Faster Slab Reassignment in memcached

Daniel Byrne djbyrne@mtu.edu
Nilufer Onder nilufer@mtu.edu
Zhenlin Wang zlwang@mtu.edu

Department of Computer Science
Michigan Technological University

MEMSYS 2019

Background

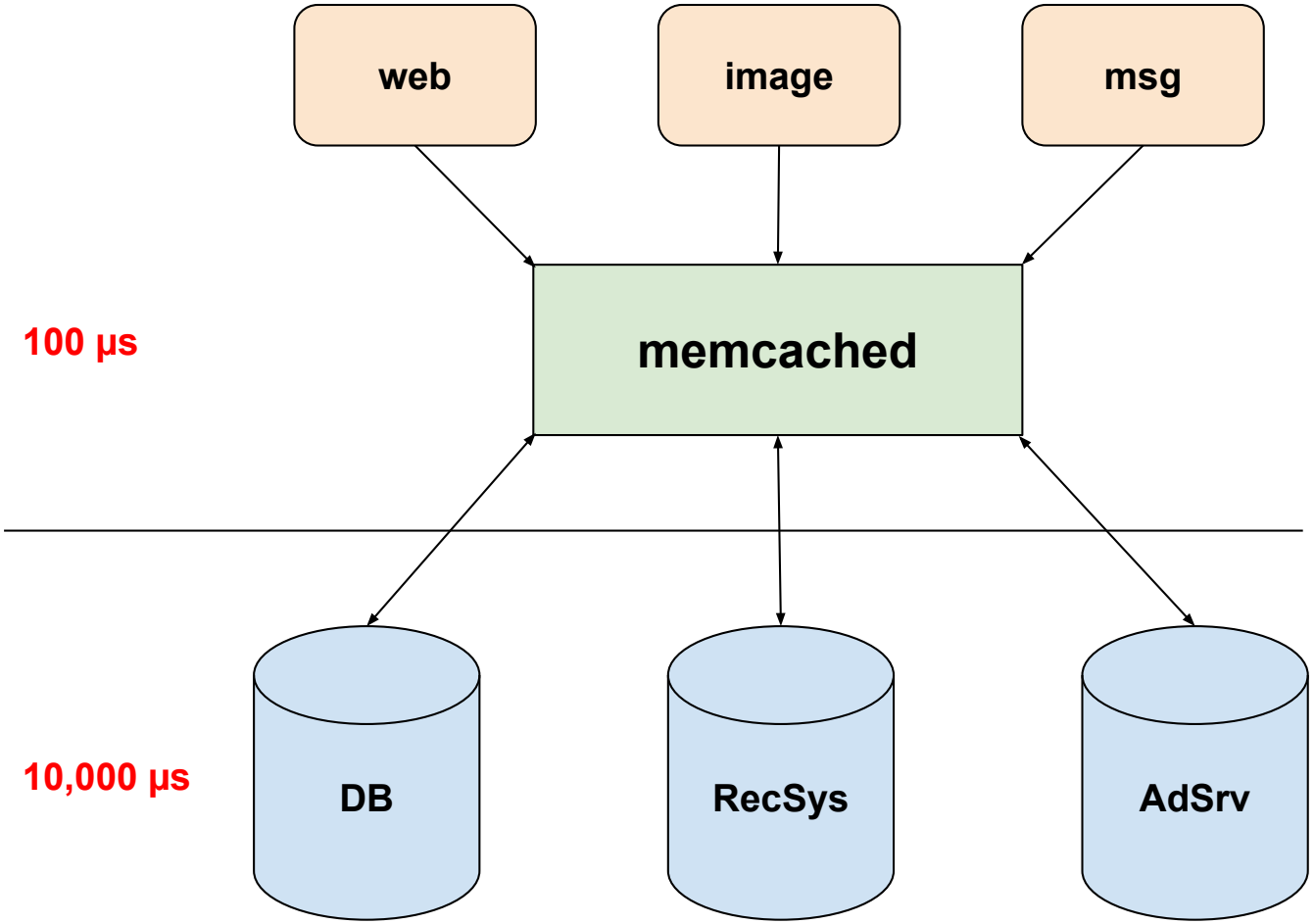
memcached

Faster Slab Reassignment

Experimental Evaluation

Conclusion

Cache Data from Backend Systems



Cache Miss Ratio Drives Performance

		Access Time (µs)		
System	**Miss Ratio**	**Cache**	**Backend**	**End-to-End**
A	2%	100	10,000	298
B	1%	100	10,000	199

Cache Miss Ratio Drives Performance

		Access Time (µs)		
System	**Miss Ratio**	**Cache**	**Backend**	**End-to-End**
A	2%	100	10,000	298
B	1%	100	10,000	199

2% → 1% in miss ratio → 33% decrease in end-to-end latency!

Background

memcached

Faster Slab Reassignment

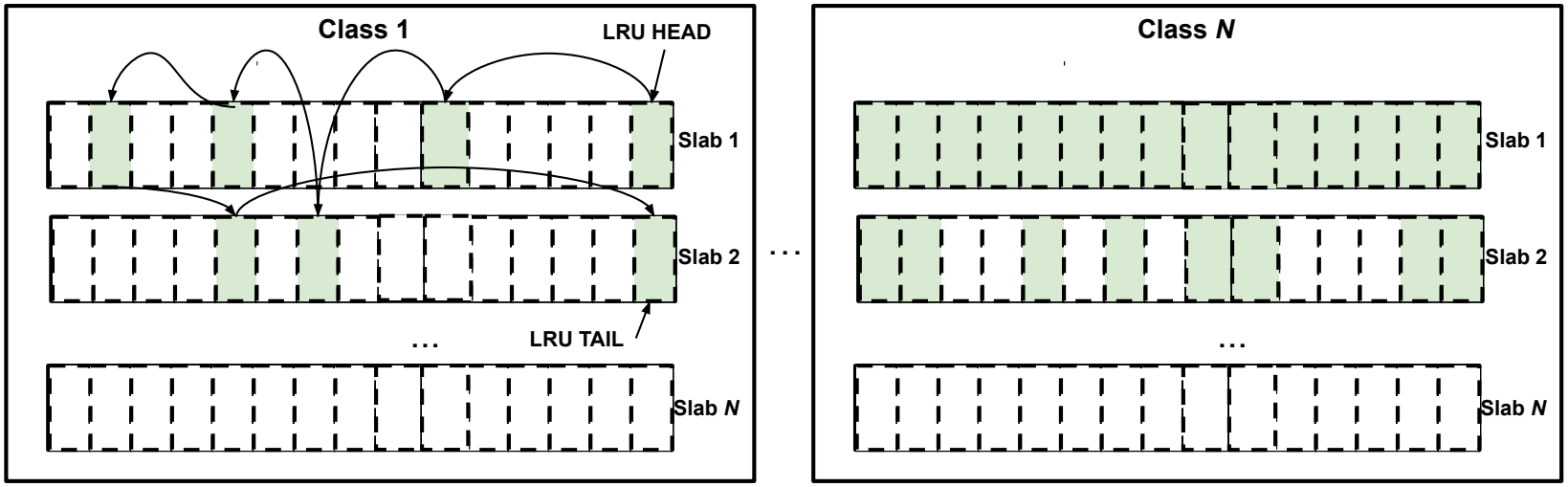
Experimental Evaluation

Conclusion

memcached

- memcached resides in server's main memory
- Data is stored as: `key,value` pair
- To retrieve an item: `GET key`
- To store/update an item: `SET key value`
- Deployed in many large scale datacenters

memcached Memory Organization



- A *class* is a collection of *slabs* that contain items
- Each class corresponds to items of a given size range
- Each class maintains its own LRU queue

Why Reassign Slabs Among Classes?

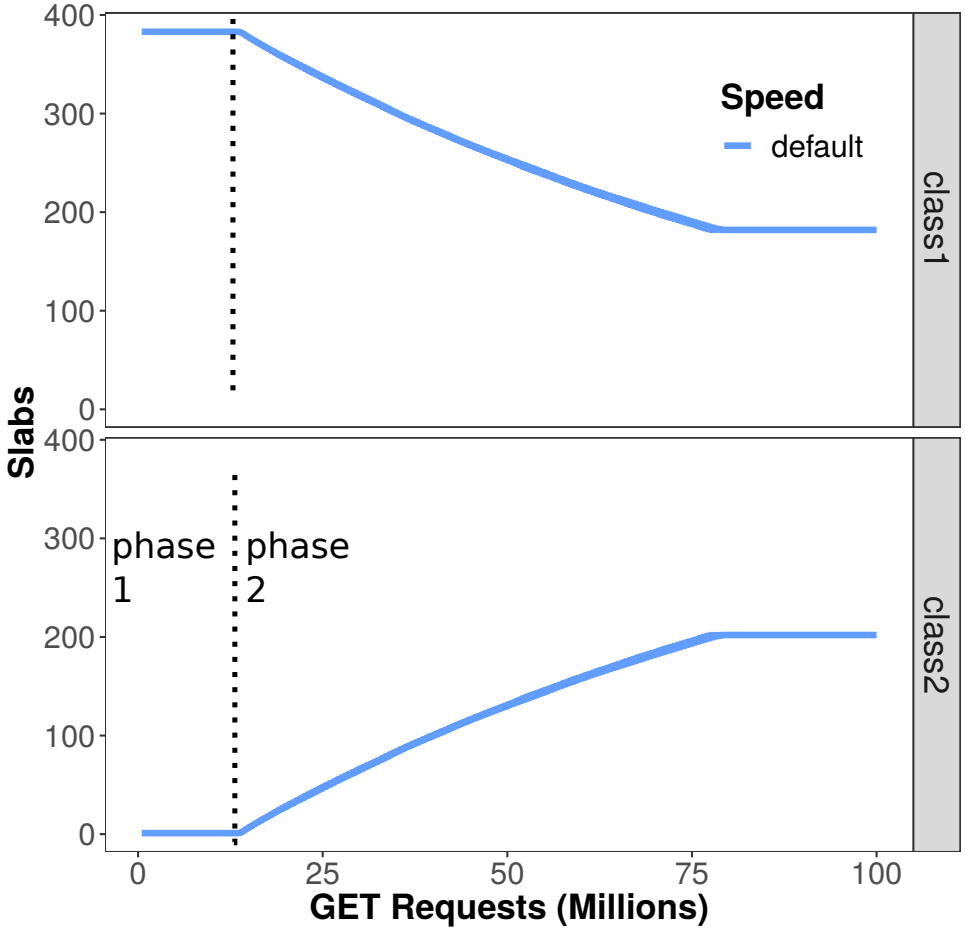
- Adapt to changes in an application's workload
  - Working set sizes can change over time
  - An application may change its item size distribution
- Dynamically reassign slabs when new applications enter the cache
- Miss ratio curves can be used to find optimal allocation among classes
  - LAMA, ATC '15
  - mPart, ISMM '18
- Our work focuses on the process of reassigning slabs from one class to another

Adapting to a New Class of Items

- Two-Phase Workload
- Fix the total memory size at 384MB

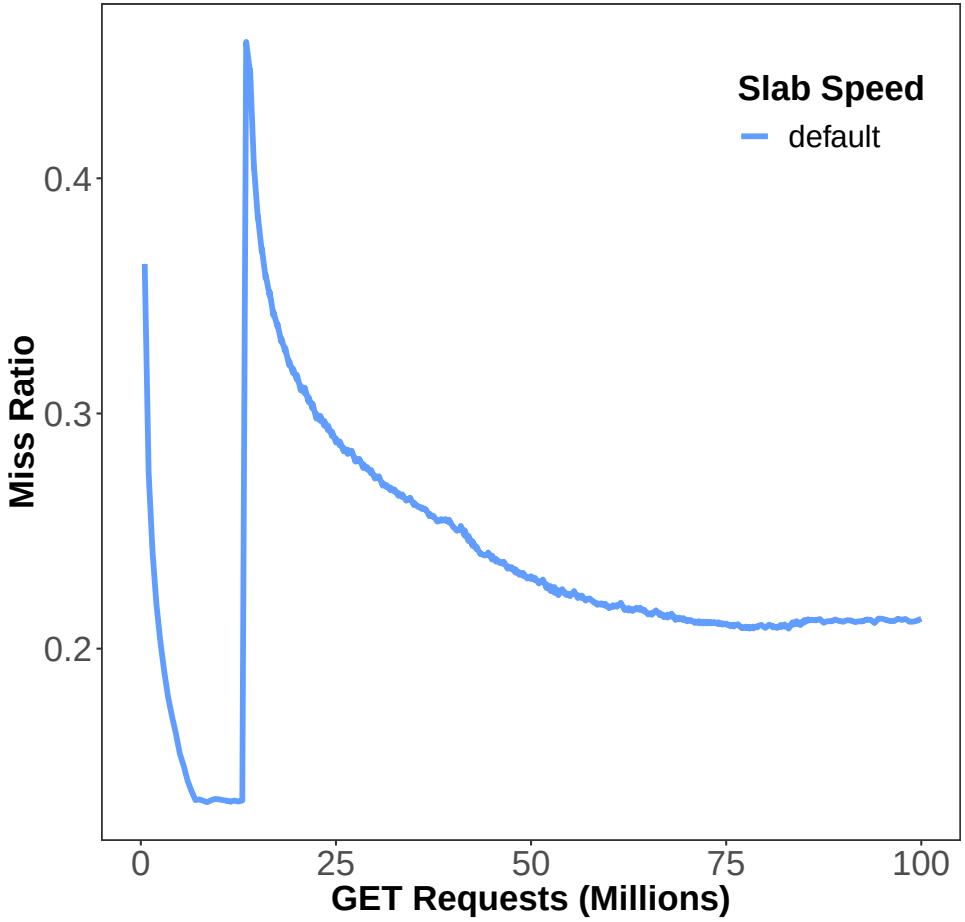
phase	% class 1	% class 2	reqs
1	100	0	13 million
2	33	67	87 million

Default Slab Reassignment



- 65 million requests to reassign over 200 slabs

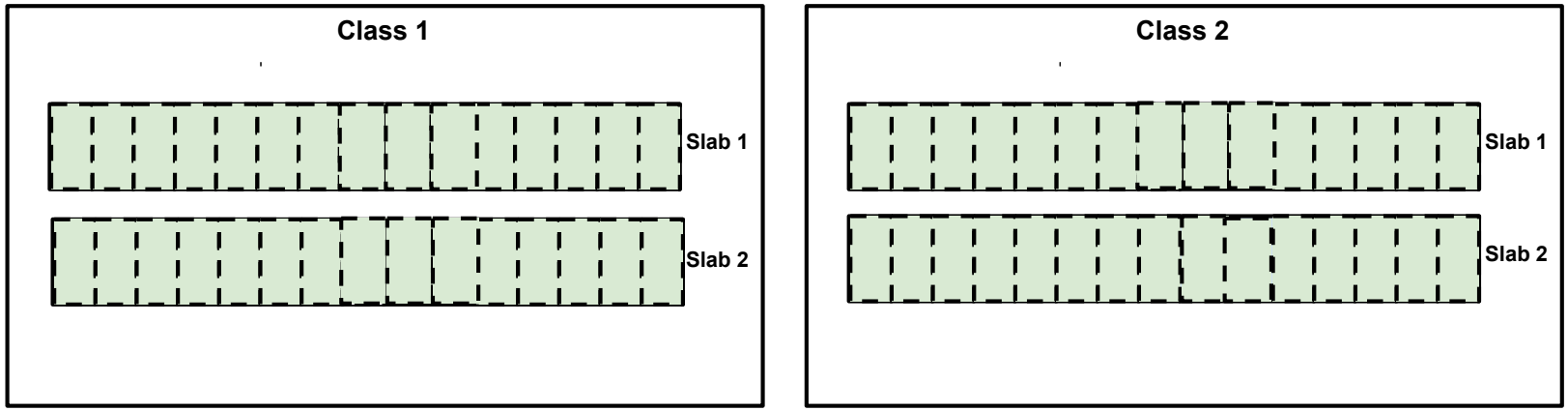
Impact on Overall Miss Ratio



- 60 million requests to reach steady state miss ratio

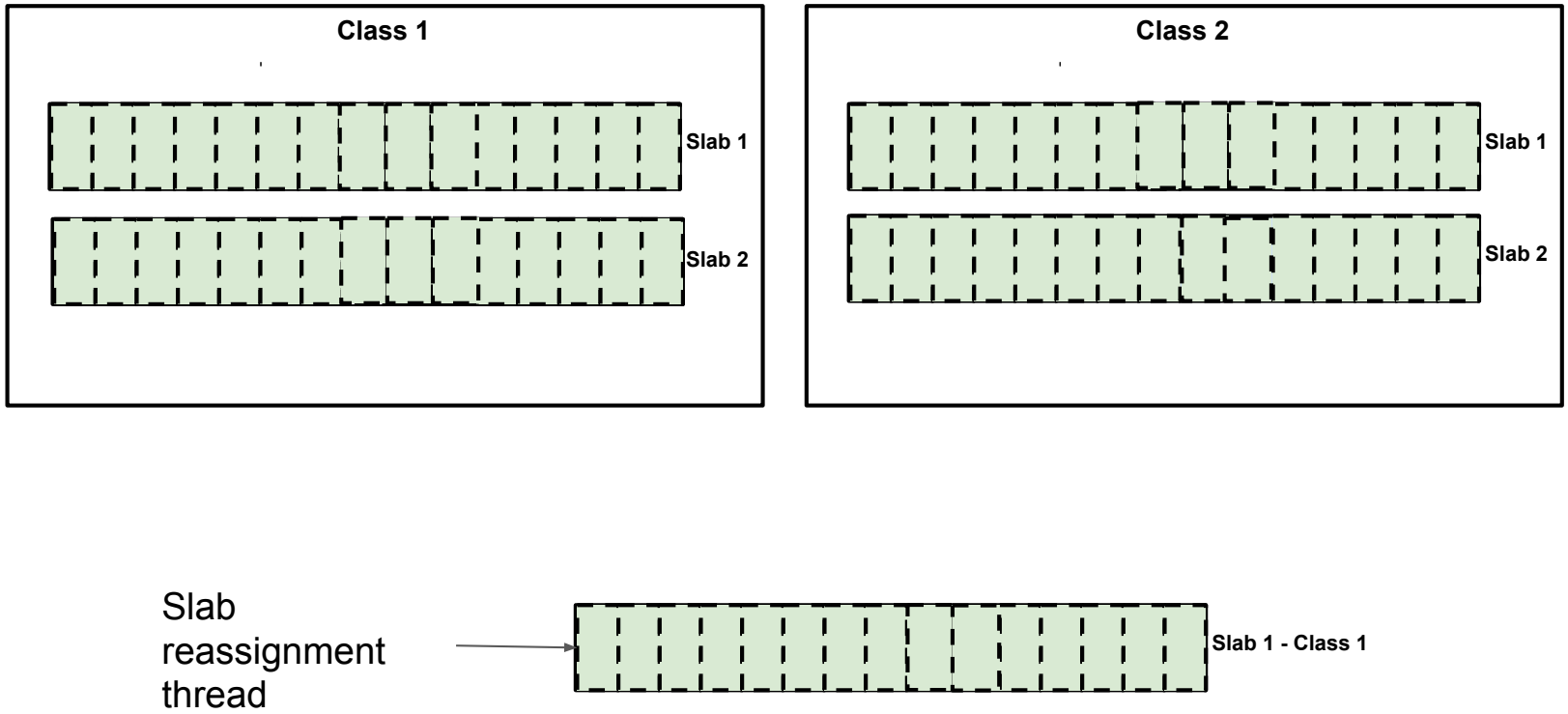
Current Reassignment Algorithm

- Goal: Reassign a slab from class 1 to class 2



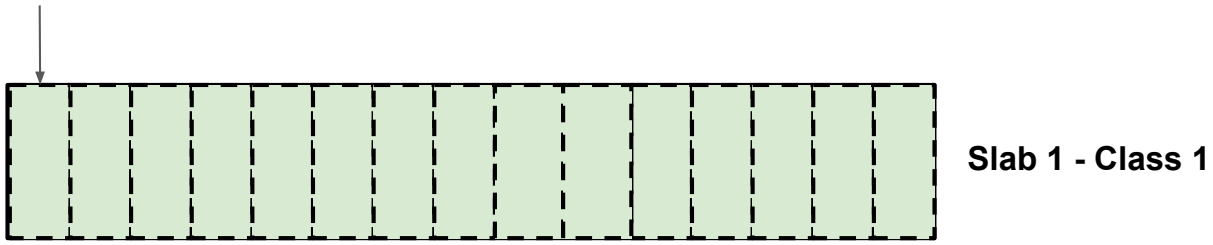
Current Reassignment Algorithm

- Goal: Reassign a slab from class 1 to class 2



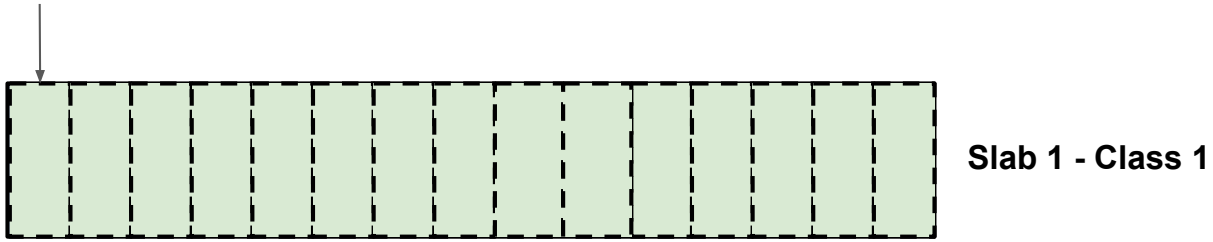
Current Reassignment Algorithm

1. **Acquire item lock**



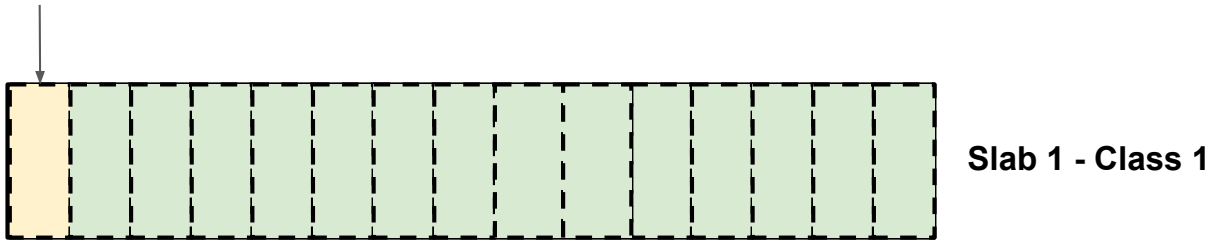
Current Reassignment Algorithm

1. Acquire item lock
2. **Check that no other threads reference this item**



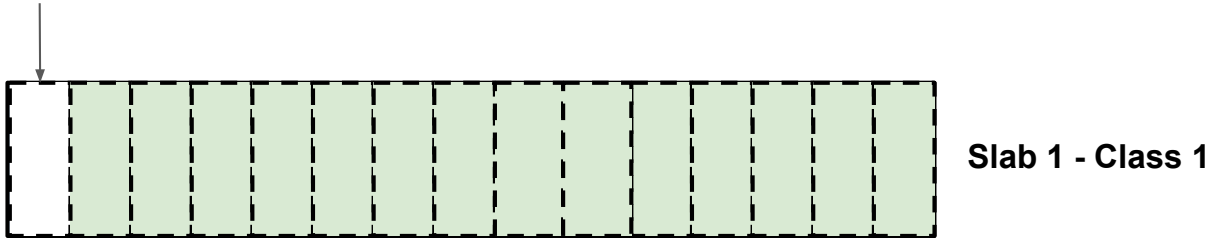
Current Reassignment Algorithm

1. Acquire item lock
2. Check that no other threads reference this item
3. **Unlink the item from the LRU queue**



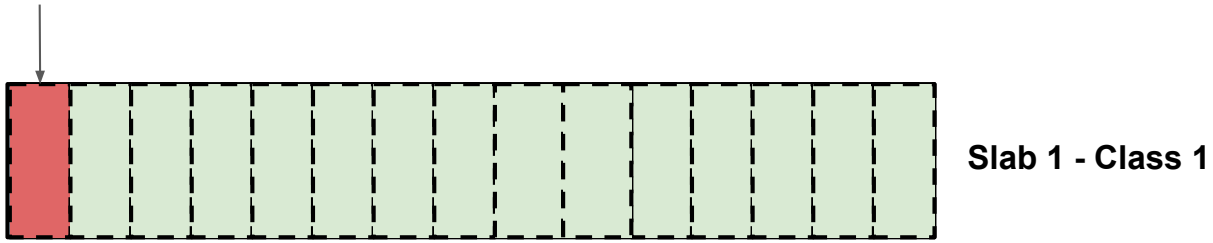
Current Reassignment Algorithm

1. Acquire item lock
2. Check that no other threads reference this item
3. Unlink the item from the LRU queue
4. **Free the item**



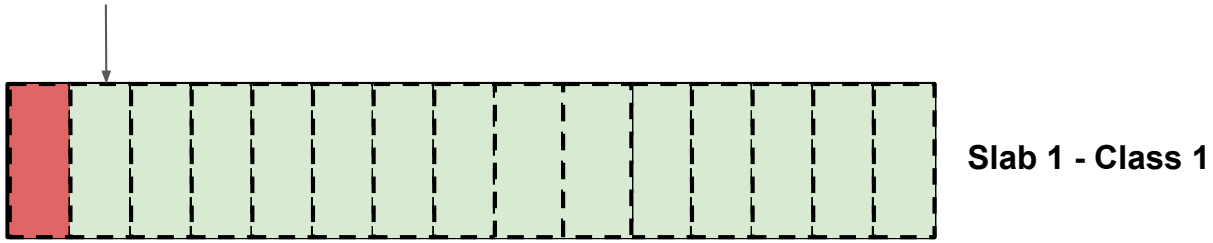
Current Reassignment Algorithm

1. Acquire item lock
2. Check that no other threads reference this item
3. Unlink the item from the LRU queue
4. Free the item
5. **Mark the item as *was busy***



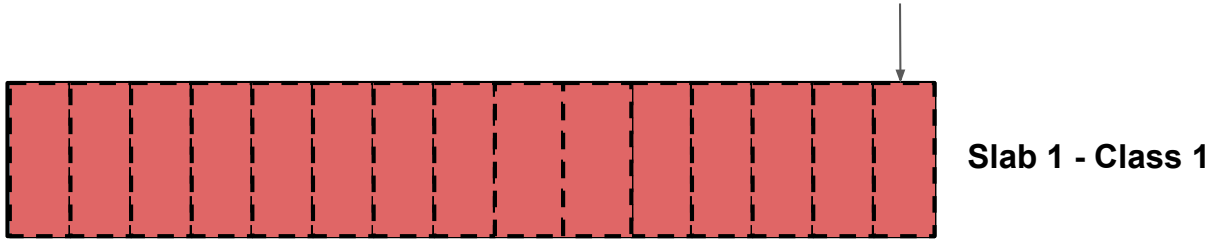
Current Reassignment Algorithm

1. Acquire item lock
2. Check that no other threads reference this item
3. Unlink the item from the LRU queue
4. Free the item
5. Mark the item as *was busy*
6. **Repeat for next item**



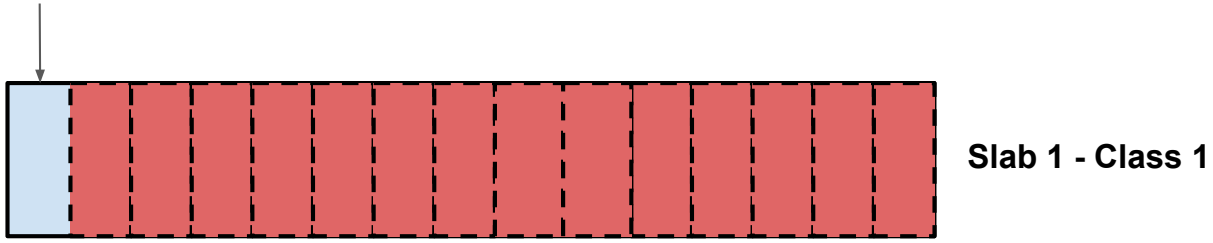
Current Reassignment Algorithm

1. Acquire item lock
2. Check that no other threads reference this item
3. Unlink the item from the LRU queue
4. Free the item
5. Mark the item as *was busy*
6. Repeat for next item
7. **Return to head**



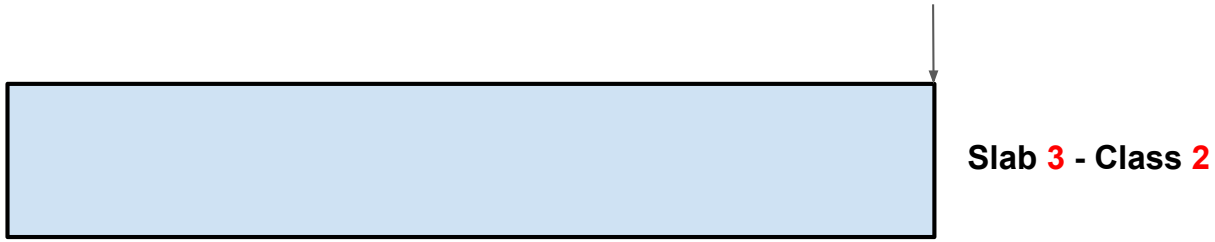
Current Reassignment Algorithm

1. Acquire item lock
2. Check that no other threads reference this item
3. Unlink the item from the LRU queue
4. Free the item
5. Mark the item as *was busy*
6. Repeat for next item
7. Return to head
8. **Remove item slot from class freelist**



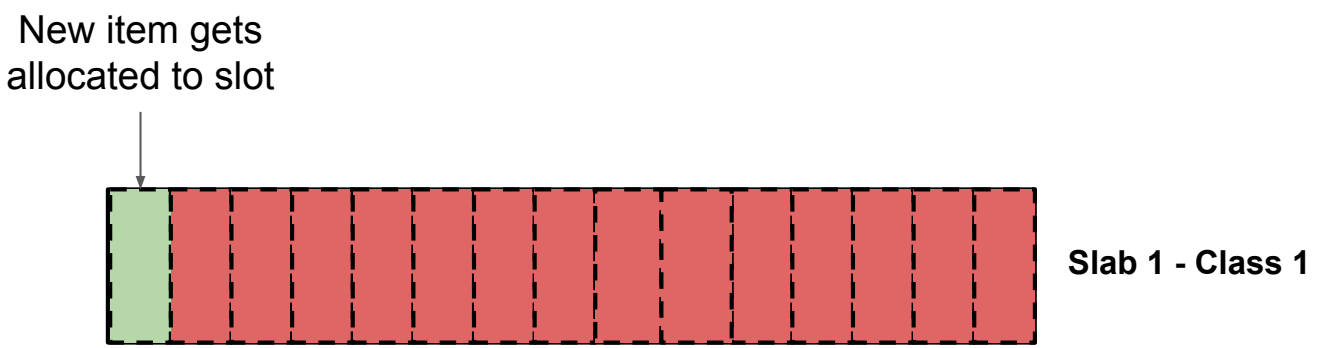
Current Reassignment Algorithm

1. Acquire item lock
2. Check that no other threads reference this item
3. Unlink the item from the LRU queue
4. Free the item
5. Mark the item as *was busy*
6. Repeat for next item
7. Return to head
8. Remove item slot from class freelist
9. **Assign to class 2**



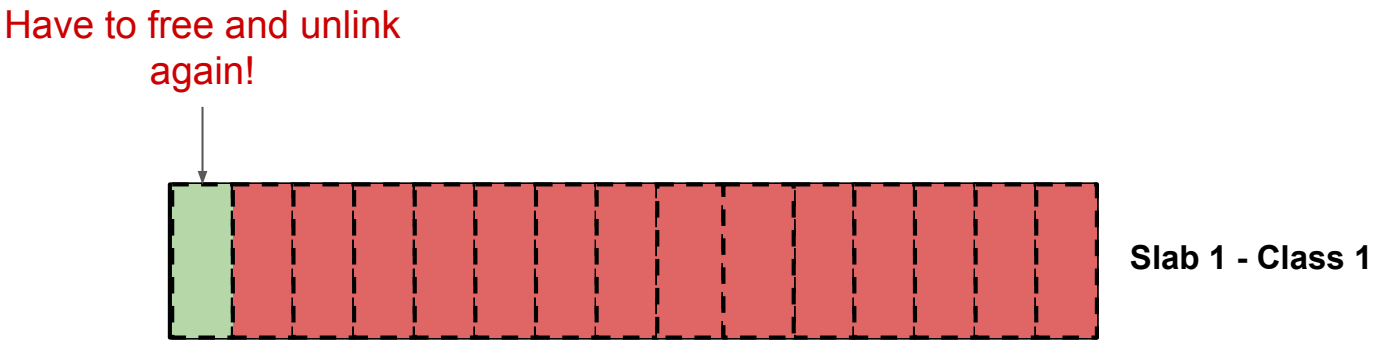
What Slows Down Reassignment?

- Each *was busy* causes the thread to sleep
  - Slab reassign thread detected that the item was in use and cannot be cut from the class's freelist at this moment
- During thread sleep, an item can be allocated to the item slot



What Slows Down Reassignment?

- Each *was busy* causes the thread to sleep
  - Slab reassign thread detected that the item was in use and cannot be cut from the class's freelist at this moment
- During thread sleep, an item can be allocated to the item slot



Sleeping for a Shorter Period

algorithm	sleep interval (µs)	slabs/s
default	1	36.93
default	10	35.5
default	100	26.55
default	1000	4.12

- Moving hundreds of slabs still requires several seconds of waiting on the slab reassingment thread to complete

Background

memcached

Faster Slab Reassignment

Experimental Evaluation

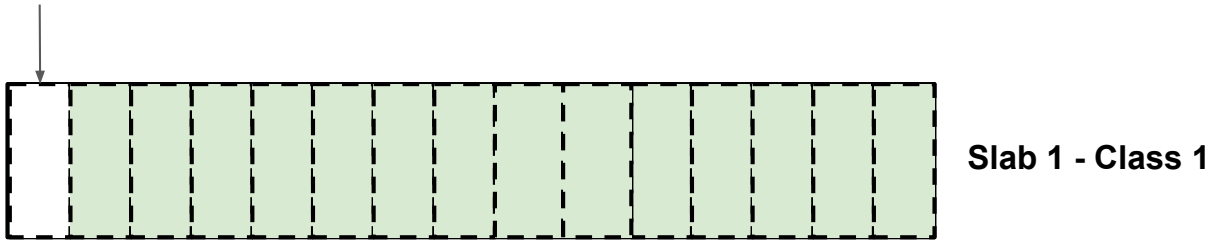
Conclusion

Faster Slab Reassignment

- Remove the items immediately from the class's freelist
  - Removes *was busy* waiting on recently freed items
  - Stops items from being allocated to recently freed slots

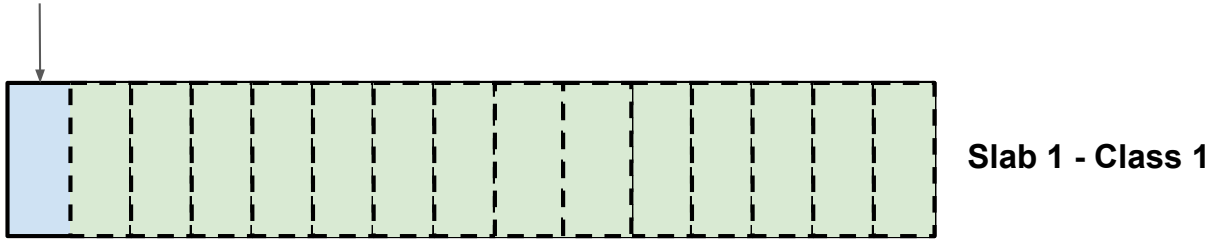
Faster Slab Reassignment Algorithm

1. Acquire item lock
2. Check that no other threads reference this item
3. Unlink the item from the LRU queue
4. **Free the item**



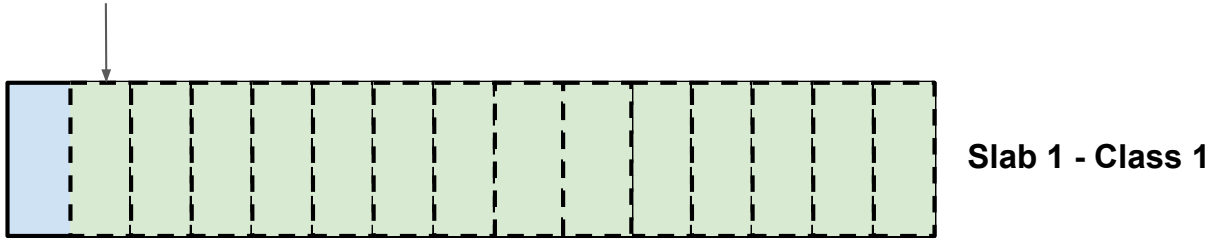
Faster Slab Reassignment Algorithm

1. Acquire item lock
2. Check that no other threads reference this item
3. Unlink the item from the LRU queue
4. Free the item
5. **Remove item slot from class freelist**



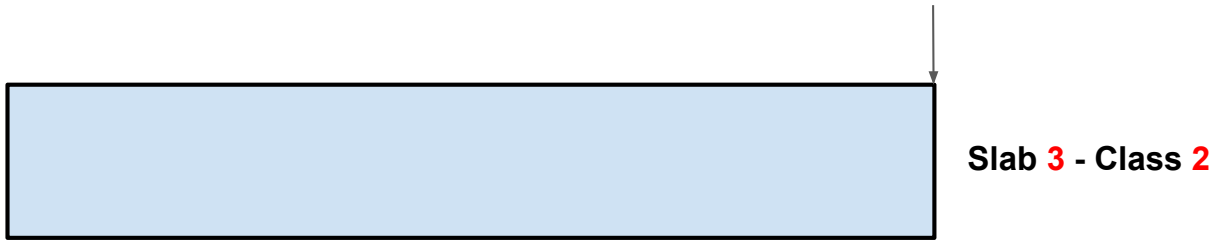
Faster Slab Reassignment Algorithm

1. Acquire item lock
2. Check that no other threads reference this item
3. Unlink the item from the LRU queue
4. Free the item
5. Remove item slot from class freelist
6. **Repeat for next item**



Faster Slab Reassignment Algorithm

1. Acquire item lock
2. Check that no other threads reference this item
3. Unlink the item from the LRU queue
4. Free the item
5. Remove item slot from class freelist
6. Repeat for next item
7. **Assign to class 2**



Background

memcached

Faster Slab Reassignment

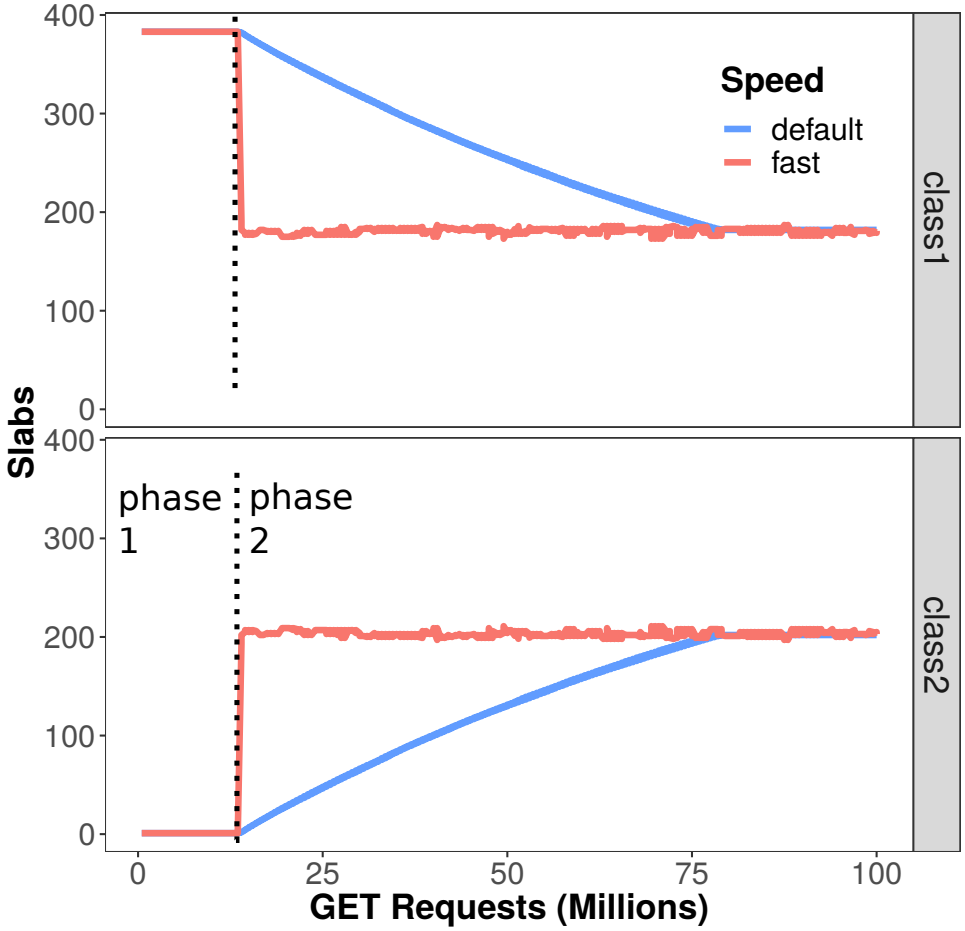
Experimental Evaluation

Conclusion

Experimental Setup

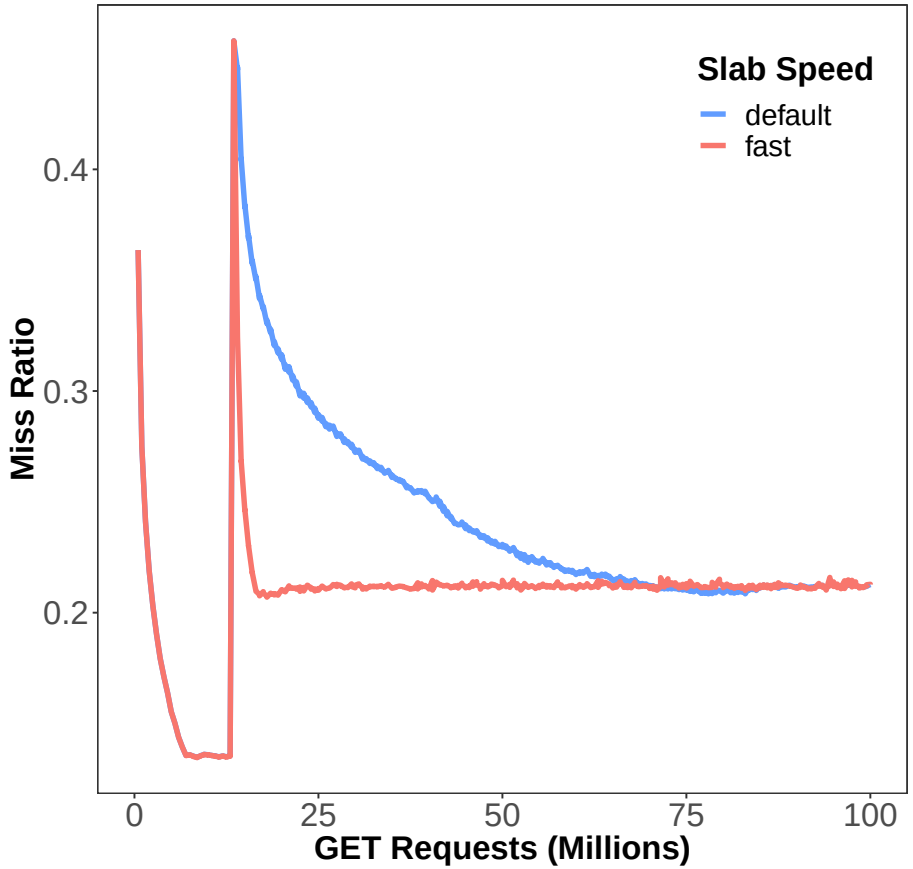
- Implemented fast slab reassignment algorithm in memcached
- Use miss ratio curve partitioning to assign memory among classes
- 2 different workloads - See paper for multi-tenant evaluation
  - Two-Phase
  - Time-Varying
- Record the overall miss ratio and slab assignments over entire trace

Slab Movement in Two-Phase Workload



- Over 95% reduction in the time needed to reallocate slabs

Miss Ratio in Two-Phase Workload



- Over 95% reduction time to steady state
- 11.5% improvement in the mean miss ratio

CPU Usage vs. Slab Move Rate in Two-Phase Workload

algorithm	sleep interval (µs)	cpu usage %	slabs/s
default	1	13.8	36.93
default	10	10.6	35.5
default	100	8.0	26.55
default	**1000**	**3.1**	**4.12**
fast	**1000**	**97.0**	**252.31**

- Tradeoff: Increased CPU usage for significantly faster reassignment
- Allows the reassingment thread to complete execution sooner

Time-Varying Workload

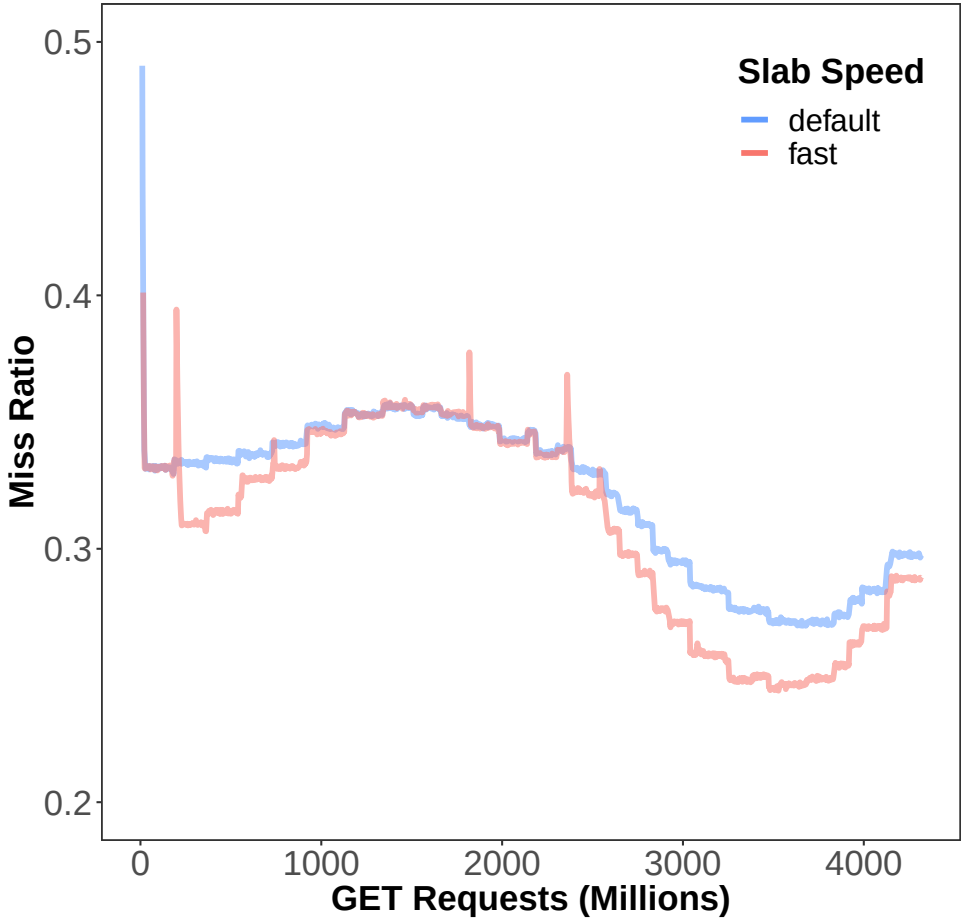
- Simulates two applications over an entire 24 hour period following the request rate distribution at Facebook
- Each application sends 2.16B requests totally, drawing from 7 million unique items

Slab Move Rate in Time-Varying Workload

	Move Rate (GETs)	Min	Max
default	19,454,304	17	26,333,587
fast	77.62	4	220

- 99.99% decrease in the average time required to reassign a slab

Miss Ratio in Time-Varying Workload



- 3.42% improvement in overall miss ratio

Background

memcached

Faster Slab Reassignment

Experimental Evaluation

Conclusion

Conclusion

- Improved miss ratio in memcached as a result of faster slab reassignment among classes
- Orthogonal to other works that decide *how many slabs* to assign to each class
- Submitted our implementation as patch to current memcached source code

Thank you, questions?

Faster Slab Reassignment in memcached

Daniel Byrne djbyrne@mtu.edu
Nilufer Onder nilufer@mtu.edu
Zhenlin Wang zlwang@mtu.edu

Department of Computer Science
Michigan Technological University

MEMSYS 2019