Enabling Efficient Data Infrastructure and Analytics on HPC Systems

Huansong Fu Weikuan Yu

Department of Computer Science
Florida State University
{*fu,yuw*}*@cs.fsu.edu*

***Abstract*—We propose to leverage PGAS and one-sided communication for building data infrastructure and analytics frameworks on HPC systems. Specifically, we have developed SHMEMCache, a distributed in-memory key-value store and SHMEMGraph, a balanced graph processing framework. We have tackled unique challenges in data consistency, load balancing, etc. with novel design features. The experimental results show that SHMEMCache achieves significant performance improvements over other KV stores such as Memcached in terms of latency and throughput. In addition, SHMEMGraph achieves an average improvement of 35.5% in terms of job completion time compared to a state-of-the-art graph processing system called Gemini.**

I. Introduction

There has been a rise in the interest to leverage HPC systems for big data analytics. Several prior studies have attempted to use HPC mechanisms in order to develop various components for high-speed distributed data analytics, including data infrastructure such as key-value (KV) stores and data analytics applications such as graph processing.

In the meantime, PGAS (Partitioned Global Address Space) programming models have gained wide popularity on many HPC systems. As an important communication paradigm supported by most PGAS libraries, one-sided communication features several advantages compared to its two-sided counterpart. Firstly, most one-sided communication libraries can naturally leverage RDMA and the underlying high-performance interconnect (e.g. Infiniband) for better communication efficiency. Secondly, in contrast to the costly data movement in two-sided communication, one-sided communication has zero-copy and kernel-bypass abilities. Thirdly, without explicit synchronization operation on the receiver's side, one-sided operations carries much less synchronization requirement.

In light of its increasing popularity and distinct advantages, my work has examined the potential of PGAS model and one-sided communication in building data infrastructure and supporting data analytics on HPC systems. Particularly, we have found that it offers a number of great opportunities for developing key-value stores and graph processing frameworks. Firstly, PGAS has similar memory-style addressing that makes it a good match for building in-memory storage, which is essential to both KV stores and in-memory graph processing. In addition, the fact that one-sided communication can relieve the receiver of participation in communication can greatly increase a KV store server's processing capacity. Moreover, the relaxed synchronization requirement can alleviate the imbalance issue existed in graph processing. Furthermore, many one-sided communication libraries, such as OpenSHMEM [1], can deliver good performance on a variety of platforms. Such portability can facilitate the deployment of data infrastructure and analytics frameworks on both commodity servers and the HPC systems without significant increase in the amount of effort required to do so.

II. Design Highlights

A. SHMEMCache

While SHMEMCache can manage KV pairs in a global memory space and support remote read or write operations through one-sided operations, conflicts can happen when multiple concurrent operations are issued to the same KV pair. These are often categorized as *read-write* and *write-write races* [2].To resolve these two races, we have introduced a mechanism called *read-friendly exclusive write*. It allows concurrent reads to verify the data consistency through a lightweight version check while precluding concurrent writes using locks.

Moreover, traditional KV store such as Memcached [3] uses linked lists to resolve hash collisions, causing the problem of *pointer chasing*, where it needs to follow multiple pointers to locate a KV pair. This issue becomes costly when the client tries to locate a KV pair remotely in SHMEMCache, which we refer to as *remote pointer chasing*. To mitigate its impact, we have designed a *set-associative hash table* that accommodates multiple pointers in one hash entry and a per-client *pointer directory* that caches remote pointers for fast access.

Furthermore, managing cached KV pairs in SHMEMCache is also challenging. In SHMEMCache, a server may not be aware of clients accessing KV pairs while it is deciding whether to evict KV pairs based on the least recently used (LRU) policy. On the other hand, the client is unaware of the server's decision to evict KV pairs, so it cannot invalidate its cached pointers for those KV pairs. Explicit synchronization about the access or eviction history between the server and the client could incur significant network and processing overheads. To solve those issues, we relax the definition of

the *recency* of a KV pair from an exact time point of its last access to a small *time range* that its last access belongs to. This relaxation enables us to update the recency of KV pairs and inform client about evicted KV pairs in a coarse-grained manner, without incurring excessive overheads.

B. SHMEMGraph

SHMEMGraph is an efficient and balanced graph processing framework that uses one-sided operations to exchange remotely-accessible graph data between different nodes in a shared-memory style Several benefits comes with such a design. Firstly, the memory-style addressing matches up the distributed in-memory graph processing better than the conventional view. Secondly, the one-sided operations that are used in the shared memory model can improve the communication efficiency of distributed graph processing, especially with RDMA. Thirdly, by leveraging the flexibility of one-sided operations, the aforementioned imbalance issues can be mitigated through more flexible and fine-grained load balancing, which is prohibitively difficult with the traditional two-sided communication.

In our work, we show SHMEMGraph's design over the state-of-the-art Gemini framework [4] as proof-of-concept. SHMEMGraph addresses the inefficiency and imbalance issues by incorporating several novel techniques. First of all, its basic communication channel uses only one single thread to conduct one-sided `Put` for transferring data, which is more efficient and scalable than the existing solutions. Secondly, we divide the long and imbalanced computation process into finer-grained pieces so that they can be better overlapped and cause less node idleness. Thirdly, we further exploit the flexibility of one-sided `Get` to implement a communication balancing mechanism to further reduce the delaying time.

III. Experimental Setup

A. SHMEMCache

The experiments of SHMEMCache are conducted on two systems. The first one is an in-house cluster of 21 server nodes connected through an FDR Infiniband interconnect with the ConnectX-3 NIC. The second is the *Titan* supercomputer at Oak Ridge National Laboratory, which has a Gemini interconnect. We use the OpenSHMEM implementation in OpenMPI v2.1 on Innovation and Cray SHMEM v7.4.0 on Titan. Unless otherwise specified, on each machine we run only one client or server, without collocating them. The microbenchmark we use consists of 100 thousand operations including either `SET` or `GET` conducted on 1 thousand randomly generated KV pairs of varying sizes. We compare the results to the original Memcached [3] (version 1.4.25) and HiBD, which leverages one-sided RDMA operations for Memcached [5]. But unlike SHMEMCache, the HiBD server needs to conduct RDMA write to return the `GET` result to the client, and RDMA read to retrieve the KV pair from the client for `SET` [6]. This evaluation aims to demonstrate SHMEMcache's improvement compared to the traditional Memcached over socket, and also RDMA-accelerated HiBD due to more salient design features.

B. SHMEMGraph

Due to space limitation, we only focus on the comparison between SHMEMGraph and Gemini [4]. But we have confirmed that both Gemini and SHMEMGraph outperform other state-of-the-art frameworks such as PowerGraph [7] and PowerLyra [8]. The experiments for SHMEMGraph are conducted on the same in-house cluster and Titan supercomputer as for SHMEMCache. We use Open MPI [9] v3.0.0 for both thread-safe (MPI) and non-thread-safe (OpenSHMEM) one-sided operations used in SHMEMGraph. We use four real-world graph loads, varying from 0.1 to 3.8 billion edges. We evaluate five representative graph algorithms, including PageRank (PR), Connected Components (CC), Single Source Shortest Path (SSSP), Betweenness Centrality (BC) and Breadth-First Search (BFS). For results that are shown in the poster, they are the results of running 10 iterations of PageRank or other algorithms on twitter-2010 on 4 nodes. Similar to Gemini, SHMEMGraph can scale up to 4 or 8 nodes for various applications and beyond that, per-node communication cost can no longer be overlapped by per-node computation cost.

IV. Conclusion and Future Plan

Our progress and plan in supporting data infrastructure and analytics with SHMEMCache and SHMEMGraph on HPC systems is presented below.

- Building and evaluating a high-performance distributed KV store, enabling the YCSB workload on it (finished).
- Building and evaluating an efficient and balanced graph processing framework based on Gemini (finished).
- Showing portability of SHMEMCache and SHMEMGraph for various lower-level communication libraries and higher-level graph processing engines (finished for SHMEMCache).
- Showing integration of SHMEMGraph and SHMEMCache for graph processing problems running on top of KV stores.

References

[1] S. Pophale, R. Nanjegowda, T. Curtis, B. Chapman, H. Jin, S. Poole, and J. Kuehn, "Openshmem performance and potential: A npb experimental study," in *The 6th Conference on Partitioned Global Address Space Programming Models (PGAS12)*, Citeseer, 2012.

[2] C. Mitchell, Y. Geng, and J. Li, "Using one-sided rdma reads to build a fast, cpu-efficient key-value store.," pp. 103–114, 2013.

[3] "Memcached." https://memcached.org/downloads.

[4] X. Zhu, W. Chen, W. Zheng, and X. Ma, "Gemini: A computation-centric distributed graph processing system," in *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)*, 2016.

[5] "Hibd." http://hibd.cse.ohio-state.edu/.

[6] J. Jose, H. Subramoni, M. Luo, M. Zhang, J. Huang, M. Wasi-ur Rahman, N. S. Islam, X. Ouyang, H. Wang, S. Sur, *et al.*, "Memcached design on high performance rdma capable interconnects," pp. 743–752, 2011.

[7] J. E. Gonzalez, Y. Low, H. Gu, D. Bickson, and C. Guestrin, "Powergraph: Distributed graph-parallel computation on natural graphs.," in *OSDI*, vol. 12, p. 2, 2012.

[8] R. Chen, J. Shi, Y. Chen, and H. Chen, "Powerlyra: Differentiated graph computation and partitioning on skewed graphs," in *Proceedings of the Tenth European Conference on Computer Systems*, p. 1, ACM, 2015.

[9] "Open MPI." https://www.open-mpi.org/.