HIGH-PERFORMANCE AND SCALABLE SUPPORT FOR BIG DATA STACKS WITH MPI

Aamir Shafi, Jinghan Yao, Kinan Al Attar, Dhabaleswar K. (DK) Panda
Network Based Computing Laboratory
The Ohio State University
http://nowlab.cse.ohio-state.edu/
Presentation Outline

• Introduction to Big Data Analytics and MVAPICH2

• 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**
Overview of the MVAPICH Project

- High Performance open-source Message Passing Interface (MPI) Library
- Support for multiple interconnects
  - InfiniBand, Omni-Path, Ethernet/iWARP, RDMA over Converged Ethernet (RoCE), AWS Elastic Fabric Adapter, Omni-Path Express, Broadcom RoCE, Intel Ethernet, Rockport Networks, Slingshot 10/11
- Support for multiple platforms
  - x86, OpenPOWER, ARM, Xeon-Phi, GPGPUs (NVIDIA and AMD)
- Started in 2001, first open-source version demonstrated at SC ‘02
- Supports the latest MPI-3.1 standard
- http://mvapich.cse.ohio-state.edu
- Additional optimized versions for different systems/environments:
  - MVAPICH2-X (Advanced MPI + Partitioned Global Address Space), since 2011
  - MVAPICH2-GDR with support for NVIDIA (since 2014) and AMD (since 2020) GPUs
  - MVAPICH2-MIC with support for Intel Xeon-Phi, since 2014
  - MVAPICH2-Virt with virtualization support, since 2015
  - MVAPICH2-EA with support for Energy-Awareness, since 2015
  - MVAPICH2-Azure for Azure HPC InfiniBand instances, since 2019
  - MVAPICH2-X-AWS for AWS HPC+Elastic Fabric Adapter instances, since 2019
- Tools:
  - OSU MPI Micro-Benchmarks (OMB), since 2003
  - OSU InfiniBand Network Analysis and Monitoring (INAM), since 2015
- Used by more than 3,300 organizations in 90 countries
- More than 1.66 Million downloads from the OSU site directly
- Empowering many TOP500 clusters (Nov ‘22 ranking)
  - 7th, 10,649,600-core (Sunway TaihuLight) at NSC, Wuxi, China
  - 19th, 448, 448 cores (Frontera) at Texas Advanced Computing Center
  - 34th, 288,288 cores (Lassen) at Lawrence Livermore National Lab
  - 46th, 570,020 cores (Nurion) in South Korea and many others
- Available with software stacks of many vendors and Linux Distros (RedHat, SuSE, OpenHPC, and Spack)
- Partner in the 19th ranked TACC Frontera system
- Empowering Top500 systems for more than 17 years
### Architecture of MVAPICH2 Software Family for HPC, DL/ML, and Data Science

#### High Performance Parallel Programming Models

- **Message Passing Interface (MPI)**
- **PGAS** *(UPC, OpenSHMEM, CAF, UPC++)*
- **Hybrid --- MPI + X** *(MPI + PGAS + OpenMP/Cilk)*

#### High Performance and Scalable Communication Runtime

**Diverse APIs and Mechanisms**

- **Point-to-point Primitives**
- **Collectives Algorithms**
- **Job Startup**
- **Energy-Awareness**
- **Remote Memory Access**
- **I/O and File Systems**
- **Fault Tolerance**
- **Virtualization**
- **Active Messages**
- **Introspection & Analysis**

**Support for Modern Networking Technology** *(InfiniBand, iWARP, RoCE, Omni-Path, EFA, Rockport, Slingshot)*

- **Transport Protocols**
  - RC
  - SRD
  - UD
  - DC
- **Modern Features**
  - UMR
  - ODP
  - SR-IOV
  - Multi Rail

**Support for Modern Multi-/Many-core Architectures** *(Intel-Xeon, AMD, OpenPOWER, ARM, GPU (NVIDIA, AMD), DPU)*

- **Transport Protocols**
- **Modern Features**
  - Shared Memory
  - CMA
  - IVSHMEM
  - XPMEM
  - Optane*
  - NVLink
  - CAPI*

* Upcoming
Presentation Outline

• Introduction to Big Data Analytics and MVAPICH2

• 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.
  - The key idea is to utilize MPI-based point-to-point communication inside Netty.
**MPI4Spark Interconnect Support**

- The current approach is different from its predecessor design, RDMA-Spark ([http://hibd.cse.ohio-state.edu](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_WORLD

Node View (4 nodes)

Node A
- Executor A
- Worker A

Node B
- Executor B
- Worker B

Node C
- Master
- Driver

Node D
- Master
- Driver
MPI4Spark-Basic Design

- Modified the Netty NIO selector loop, which polls for channel state changes based on connection, read, or write events
- Inside of the selector loop checks were implemented with MPI non-blocking probing method (MPI_Iprobe) for MPI_recv calls matching MPI_sends
- Netty Channels or simply Java sockets were still being used but only for connection establishment
- Too CPU-intensive, performed badly
## Types of Messages Communicated by Spark

<table>
<thead>
<tr>
<th>Message Type</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>StreamRequest</td>
<td>A request to stream data from the remote end</td>
</tr>
<tr>
<td>StreamResponse*</td>
<td>A response to a StreamRequest when the stream has been successfully opened</td>
</tr>
<tr>
<td>RpcRequest</td>
<td>A request to perform a generic Remote Procedure Call (RPC)</td>
</tr>
<tr>
<td>RpcResponse</td>
<td>A response to a RpcRequest for a successful RPC</td>
</tr>
<tr>
<td>ChunkFetchRequest</td>
<td>A request to fetch a sequence of a single chunk of a stream</td>
</tr>
<tr>
<td>ChunkFetchSuccess*</td>
<td>A response to ChunkFetchRequest when a chunk exists and has been successfully fetched</td>
</tr>
<tr>
<td>OneWayMessage</td>
<td>A RPC that does not expect a reply</td>
</tr>
</tbody>
</table>
The MPI4Spark-Optimized design avoids the pitfalls of the MPI4Spark-Basic design and is a lot simpler.

In this design, we only target shuffle messages, knowing that the shuffle phase was a performance bottleneck and can account for 80% of total execution time:

- non-blocking MPI probes are avoided
- the idea was now to trigger MPI_recv calls by parsing the headers of shuffle messages inside of ChannelHandlers that reside in ChannelPipelines in Netty.
MPI4Spark Release

- MPI4Spark 0.1 release adds support for high-performance MPI communication to Spark:
  - Can be downloaded from: [http://hibd.cse.ohio-state.edu](http://hibd.cse.ohio-state.edu)

- Features:
  - **(NEW)** Based on Apache Spark 3.3.0
  - **(NEW)** Compliant with user-level Apache Spark APIs and packages
  - **(NEW)** 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
  - **(NEW)** Built on top of the MVAPICH2-J Java bindings for MVAPICH2 family of MPI libraries
  - **(NEW)** Tested with
    - OSU HiBD-Benchmarks, GroupBy and SortBy
    - 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 and MVAPICH2

• 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
- TCP
- UCX
- MVAPICH2
- mpi4py

TCP
- UCX

Dask

Laptops/
Desktops

Dask

Dask

Dask

Dask
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: Point-to-point Communication Coroutines

- Implements communication coroutines for point-to-point MPI functions:
  - Using mpi4py (Cython wrappers) and MVAPICH2-GDR

- mpi4py provides two flavors of point-to-point communication functions:
  - `Send()/Recv()` – for exchanging data in buffers (faster and used in MPI4Dask)
  - `send()/recv()` – for communicating Python objects (pickle/unpickle)
  - GPU buffers implement the `__cuda_array_interface__`

- Implemented chunking mechanism for large messages

- The send and receive communication coroutines are as follows:

```python
request = comm.Isend([buf, size], dest, tag)
status = request.Test()

while status is False:
    await asyncio.sleep(0)
    status = request.Test()
```

```python
request = comm.Irecv([buf, size], src, tag)
status = request.Test()

while status is False:
    await asyncio.sleep(0)
    status = request.Test()
```
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 and Trends
• 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.
- 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.

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 and MVAPICH2
- 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**: $ddf1.merge(ddf2)$, using persist
  - Merge two GPU data frames, each with length of $32\times1e8$
  - 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**

- On average, MPI4Dask is:
  - 4.94x faster than UCX
  - 26.85x faster than TCP

**Aggregated Throughput**

- MPI4Dask 0.3, Dask 2022.8.1, Distributed, 2022.8.1, MVAPICH2-GDR 2.3.7, UCX v1.13.1, UCX-py 0.27.00
**NumPy Array Slicing Benchmark on TACC Frontera CPU System**

https://arxiv.org/abs/2101.08878

1.26x better on average

3.17x better on average

From 32 workers, we increase array size by 16 times

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

- Introduction to Big Data Analytics and MVAPICH2
- 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

Follow us on

https://twitter.com/mvapich

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/