High-Performance and Scalable Support for Big Data Stacks with MPI

Talk at the 2023 Annual MVAPICH User Group (MUG) Conference

by

Aamir Shafi, Kinan Al Attar, Jinghan Yao

The Ohio State University

E-mail: shafi.16@osu.edu

https://cse.osu.edu/people/shafi.16
Presentation Outline

• Introduction to Big Data Analytics

• Overview, Design and Implementation
  – MPI4Spark
  – MPI4Dask

• Performance Evaluation
  – MPI4Spark
  – MPI4Dask

• Related Publications and Summary
Introduction to Big Data Analytics

• Big Data has changed the way people understand and harness the power of data, both in the business and research domains
• Big Data has become one of the most important elements in business analytics
• Big Data and High Performance Computing (HPC) are converging to meet large scale data processing challenges
• Dask and Spark are two popular Big Data processing frameworks
• Sometimes also called Data Science
Presentation Outline

• Introduction to Big Data Analytics

• Overview, Design and Implementation
  – MPI4Spark
  – MPI4Dask

• Performance Evaluation
  – MPI4Spark
  – MPI4Dask

• Related Publications and Summary
MPI4Spark: Using MVAPICH2 to Optimize Apache Spark

- The main motivation of this work is to utilize the communication functionality provided by MVAPICH2 in the Apache Spark framework
  - MPI4Spark relies on Java bindings of the MVAPICH2 library
- Spark’s default Shuffle Manager relies on Netty for communication:
  - Netty is a Java New I/O (NIO) client/server framework for event-based networking applications
MPI4Spark Interconnect Support

- The current approach is different from its predecessor design, RDMA-Spark (http://hibd.cse.ohio-state.edu)
  - RDMA-Spark supports only InfiniBand and RoCE
  - Requires new designs for new interconnect

- MPI4Spark supports multiple interconnects/systems through a common MPI library
  - Such as InfiniBand (IB), Intel Omni-Path (OPA), HPE Slingshot, RoCE, and others
  - No need to re-design the stack for a new interconnect as long as the MPI library supports it
Launching Spark using MPI with Dynamic Process Management

**Step A:** Launch 4 Wrapper Processes
(for e.g. mpiexec -np 4 .. SparkMPI.java)

**Step B:** Each Wrapper Process Forks Spark Processes

**Step C:** Launch 2 Executor Processes
MPI_Comm_spawn_multiple()
Next MPI4Spark Release (v0.2)

• MPI4Spark 0.2 release adds support for the YARN cluster manager:
  - Will be available from: http://hibd.cse.ohio-state.edu

• Features:
  • Based on Apache Spark 3.3.0
  • (NEW) Support for YARN cluster manager
  • Compliant with user-level Apache Spark APIs and packages
  • High performance design that utilizes MPI-based communication
    • Utilizes MPI point-to-point operations
    • Relies on MPI Dynamic Process Management (DPM) features for launching executor processes for the standalone cluster manager
  • (NEW) Relies on Multiple-Program-Multiple-Data (MPMD) launcher mode for the YARN cluster manager
  • Built on top of the MVAPICH2-J Java bindings for MVAPICH2 family of MPI libraries
  • Tested with
    • (NEW) OSU HiBD-Benchmarks, GroupBy and SortBy
    • (NEW) Intel HiBench Suite, Micro Benchmarks, Machine Learning and Graph Workloads
    • Mellanox InfiniBand adapters (EDR and HDR 100G and 200G)
    • HPC systems with Intel OPA interconnects
    • Various multi-core platforms
Presentation Outline

• Introduction to Big Data Analytics

• Overview, Design and Implementation
  – MPI4Spark
  – MPI4Dask

• Performance Evaluation
  – MPI4Spark
  – MPI4Dask

• Related Publications and Summary
MPI4Dask: MPI backend for Dask

• Dask is a popular task-based distributed computing framework:
  – Scales Python applications from laptops to high-end systems
  – Builds a task-graph that is executed lazily on parallel hardware

• Dask Distributed library historically had two communication backends:
  – TCP: Tornado-based
  – UCX: Built using a GPU-aware Cython wrapper called UCX-Py

• Designed and implemented MPI4Dask communication device:
  – MPI-based backend for Dask
  – Implemented using mpi4py (Cython wrappers) and MVAPICH2
  – Uses Dask-MPI to bootstrap execution of Dask programs
Dask Distributed Execution Model

• Key characteristics:
  1. Scalability
  2. Elasticity
  3. Support for coroutines
  4. Serialization/De-serialization to data to/from GPU memory
MPI4Dask in the Dask Architecture

Dask

- Dask Bag
- Dask Array
- Dask DataFrame
- Delayed
- Future

Task Graph

Dask-MPI
- Dask-CUDA
- Dask-Jobqueue

Distributed

Scheduler
Worker
Client

Comm Layer
- tcp.py
- ucx.py
- MPI4Dask

High Performance Computing Hardware

Laptops/ Desktops

TCP
UCX
MVAPICH2
UCX-Py (Cython wrappers)
mpi4py

Network Based Computing Laboratory
MUG 2023
MPI4Dask: Bootstrapping and Dynamic Connectivity

• Several ways to start Dask programs:
  − Manual
  − Utility classes:
    • LocalCUDACluster, SLURMCluster, SGECluster, PBCCluster, and others

• MPI4Dask uses the Dask-MPI to bootstrap execution of Dask programs

• Dynamic connectivity is established using the asyncio package in MPI4Dask:
  − Scheduler and workers listen for incoming connections by calling asyncio.start_server()
  − Workers and client connect using asyncio.open_connection()
MPI4Dask Release

• MPI4Dask 0.3 was released in Feb ‘23 adding support for high-performance MPI communication to Dask:
  − Can be downloaded from: http://hibd.cse.ohio-state.edu

• Features:
  − (NEW) Based on Dask Distributed 2022.8.1
  − Compliant with user-level Dask APIs and packages
  − Support for MPI-based communication in Dask for cluster of GPUs
    • Implements point-to-point communication co-routines
    • Efficient chunking mechanism implemented for large messages
  − Built on top of mpi4py over the MVAPICH2-GDR library
  − Supports starting execution of Dask programs using Dask-MPI
  − Tested with
    • Mellanox InfiniBand adapters (FDR, EDR, and HDR)
    • (NEW) Various benchmarks used by the community (MatMul, Slicing, Sum Transpose, cuDF Merge, etc.)
    • (NEW) Various multi-core platforms
    • (NEW) NVIDIA V100 and A100 GPUs
Presentation Outline

• Introduction to Big Data Analytics
• Overview, Design and Implementation
  – MPI4Spark
  – MPI4Dask
• Performance Evaluation
  – MPI4Spark
  – MPI4Dask
• Related Publications and Summary
Weak Scaling Evaluation with OSU HiBD Benchmarks (OHB)

- The above are weak-scaling performance numbers of OHB benchmarks (GroupByTest and SortByTest) executed on the TACC Frontera system using the Standalone cluster manager in Spark.

- Speed-ups for the overall total execution time for 448GB with GroupByTest is 4.1x and 2.2x compared to IPoIB and RDMA, and for SortByTest the speed-ups are 3.8x and 1.5x, respectively.

- Speed-ups for the shuffle read stage for 112GB with GroupByTest are 13x compared with IPoIB and 5.6x compared to RDMA, while for SortByTest the speed-ups are 12.8x and 3.2x, respectively.

The above are weak-scaling performance numbers of OHB benchmarks (GroupByTest and SortByTest) executed on the TACC Frontera system using the YARN cluster manager in Spark.

- Speed-ups for the overall total execution time for SortByTest, 64 NodeManagers, are 4.5x and 2.3x compared to IPoIB and RDMA, and for GroupByTest, also 64 NodeManagers, the speed-ups are 3.8x and 2.5x, respectively.
- Speed-ups for the shuffle read stage for 896GB with GroupByTest are 6.8x compared with IPoIB and 4.4x compared to RDMA, while for SortByTest the speed-ups are 8.4x and 3.9x, respectively.
Performance Evaluation with Intel HiBench Workloads

- This evaluation was done on the TACC Frontera (IB) and the TACC Stampede2 (OPA) Systems
- This illustrates the portability of MPI4Spark on different interconnects
- We see a speed-up for the LR machine learning workload on Stampede2 of about 2.2x
- Speed-ups for the LDA machine learning workload on Frontera are 1.7x for both IPoIB and RDMA

Presentation Outline

• Introduction to Big Data Analytics

• Overview, Design and Implementation
  – MPI4Spark
  – MPI4Dask

• **Performance Evaluation**
  – MPI4Spark
  – MPI4Dask

• Related Publications and Summary
cuDF Merge Benchmark on the Cambridge Wilkes-3 System

- GPU-based Operation: using persist
  - Merge two GPU data frames, each with length of 32*1e8
  - Compute() will gather the data from all worker nodes to the client node, and make a copy on the host memory.
  - Persist() will leave the data on its current nodes without any gathering.

Wilke3 GPU System:
- 80 nodes
- 2x AMD EPYC 7763 64-core Processors
- 1000 GiB RAM
- Dual-rail Mellanox HDR200 IB
- 4x NVIDIA A100 SXM4 80 GB

MPI4Dask 0.3, Dask 2022.8.1, Distributed, 2022.8.1, MVAPICH2-3.0, UCX v1.13.1, UCX-py 0.27.00
MUG 2023

Network Based Computing Laboratory

- GPU-based Operation: using persist
  - Arrays are distributed on multiple GPUs
  - Compute() will gather the data from all worker nodes to the client node, and make a copy on the host memory.
  - Persist() will leave the data on its current nodes without any gathering.

Wilke3 GPU System:
- 80 nodes
- 2x AMD EPYC 7763 64-core Processors
- 1000 GiB RAM
- Dual-rail Mellanox HDR200 IB
- 4x NVIDIA A100 SXM4 80 GB

**Execution Time**

<table>
<thead>
<tr>
<th>Number of Dask Workers</th>
<th>MPI4Dask</th>
<th>UCX</th>
<th>IPoIB</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>35</td>
<td>30</td>
<td>25</td>
</tr>
<tr>
<td>4</td>
<td>30</td>
<td>25</td>
<td>20</td>
</tr>
<tr>
<td>8</td>
<td>25</td>
<td>20</td>
<td>15</td>
</tr>
<tr>
<td>16</td>
<td>20</td>
<td>15</td>
<td>10</td>
</tr>
<tr>
<td>32</td>
<td>15</td>
<td>10</td>
<td>5</td>
</tr>
</tbody>
</table>

On average, MPI4Dask is:
- 3.83x faster than UCX
- 8.70x faster than TCP

**Aggregated Throughput**

<table>
<thead>
<tr>
<th>Number of Dask Workers</th>
<th>MPI4Dask</th>
<th>UCX</th>
<th>IPoIB</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>5000</td>
<td>4000</td>
<td>3000</td>
</tr>
<tr>
<td>4</td>
<td>3000</td>
<td>2000</td>
<td>1000</td>
</tr>
<tr>
<td>8</td>
<td>2000</td>
<td>1000</td>
<td>500</td>
</tr>
<tr>
<td>16</td>
<td>1000</td>
<td>500</td>
<td>250</td>
</tr>
<tr>
<td>32</td>
<td>500</td>
<td>250</td>
<td>125</td>
</tr>
</tbody>
</table>

1.3x

2.1x

cupy GEMM Benchmark on the Cambridge Wilkes-3 System

MPI4Dask 0.3, Dask 2022.8.1, Distributed, 2022.8.1, MVAPICH2-3.0, UCX v1.13.1, UCX-py 0.27.00
**NumPy Array Slicing Benchmark on TACC Frontera CPU System**

1. **1.26x better on average**
   - On average, MPI4Dask is:
     - 1.37x faster than UCX
     - 1.51x faster than TCP

2. **3.17x better on average**
   - From 32 workers, we increase array size by 16 times
   - MPI4Dask 0.3 release

---

A. Shafi, J. Hashmi, H. Subramoni, and D. K. Panda, Efficient MPI-based Communication for GPU-Accelerated Dask Applications, CCGrid ’21
https://arxiv.org/abs/2101.08878

MPI4Dask 0.3 release
(http://hibd.cse.ohio-state.edu)
Presentation Outline

• Introduction to Big Data Analytics
• Overview, Design and Implementation
  – MPI4Spark
  – MPI4Dask
• Performance Evaluation
  – MPI4Spark
  – MPI4Dask
• Related Publications and Summary
Related Publications


• Towards Java-based HPC using the MVAPICH2 Library: Early Experiences K. Al Attar, A. Shafi, H. Subramoni, D. Panda HIPS '22 (IPDPSW), May 2022.


Summary

• Apache Spark and Dask are two popular Big Data processing frameworks
• There is existing support for parallel and distributed on HPC systems:
  – One bottleneck is the lack of support for low-latency and high-bandwidth interconnects
• This talk presented latest developments in the MPI4Dask (MPI-based Dask ecosystem) and MPI4Spark (MPI-based Spark ecosystem)
• Provided an overview of issues, challenges, and opportunities for designing efficient communication runtimes
  – Efficient, scalable, and hierarchical designs are crucial for Big Data/Data Science frameworks
  – Co-design of communication runtimes and BigData/Data Science frameworks will be essential
Thank You!

{shafi.16}@osu.edu

Network-Based Computing Laboratory
http://nowlab.cse.ohio-state.edu/

The MVAPICH2 Project
http://mvapich.cse.ohio-state.edu/

The High-Performance Deep Learning Project
http://hidl.cse.ohio-state.edu/

Follow us on

https://twitter.com/mvapich