MVAPICH
MPI, PGAS and Hybrid MPI+PGAS Library

*Follow us on*

https://twitter.com/mvapich

HiBD
High-Performance Big Data

# Accelerating Spark and Dask using MVAPICH2

## Talk at OSU Booth SC '22

by


**Aamir Shafi**

Network Based Computing Laboratory

The Ohio State University

Shafi.16@osu.edu

# Presentation Outline

- **The Landscape of Big Data Frameworks**
- Overview of the Apache Spark Project
- Overview of the Dask Project
- MPI4Spark: Using MVAPICH2 to Optimize Apache Spark
- MPI4Dask: Using MVAPICH2 to Optimize Dask
- Summary

# Overview of Big Data Framework

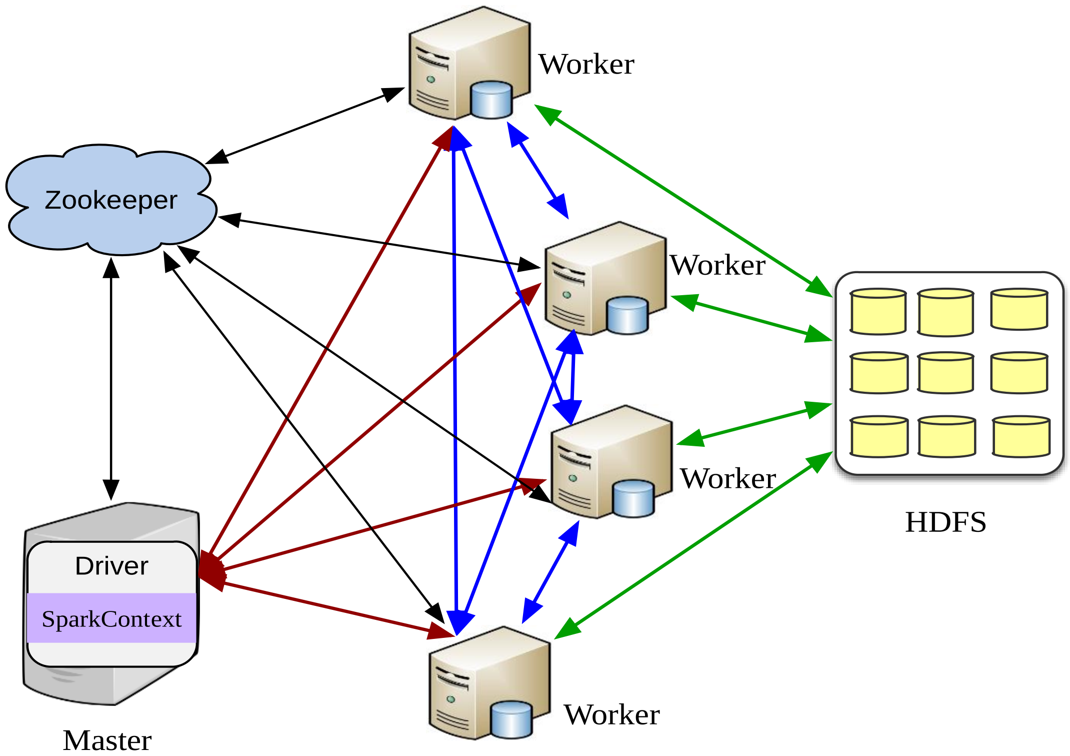
- Some of the popular Big Data processing frameworks include Apache Spark, Dask, Ray
- Apache Spark is an in-memory data processing framework that is written in Scala and Java:
  - Has support for Python using PySpark
- Dask is a task-based distributed computing framework that scales Python applications from laptops to high-end systems
- Motivation of this work:
  - The support for efficient execution on high-speed interconnects is lagging:
    - Vanilla Spark has no support (still relies on TCP/IP based sockets via Netty)
    - Dask provides two communication devices: TCP/IP and UCX
- The main goal of this work is to utilize the MVAPICH2 library for optimizing communication in Spark and Dask:
  - This allows exploiting supported high-speed interconnects – like InfiniBand, Omni Path, Slingshot, and others – in Big Data ecosystems

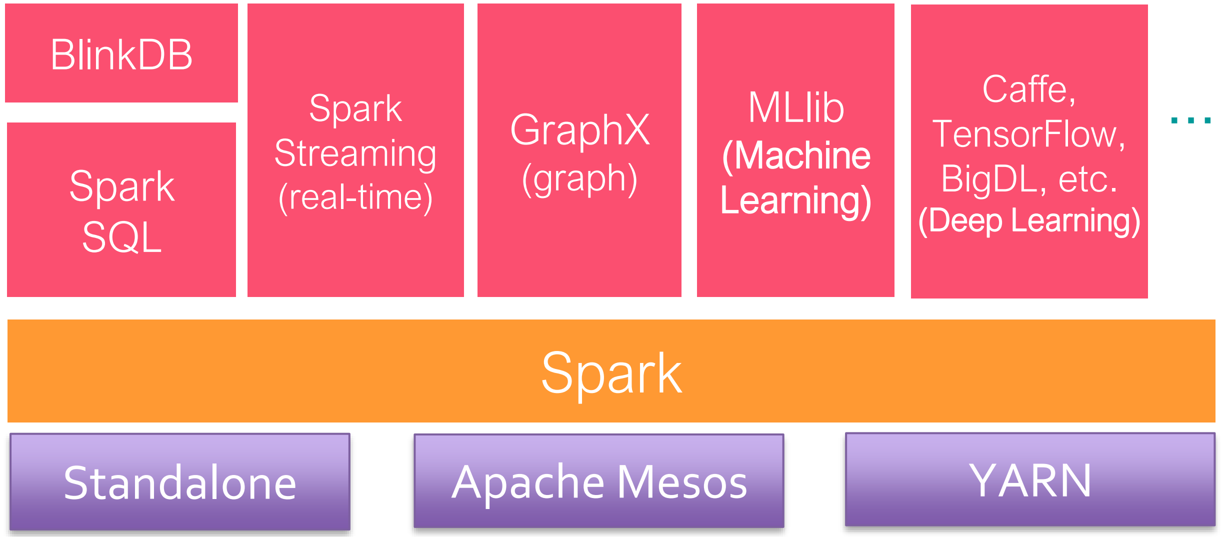
# Presentation Outline

- The Landscape of Big Data Frameworks
- **Overview of the Apache Spark Project**
- Overview of the Dask Project
- MPI4Spark: Using MVAPICH2 to Optimize Apache Spark
- MPI4Dask: Using MVAPICH2 to Optimize Dask
- Summary

# The Apache Spark Framework

- An in-memory data-processing framework
  - Iterative machine learning jobs
  - Interactive data analytics
  - Scala based Implementation
  - Standalone, YARN, Mesos
- A unified engine to support Batch, Streaming, SQL, Graph, ML/DL workloads
- Scalable and communication intensive
  - Wide dependencies between Resilient Distributed Datasets (RDDs)
  - MapReduce-like shuffle operations to repartition RDDs
  - Sockets based communication




http://spark.apache.org

# RDD Programming Model in Spark

- Key idea: *Resilient Distributed Datasets* (***RDDs***)
  - Immutable distributed collections of objects that can be cached in memory across cluster nodes
  - Created by transforming data in stable storage using data flow operators (map, filter, groupBy, …)
  - Manipulated through various parallel operators
  - Automatically rebuilt on failure
    - rebuilt if a partition is lost
- Interface
  - Clean language-integrated API in Scala (Python & Java)
  - Can be used *interactively* from Scala console

# RDD Operations

| Transformations (define a new RDD) | Actions (return a result to driver) |
| --- | --- |
| map | reduce |
| filter | collect |
| sample | count |
| union | first |
| groupByKey | Take |
| reduceByKey | countByKey |
| sortByKey | saveAsTextFile |
| join | saveAsSequenceFile |
| … | … |

**More Information:**

- https://spark.apache.org/docs/latest/programming-guide.html#transformations
- https://spark.apache.org/docs/latest/programming-guide.html#actions

# Presentation Outline

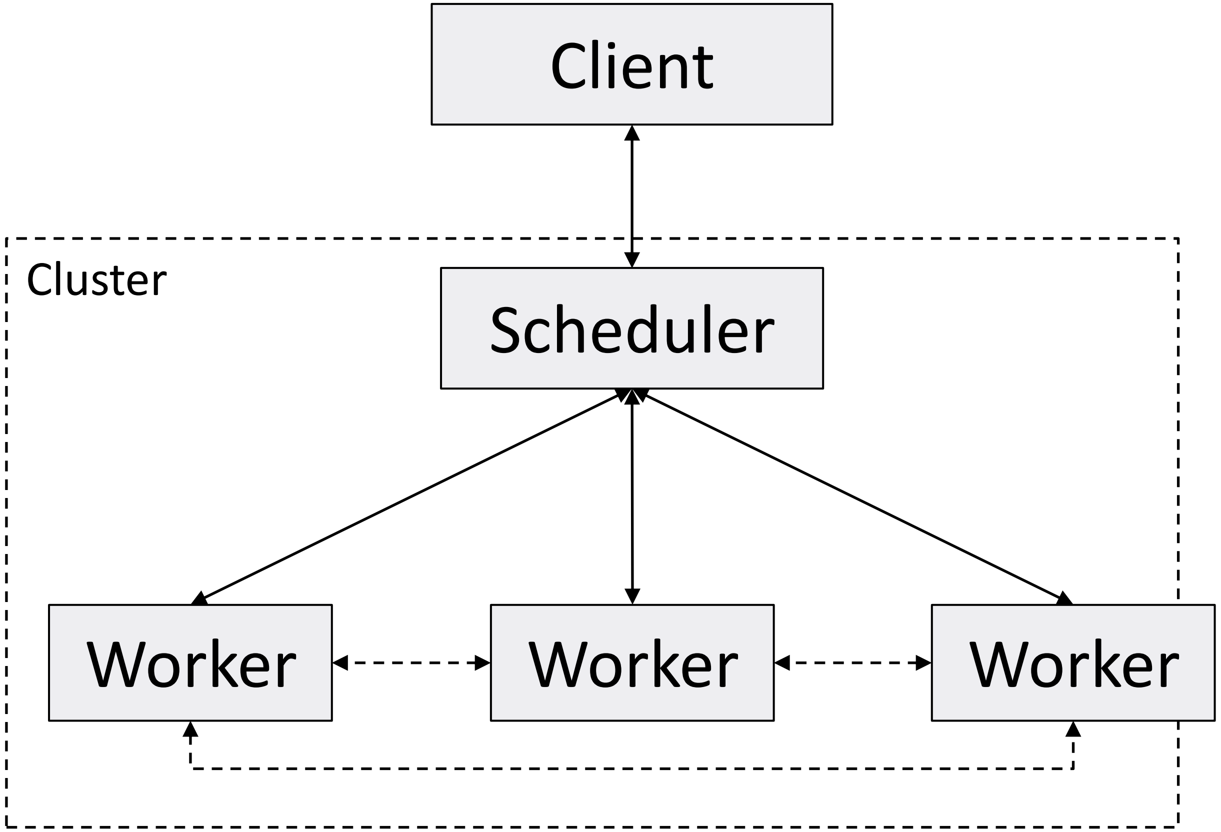
- The Landscape of Big Data Frameworks
- Overview of the Apache Spark Project
- **Overview of the Dask Project**
- MPI4Spark: Using MVAPICH2 to Optimize Apache Spark
- MPI4Dask: Using MVAPICH2 to Optimize Dask
- Summary

# Introduction to Dask

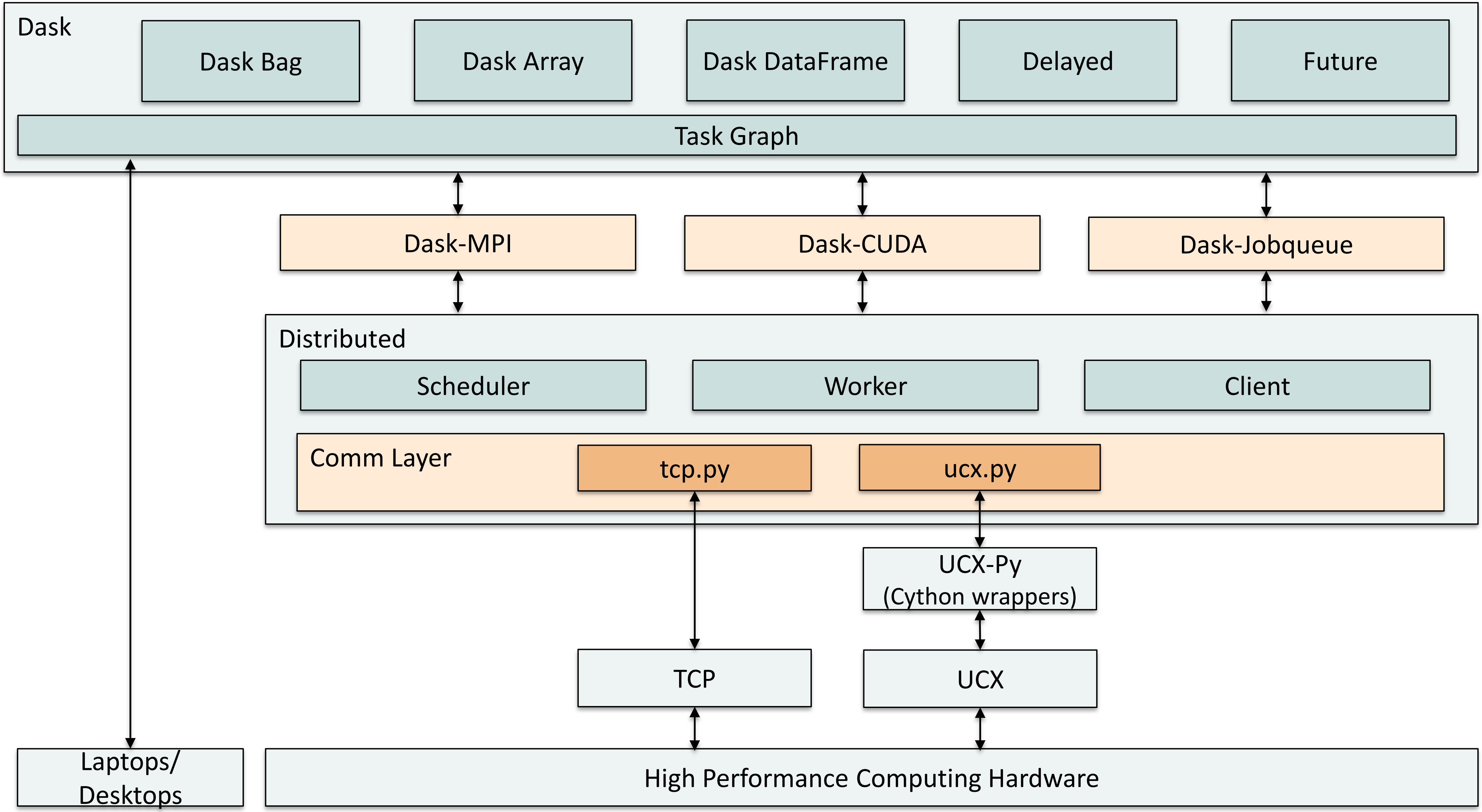
- Dask is a popular task-based distributed computing framework:
  - Scales Python applications from laptops to high-end systems
  - Builds a task-graph that is executed lazily on parallel hardware
  - Natively extends popular data processing libraries like numPy, Pandas
- Dask Distributed library supports parallel and distributed execution:
  - Built using the asyncio package that allows execution of asynchronous/non-blocking/concurrent operations called *coroutines:*
    - These are defined using async and invoked using await
  - Dask Distributed library originally has two communication backends:
    - TCP: Tornado-based
    - UCX: Built using a Cython wrapper called UCX-Py

# Dask Distributed Execution Model

- Key characteristics:
  1. Scalability
  2. Elasticity
  3. Support for coroutines
  4. Serialization/De-serialization to data to/from GPU memory



# Dask Architecture



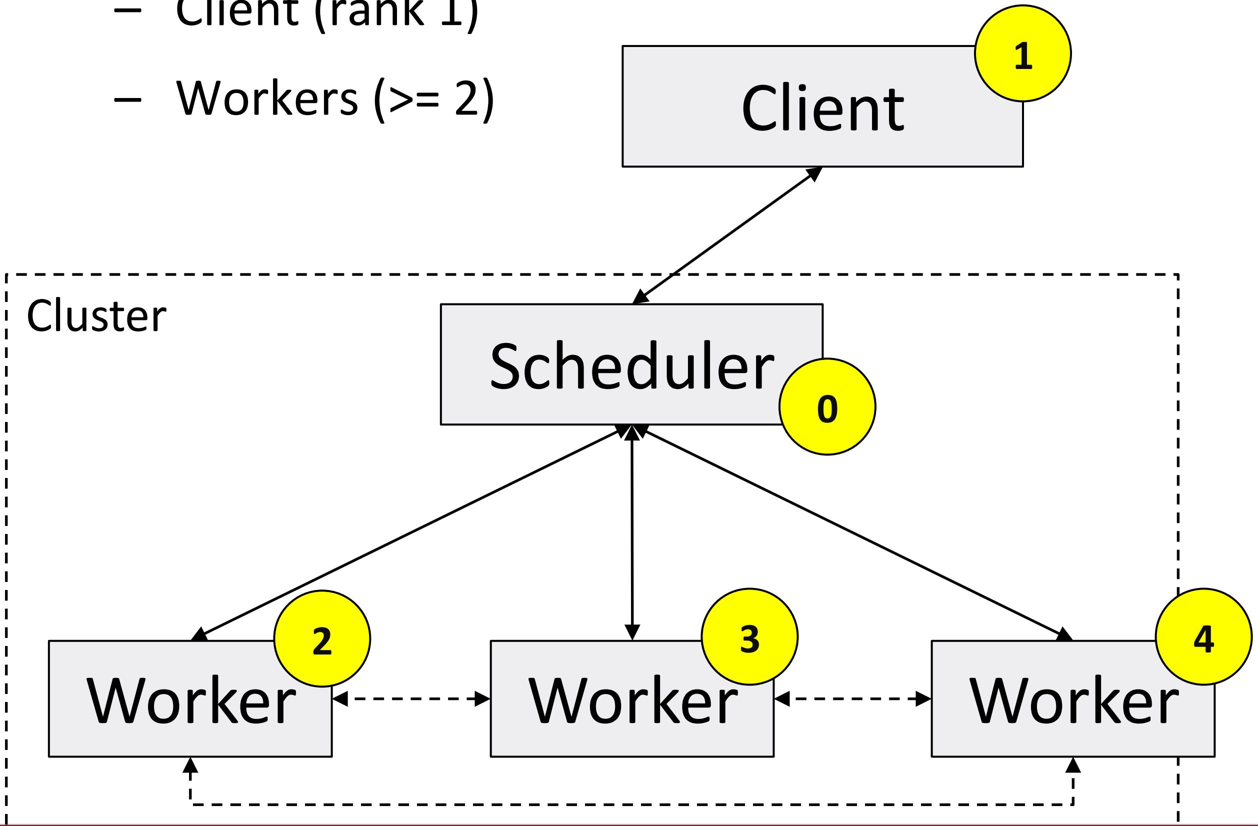
# Running Dask Programs

### Dask way: Using Cluster Objects

- Clusters are pre-defined utility classes to help with bootstrapping Dask on different environments:
  - Dask-CUDA provides LocalCUDACluster, DGX
  - Dask-Jobqueue provides PBSCluster, SLURMCluster, LSFCluster, KubeCluster, ECSCluster, YARNCluster
- Steps
  - Step 1: Start the Cluster
  - Step 2: Start client
  - Step 3: Submit work to workers through client

### MPI way: Using dask-mpi

- Using dask-mpi, start an MPI job where:
  - Scheduler (rank 0)
  - Client (rank 1)
  - Workers (>= 2)

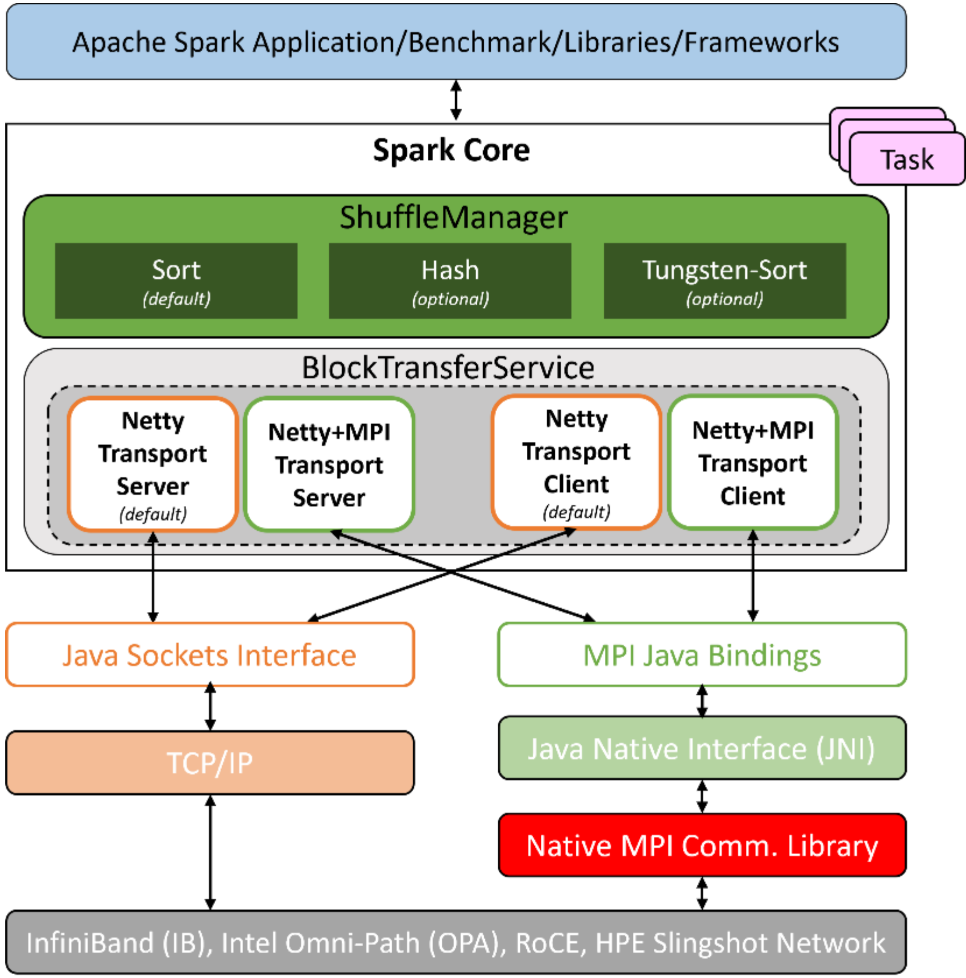


# Presentation Outline

- The Landscape of Big Data Frameworks
- Overview of the Apache Spark Project
- Overview of the Dask Project
- **MPI4Spark: Using MVAPICH2 to Optimize Apache Spark**
- MPI4Dask: Using MVAPICH2 to Optimize Dask
- Summary

# MPI4Spark: Using MVAPICH2 to Optimize Apache Spark

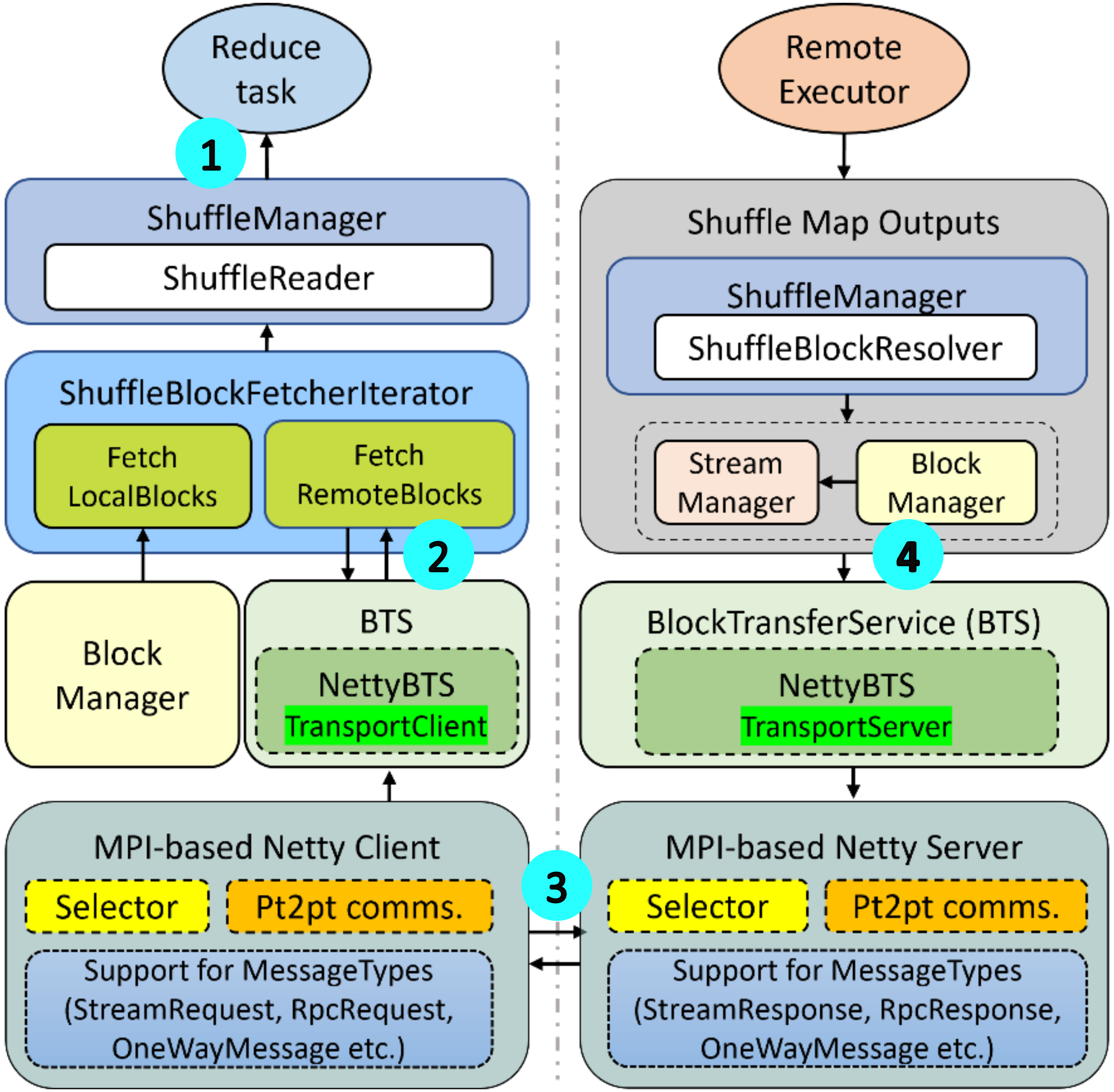
- The main motivation of this work is to utilize the communication functionality provided by MVAPICH2 in the Apache Spark framework
- MPI4Spark relies on Java bindings of the MVAPICH2 library
- Spark's default ShuffleManager relies on Netty for communication:
  - Netty is a Java New I/O (NIO) client/server framework for event-based networking applications
  - The key idea is to utilize MPI-based point-to-point communication inside Netty



# MPI4Spark: Optimizing the Communication (Shuffle) Phase

- Dataflow for two executors
  - One of the executors performs a reduce task that requires fetching of remote blocks

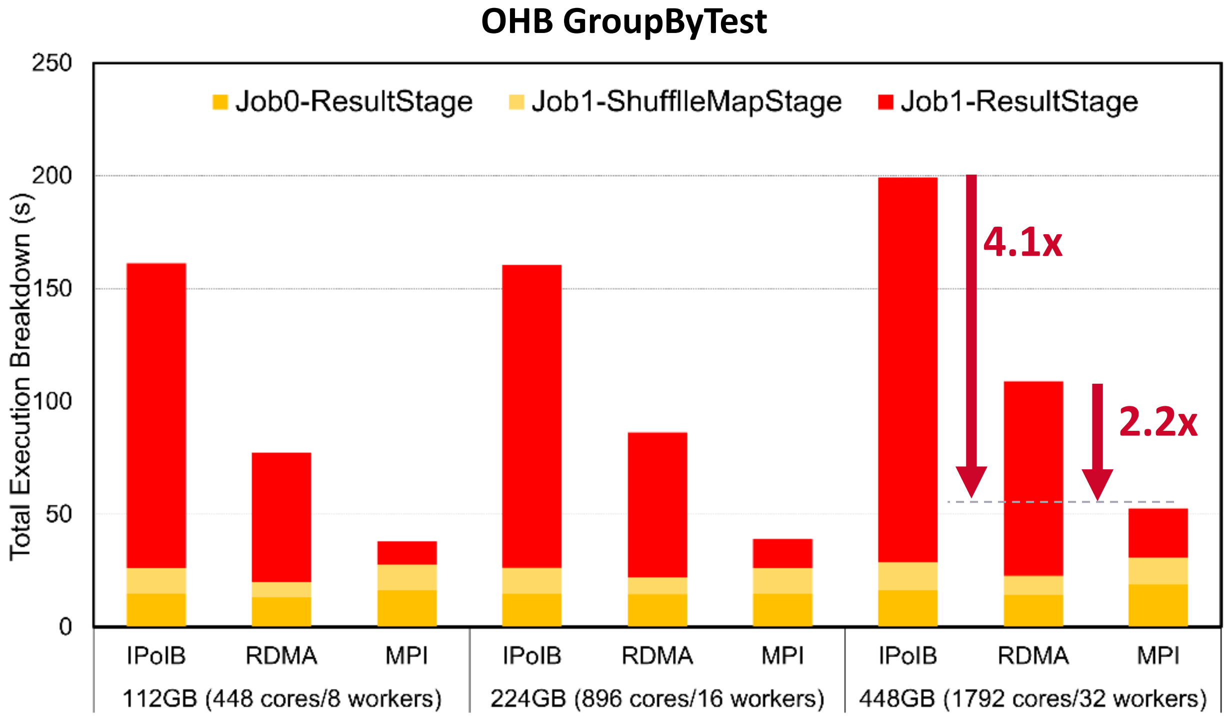
1. The reduce task starts with reading records inside of ShuffleReader
2. ShuffleBlockFetcherIterator is used to fetch data blocks locally or remotely
3. When remote fetches take place, the ShuffleBlockFetcherIterator will send requests to the underlying NettyBlockTransferService
4. MPI-based Netty will then be used to communicate the remote data block using the ShuffleBlockResolver

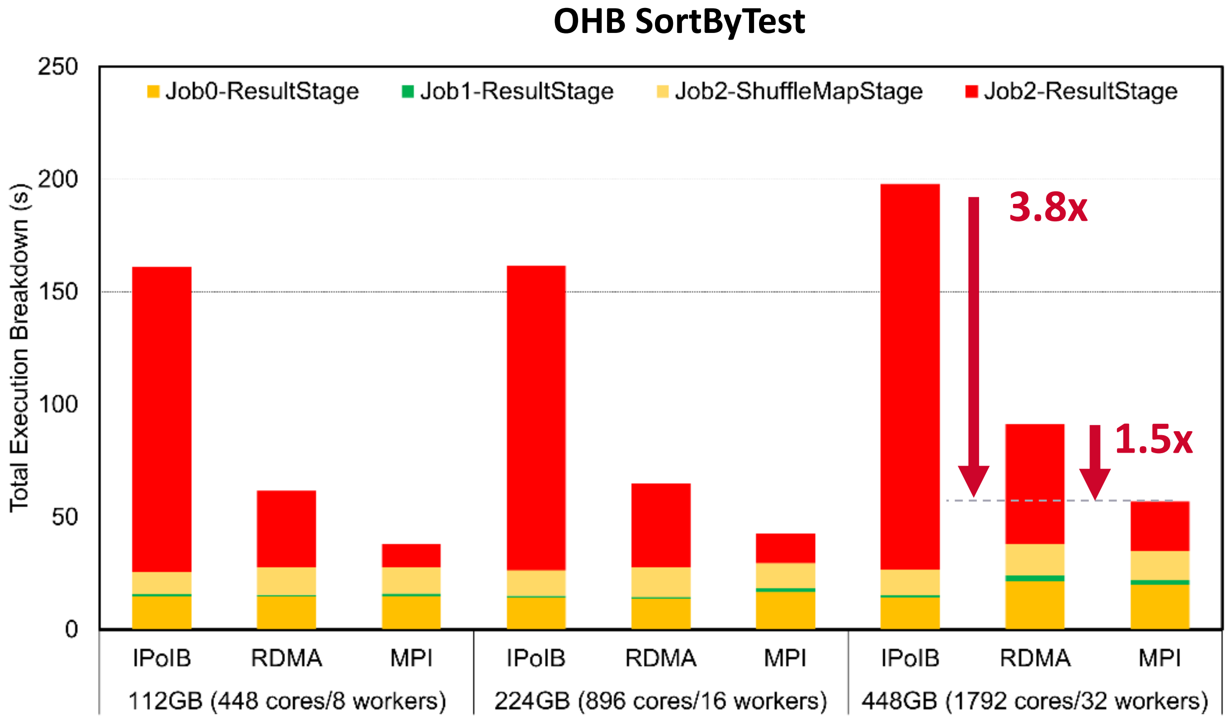


# MPI4Spark Release

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

# Weak Scaling Evaluation with OSU HiBD Benchmarks (OHB)



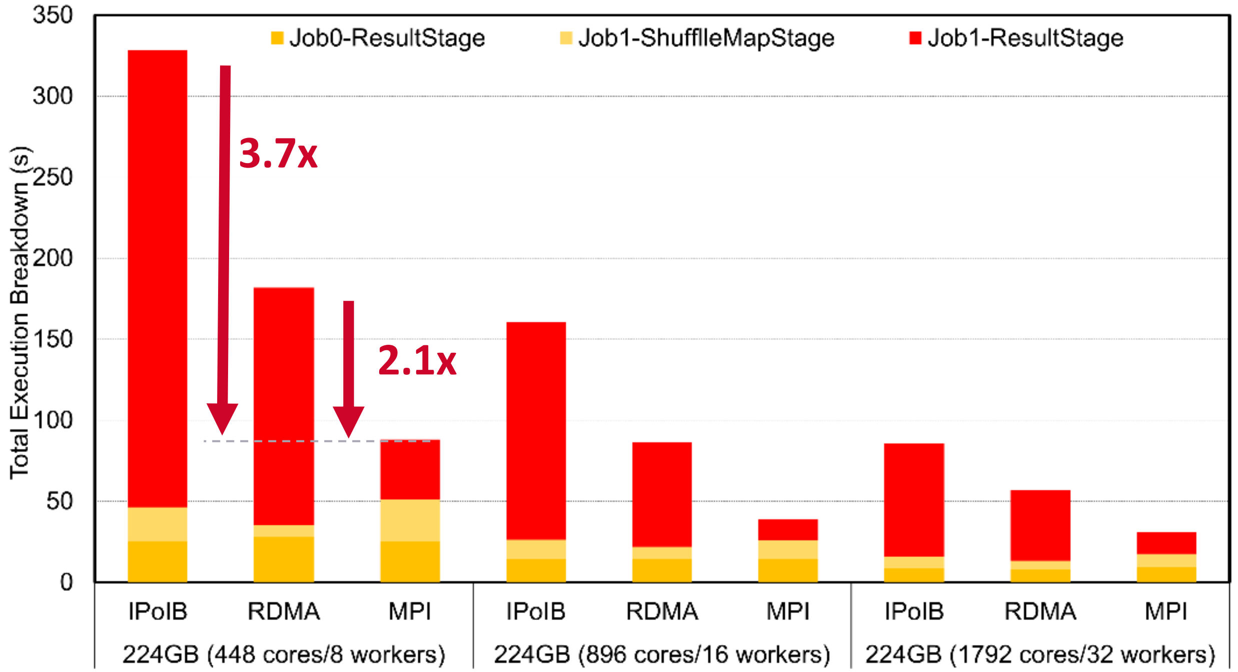




- The above are **weak-scaling** performance numbers of OHB benchmarks (GroupByTest and SortByTest) executed on the TACC Frontera system
- Speed-ups for the overall total execution time for 448GB with GroupByTest is **4.1x** and **2.2x** compared to IPoIB and RDMA, and for SortByTest the speed-ups are **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

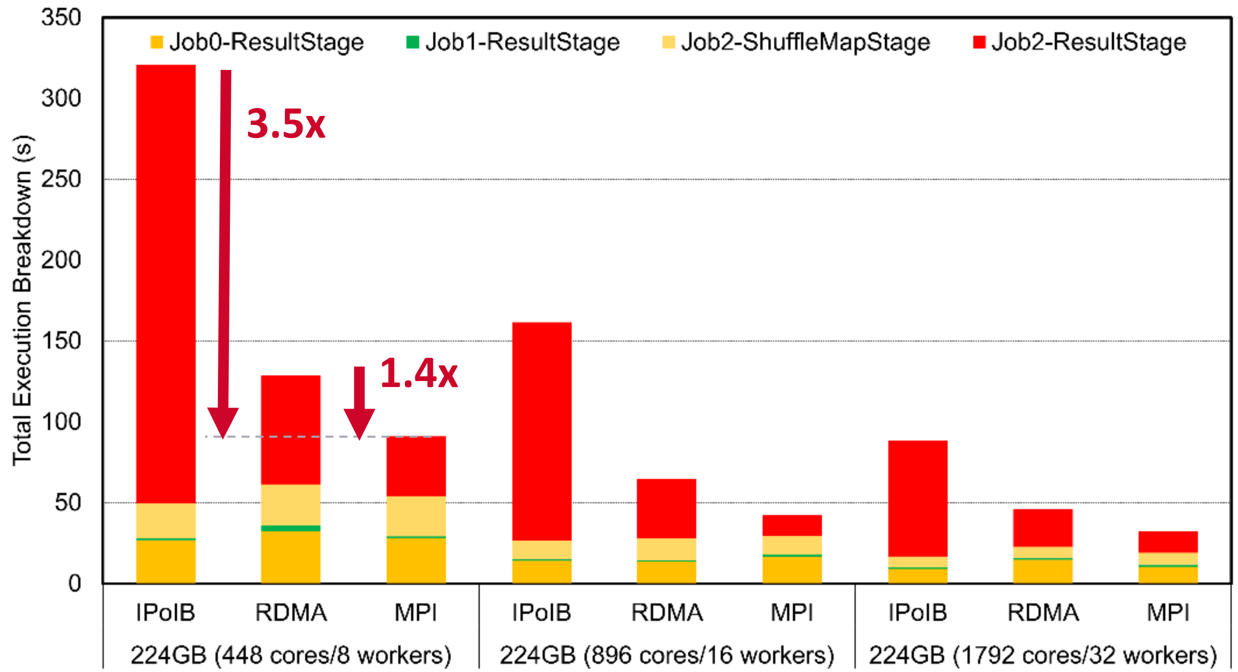
K. Al Attar, A. Shafi, M. Abduljabbar, H. Subramoni, D. Panda, Spark Meets MPI: Towards High-Performance Communication Framework for Spark using MPI, IEEE Cluster '22, Sep 2022.

# Strong Scaling Evaluation with OSU HiBD Benchmarks (OHB)

OHB GroupByTest




OHB SortByTest

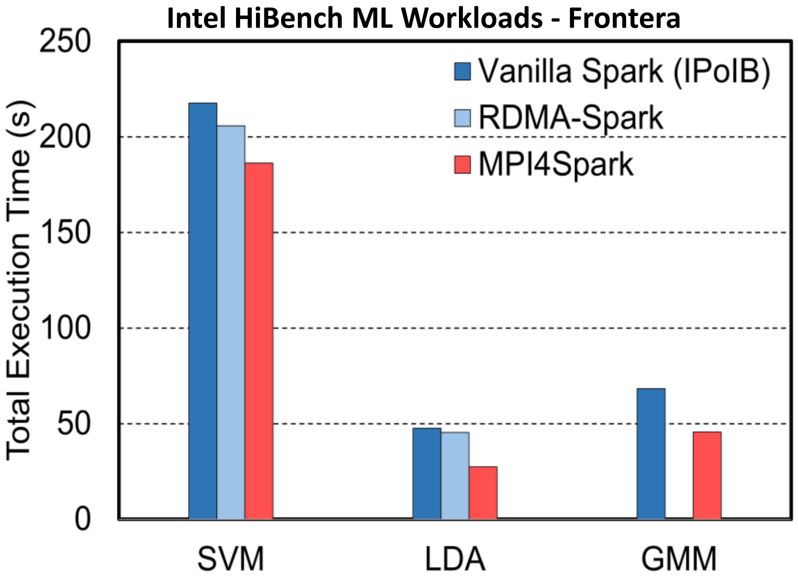



- The above are **strong-scaling** performance numbers of OHB benchmarks (GroupByTest and SortByTest) also executed on the TACC Frontera System
- Speed-ups for the overall total execution time for 8 workers with GroupByTest is **3.7x** and **2.1x** compared to IPoIB and RDMA, and for SortByTest the speed-ups are **3.5x** and **1.4x**, respectively
- Speed-ups for the shuffle read stage for 8 workers GroupByTest between MPI4Spark and IPoIB is **7.6x** and between MPI4Spark and RDMA is **4x**, while for SortByTest the speed-ups are **7.3x** and **1.8x**, respectively

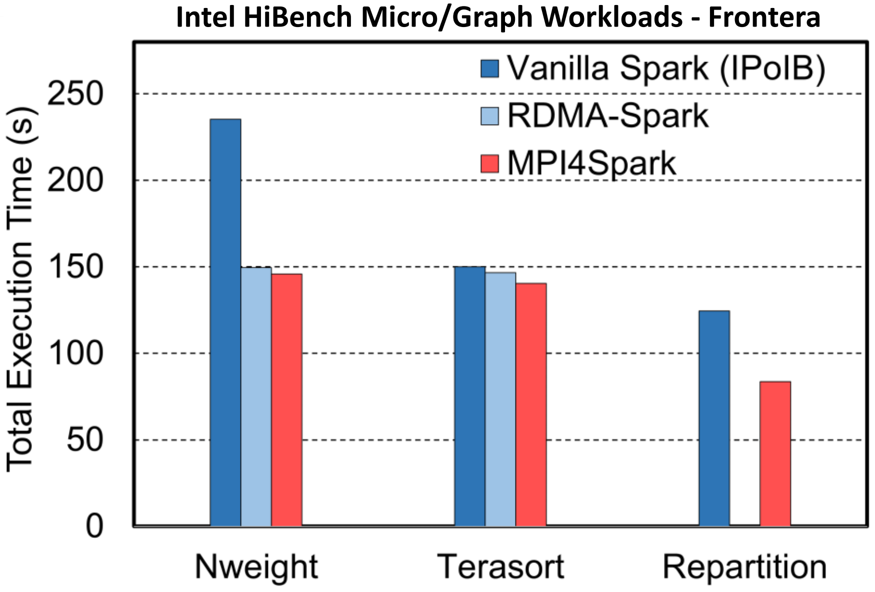
K. Al Attar, A. Shafi, M. Abduljabbar, H. Subramoni, D. Panda, Spark Meets MPI: Towards High-Performance Communication Framework for Spark using MPI, IEEE Cluster '22, Sep 2022.

# Performance Evaluation with Intel HiBench Workloads

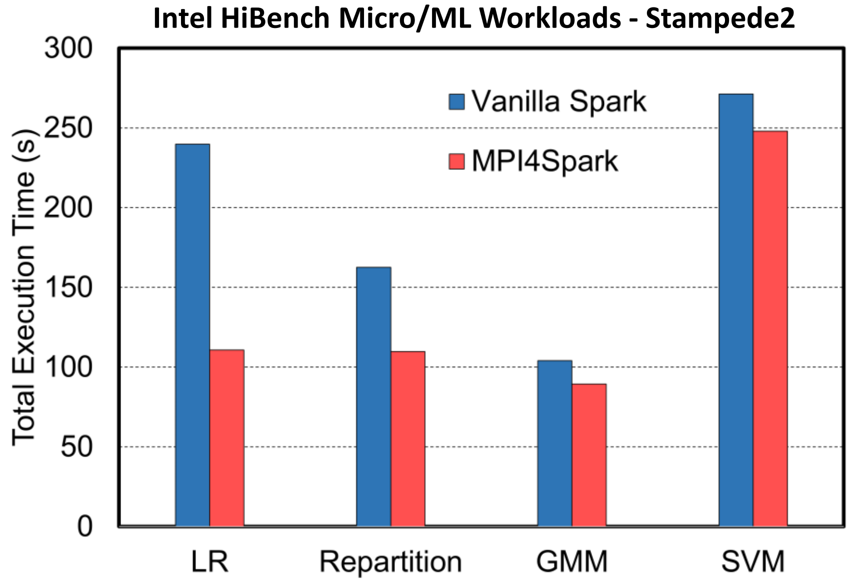
1.4x on average than RDMA-Spark




1.4x on average than Vanilla Spark




1.5x on average than Vanilla Spark

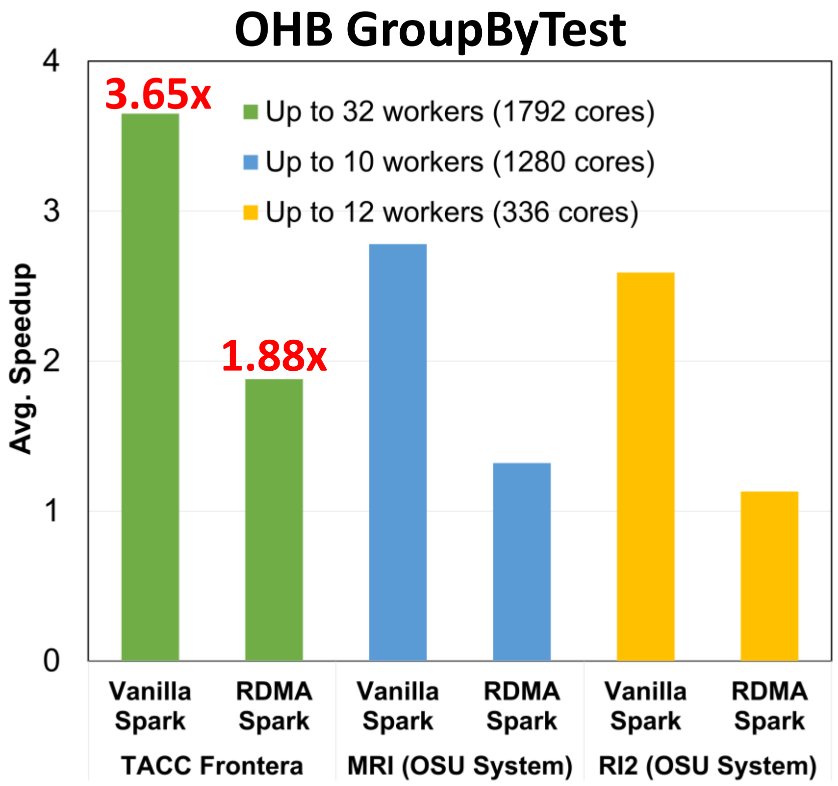


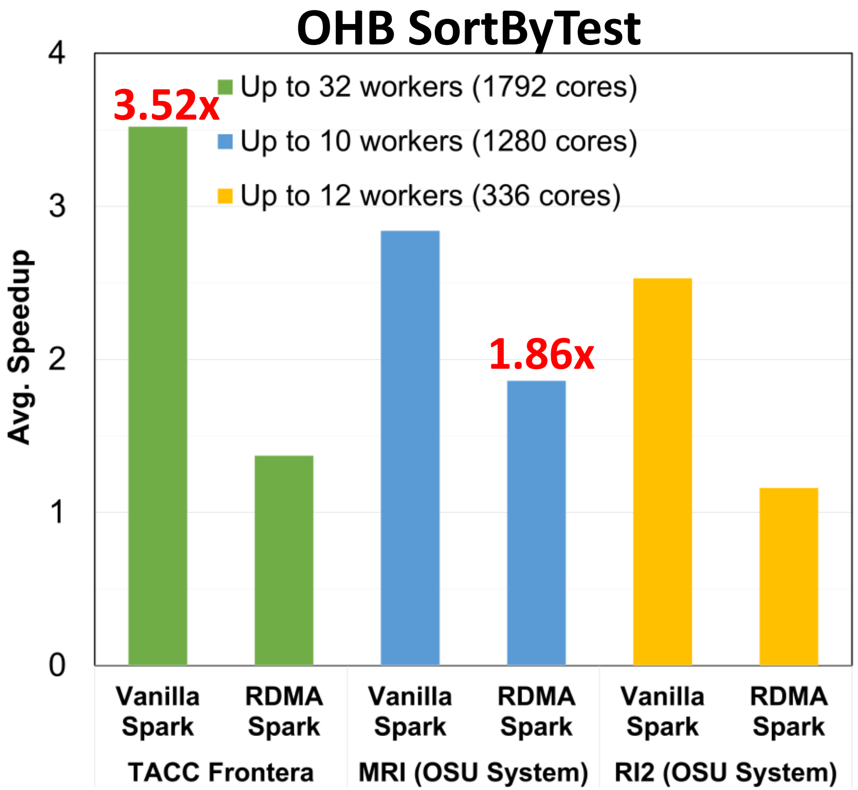

- This evaluation was done on the TACC Frontera (IB) and the TACC Stampede2 (OPA) Systems
- This illustrates the portability of MPI4Spark on different interconnects
- We see a speed-up for the LR machine learning workload on Stampede2 of about **2.2x**
- Speed-ups for the LDA machine learning workload on Frontera are **1.7x** and **1.66x** compared with IPoIB and RDMA, respectively

K. Al Attar, A. Shafi, M. Abduljabbar, H. Subramoni, D. Panda, Spark Meets MPI: Towards High-Performance Communication Framework for Spark using MPI, IEEE Cluster '22, Sep 2022.

# MPI4Spark: Relative Speedups to Vanilla Spark and RDMA-Spark on Three HPC Systems

| System Name | Nodes Used | Processor | Cores Used | Sockets | Cores/socket | RAM | Interconnect |
|---|---|---|---|---|---|---|---|
| TACC Frontera | 34 | Xeon Platinum | 1792 | 2 | 28 | 192 GB | HDR (100G) |
| RI2 (OSU System) | 14 | Xeon Broadwell | 336 | 2 | 14 | 128 GB | EDR (100G) |
| MRI (OSU System) | 12 | AMD EPYC 7713 | 1280 | 2 | 64 | 264 GB | 200 Gb/sec (4X HDR) |



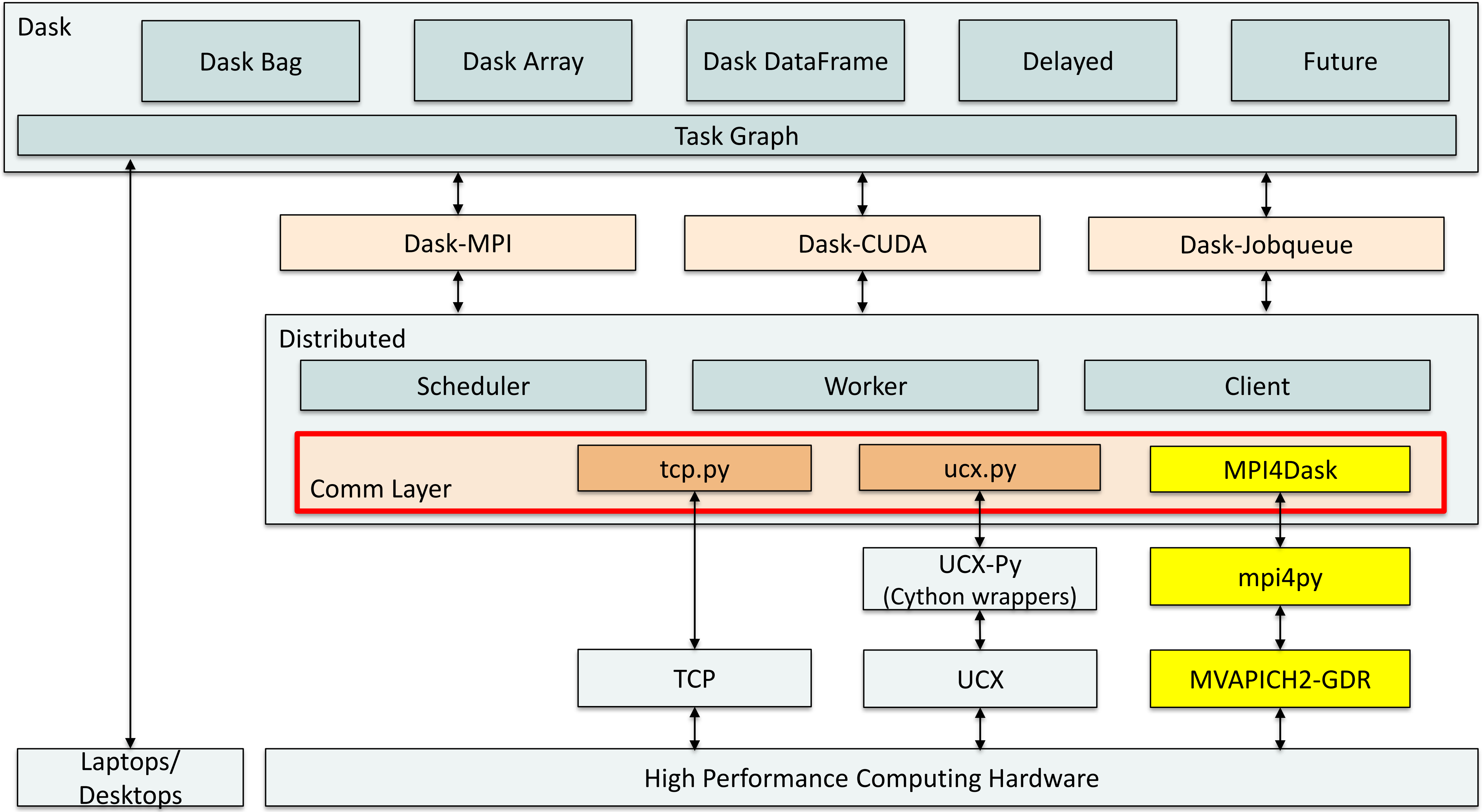
# Presentation Outline

- The Landscape of Big Data Frameworks
- Overview of the Apache Spark Project
- Overview of the Dask Project
- MPI4Spark: Using MVAPICH2 to Optimize Apache Spark
- **MPI4Dask: Using MVAPICH2 to Optimize Dask**
- Summary

# MPI4Dask: MPI backend for Dask

- 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-GDR
  - Uses Dask-MPI to bootstrap execution of Dask programs
  - Implements communication coroutines for point-to-point MPI functions
  - Provides mapping of process endpoints to MPI ranks

# MPI4Dask in the Dask Architecture

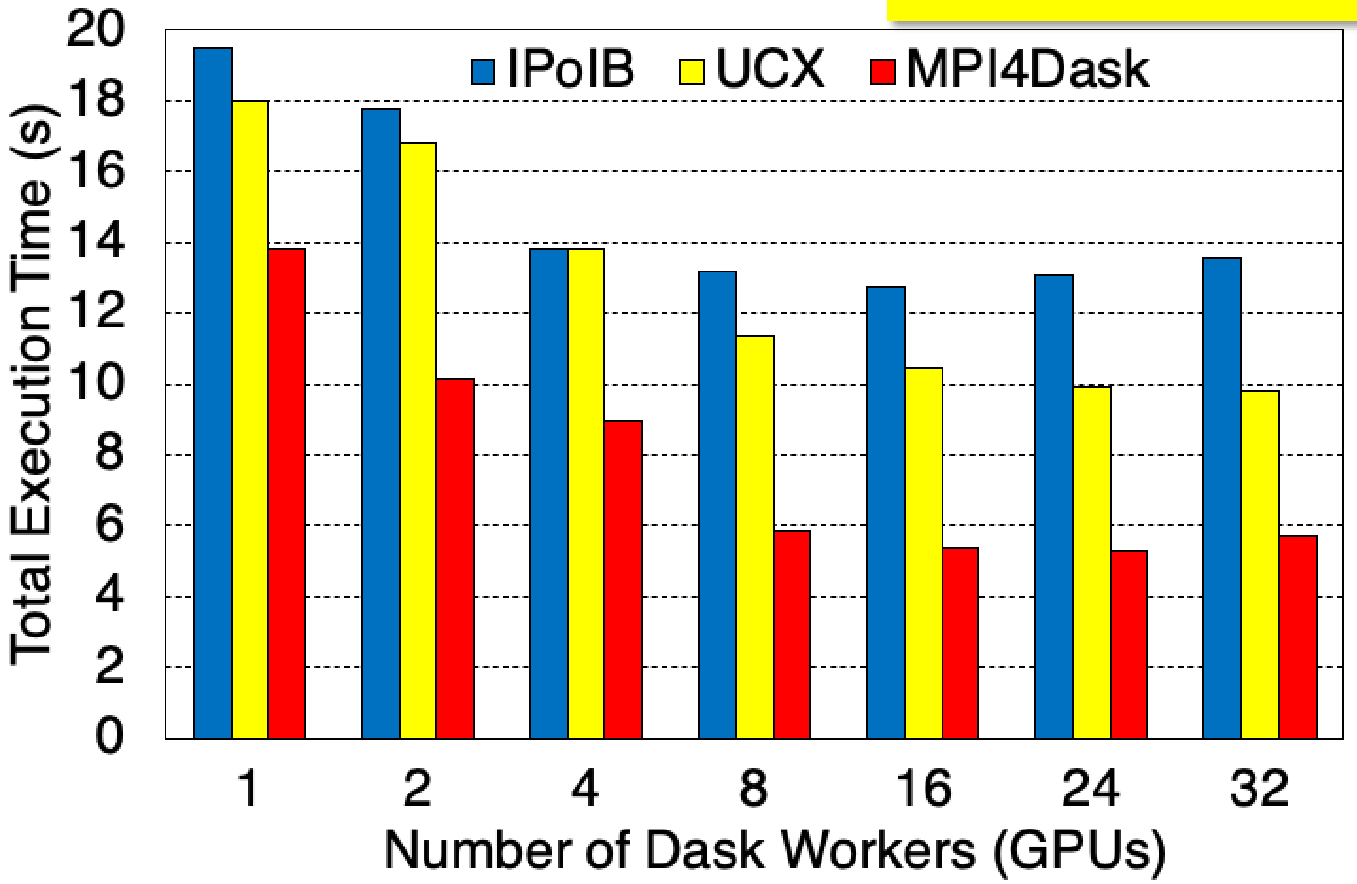


# MPI4Dask Release

- MPI4Dask 0.2 release adds support for high-performance MPI communication to Dask:
  - Can be downloaded from: http://hibd.cse.ohio-state.edu
- Features:
  - Based on Dask Distributed 2021.01.0
  - 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
  - (NEW) Built on top of mpi4py over the MVAPICH2, MVAPICH2-X, and MVAPICH2-GDR libraries
  - (NEW) Support for MPI-based communication for CPU-based Dask applications
  - Supports starting execution of Dask programs using Dask-MPI
  - Tested with
    - (NEW) CPU-based Dask applications using numPy and Pandas data frames
    - (NEW) GPU-based Dask applications using cuPy and cuDF
    - Mellanox InfiniBand adapters (FDR and EDR)
    - Various multi-core platforms
    - NVIDIA V100 and Quadro RTX 5000 GPUs
- **MPI4Dask 0.3 release (upcoming)**

# Benchmark #1: Sum of cuPy Array and its Transpose (TACC Frontera GPU Subsystem)

**1.71x better on average**



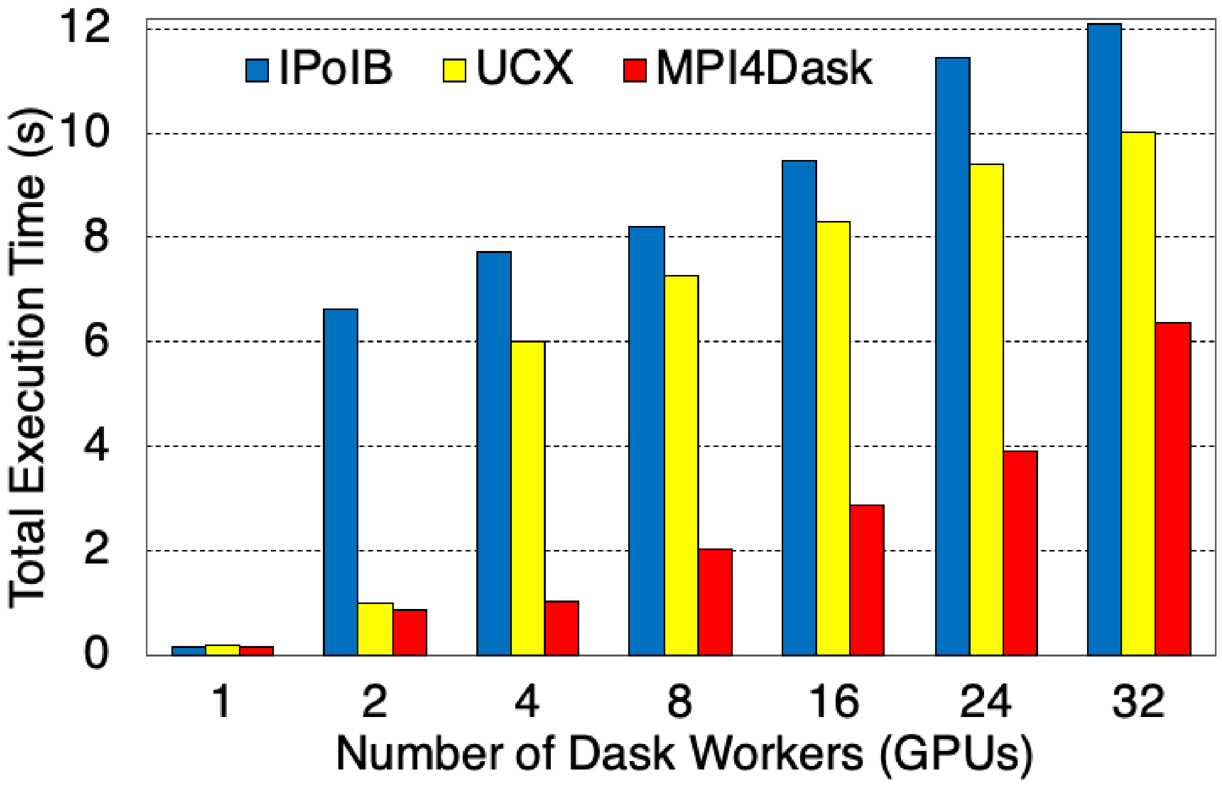

A. Shafi , J. Hashmi , H. Subramoni , and D. K. Panda, Efficient MPI-based Communication for GPU-Accelerated Dask Applications, CCGrid '21
https://arxiv.org/abs/2101.08878

**MPI4Dask 0.2 release**

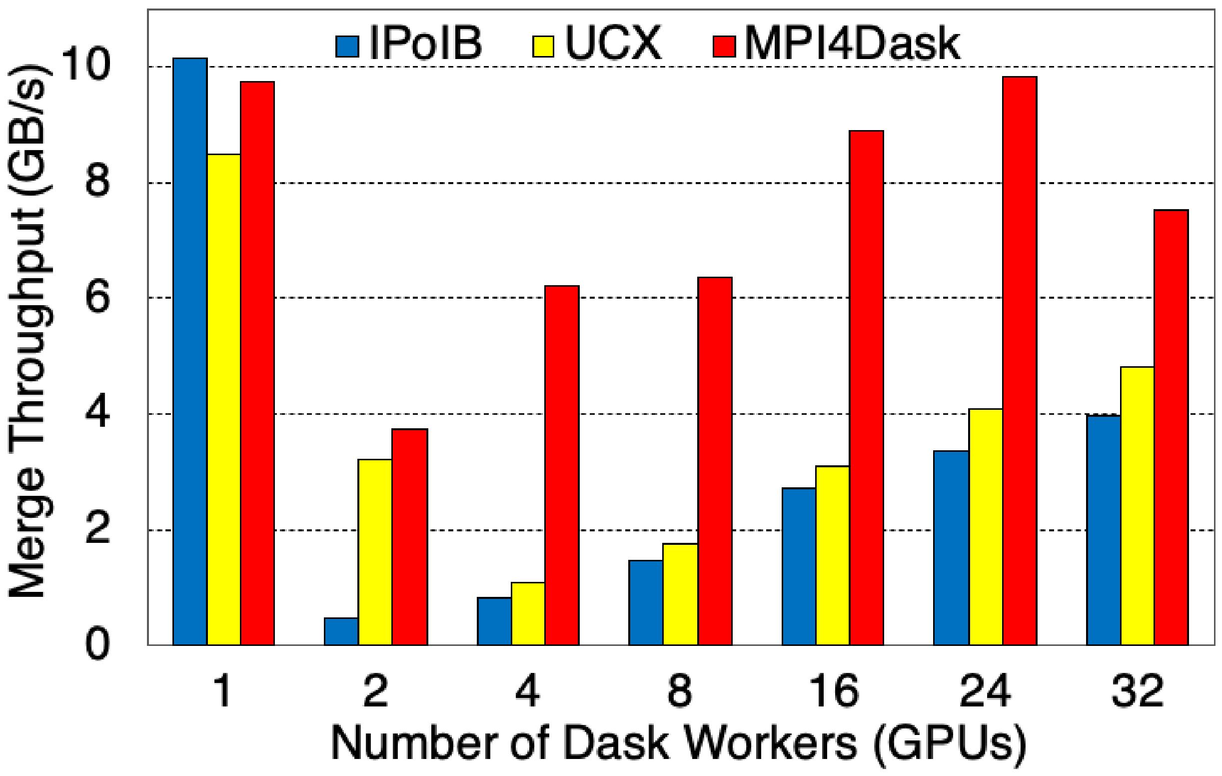
**(http://hibd.cse.ohio-state.edu)**

# Benchmark #2: cuDF Merge (TACC Frontera GPU Subsystem)

**2.91x better on average**




**2.90x better on average**




**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 0.2 release**

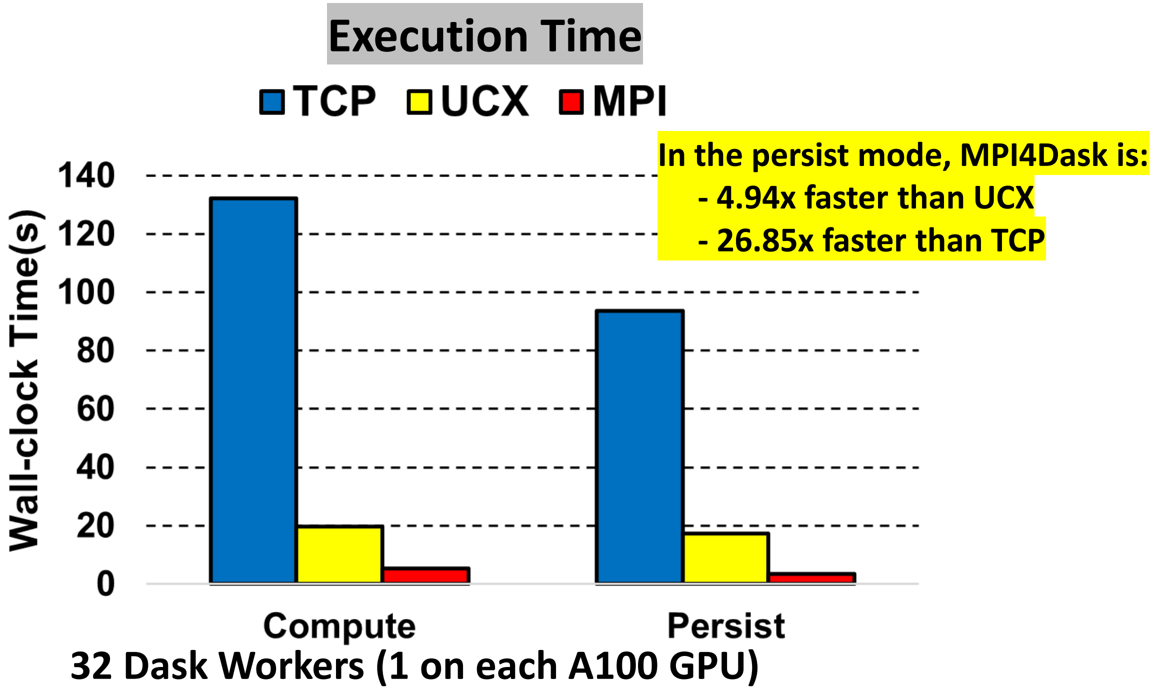
**(http://hibd.cse.ohio-state.edu)**

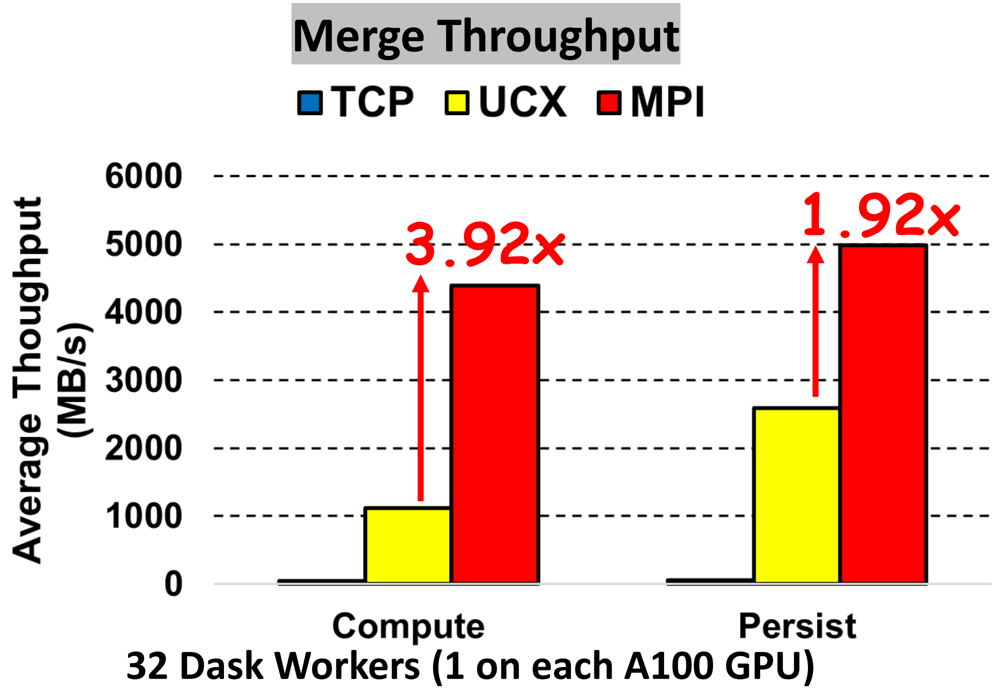
# Benchmark #2: cuDF Merge Operation (Wilkes-3 System)

- GPU-based Operation: $ddf1.merge(ddf2)$, 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**



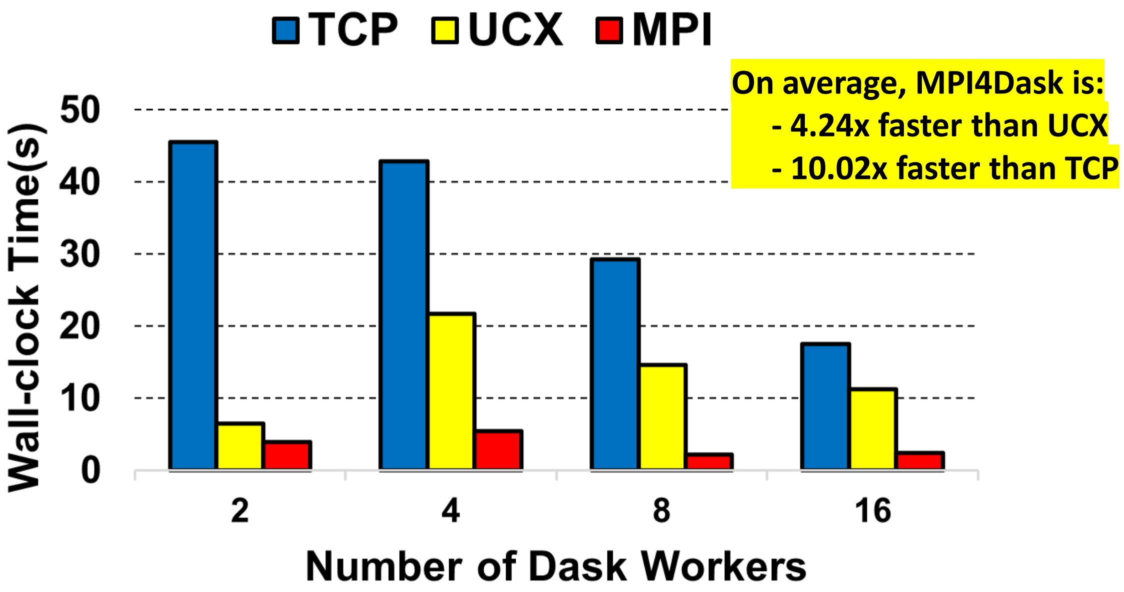

MPI4Dask 0.3* (soon to be released), Dask 2022.8.1, Distributed, 2022.8.1, MVAPICH2-GDR 2.3.7, UCX v1.13.1, UCX-py 0.27.00

# Benchmark #3: Matrix Dot Operation (Wilkes-3 System)

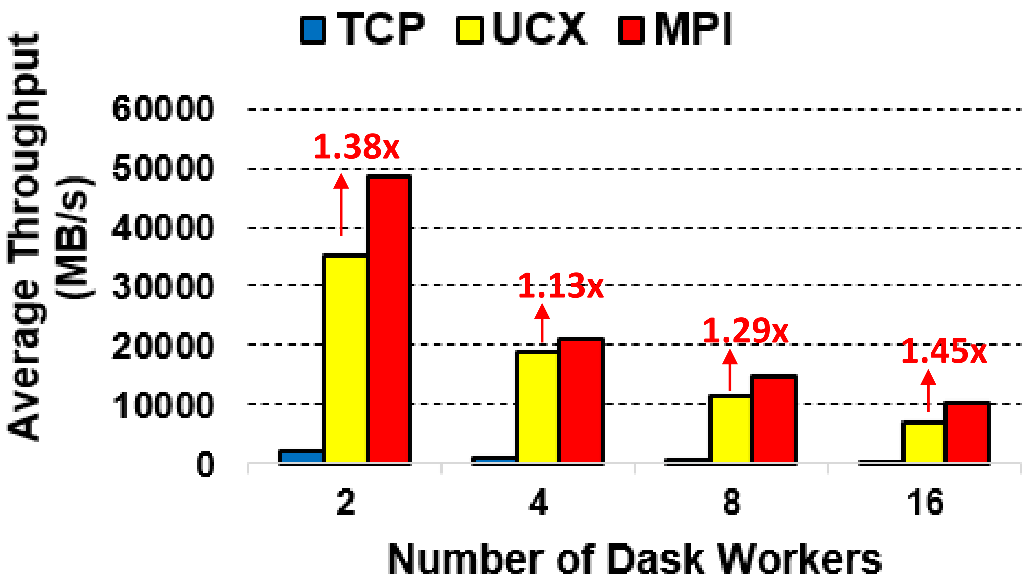
- GPU-based Operation: $a.dot(b)$, using persist()
  - Dot multiply two matrices, each with size of 4GB
  - Compute() will gather the data from all worker nodes to the client node, and make a copy on the host memory.
  - Persist() will leave the data on its current nodes without any gathering

**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**




**Multiplication Throughput**




MPI4Dask 0.3* (soon to be released), Dask 2022.8.1, Distributed, 2022.8.1, MVAPICH2-GDR 2.3.7, UCX v1.13.1, UCX-py 0.27.00

# Presentation Outline

- The Landscape of Big Data Frameworks
- Overview of the Apache Spark Project
- Overview of the Dask Project
- MPI4Spark: Using MVAPICH2 to Optimize Apache Spark
- MPI4Dask: Using MVAPICH2 to Optimize Dask
- **Summary**

# Related Publications

- Spark Meets MPI: Towards High-Performance Communication Framework for Spark using MPI K. Al Attar, A. Shafi, M. Abduljabbar, H. Subramoni, D. Panda IEEE Cluster '22, Sep 2022.

- Efficient MPI-based Communication for GPU-Accelerated Dask Applications A. Shafi, J. Hashmi, H. Subramoni, D. Panda The 21st IEEE/ACM International Symposium on Cluster, Cloud and Internet Computing, May 2021. https://arxiv.org/abs/2101.08878

- Blink: Towards Efficient RDMA-based Communication Coroutines for Parallel Python Applications A. Shafi, J. Hashmi, H. Subramoni, D. Panda 27th IEEE International Conference on High Performance Computing, Data, and Analytics, Dec 2020.

# Summary

- This talk presented MPI4Spark and MPI4Dask
  - These are optimized versions of Spark and Dask, respectively, that exploit high-performance communication provided by the MVAPICH2 library
- Both software stacks can execute on all MVAPICH2 support low-latency and high-bandwidth interconnects including InfiniBand, Omni Path, Slingshot, etc.
- Performance evaluation of MPI4Spark and MPI4Dask showed that these designs outperform the state-of-the-art communication devices in Spark and Dask framework
- MPI4Spark and MPI4Dask are available for download from the HiBD project website:
  - http://hibd.cse.ohio-state.edu

# Thank You!

shafi.16@cse.ohio-state.edu

Network-Based Computing Laboratory

http://nowlab.cse.ohio-state.edu/

The High-Performance MPI/PGAS Project

http://mvapich.cse.ohio-state.edu/

The High-Performance Big Data Project

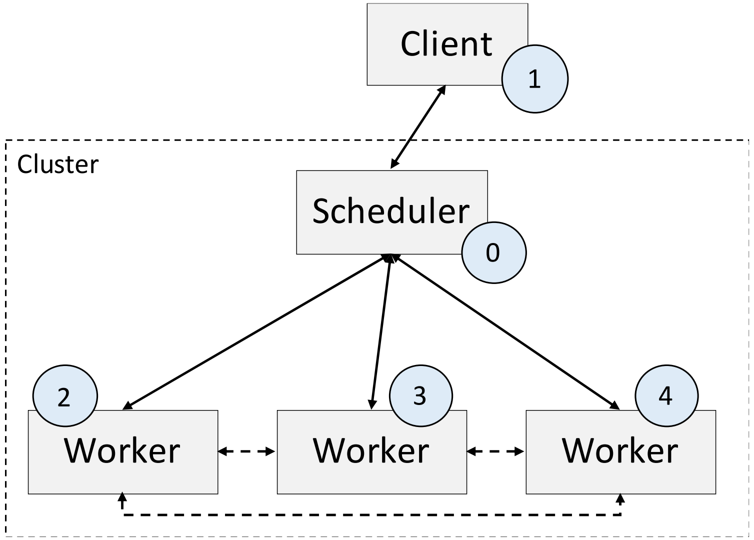
http://hibd.cse.ohio-state.edu/

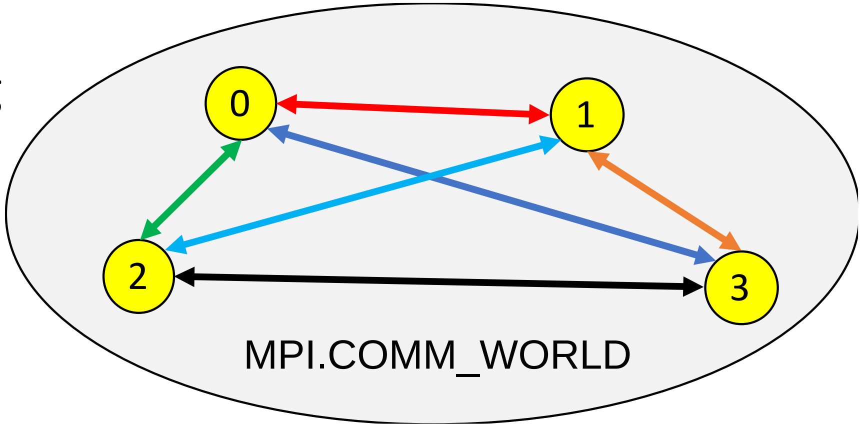
The High-Performance Deep Learning Project

http://hidl.cse.ohio-state.edu/

# 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()



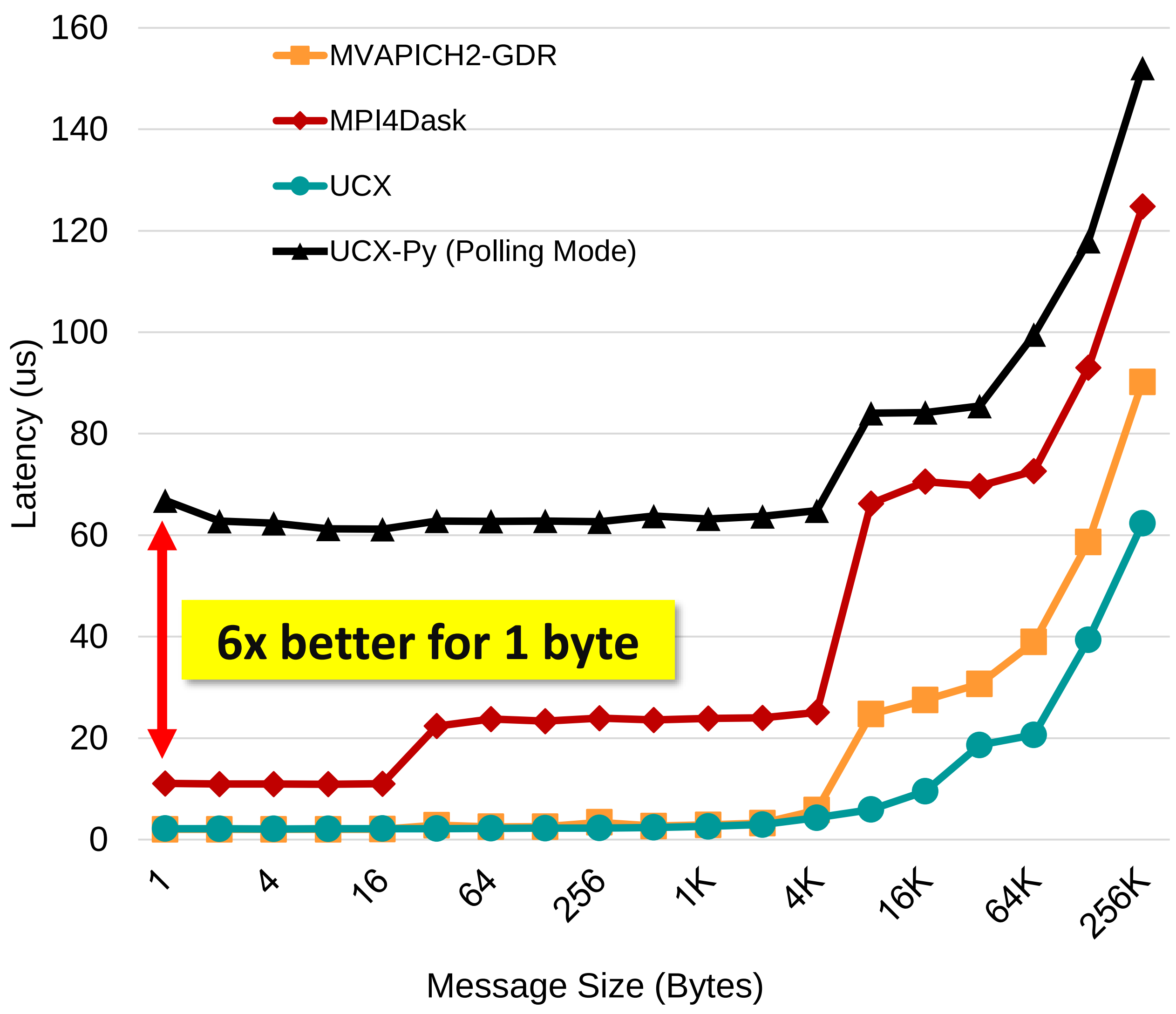
# MPI4Dask: Point-to-point Communication Coroutines

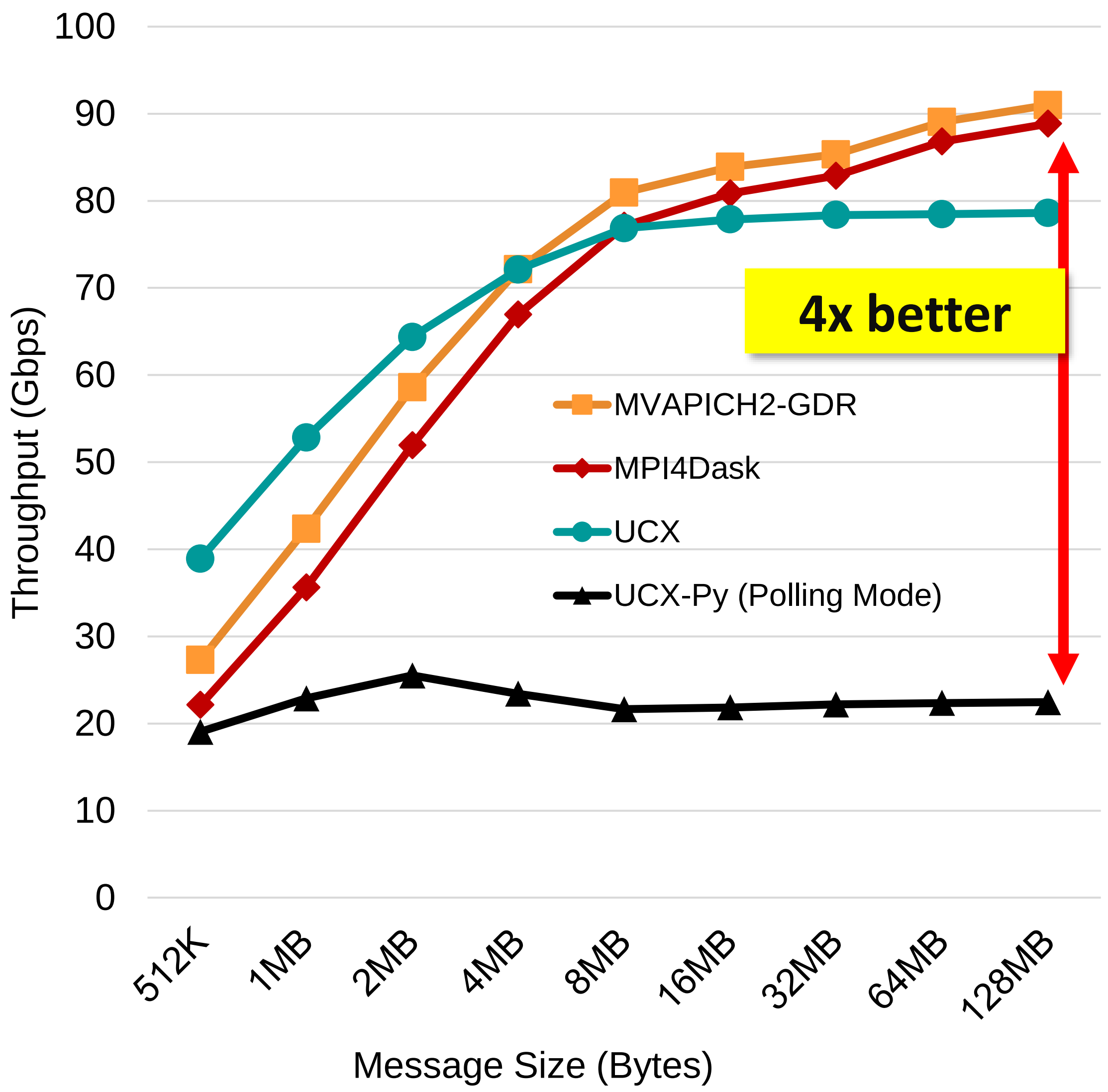
- Implements communication coroutines for point-to-point MPI functions:
  - Using mpi4py (Cython wrappers) and MVAPICH2-GDR
- mpi4py provides two flavors of point-to-point communication functions:
  - Send()/Recv() – for exchanging data in buffers (faster and used in MPI4Dask)
  - send()/recv() – for 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:

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

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

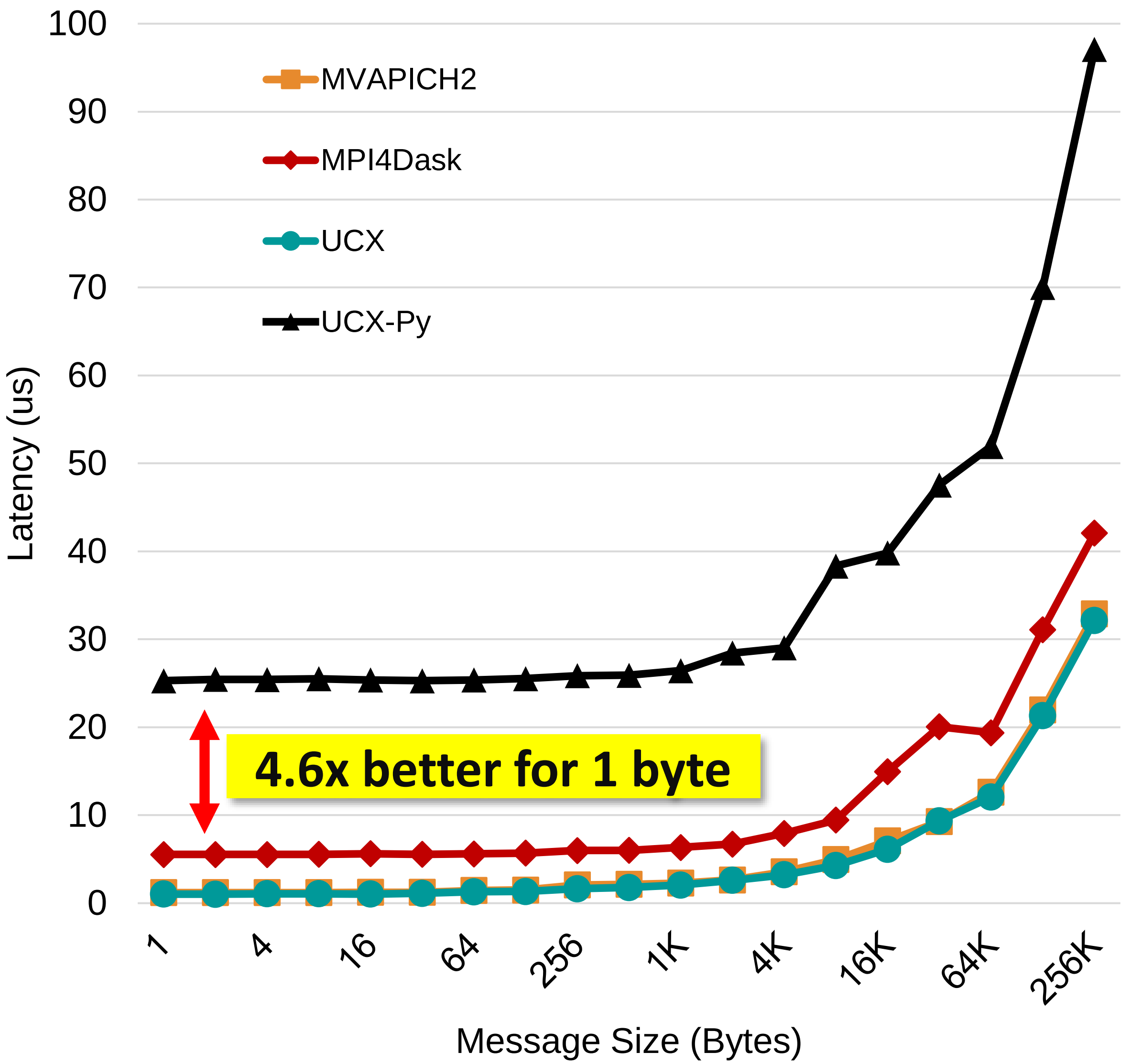
```
request = comm.Irecv([buf, size], src, tag)
status = request.Test()

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

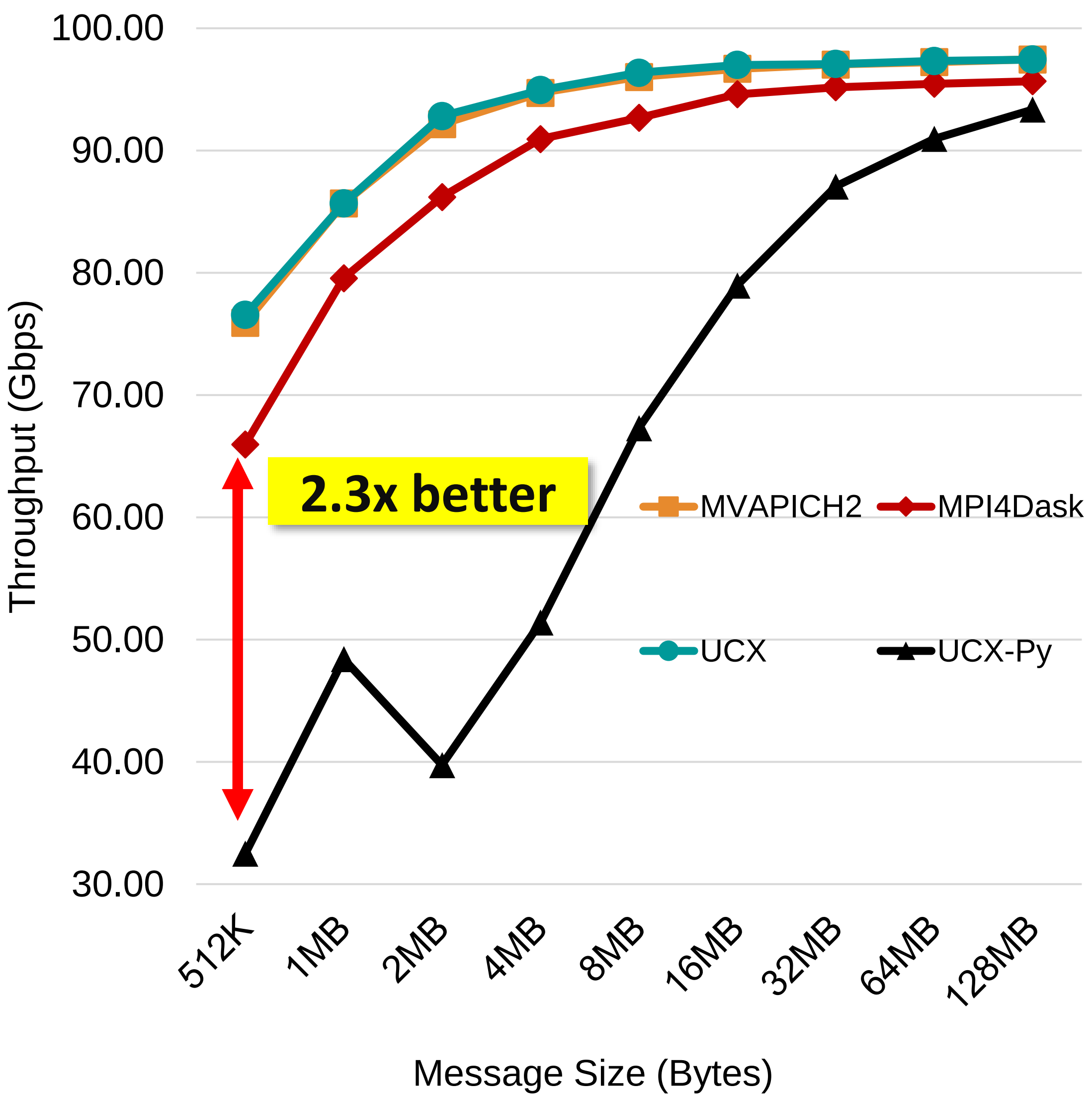
# Latency/Throughput Comparison (UCX-Py vs. MPI4Dask)



# CPU-to-CPU Communication Comparison

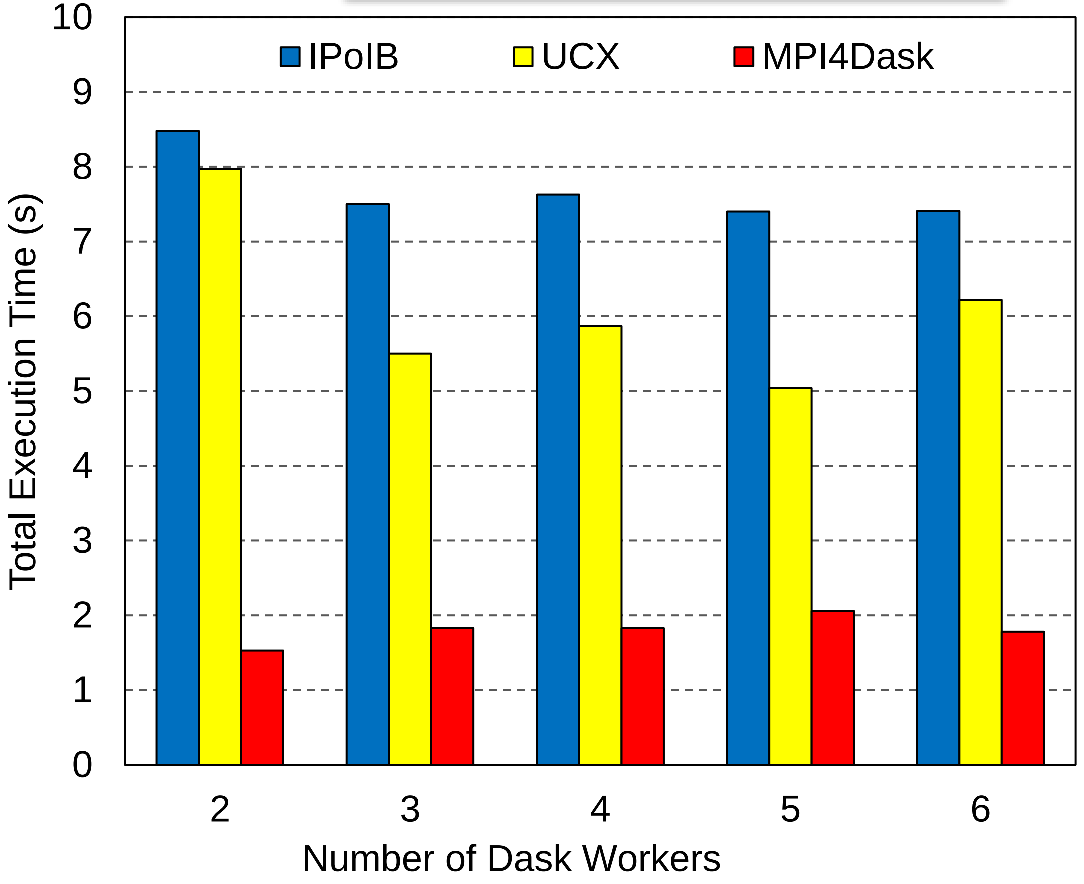


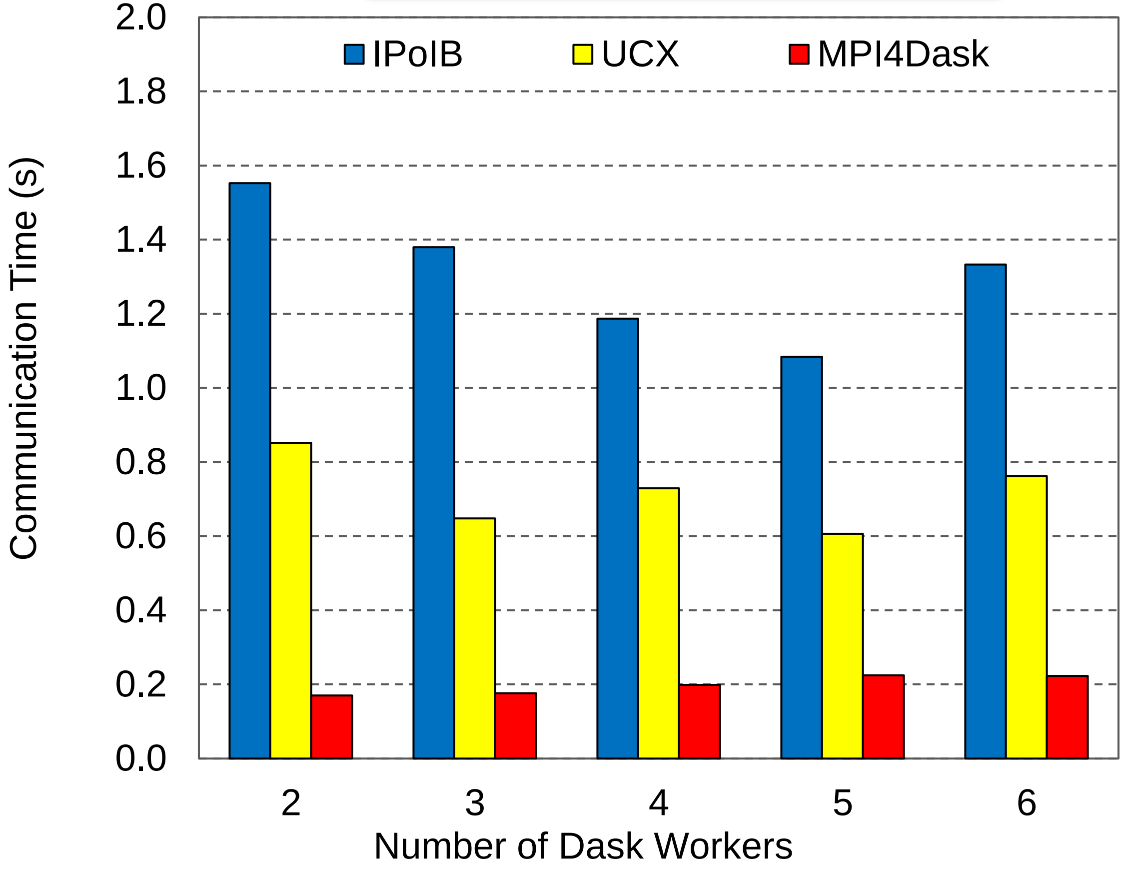

**UCX: v1.8.0, CUDA: 10.2, UCX-Py: v0.17, MPI4Dask: 0.2, and MVAPICH2: 2.3.5**

# Benchmark #1: Sum of cuPy Array and its Transpose (RI2)

**3.47x better on average**




**6.92x better on average**





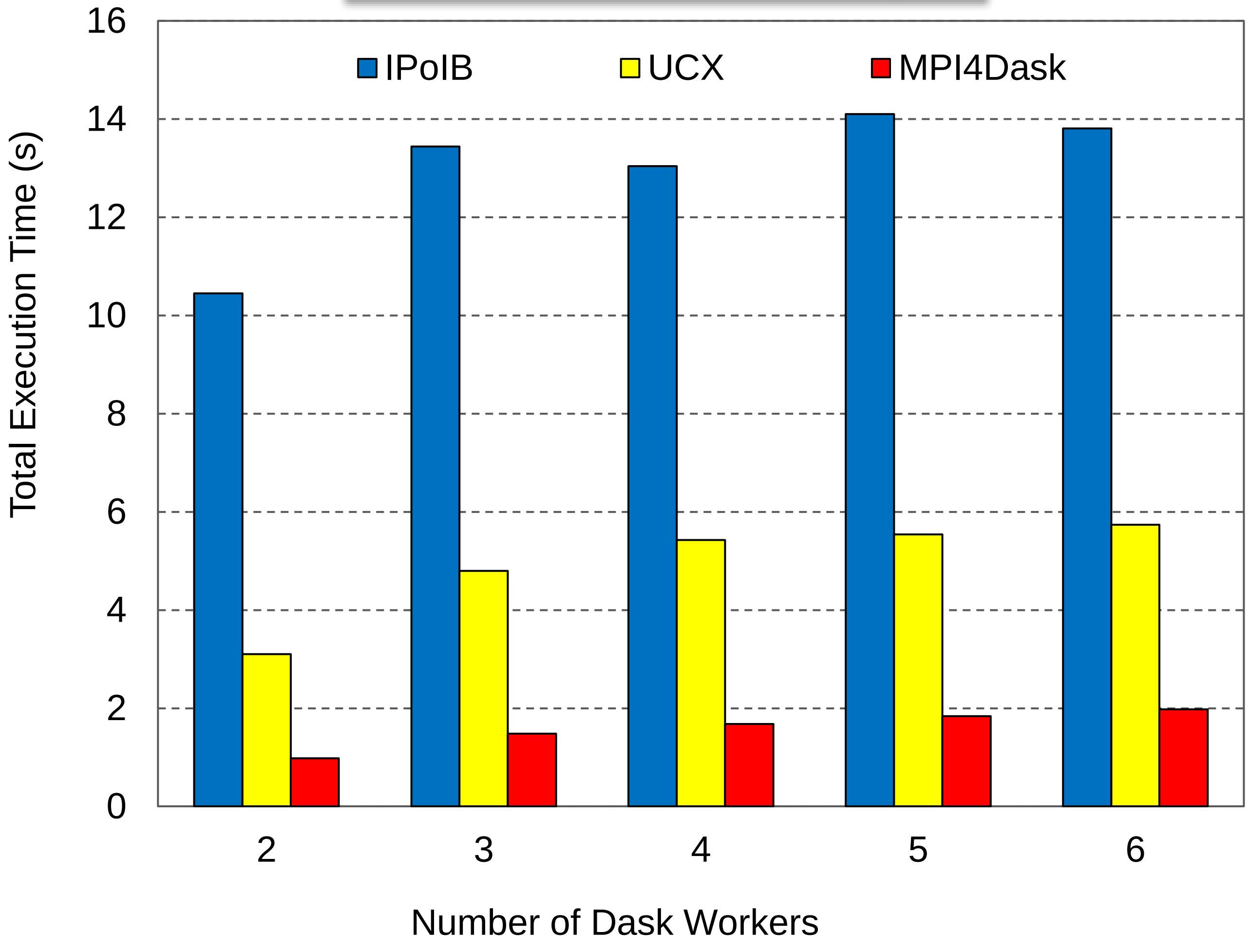
**A. Shafi , J. Hashmi , H. Subramoni , and D. K. Panda, Efficient MPI-based Communication for GPU-Accelerated Dask Applications, The 21st IEEE/ACM International Symposium on Cluster, Cloud and Internet Computing, May '21.**


**MPI4Dask 0.2 release**

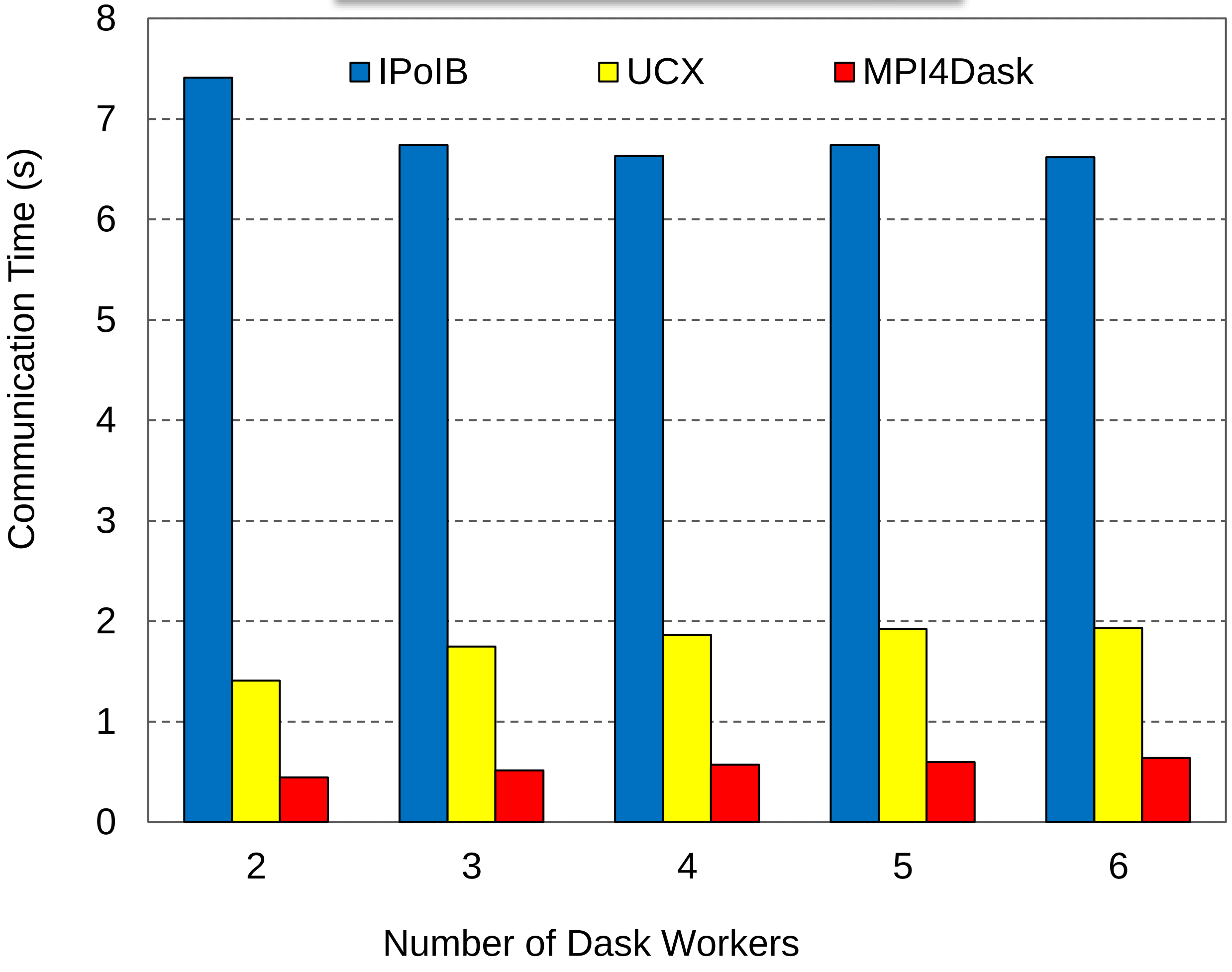
**(http://hibd.cse.ohio-state.edu)**

# Benchmark #2: cuDF Merge Operation

**3.11x better on average**




**3.22x better on average**





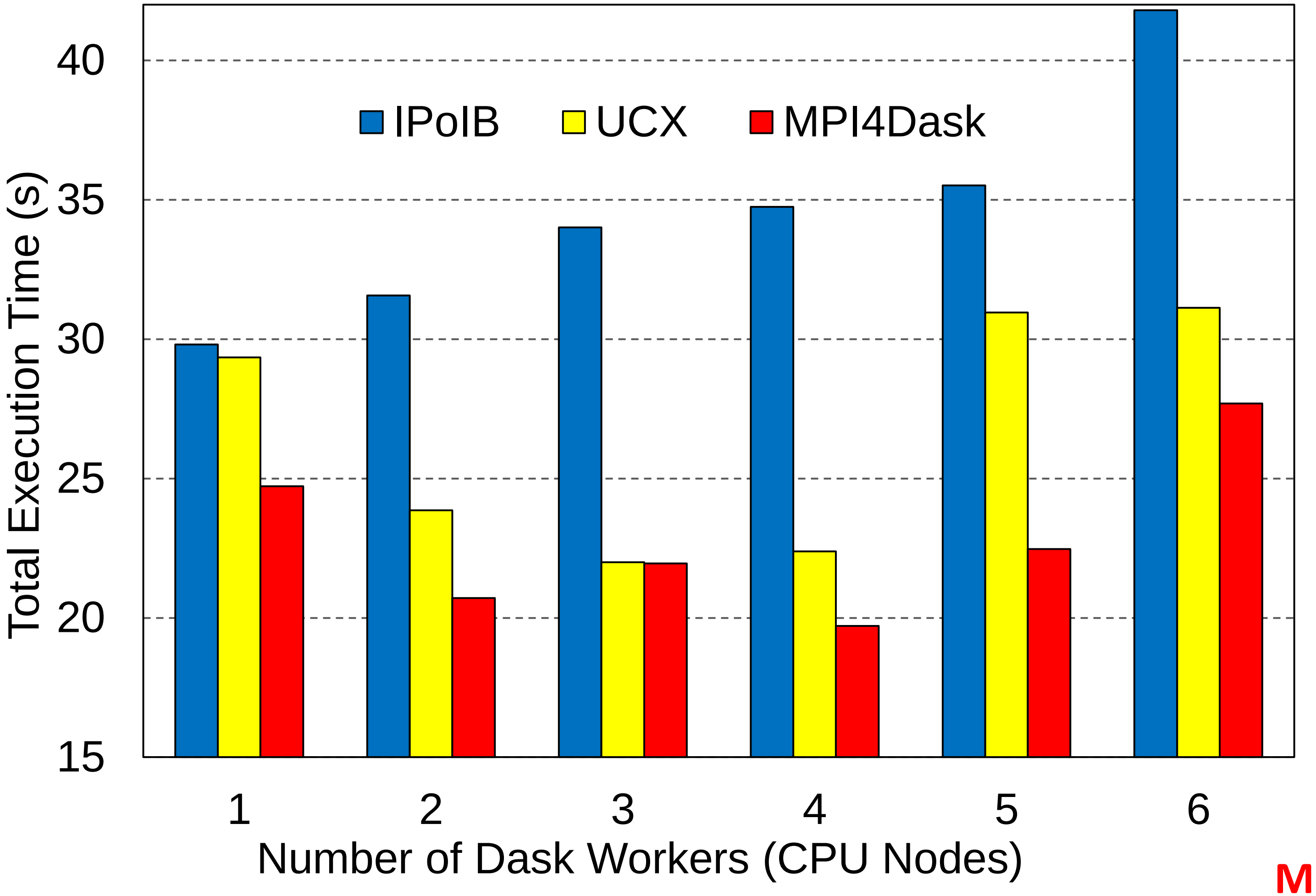
**A. Shafi , J. Hashmi , H. Subramoni , and D. K. Panda, Efficient MPI-based Communication for GPU-Accelerated Dask Applications, 21st IEEE/ACM CCGrid, May 21, https://arxiv.org/abs/2101.08878**


**MPI4Dask 0.2 release**

**(http://hibd.cse.ohio-state.edu)**

# Benchmark #4: Sum of numPy Array and its Transpose (RI2)

**1.16x better on average**

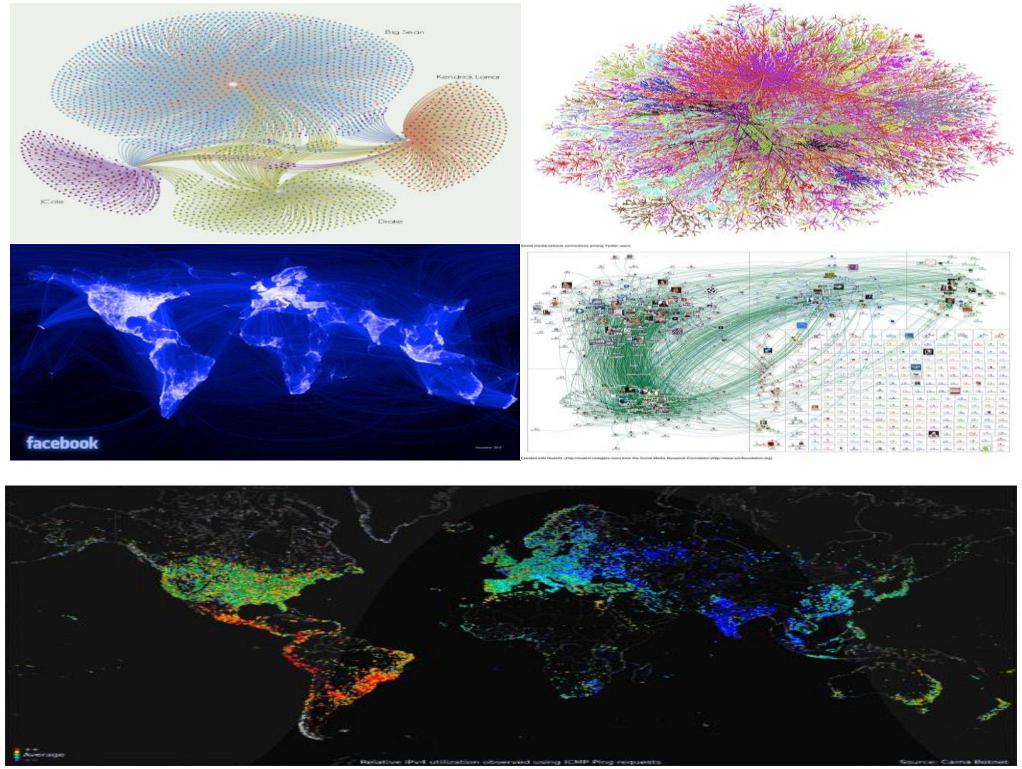



**MPI4Dask 0.2 release**

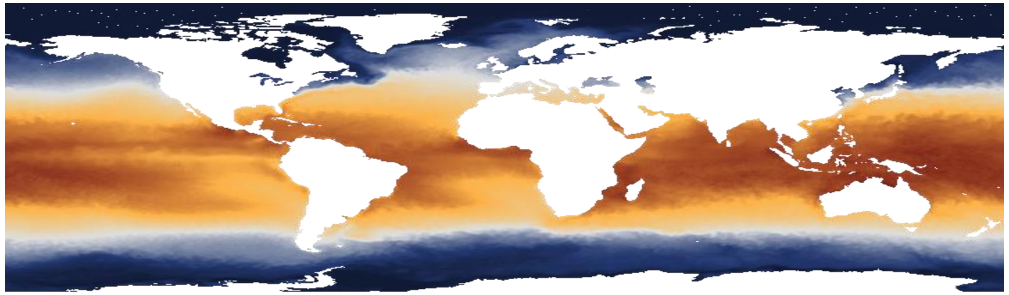
**(http://hibd.cse.ohio-state.edu)**

# Introduction to Big Data Analytics and Trends

- 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
- Running High Performance Data Analysis (HPDA) workloads in the cloud is gaining popularity
    - According to the latest OpenStack survey, 27% of cloud deployments are running HPDA workloads
- Sometimes also called Data Science



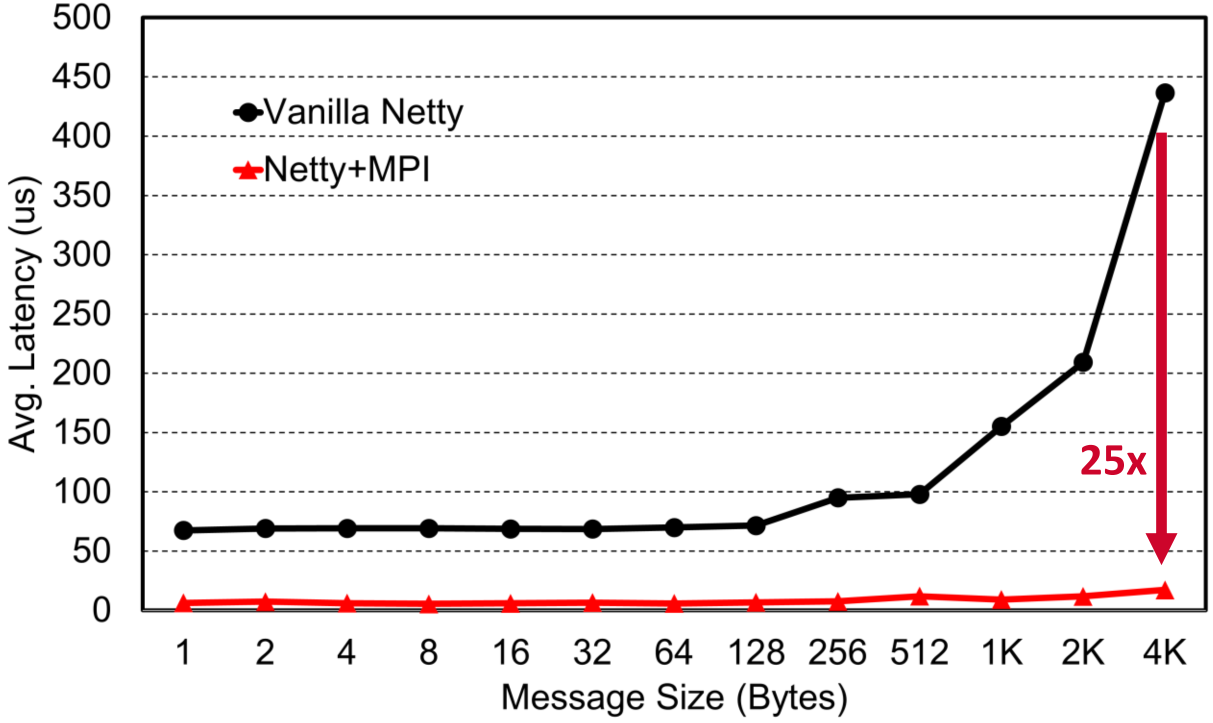

http://www.coolinfographics.com/blog/tag/data?currentPage=3

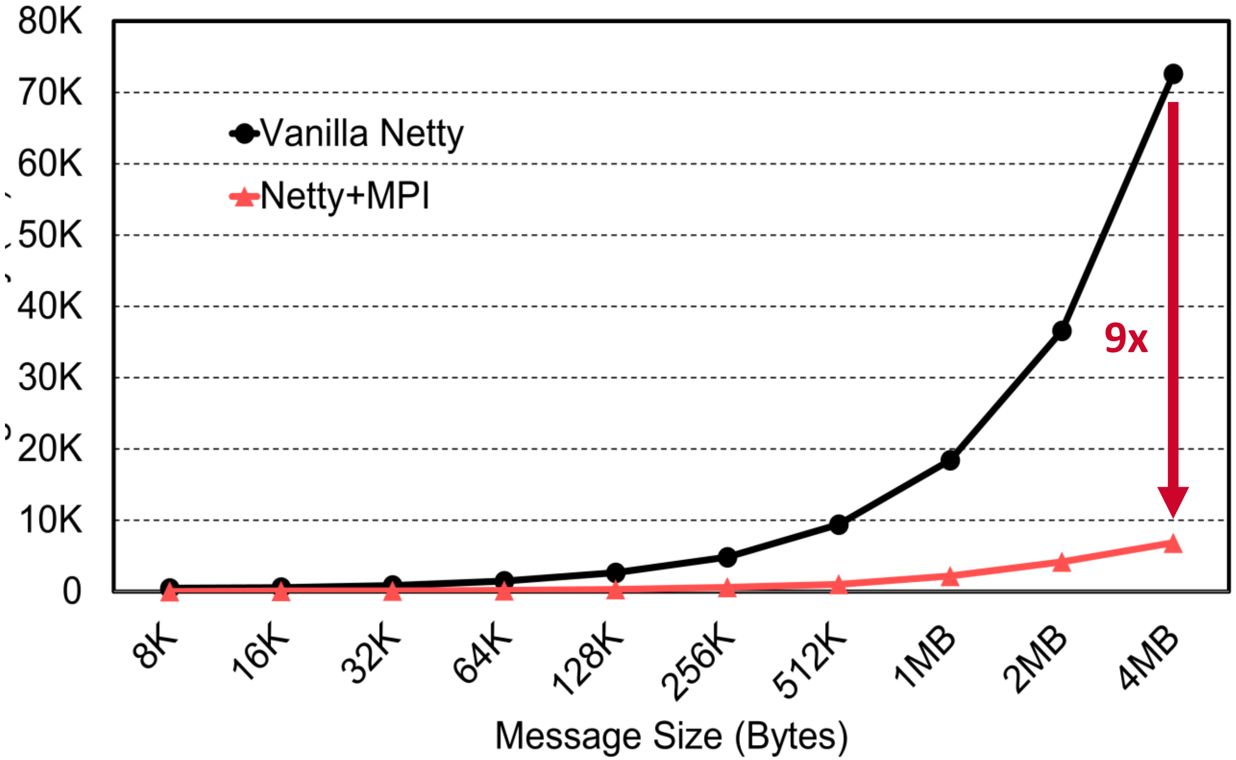


http://www.climatecentral.org/news/white-house-brings-together-big-data-and-climate-change-17194

# MPI4Spark: Performance of MPI-based Netty

- These figures represent the latency numbers for small and large message sizes
- The performance was analyzed using a ping pong Netty benchmark
- For small messages, we see a speed-up of **25x** at 4K
- For large messages, we see a speed-up of **9x** at 4MB




K. Al Attar, A. Shafi, M. Abduljabbar, H. Subramoni, D. Panda, Spark Meets MPI: Towards High-Performance Communication Framework for Spark using MPI, IEEE Cluster '22, Sep 2022.