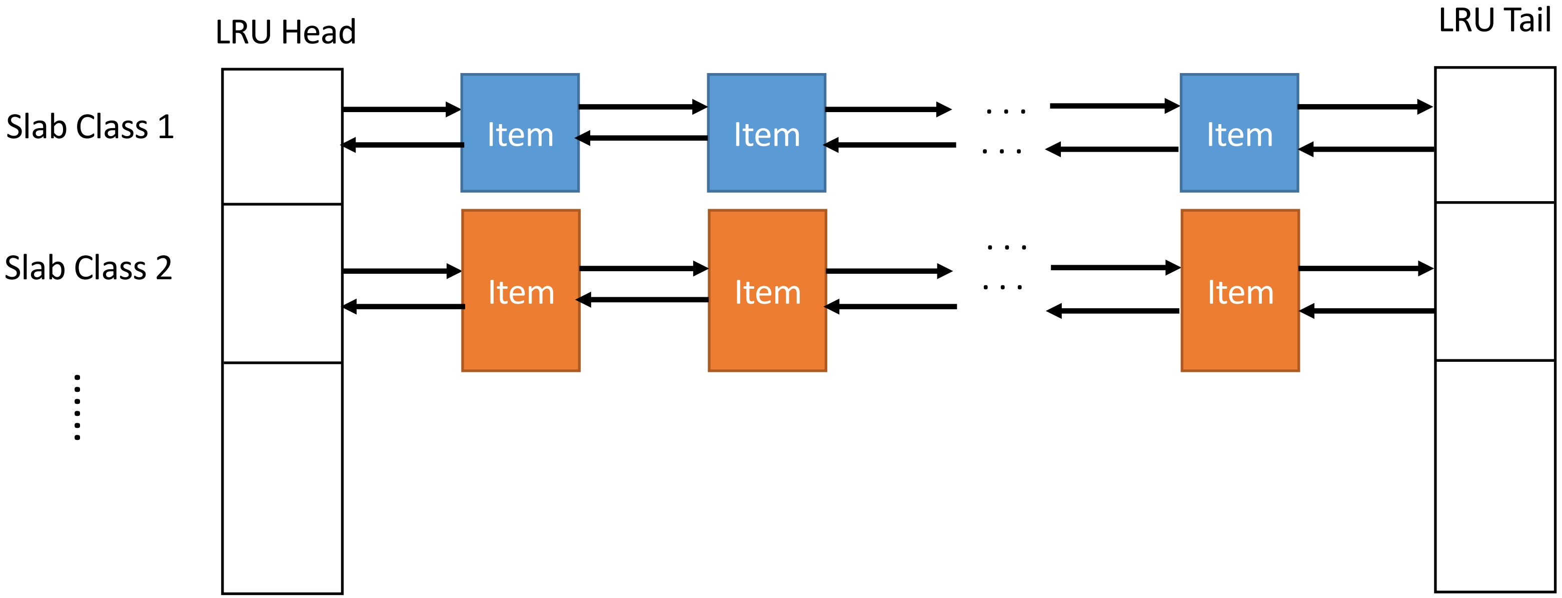
LRU Replacement in Memcached

- Once the Memcached system reaches its memory limit, LRU replacement is done separately for every slab class.



Slab-Aware Reuse Distance Profiling

- Rather than analyzing the whole Memcached system in a single reuse distance model, we model each slab class separately.
- We compose the MRCs from different slab classes.



How To Solve
Resource Provisioning?

Resource Provisioning as a Linear Program

- The resource provisioning problem can be described in one of the two ways.
  - Minimize *Cost* such that *Lat ≤ SLA*.
  - Minimize *Lat* such that *Cost ≤ TCO*.

Resource Provisioning as a Linear Program

- The resource provisioning problem can be described in one of the two ways.
  - Minimize *Cost* such that *Lat ≤ SLA*.
  - Minimize *Lat* such that *Cost ≤ TCO*.

Linear Program?

Resource Provisioning as a Linear Program

- The resource provisioning problem can be described in one of the two ways.
  - Minimize *Cost* such that *Lat ≤ SLA*.
  - Minimize *Lat* such that *Cost ≤ TCO*.

Linear Program?

- Cost is already linear in terms of DRAM/SSD capacities. ✓

Resource Provisioning as a Linear Program

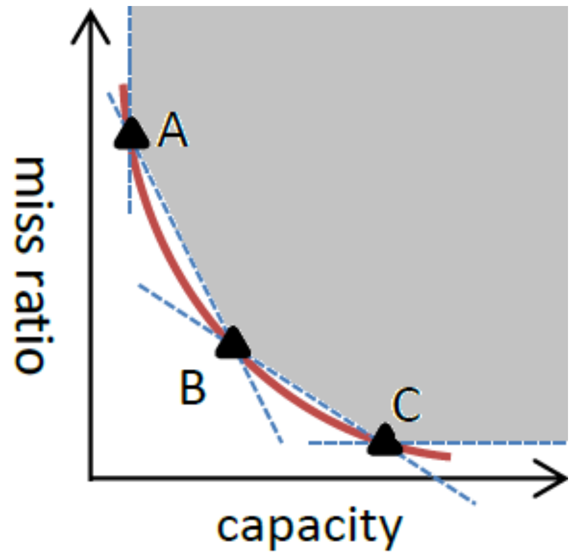
- The resource provisioning problem can be described in one of the two ways.
  - Minimize *Cost* such that *Lat ≤ SLA*.
  - Minimize *Lat* such that *Cost ≤ TCO*.

Linear Program?

- Cost is already linear in terms of DRAM/SSD capacities. ✓
- Latency is linear only in terms of the miss ratio function. ✗

Resource Provisioning as a Linear Program

- We observe that the miss ratio curves in our workloads are always convex.
- We formulate the miss ratio curve using linear constraints.



Evaluation

- We compare ElCached against a proportional approach that fixes the ratio between DRAM and SSD capacities to 1:4 (Pareto principle).

Evaluation

- We compare ElCached against a proportional approach that fixes the ratio between DRAM and SSD capacities to 1:4 (Pareto principle).

cost ($/GB)		latency	
DRAM	10	Memcached	100μs
SSD	0.68	KVD	200μs
		Back-end DB	10ms

Evaluation

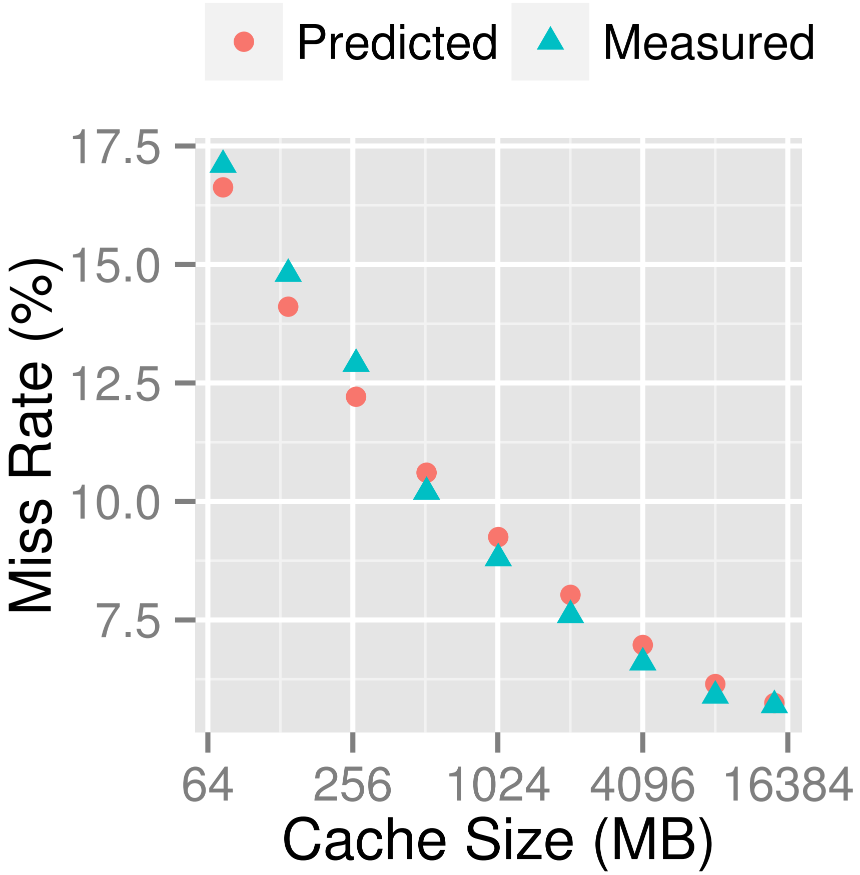
- We compare ElCached against a proportional approach that fixes the ratio between DRAM and SSD capacities to 1:4 (Pareto principle).

cost ($/GB)		latency	
DRAM	10	Memcached	$100\mu s$
SSD	0.68	KVD	$200\mu s$
		Back-end DB	10ms

- Workloads:
  - Zipfian key distribution with $\alpha = 1.15$
  - Exponential key distribution with $\lambda = 10^{-6}$
  - Both workloads issue 800 million requests to a range of 4 billion keys.

Miss Ratio Prediction Accuracy

- Mean relative error on Zipfian workload: 4%

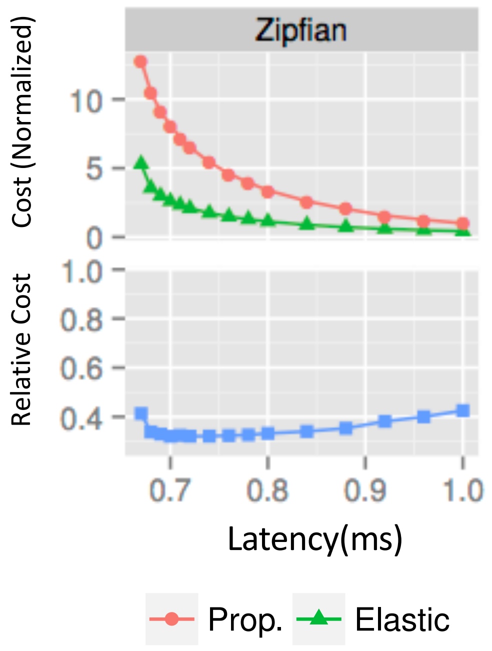


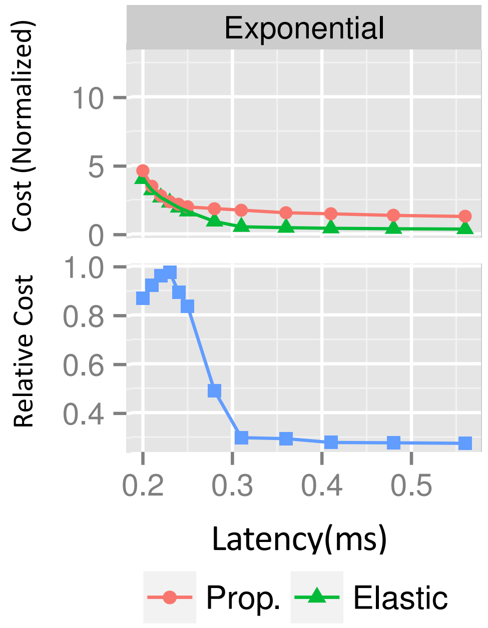
Experiment 1:
Elastic Resource Provisioning

- For each workload
  - For each latency limit, we find the minimum cost resource provisioning.

Experiment 1: Elastic Resource Provisioning

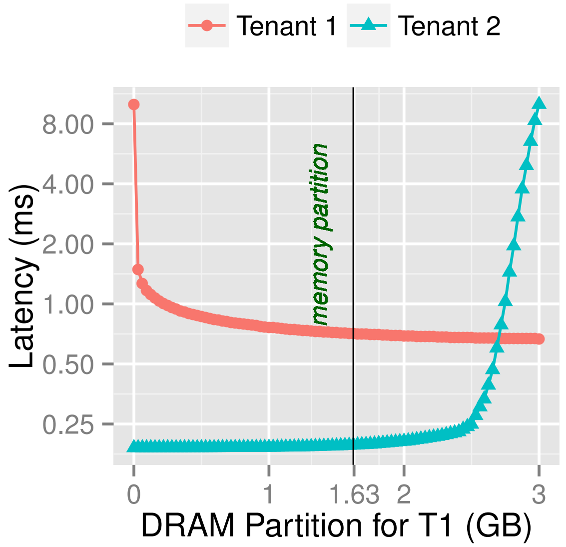
- For each workload
  - For each latency limit, we find the minimum cost resource provisioning.
- Elastic saves cost by around 60% for both workloads.





Experiment 2:
Multitenant Resource Provisioning

- First, we use the proportional scheme to partition a fixed 3GB memory between the two tenants.



Proportional

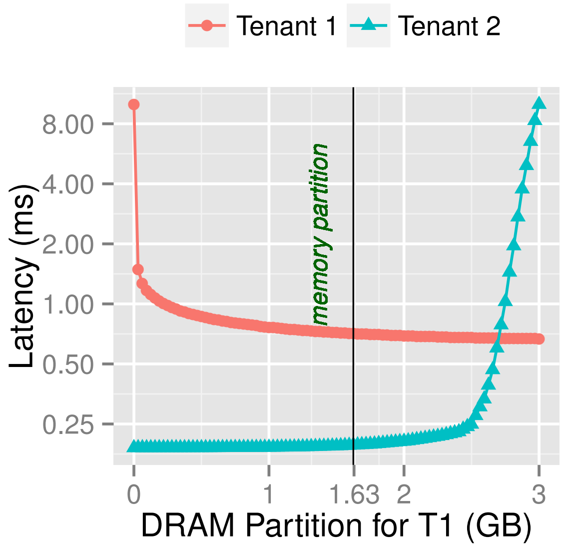
	T1	T2
DRAM	1.63	1.37
Lat.	0.71	0.20
Cost	7.20	5.41

Experiment 2:
Multitenant Resource Provisioning

- Then, we use ElCached to find the latency-optimal DRAM/SSD partitioning.

Elastic

	T1	T2
DRAM	0.46	1.17
Lat.	0.67	0.20
Cost	5.30	4.01



Proportional

	T1	T2
DRAM	1.63	1.37
Lat.	0.71	0.20
Cost	7.20	5.41

Experiment 2: Multitenant Resource Provisioning

- Elasticity improves
  - tenant 1's latency by 5%,
  - both tenants' cost by 26%, and
  - total memory consumption by 46%.

Summary

- ElCached is a multi-level key-value caching system.
- It uses a **reuse distance profiler** to estimate the miss rate curve across all capacity limits.
- It reduces the total **cost** by up to around 60% compared to a proportional scheme.
- Multi-tenant experiment indicates that we can improve **latency**, **cost**, and total **DRAM usage**, compared to the proportional scheme.

Thank You

Any Questions?