

SHMemCache: Enabling Memcached on the OpenSHMEM Global Address Model



Huansong Fu^{*}, Kunal SinghaRoy^{*},
Manjunath Gorentla Venkata[†],
Yue Zhu^{*}, Weikuan Yu^{*}

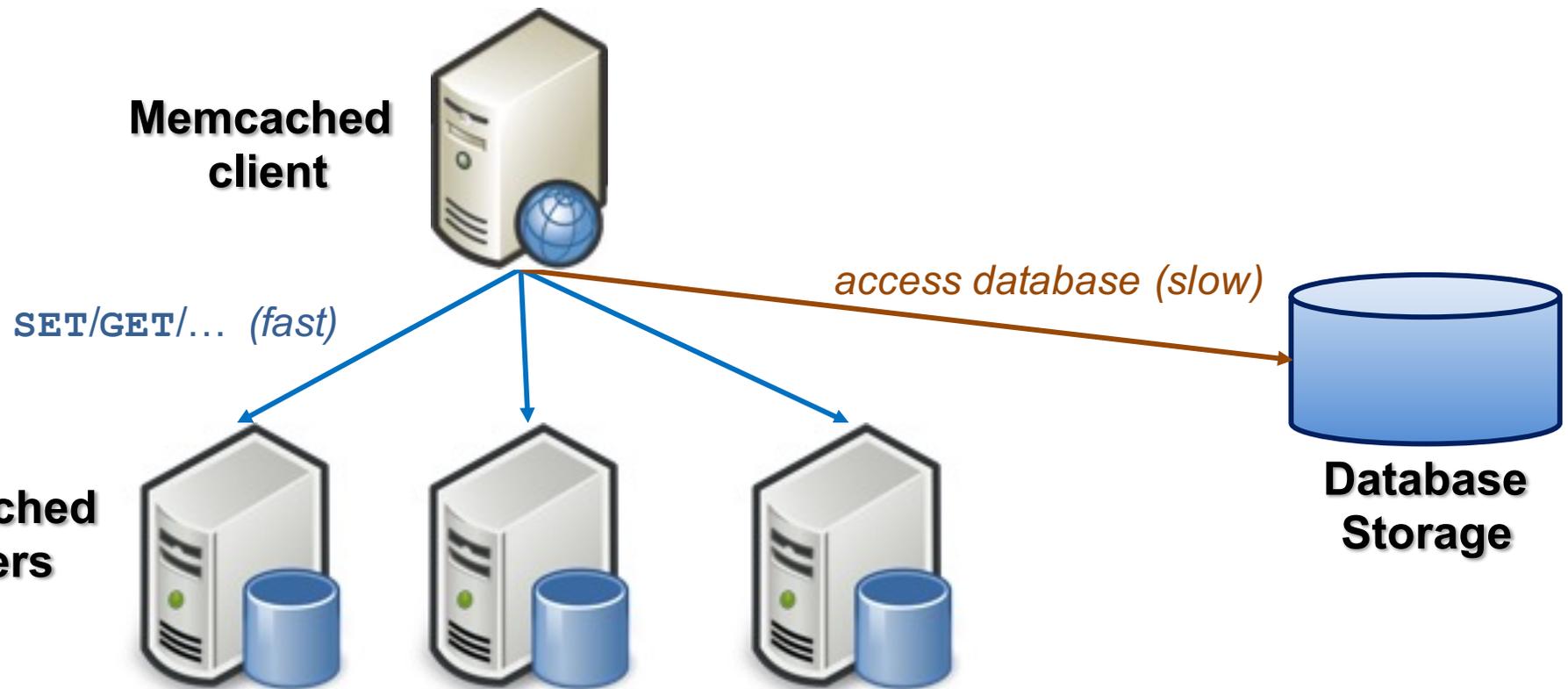
^{*}Florida State University

[†]Oak Ridge National Laboratory



Big Data and Memcached

- Enormous datasets need efficient data analytics tools. Big opportunities but also challenges come along.
- Memcached is an important in-memory key/value store that can cache hot key/value pairs for fast access.



Challenges of Big Data Analytics

- Performance is often limited by network or storage.
 - E.g., the existing open-source Memcached implementation uses TCP/IP socket communication.
- More generally: not embracing or being embraced by the fast-growing HPC systems.
 - Big data applications are not widely used on HPC systems and leveraging high performance interconnects.
 - HPC systems have not met the needs of big data analytics middleware very well.
- **Proposed solution** - **SHMemCache**: use OpenSHMEM to enable **Memcached**.



Why OpenSHMEM + Memcached?

- Memcached has SET and GET APIs that match with OpenSHMEM's one-sided communication very well.
- The integration can provide great portability for Memcached (and possibly other data analytics frameworks) to be deployed on various leadership facilities with the OpenSHMEM run-time, without knowing low-level details.
- The programming compatibility of OpenSHMEM and Memcached's similar memory-style addressing models can be explored.



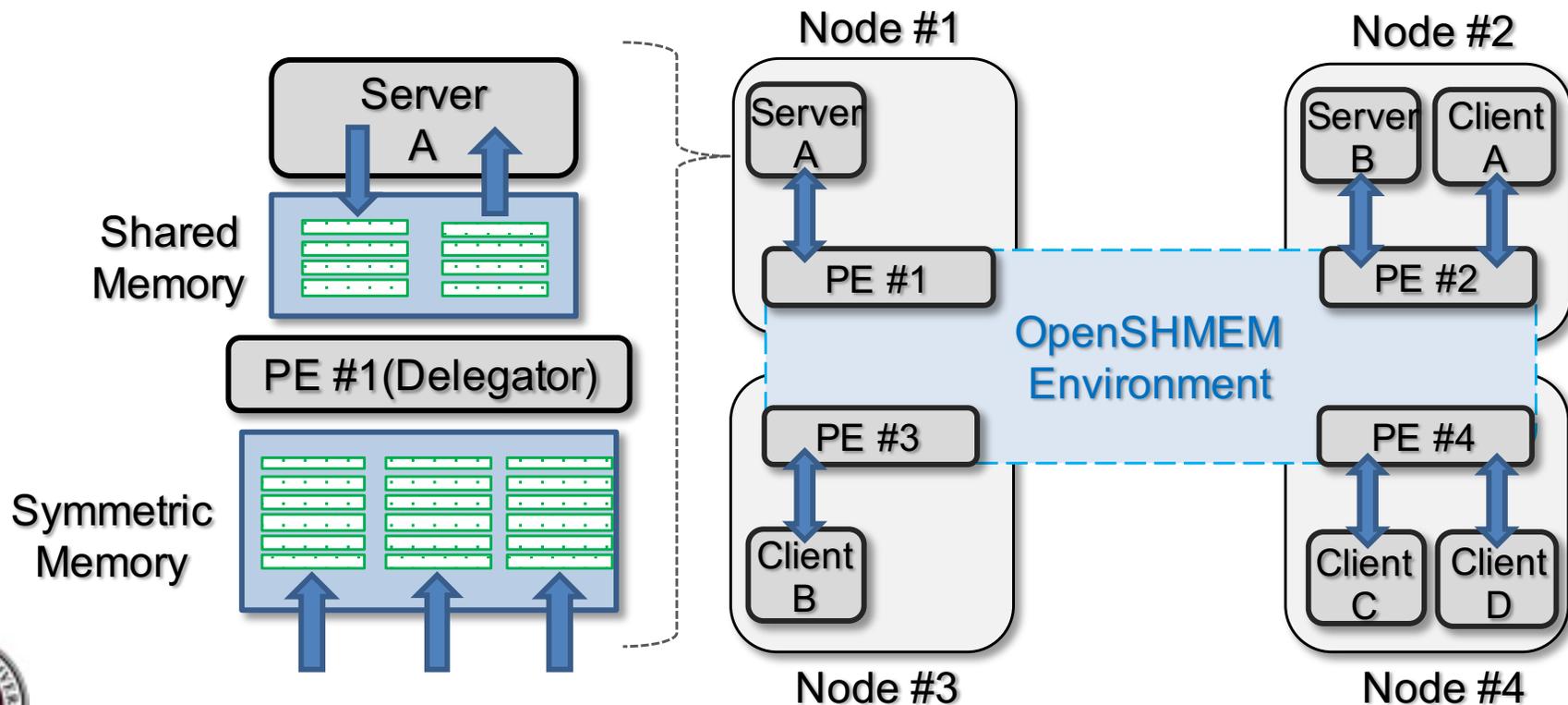
Outline

- Design of SHMemCache.
 - Overall structure.
 - Communication via shared & symmetric memory.
 - Server & client.
- Experiments
 - Latency and processing dissection.
 - Throughput.
- Conclusion



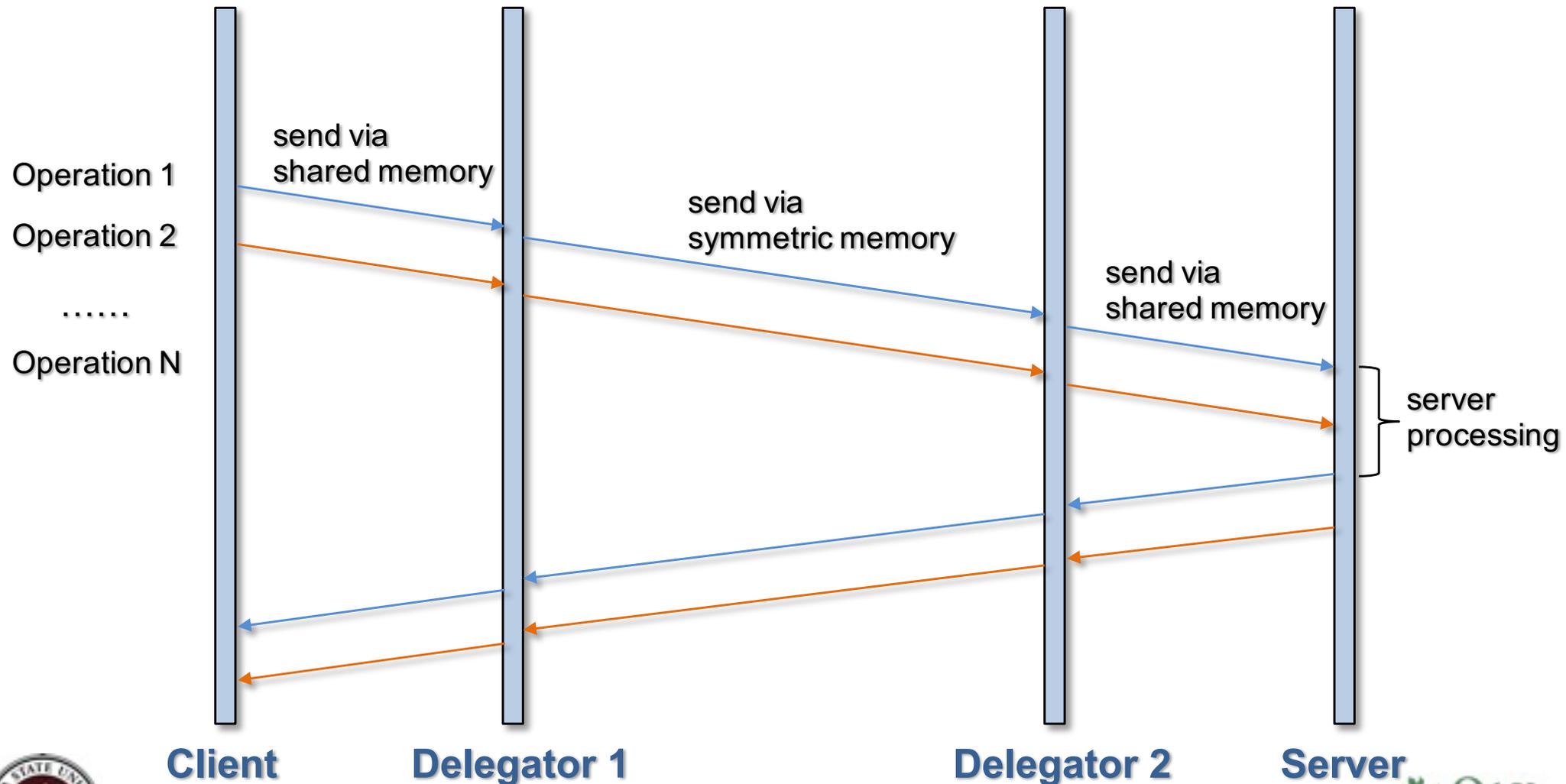
Overall Structure

- Client and server send/receive messages by accessing its own shared memory region.
- Per-node *delegators* forward the messages by sending to symmetric memory of destination PE.



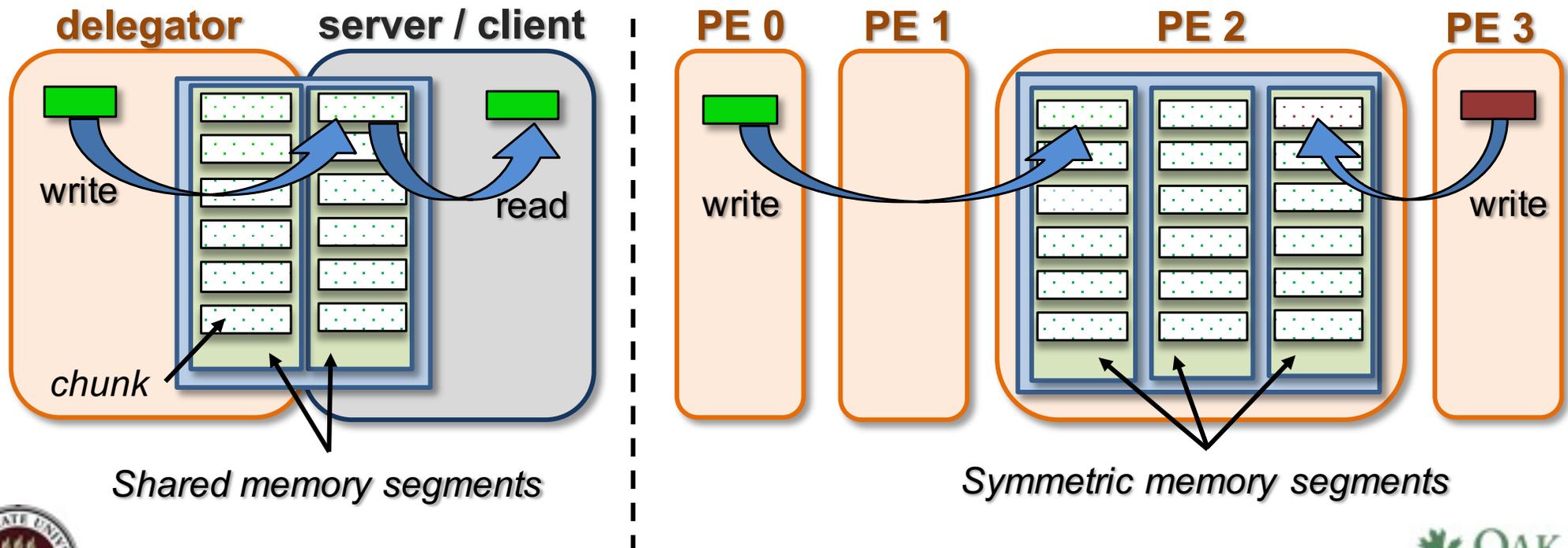
Operation Stages and Pipelining

- Pipeline multiple stages of a Memcached operation.



Memory Communication

- Every memory region includes a number of *segments*. Every segment is divided into a number of *chunks*. Each chunk can receive an *operation item*.
- Memory consumption depends on demand.
 - E.g., a 4-node system has 3 symmetric memory segments per node and 2 shared memory segments per server/client.



Send/Recv via Shared Memory

- Mutex-free for better performance.
 - Sender writes to a chunk if it is available for writing.
 - Receiver polls from a chunk if there is data (polls `use_flag`) and data is complete (polls `comp_flag`).
- Mutex remains as an option.

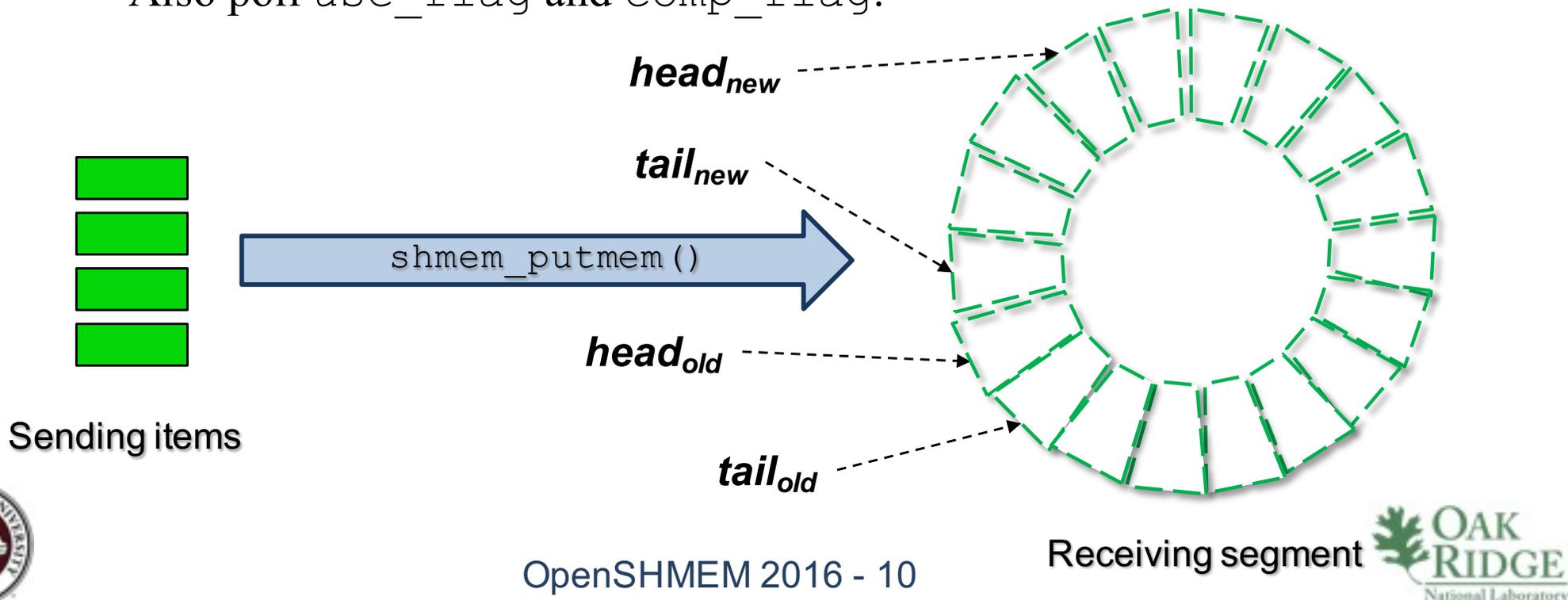
```
1: function MEM_SEND(des , src)
2:   tmp_flag ← src.use_flag
3:   src.use_flag ← OCCUPIED
4:   memcpy(des, src, copy_size)
5:   des.use_flag ← tmp_flag
6:   src.use_flag ← tmp_flag
7: end function
```

```
1: function MEM_RECV(s)
2:   if s.use_flag ≠ UNUSED then
3:     calculate position of comp_flag
4:     while s.completion_flag ≠ SET do
5:       continue
6:     end while
7:   end if
8:   memcpy(buf, s, copy_size)
9:   s.use_flag ← UNUSED
10: end function
```



Send/Recv via Symmetric Memory

- Sender maintains a *head* and a *tail* pointing to the receiving segment. Head never moves beyond *tail*.
 - *Head*: the next available chunk to send data.
 - *Tail*: the last available chunk before head.
- Receiver polls data and recycles the chunk so the chunk becomes available to the sender again.
 - Also poll `use_flag` and `comp_flag`.



Server and Client

- Main functionality of the existing Memcached server and client are maintained.
 - Server: two-step hashing, memory slabs, etc.
 - Client (*libMemcached*): random k/v generation, benchmarks, etc.
- Minimal changes but careful tuning.
 - Only add the new communication interfaces.
 - Server use multiple working threads.
 - Important parameters: number of slots, number of workers, etc.



Outline

- Design of SHMemCache.
 - Overview.
 - Goals and principles.
 - Overall structure.
 - Communication interfaces.
 - Memory communication.
 - Symmetric Ring Buffer.
 - Server & client.
- Experiments
 - Latency and time dissection.
 - Throughput.
- Conclusion



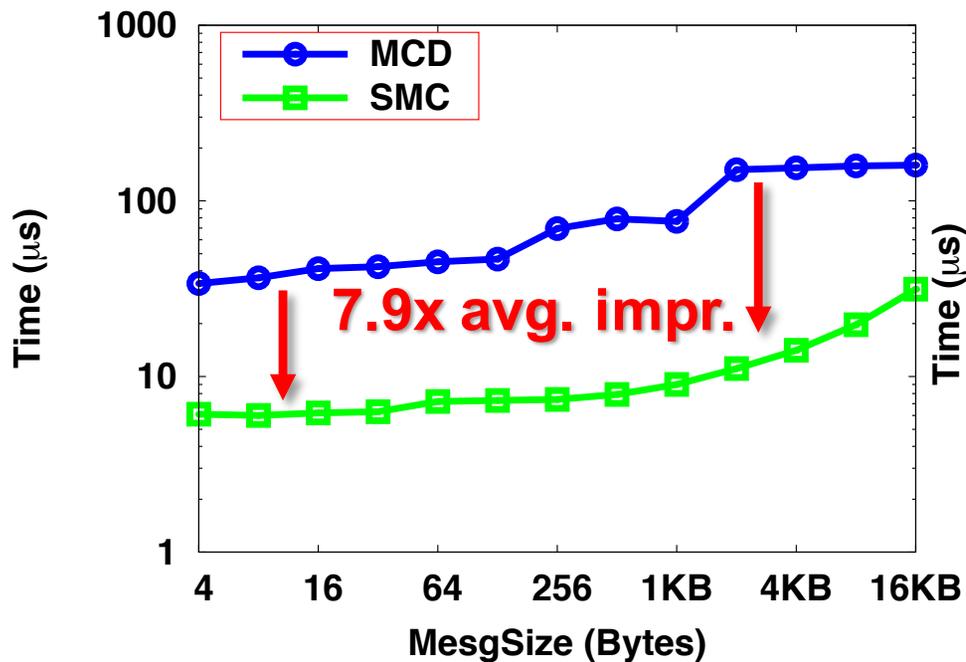
Experiment Setup

- 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.
- Software versions: OpenSHMEM in OpenMPI v1.10.3, Memcached v1.4.25 and libMemcached 1.0.18.

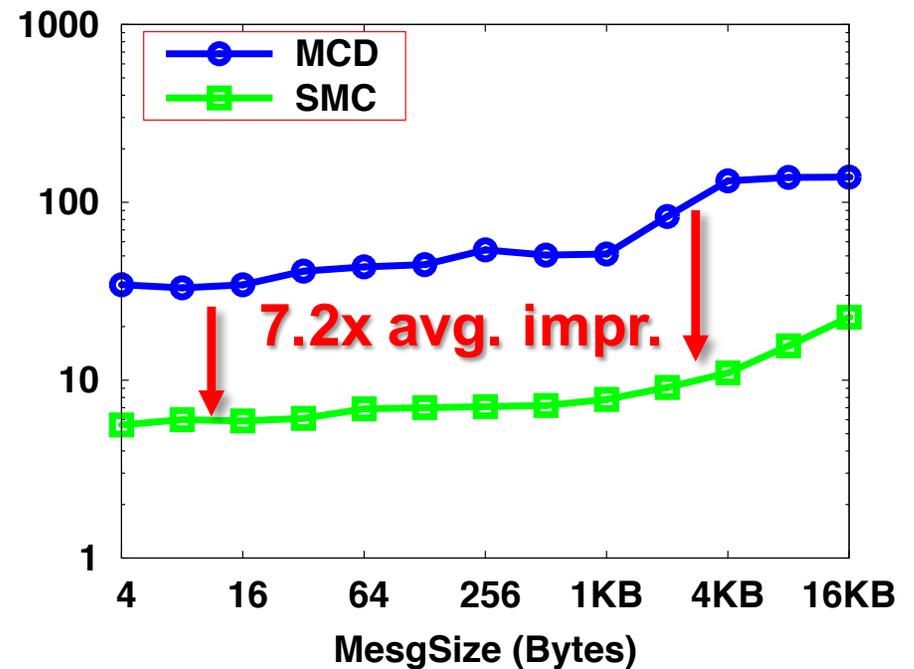


Latency

- Memcached (**MCD**) vs. SHMemCache (**SMC**).
 - As low as $6 \mu\text{s}$ for SET, and $5 \mu\text{s}$ for GET.
 - In average, SMC outperforms MCD by 7.9 times for SET and 7.2 times for GET.



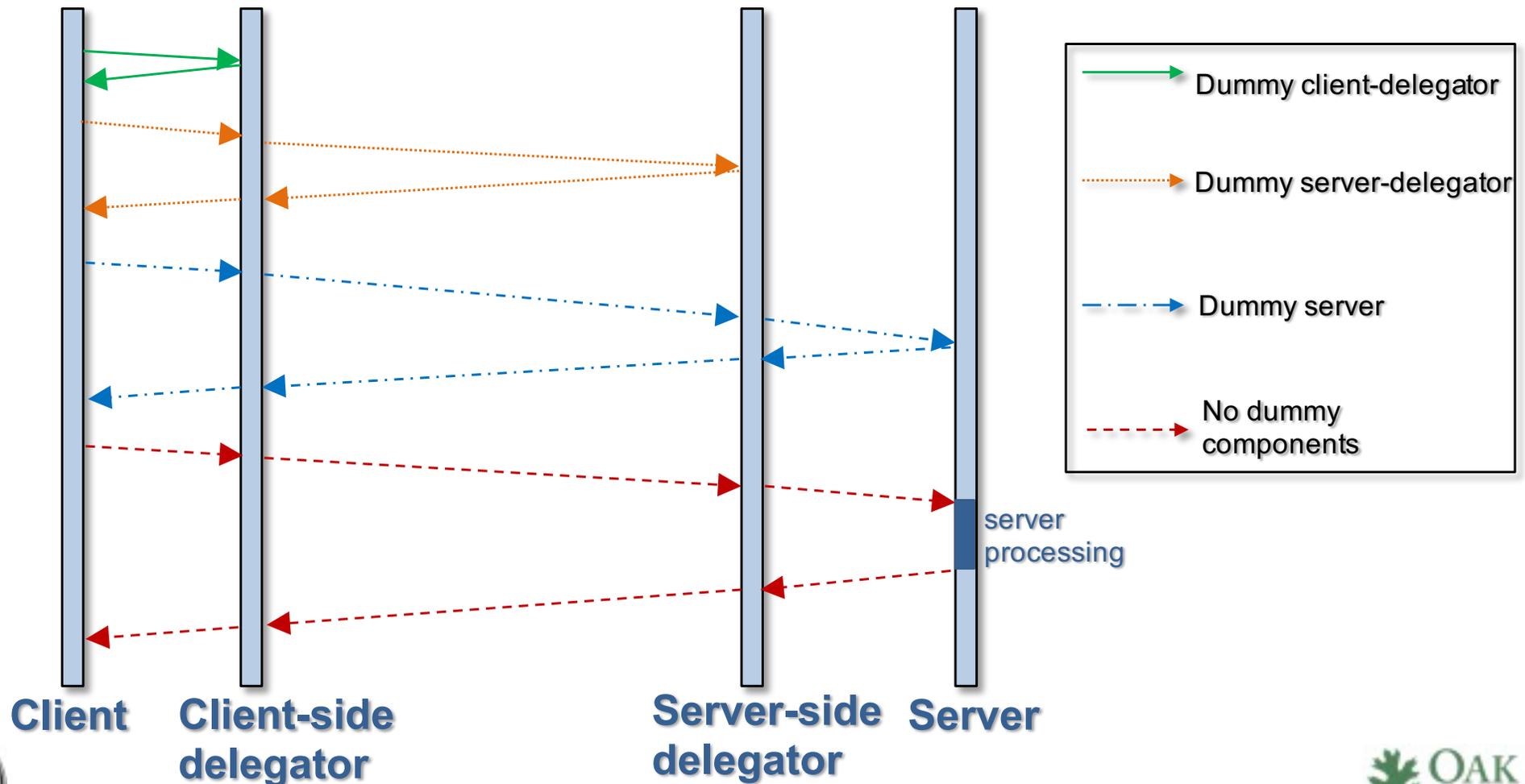
(1) SET



(2) GET

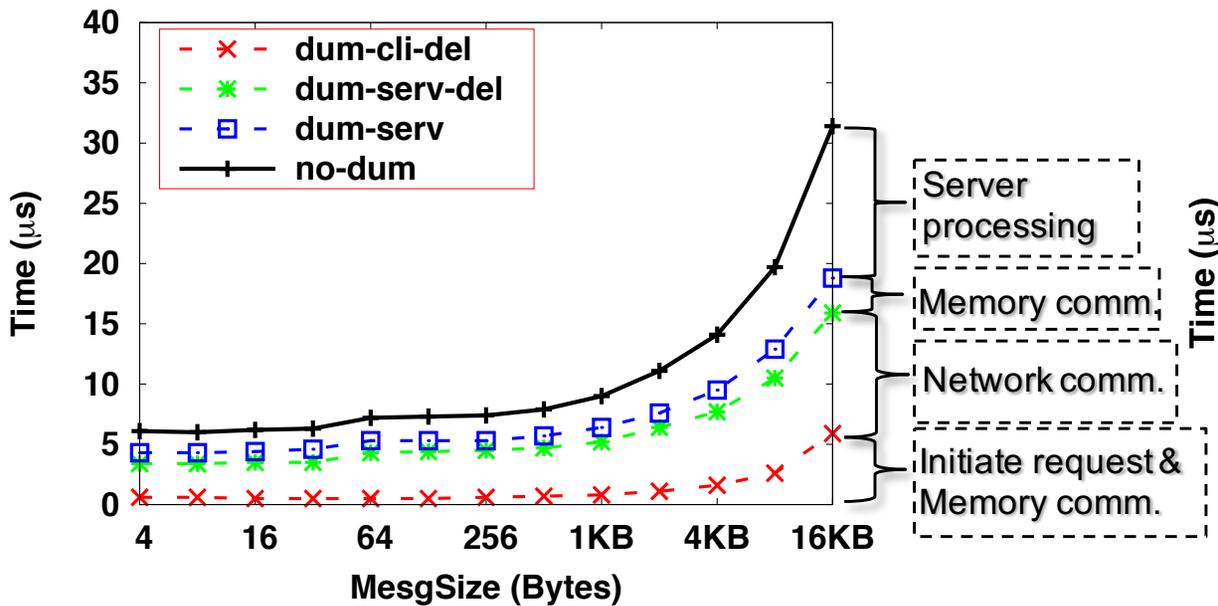
Processing Dissection

- Dummy component responses messages immediately without passing them forward.

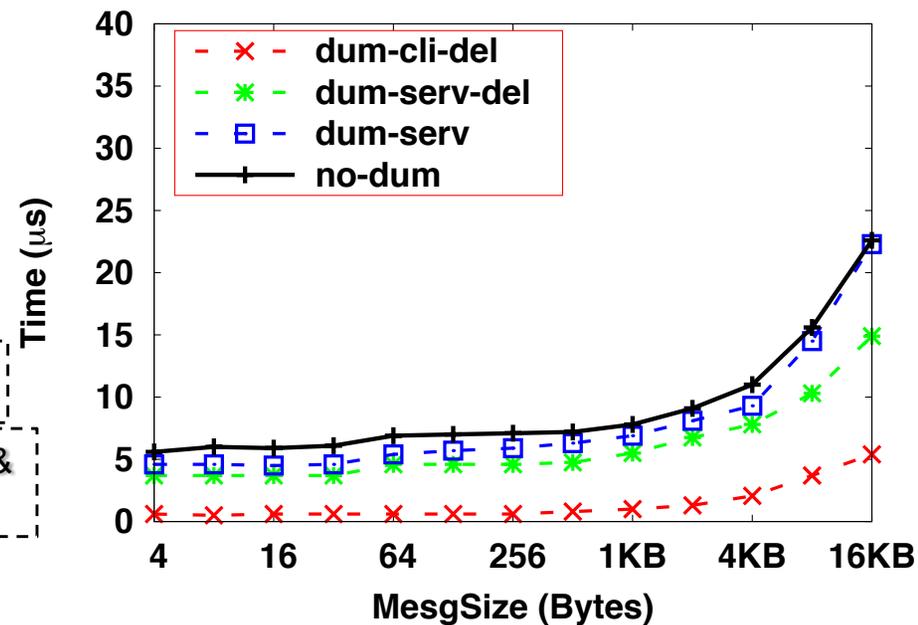


Processing Dissection (cont.)

- Dummy test echoes with our expectation.
 - In general, network communication is the most time-consuming part.
 - SET takes more time for server processing.
 - **SHMemCache adds little extra overheads.**



(1) SET

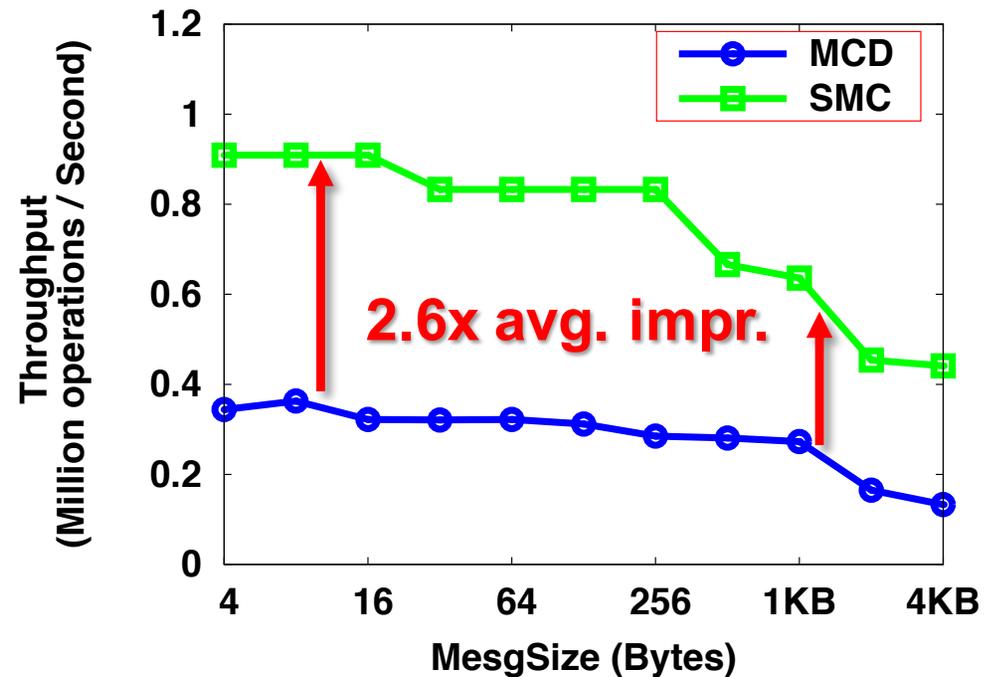
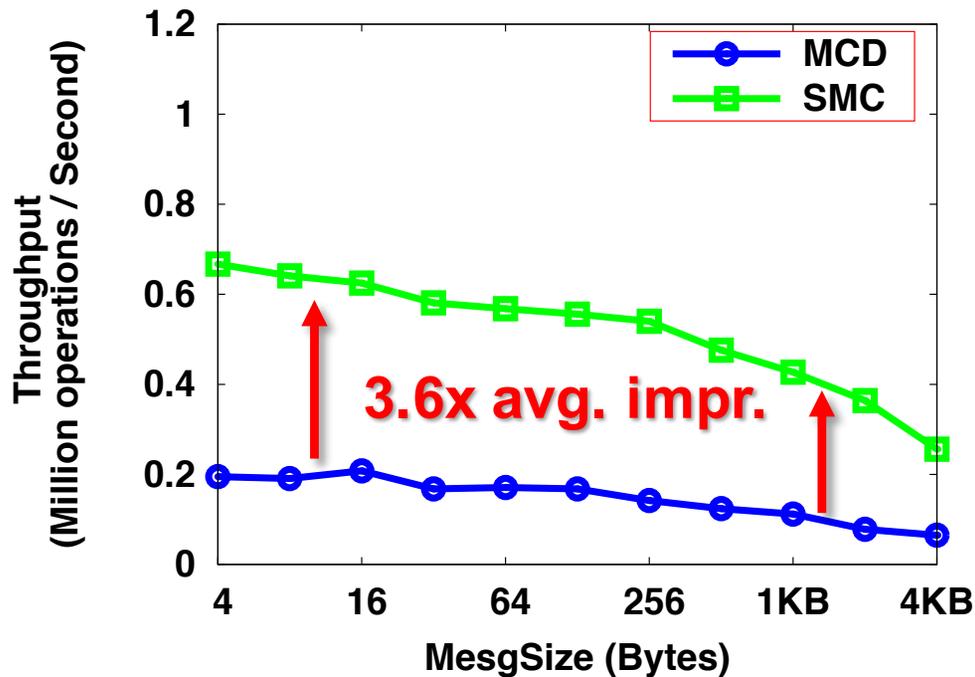


(2) GET



Throughput

- SET and GET throughput (operations per second) with one client and one server.
 - More than 650,000 operations/second for SET and 900,000 for GET.
 - In average, SHMemCache outperforms Memcached by a factor of 3.6 for SET and 2.6 for GET.



GET throughput

OpenSHMEM 2016 - 17



Conclusion and Future Work

- OpenSHMEM's one-sided communication and global memory pool can be used to improve big data analytics frameworks such as Memcached.
- SHMemCache's communication interfaces are simple and easily portable.
- SHMemCache can achieve much better performance for both latency and throughput than Memcached.
- In future, we will further evaluate the scalability and elasticity of SHMemCache. We will also try to directly program Memcached operations with OpenSHMEM.



Acknowledgment

- Guidance and comments from Dr. Neena Imam and Dr. Manjunath Gorentla Venkata.



Thank You and Questions?

