Portable SHMEMCache: A High-Performance Key-Value Store on OpenSHMEM and MPI

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

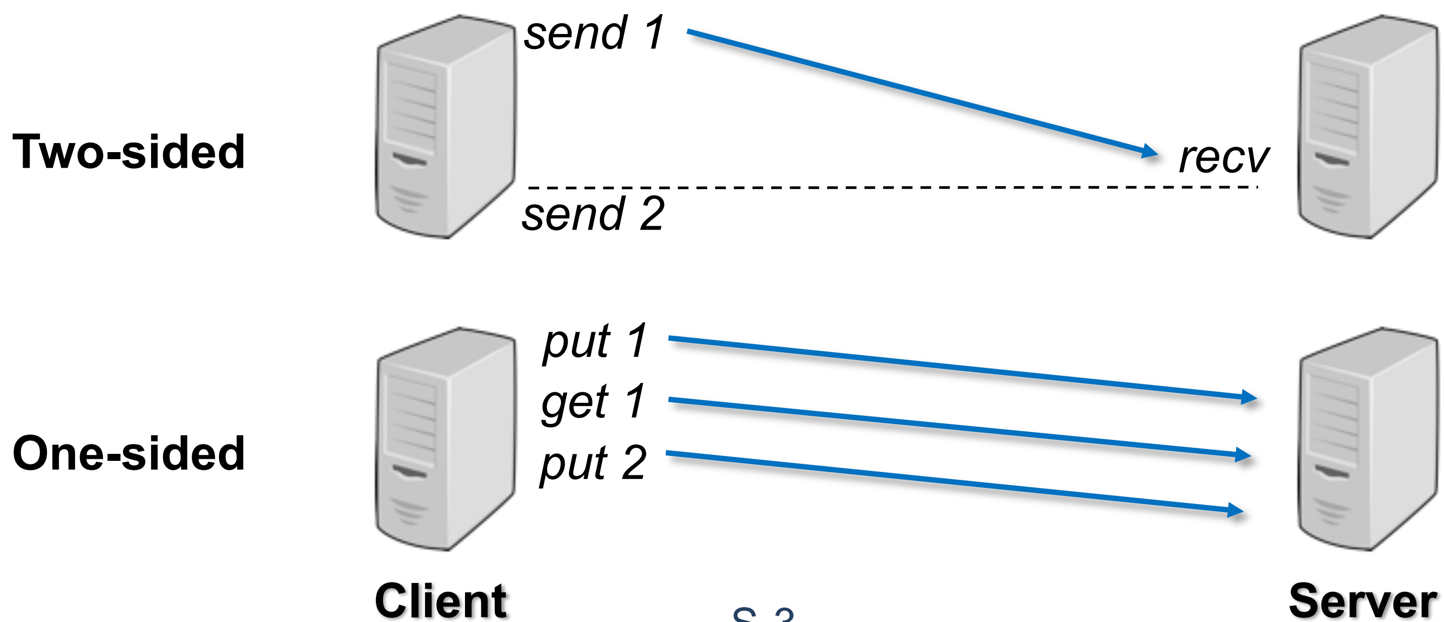
*Florida State University
†Oak Ridge National Laboratory

Outline

- Background and Motivation
  - SHMEMCache
  - Why Portable SHMEMCache
- Design and Implementation
  - Modular architecture
  - Portable interface
  - Leveraging OpenSHMEM and MPI
- Experiment
- Conclusion and Future Work

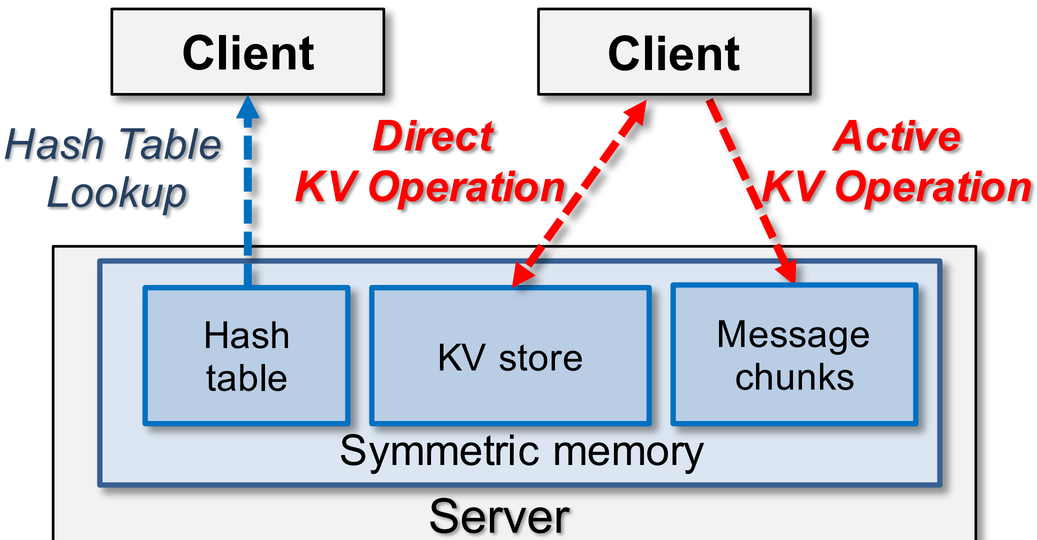
One-sided Communication for KV Store

- Distributed in-memory key-value (KV) store caches KV pairs in memory for fast access.
- **One-sided communication** has been popularly used for distributed in-memory KV store.
  - More relaxed synchronization requirements
  - Low-latency and high-throughput operations with RDMA



SHMEMCache

- SHMEMCache is a high-performance distributed key-value store built on OpenSHMEM.
  - Data are stored in symmetric memory of servers and can be accessed by clients through one-sided operations.
    - **Both `SET` and `GET`** can be conducted directly by clients.
    - **Low-cost** coarse-grained cache management.
  - Good trend of scalability to more than **one thousand nodes**.



Opportunity for Portable SHMEMCache

- Besides OpenSHMEM, one-sided communication is available through a wide range of libraries.
  - MPI, UPC, Co-Array Fortran/C++, etc.

- By leveraging them in SHMEMCache, we can have...
  - **Higher portability** of SHMEMCache.
  - Potential **performance improvement**.
  - **More understanding** about how different one-sided communications fit in with SHMEMCache or even other distributed systems that use one-sided communication.

Designing Portable SHMEMCache

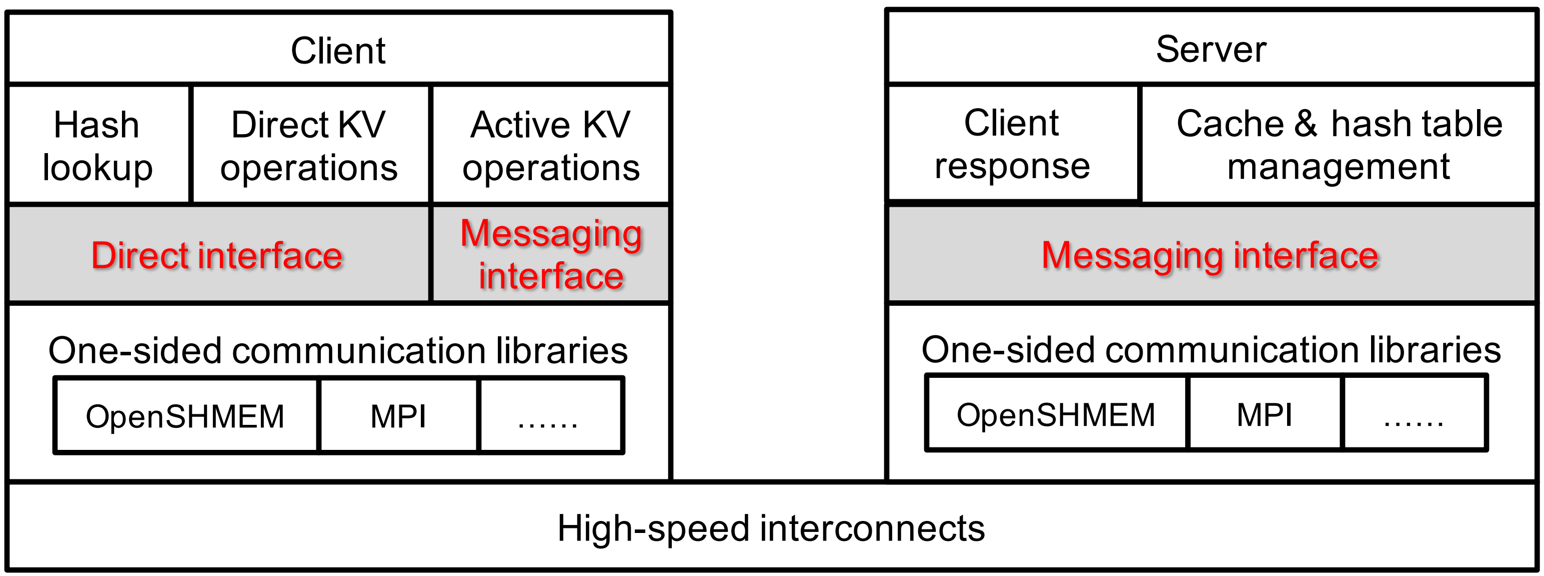
- **Modular** communication architecture
  - Needs to be able to accommodate new one-sided communication libraries.
- **Portable** interface
  - More general and easy to implement.
- Examining the **suitability** and choosing the **best implementation approach** for each library.
  - Memory semantics: visibility of remote memory, ways to access remote memory.
  - Synchronization method: delivery of data, involvement of remote process, synchronization overhead.

Outline

- Background and Motivation
  - SHMEMCache
  - Why Portable SHMEMCache
- Design and Implementation
  - Modular architecture
  - Portable interface
  - Leveraging OpenSHMEM and MPI
- Experiment
- Conclusion and Future Work

Modular Architecture

- A layer of communication interface is added to abstract the communication between client and server.
  - Modularizes the work of supporting new one-sided communication libraries.



Portable Interface

- Direct interface
  - Akin to common one-sided `Put` and `Get` but more general.
  - *Target memory = ID + offset*

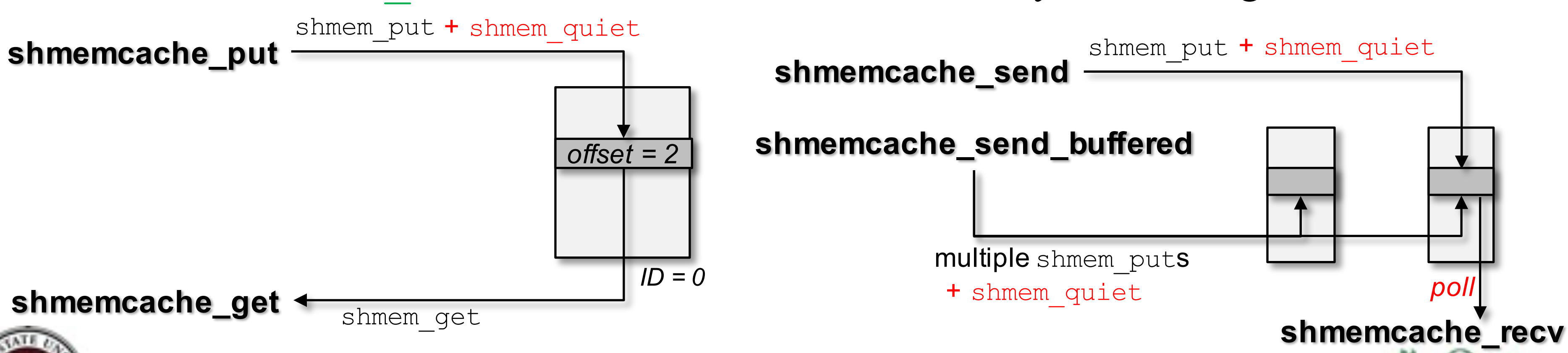
```
int shmemcache_put(void * src_buf, size_t length,
                   ProcessID dst_proc, MemoryID dst_mem,
                   size_t offset);
int shmemcache_get(void * dst_buf, size_t length,
                   ProcessID dst_proc, MemoryID dst_mem,
                   size_t offset);
```

- Messaging interface
  - Either one or multiple buffered messages of a *window* size.
    - Buffering enabled accordingly (e.g. when no response is required).

```
int shmemcache_send(Message * msg, ProcessID dst_proc);
int shmemcache_send_buffered(Message ** msgs, ProcessID dst_proc);
Message * shmemcache_recv(ProcessID dst_proc);
```

Leveraging OpenSHMEM

- Memory semantics
  - Shared memory model fits in nicely. Visible remote memory.
  - Translate memory address to memory ID + offset.
- Synchronization
  - Source PE uses `shmem_quiet` to assure data delivery.
    - `shmem_fence` NOT suitable: only assuring ordering.
  - Target PE simply polls local symmetric memory.
    - `shmem_wait` NOT suitable: less flexibility for the target PE



Direct Interface

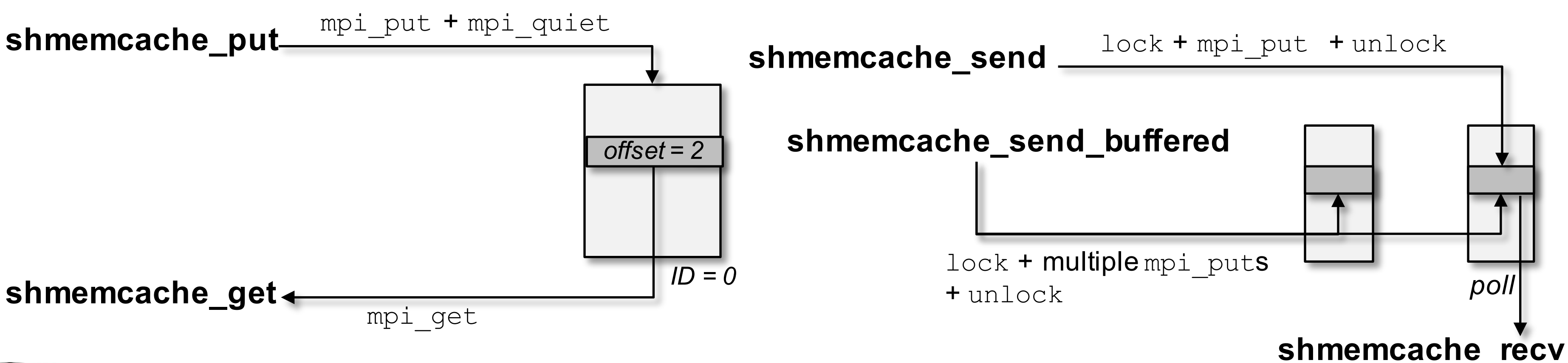
Active Interface

Leveraging MPI

- Memory semantics
  - RMA unified over RMA separate. Need hardware support.
  - Associate MPI windows with memory IDs.
- Synchronization
  - Post-and-wait : client start/complete, server post/wait.
    - NOT suitable: need exact matching of calls from client/server.
    - Similar reason to why `Isend/Irecv` is not suitable either.
  - Fence : every process synchronizes in an epoch.
    - NOT suitable: hard to determine a good duration of the epoch.
      - Short duration: high synchronization overheads for all.
      - Long duration: prolonged KV operation latency.

Leveraging MPI (cont.)

- Synchronization approach (cont.)
  - **Lock and unlock:** provide passive point-to-point synchronization, which is desired by SHMEMCache.
  - Using lighter-weight `lock-all` and `unlock-all`?
    - Not necessary. Client communicates with only one server each time.
- Implementation similar to the OpenSHMEM version.
  - But two synchronization calls are required each time.



Outline

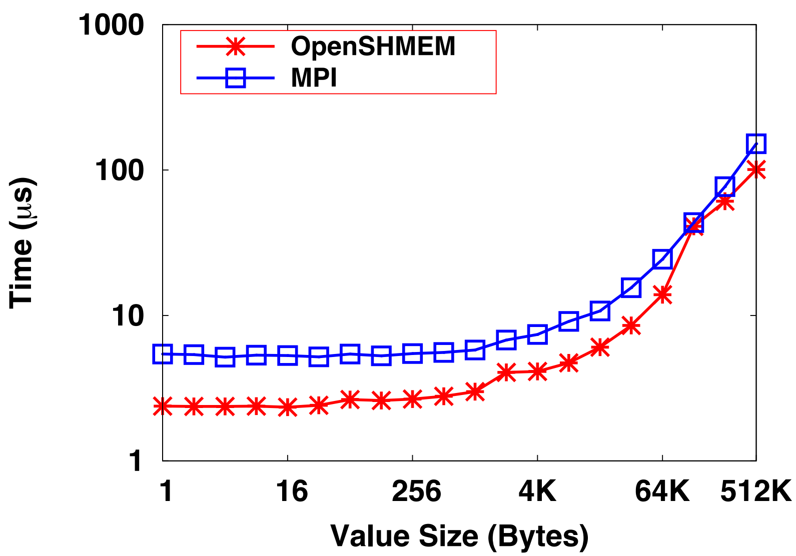
- Background and Motivation
  - SHMEMCache
  - Why Portable SHMEMCache
- Design and Implementation
  - Modular architecture
  - Portable interface
  - Leveraging OpenSHMEM and MPI
- Experiment
- Conclusion and Future Work

Experimental Setup

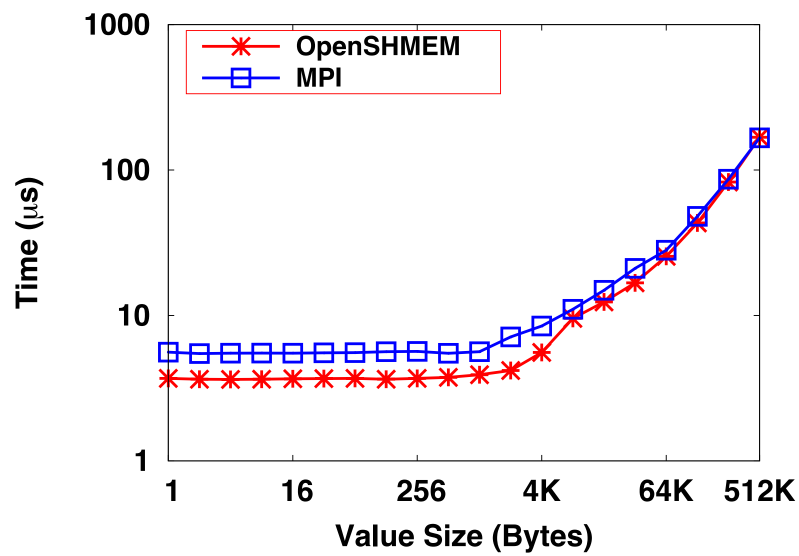
- Innovation
  - An in-house cluster with 21 dual-socket server nodes, each featuring 10 Intel Xeon(R) cores and 64 GB memory. All nodes are connected through an FDR Infiniband interconnect with the ConnectX-3 NIC.
- Titan supercomputer
  - Titan is a hybrid-architecture Cray XK7 system, which consists of 18,688 nodes and each node is equipped with a 16-core AMD Opteron CPU and 32GB of DDR3 memory.
- Workloads generated by YCSB
- Open MPI v2.1.0 for both OpenSHMEM and MPI versions of SHMEMCache

Direct KV Operation Latency

- Performance trend is similar on Innovation cluster (Inv) and Titan supercomputer (Titan).
- OpenSHMEM version has lower latency in general.
  - Key cause is MPI's higher synchronization overhead.
    - Optimization: MPI_MODE_NOCHECK assertion



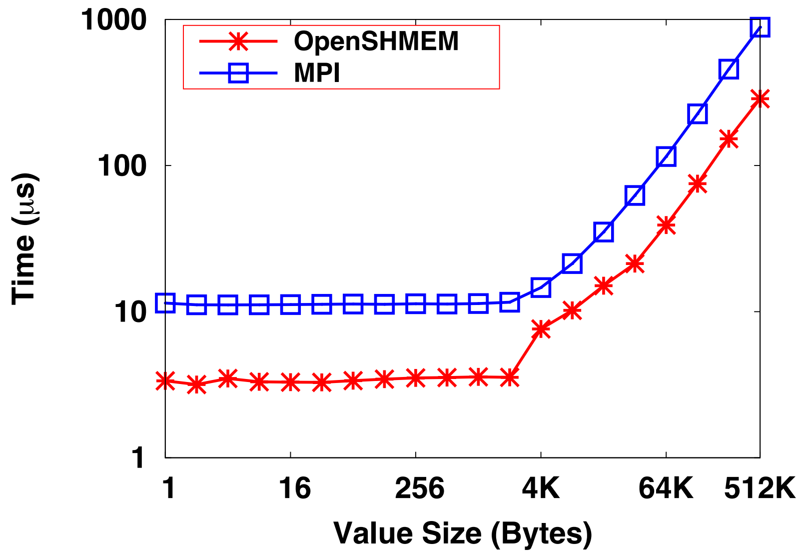
(a) Direct `GET` latency (Inv)



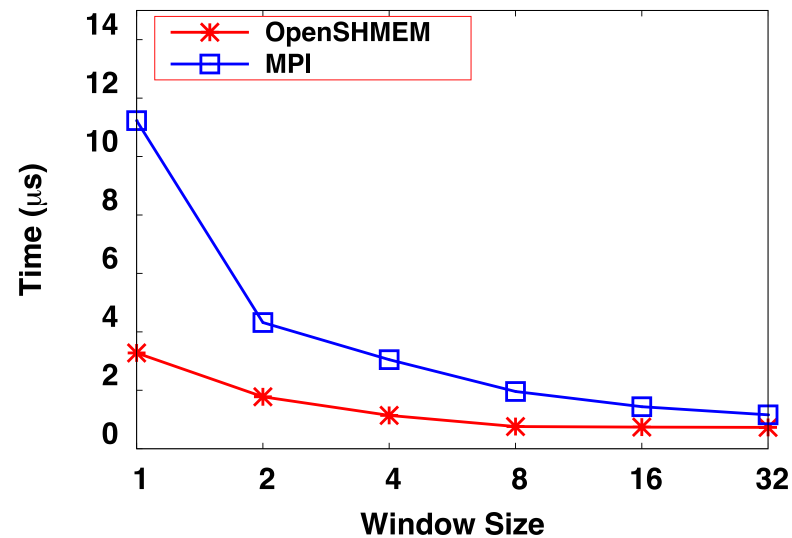
(b) Direct `SET` latency (Titan)

Active KV Operation Latency

- Active KV operation has larger performance difference between OpenSHMEM and MPI versions.
- Increasing messaging window size can mitigate the gap.
  - But only for limited scenarios.



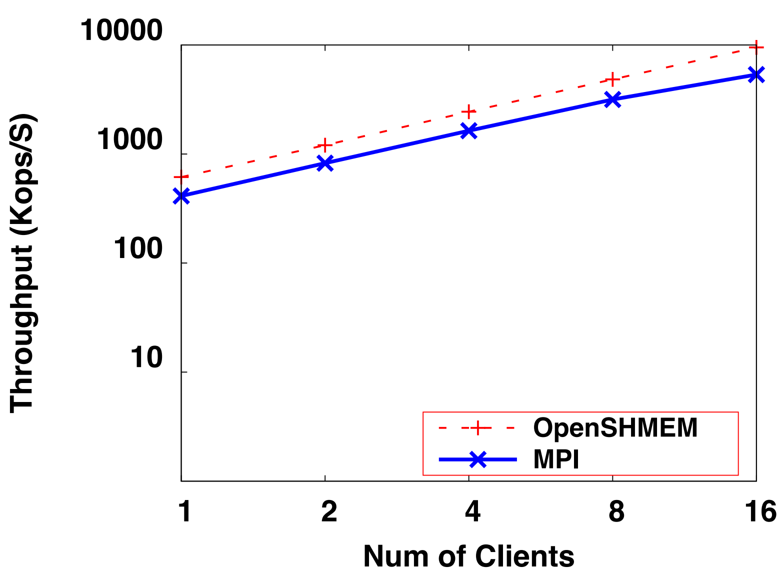
(a) Non-buffered Active `GET` latency (Inv)



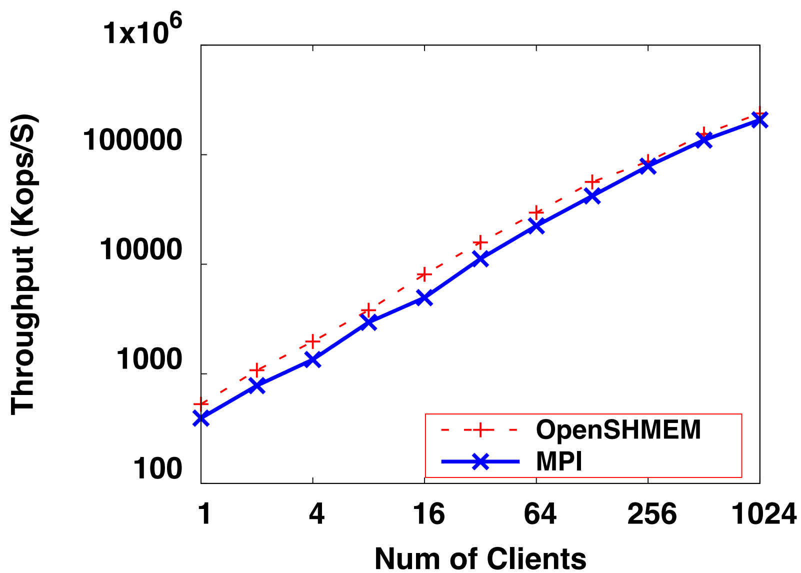
(b) Active `GET` latency with varying window sizes (Inv)

KV Operation Throughput

- OpenSHMEM version has slightly higher throughput in general.
- Both can scale well to 1024 nodes on Titan.



(a) Operation throughput (Inv)



(b) Operation throughput (Titan)

Conclusion

- We have extended SHMEMCache, a high-performance distributed key-value store to portable SHMEMCache.
- We have supported both OpenSHMEM and MPI one-sided communication for SHMEMCache.
- We have examined the performance of portable SHMEMCache on both commodity machines and Titan supercomputer.

Future Work

- In future, we will support more one-sided communication libraries.
  - The **shared memory model** and the abstraction of memory **ID+offset** are generally applicable.
    - PGAS family (CAF, UPC, etc.) have addressable remote memory similar to OpenSHMEM.
    - Similarly, lower-level communication libraries designed for PGAS (GASNet, OSPRI, etc.) also meet the needs.
  - Flexible **passive synchronization** point-to-point method is generally available.
    - CAF, UPC: lock/unlock
    - GASNet: try/wait for implicit-handle non-blocking operations
- We will also explore other use cases for one-sided communication, such as **graph processing**.

Acknowledgment

Oak Ridge National Laboratory

Department of Defense
United States of America

Thank You and Questions?