# High-Performance Key-Value Store on OpenSHMEM

Huansong Fu*,
Manjunath Gorentla Venkata†,
Ahana Roy Choudhury*,
Neena Imam†, Weikuan Yu*

*Florida State University
†Oak Ridge National Laboratory

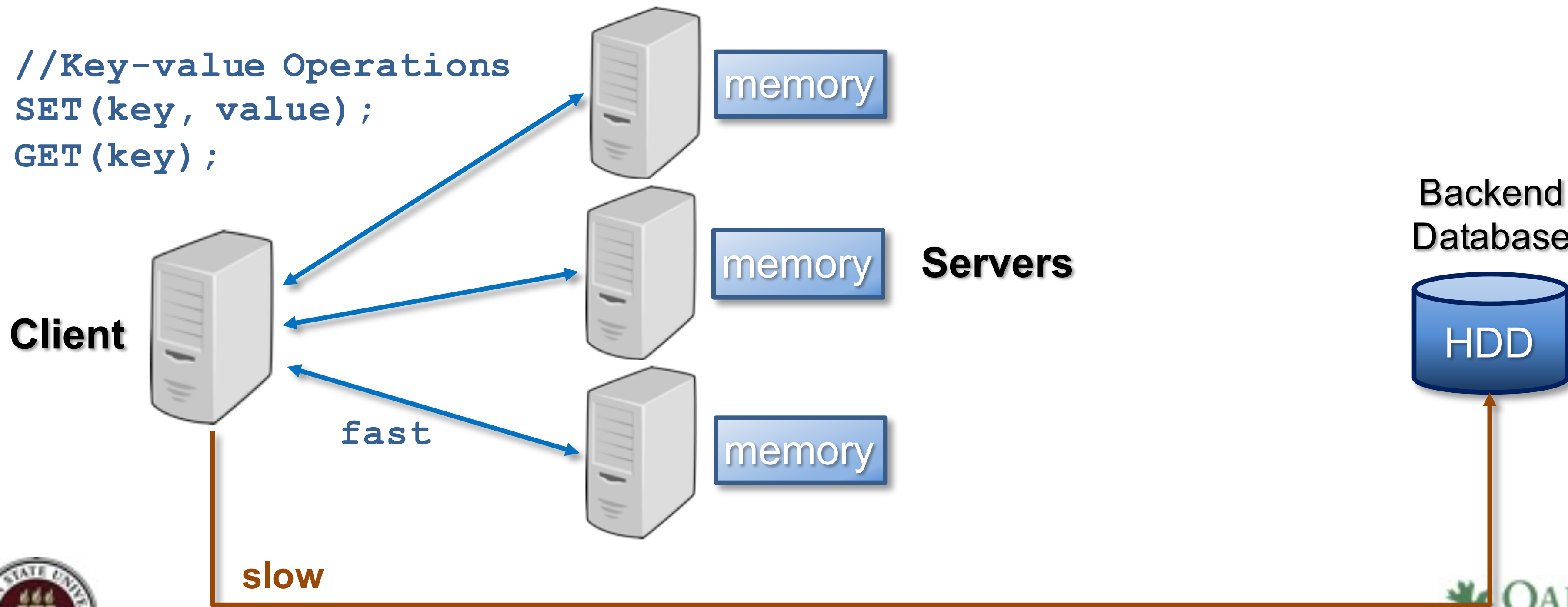
# Outline

- Background
- SHMEMCache
  - Overview
  - Challenges
  - Design
- Experiments
- Conclusion

# Distributed In-memory Key-Value Store

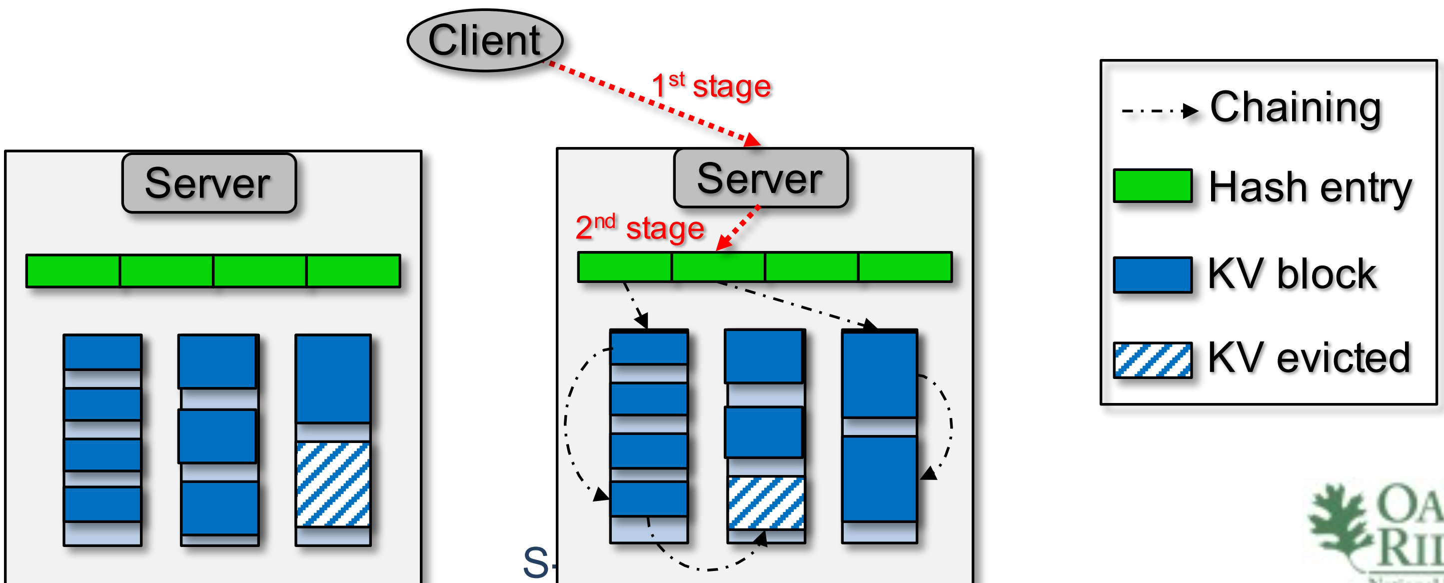
- Caching popular key-value (KV) pairs in memory in a distributed manner for fast access (mainly reads).
  - E.g. Memcached used by Facebook serves >1 billion request per second.
- Client-server architecture.
  - Client and server communicate over TCP/IP.

```
//Key-value Operations
SET(key, value);
GET(key);
```



# Memcached Structure

- Two-stage hashing and KV pair locating
  - 1st stage to a server and 2nd stage to a *hash entry* in that server.
  - A hash entry contains pointers to locate *KV pairs*.
  - Server chases *chained* pointers to find desired KV pair.
- Storage and eviction policy
  - Memory storage is divided into *KV blocks* in various sizes.
  - A KV pair is stored in the *smallest possible* KV block.
  - Least Recent Used (*LRU*) for evicting KV pairs.



# PGAS and OpenSHMEM

- Partitioned Global Address Space (PGAS) enables a large memory address space for parallel programming.
  - UPC, CAF, **SHMEM**.

- OpenSHMEM is a recent standardization effort of SHMEM.
  - Participating processes are called *Processing Elements* (PE).
  - A *symmetric memory* space that is globally-visible.
    - Variables have same name and same address on different PE.
  - Communication through *one-sided operations*.
    - `Put`: write to symmetric memory.
    - `Get`: read from symmetric memory.

# Our Objective

- There is a high suitability of leveraging OpenSHMEM's feature set for building a distributed in-memory key-value store.
  - Similar memory-style addressing: client "sees" memory storage.
  - One-sided communication relieves the server of participation in communication.
  - Portability on both commodity servers and HPC systems.

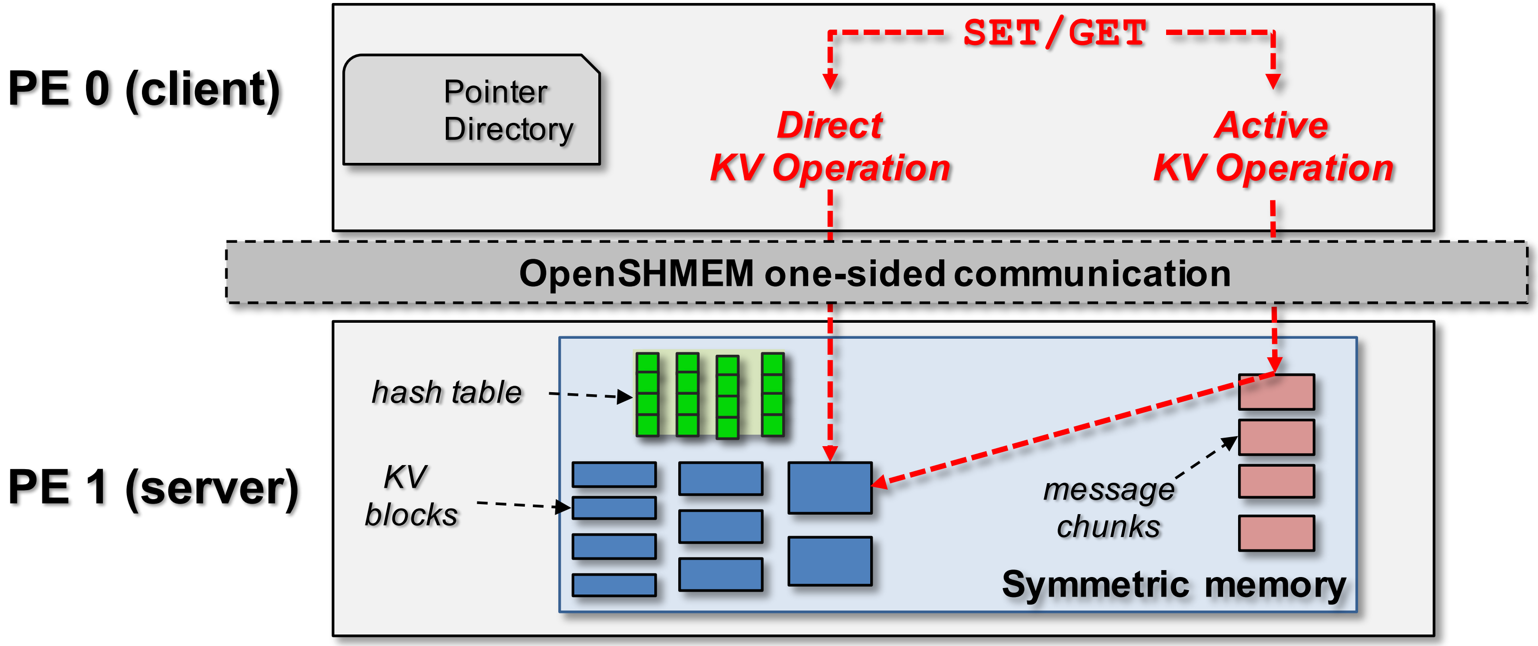
- We propose **SHMEMCache**, a high-performance key-value store built on top of OpenSHMEM.

# Outline

- Background
- SHMEMCache
  - Overview
  - Challenges
  - Design
- Experiments
- Conclusion

# Structure of SHMEMCache

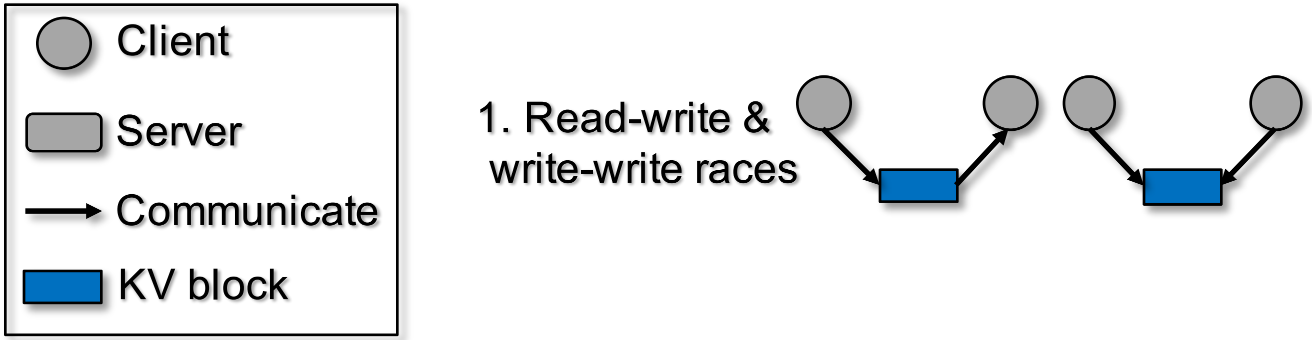
- Server stores KV pairs in symmetric memory.
- Client performs two types of KV operations.
  - Direct KV operation: client directly accessing KV pairs.
  - Active KV operation: client informs server to conduct KV operations by writing messages.




- There are **three major challenges** in realizing this structure.

# Challenges #1

- Concurrent reads/writes cause *read-write & write-write races*.
  - Solution: read-friendly exclusive write.

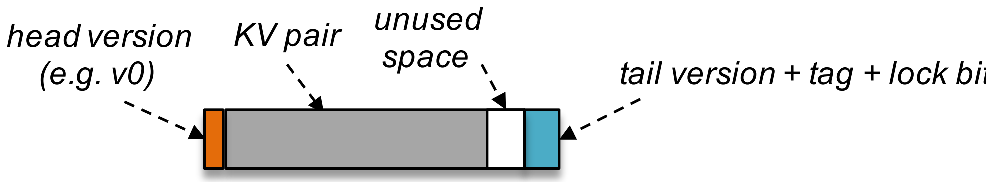


# Read-Friendly Exclusive Write

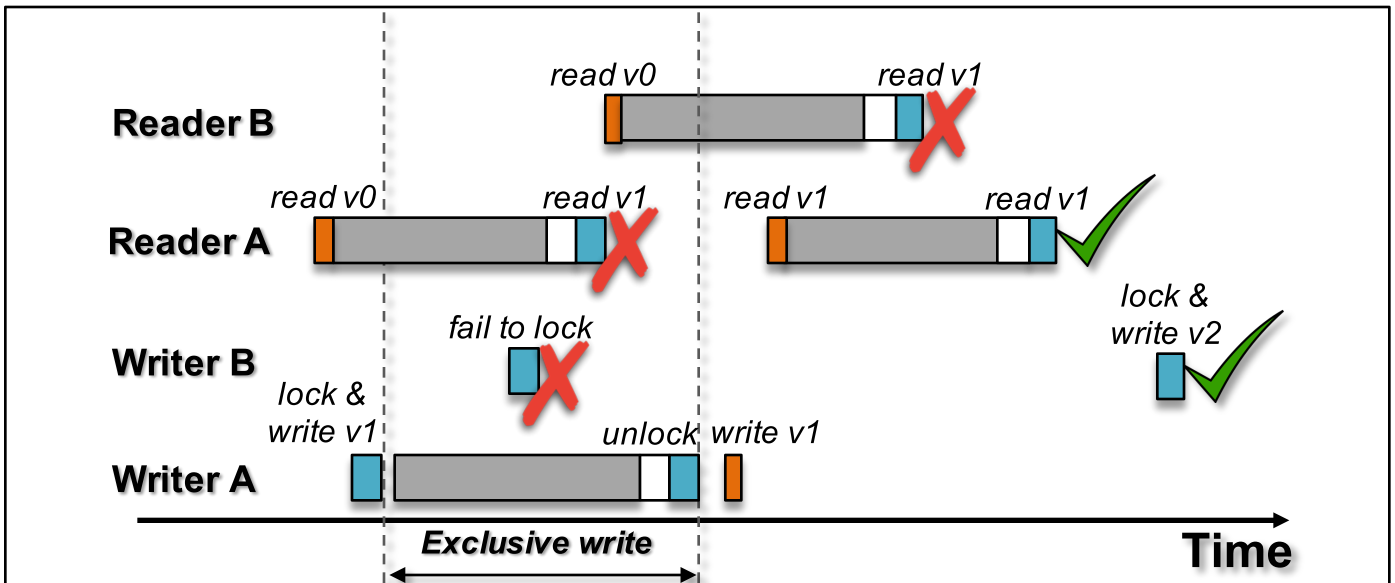
- Key issue: concurrent write can invalidate both read and write.
  - Write must be conducted exclusively.
- Solving write-write race: use locks to write exclusively.
  - Locks are expensive, so not for the much more frequent reads.
- Solving read-write race: a new optimistic concurrency control.
  - Existing solutions are heavy-weight: checksum, cacheline versioning…
  - We propose an efficient head-tail versioning mechanism.
- Combining locking and versioning operations to minimize overheads.

# Read-Friendly Exclusive Write (cont.)

- Structure of KV block

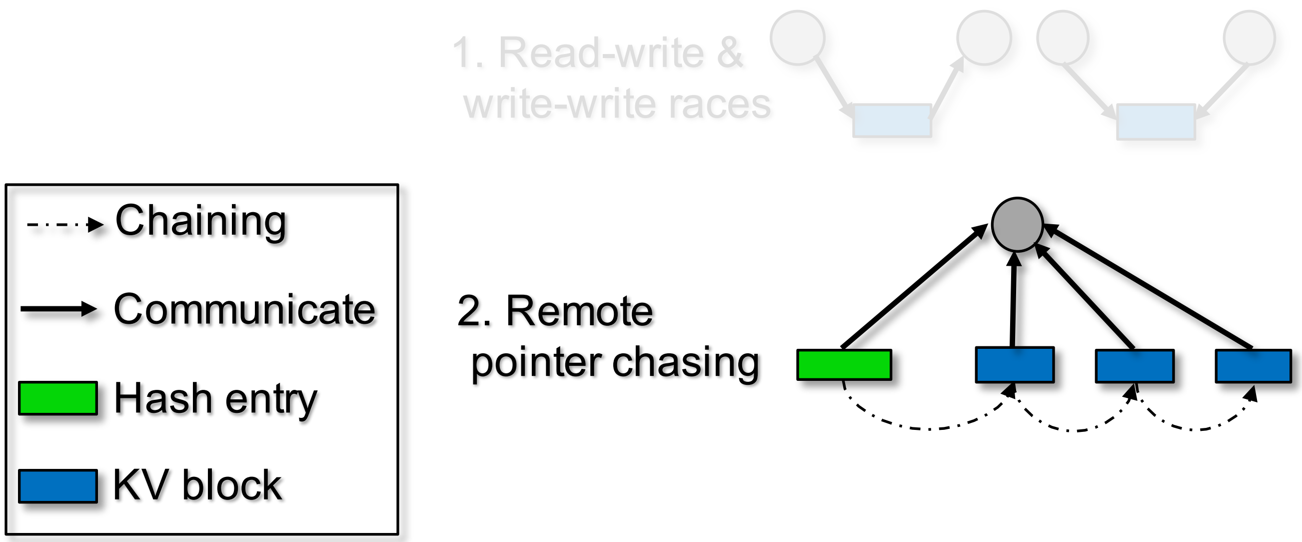



- Write operations:
  - (1) an atomic CAS for locking and writing tail version; (2) a `Put` for writing KV content and unlocking; (3) a `Get` for writing head version.
- Read operation:
  - A `Get` for reading the whole KV block.



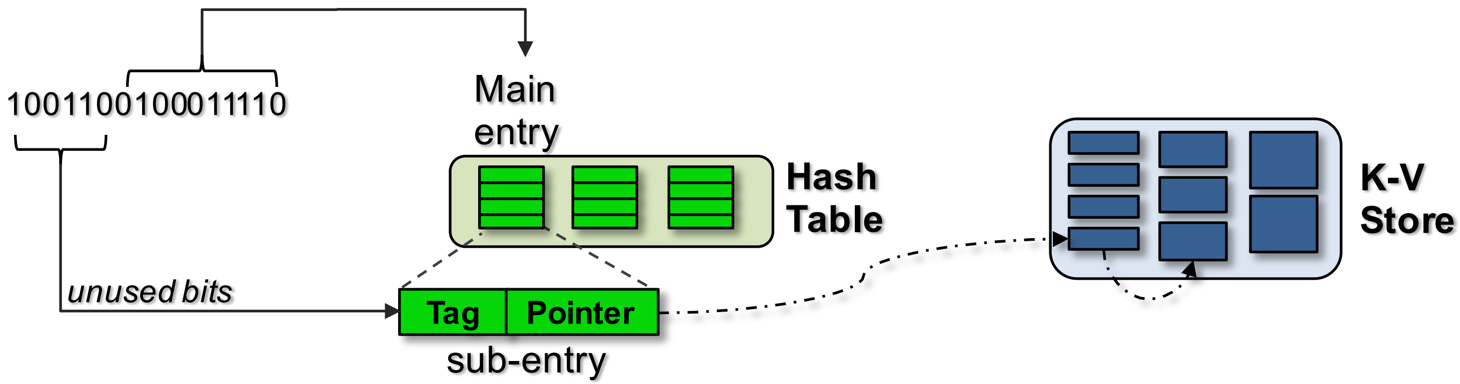
# Challenges #2

- Concurrent reads/writes cause *read/write & write/write races.*
- Remotely locating a KV pair incurs costly *remote pointer chasing*.
  - Solution: set-associative hash table and pointer directory.

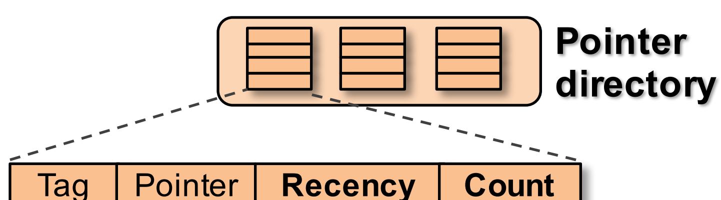


# Mitigating Remote Pointer Chasing

- Set-associative hash table
  - Multiple pointers are fetched with one hash lookup.
  - If pointer not found in a hash entry, only server chases pointers.

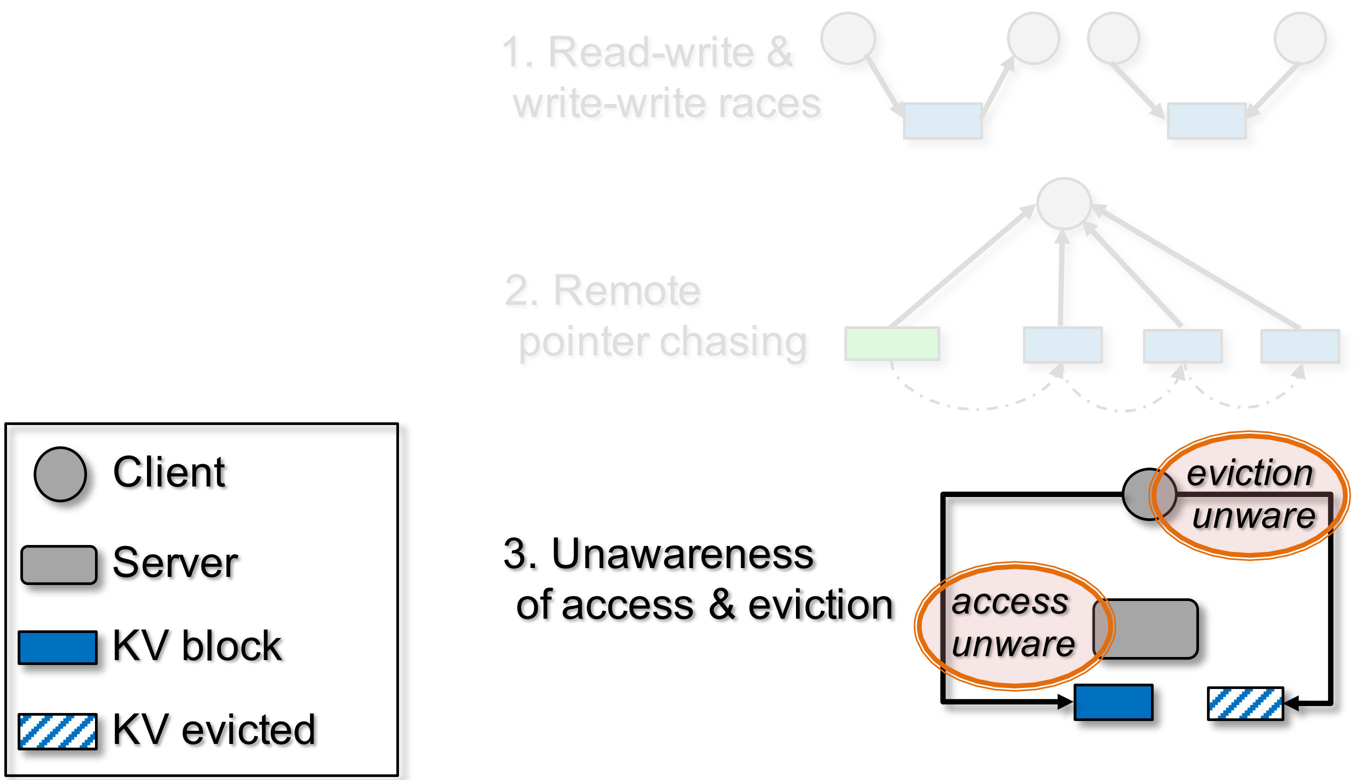



- Pointer directory
  - New pointer is stored after first read/write.
  - *Recency* and *count* indicate how recent and frequent a KV pair has been accessed. They help decide which pointer to keep in the directory.



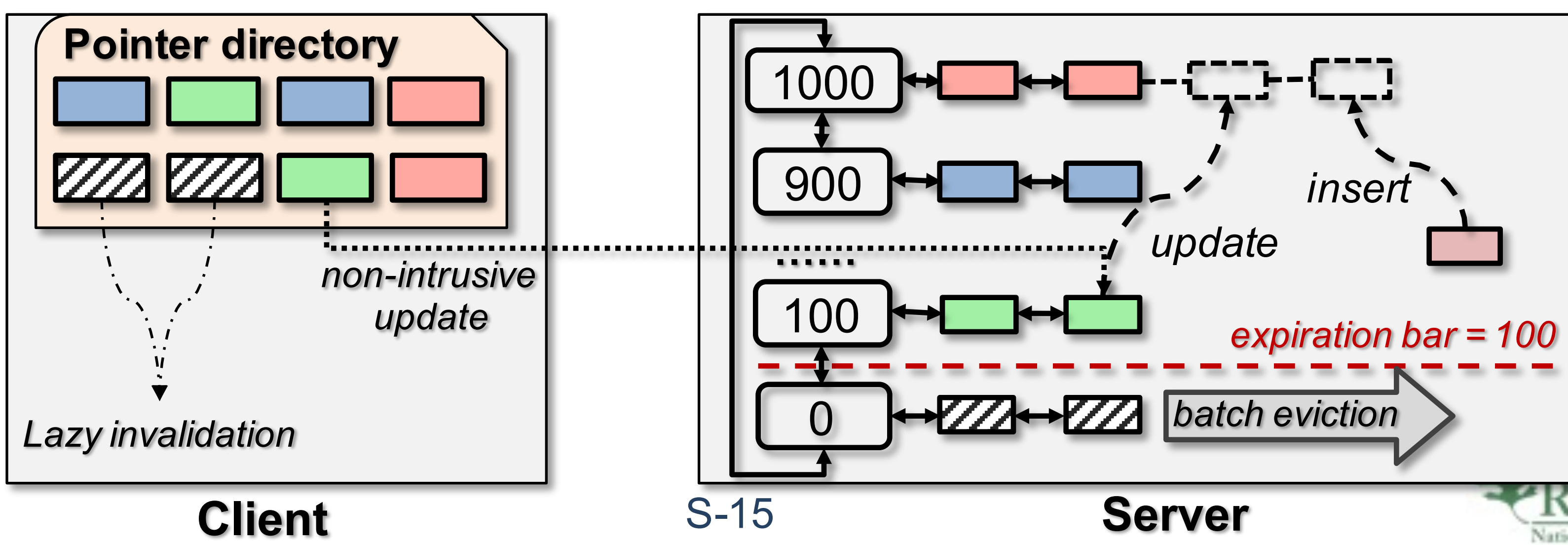
# Challenges #3

- Concurrent reads/writes cause *read/write & write/write races.*
- Remotely locating a KV pair incurs costly *remote pointer chasing*.
- Server/client is *unaware of other's access/eviction* actions.
  - Solution: Coarse-grained cache management.



# Coarse-grained Cache Management

- Relaxed recency definition from *time point* to *time range*.
- Non-intrusive recency update
  - When recency changes, client directly update it using atomic CAS.
- Batch eviction and lazy invalidation
  - Server evicts KV pairs in batches with the *oldest* recency, and notifies client the new *expiration bar*. KV pair that has higher recency than the evicted batch will be updated. New KV pair will be inserted to the top.
  - Client lazily invalidate pointers.



# Outline

- Background
- SHMEMCache
  - Overview
  - Challenges
  - Design
- Experiments
- Conclusion

# Experimental Setup

- Innovation
  - An in-house cluster with 21 dual-socket server nodes, each featuring 10 Intel Xeon(R) cores and 64 GB memory. All nodes are connected through an FDR Infiniband interconnect with the ConnectX- 3 NIC.
- Titan supercomputer
  - Titan is a hybrid-architecture Cray XK7 system, which consists of 18,688 nodes and each node is equipped with a 16-core AMD Opteron CPU and 32GB of DDR3 memory.
- Benchmarks: microbenchmark and YCSB.
- OpenSHMEM version
  - In-house cluster: Open MPI v1.10.3
  - Titan: Cray SHMEM v7.4.0

# Latency

- Innovation cluster
  - SHMEMCache outperforms Memcached by **25x** and **20x** for `SET` and `GET` latencies.
  - SHMEMCache outperforms HiBD by **128%** and **16%** for `SET` and `GET` latencies.

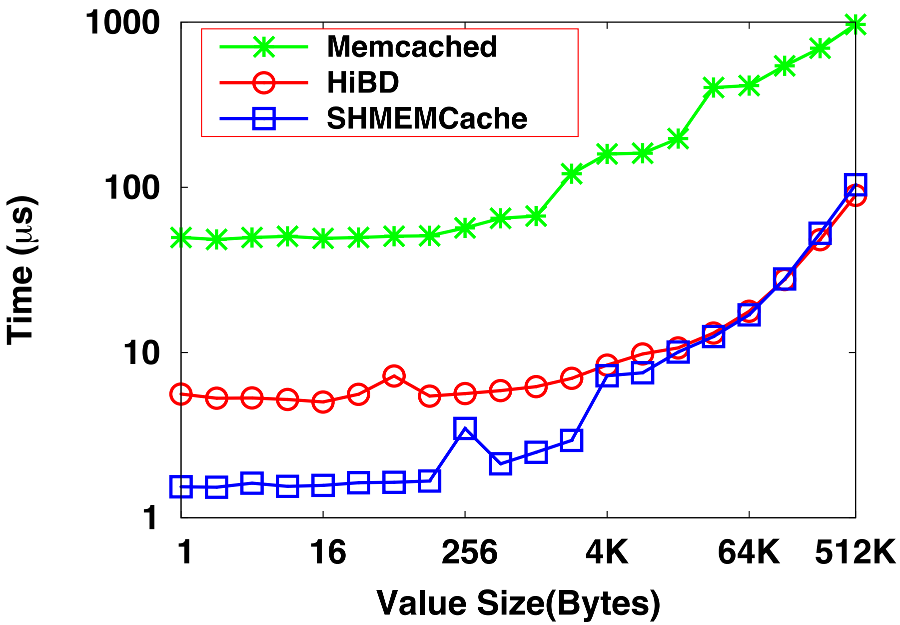



**Fig. 1 `SET` latency.**

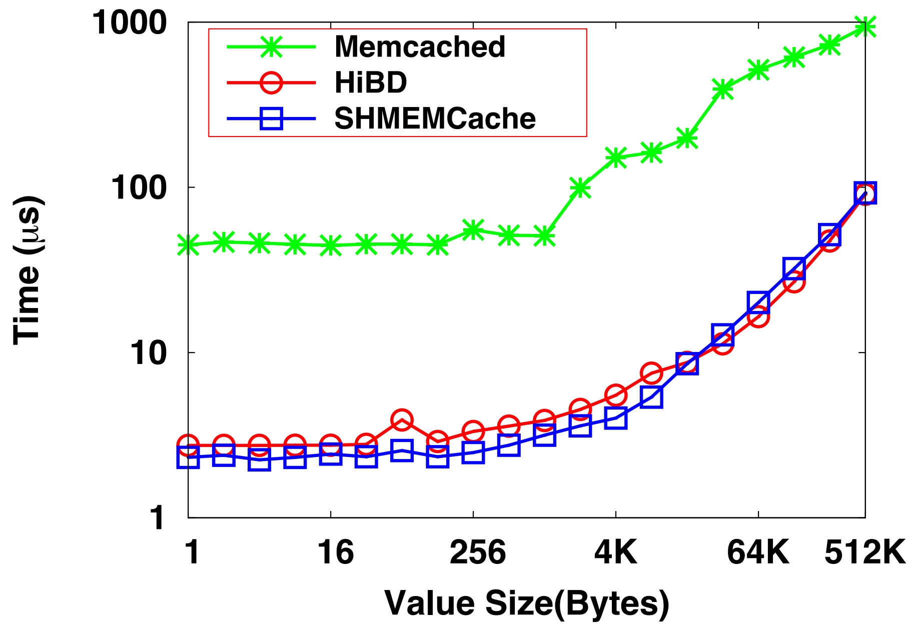



**Fig. 2 `GET` latency.**

# Latency (cont.)

- Titan supercomputer
  - Direct KV operation is consistently faster.
  - SHMEMCache's achieves comparable performance on Titan with the innovation cluster.

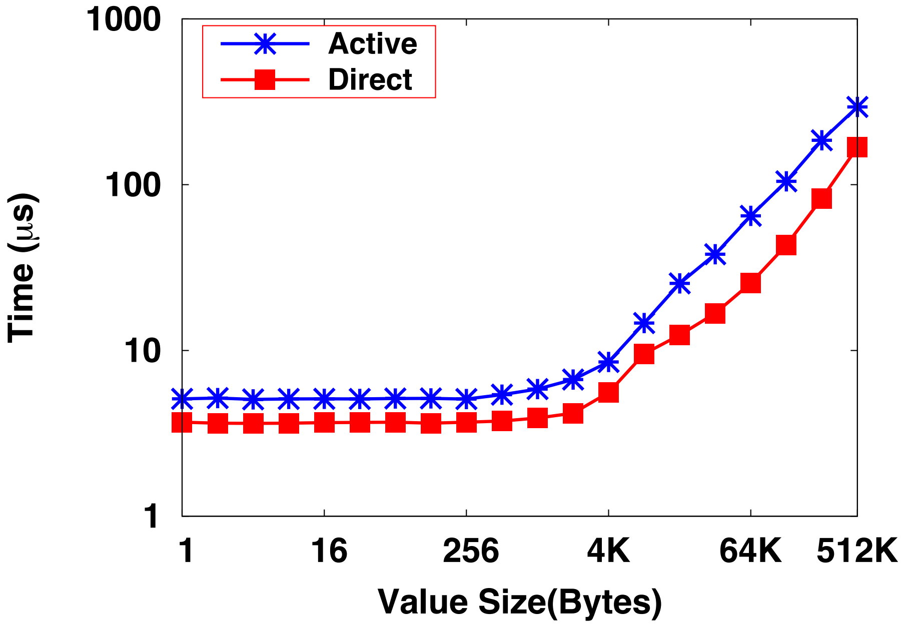



Fig. 1 **SET** latency.

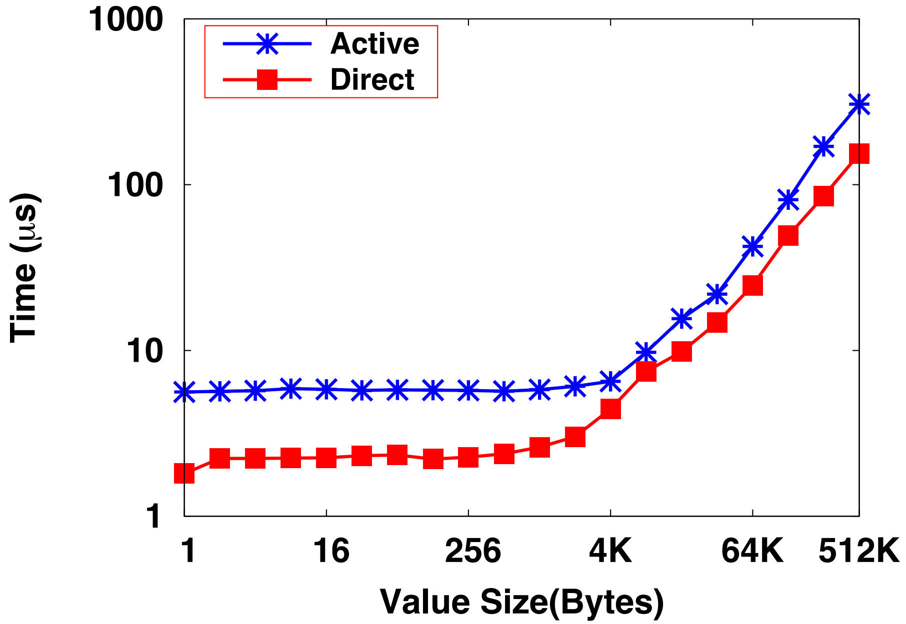



Fig. 2 **GET** latency.

# Throughput

- Innovation cluster
  - SHMEMCache outperforms Memcached by **19x** and **33x** for 32-Byte and 4-KB `SET`, and **14x** and **30x** for 32-Byte and 4-KB `GET`.

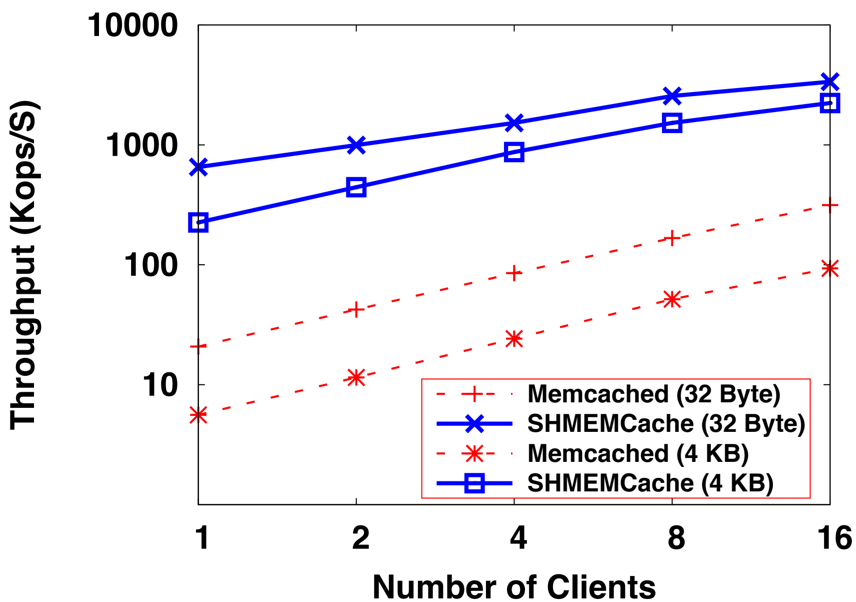



**Fig. 1 `SET` throughput.**

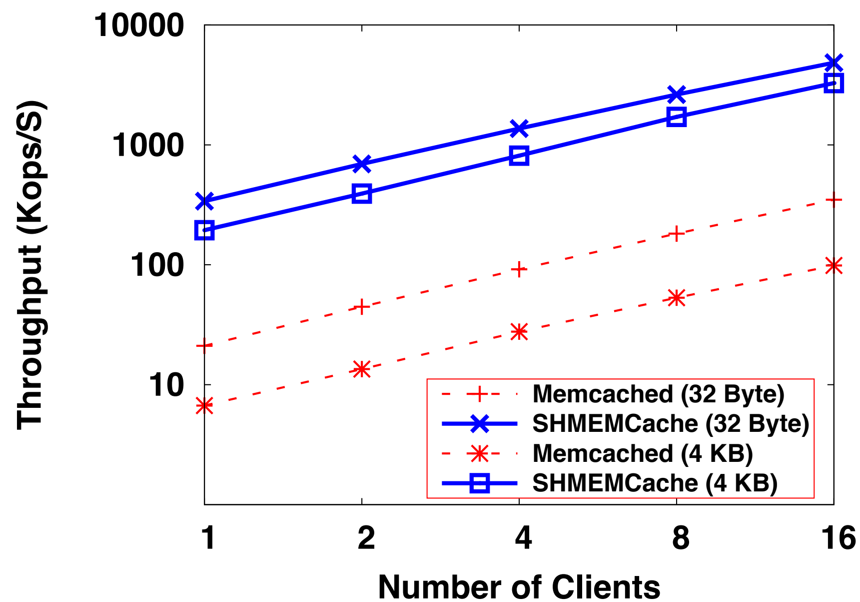



**Fig. 2 `GET` throughput.**

# Throughput (cont.)

- Titan supercomputer
  - SHMEMCache scales well on Titan supercomputer to up to 1024 machines.

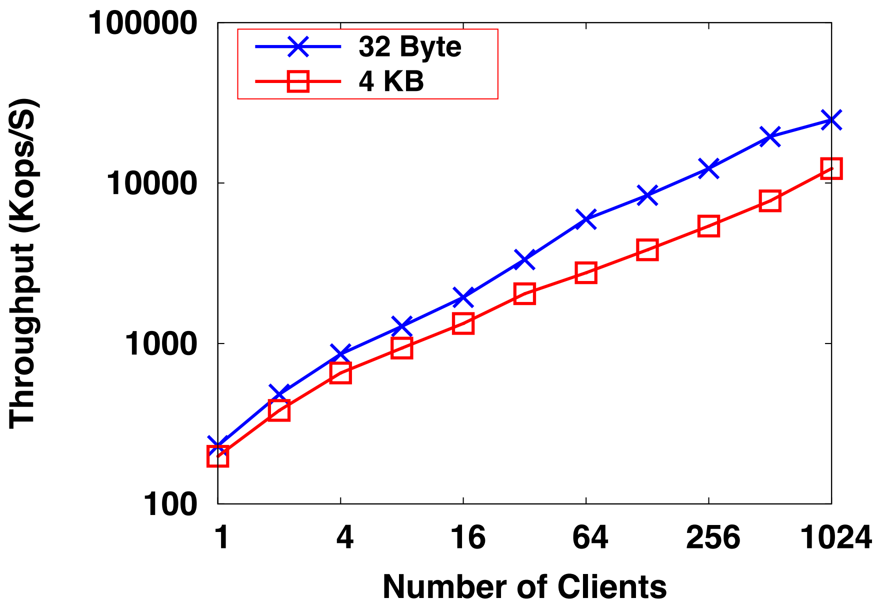



Fig. 1 **SET** throughput.

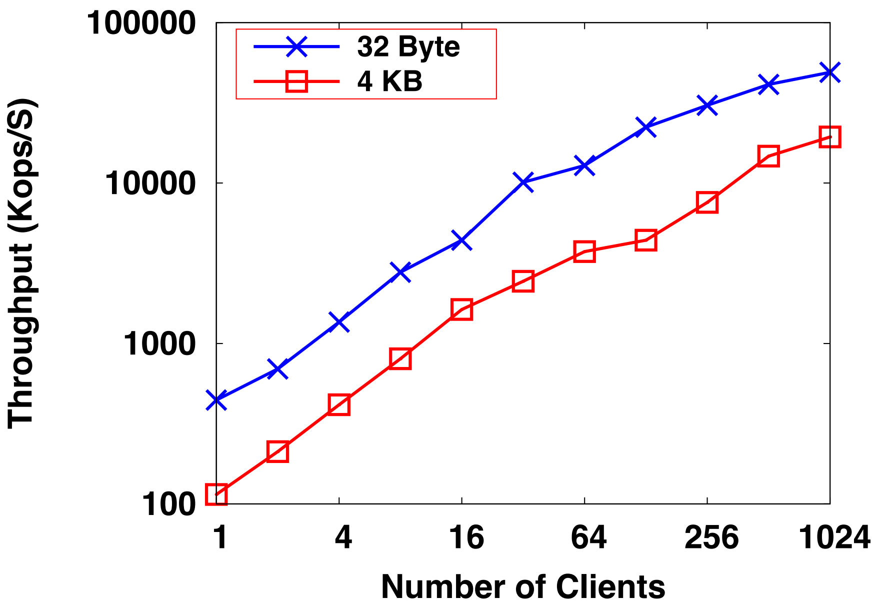
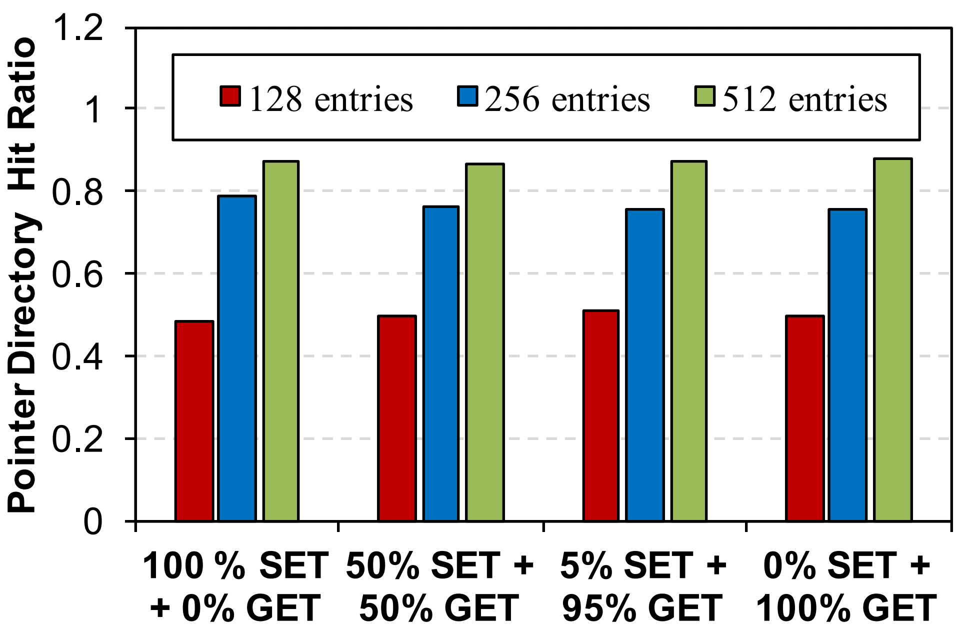



Fig. 2 **GET** throughput.

# Hit Ratio of Pointer Directory

- Varying YCSB workloads and number of entries in the pointer directory.
- Larger pointer directory helps increase hit ratio.
  - The impact is getting smaller due to less popular KV pairs.
  - No need to maximize its size at the cost of low space efficiency.



# Conclusion

- SHMEMCache, a high-performance distributed in-memory KV store built on top of OpenSHMEM.
- Novel designs to tackle three major challenges.
  - Read-friendly write for solving read-write and write-write races.
  - Set-associative hash table and pointer directory for mitigating remote pointer chasing.
  - Coarse-grained cache management for reducing high costs of informing server/client about access/eviction operations.
- Evaluation results showing that SHEMMCache achieves high-performance at scale of 1024.

# Acknowledgment

***Thank You and Questions?***