AUBURN UNIVERSITY
Samuel Ginn College of Engineering

# MPIgnite: An MPI-Like Language for Apache Spark

Brandon Morris
Auburn University
blm0026@auburn.edu

Anthony Skjellum
University of Tennessee at Chattanooga
tony-skjellum@utc.edu

Spark

## Motivation

- Computational needs and capabilities have exploded, but MPI has not followed into mainstream interest[1]
- Numerous high-level, data parallel solutions are used instead
- Can be easier to program, but lack task-parallel and granularity
- What if these frameworks incorporated traditional HPC techniques?

## Apache Spark: Cloud Data Processing

- Uses fundamental Resilient Distributed Datasets (RDDs) to handle large data[2]
  - Performs **data parallel** *actions* and *transformations* on RDDs
- Runs on commodity hardware with several cluster managers (can work on HPC infrastructure[3])
- Comes with several libraries for machine learning, graph computations, stream processing

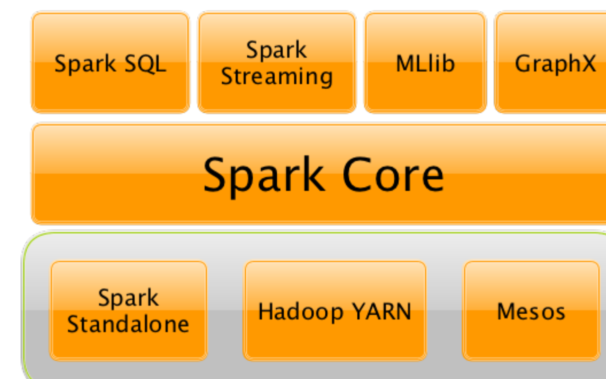
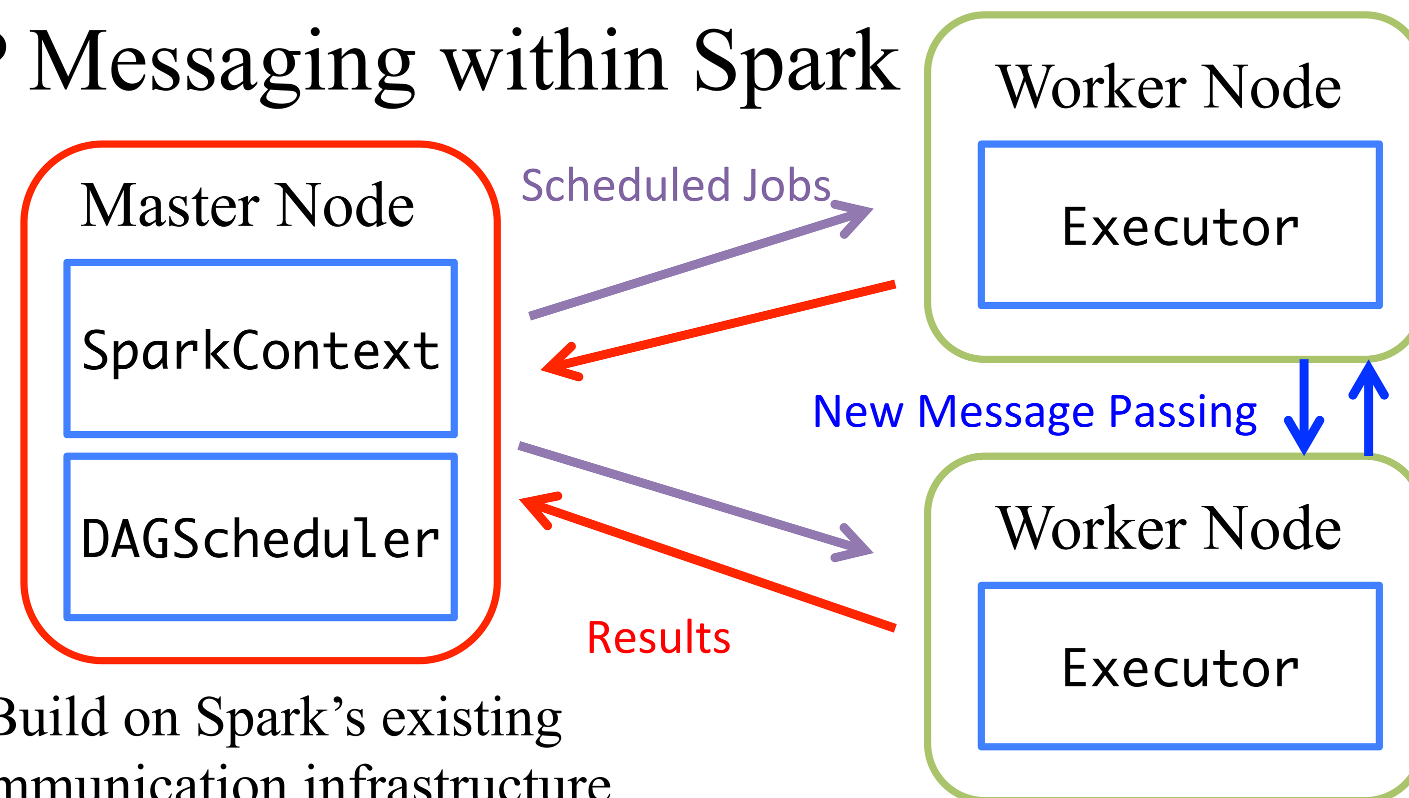



Figure 1: Diagram of Spark platform, libraries, and cluster managers (taken from [4])

## Objectives

- Introduce MPI-like message passing into open source Apache Spark
  - Direct peer-to-peer communication across worker nodes
  - Take advantage language features and existing code infrastructure
- Create a programming environment suitable for both traditional HPC and cloud developers
  - Maintain original Spark capabilities; strictly additive features
  - API and features strongly influenced by MPI standard

## P2P Messaging within Spark




Build on Spark's existing communication infrastructure

## Example: Matrix-Vector Multiplication with 2D Decomposition

```
// Multiply A*x with 2D decomposition
sc.parallelizeFunc[Array[Int]]((world: SparkComm) => {
  val worldRank = world.getRank
  val rowComm = world.split(worldRank / 3, worldRank)
  val colComm = world.split(worldRank % 3, worldRank)

  val a = A(colComm.getRank)(rowComm.getRank)
  println(s"Rank ${worldRank}: a = ${a}")

  // Distribute the x value to the appropriate process
  if (rowComm.getRank == rowComm.getSize - 1) {
    rowComm.send(colComm.getRank, 0,x(colComm.getRank))
  }

  val x_row = if (rowComm.getRank == colComm.getRank) {
    Some(rowComm.receive[Int](rowComm.getSize - 1, 0))
  } else None
  val multiplied = x_row match {
   case Some(x) =>
    colComm.broadcast[Int](colComm.getRank, x)
    x * a
   case None =>
    a * colComm.broadcast[Int](rowComm.getRank)
  }
  val result = rowComm.allReduce[Int](multiplied,
                  (a:Int, b: Int) => a + b)

  // Send the resulting vector to the root
  if (rowComm.getRank == 0) {
    world.send(0, 0, result)
  }
  if (world.getRank == 0)
    (for (i <- 0 until colComm.getSize) yield {
      world.receive[Int](i * colComm.getSize, 0)
    }).toArray
  else Array()
}).execute(9)(0) // Returns result from each process
```

## Implementation

- Function closures parallelized with `parallelize` method
- The `SparkComm` argument is the world communicator, can be split
- Utilizes existing communication to allow for process messages; augmented to allow direct peer-to-peer communication
- Parallelized functions can be repeatedly executed with arbitrary number of processes with `execute` method
- Parallel closures can be named, anonymous, imported, and chained together

## Results

- Introduces task parallelism into existing Apache Spark framework
- Developers can interchangeably switch between sophisticated Spark data parallelism (RDDs) or MPI-like task parallelism
- MPIgnite parallel closures can run alongside normal RDDs
  - Similar to Chapel *cobegin* statements[5]
- Does *not* simply augment MPI to a MapReduce model, or run Spark in HPC environment
- Incorporates HPC and MPI concepts into the popular world of cloud computing
- Several core MPI functions implemented (similar but not identical syntax), including send, receive, and some collective functions

## API Comparison

| MPIgnite | MPI |
|---|---|
| comm.send(rec, tag, data) | MPI_Send |
| comm.receive[T](sender, tag): T | MPI_Recv |
| comm.receiveAsync[T](sender, tag): Future[T] | MPI_Irecv |
| Await.result(f: Future[T]): T | MPI_Wait |
| comm.getRank | MPI_Comm_rank |
| comm.getSize | MPI_Comm_size |
| comm.split(color, key): SparkComm | MPI_Comm_split |
| comm.broadcast[T](root, data[1]): T | MPI_Bcast |
| comm.allReduce[T](data, f(a, b): T): T | MPI_Allreduce |

## References

1. Jonathan Dursi. 2015 HPC is dying, and MPI is killing it (2015). http://www.dursi.ca/hpc-is-dying-and-mpi-is-killing-it/.
2. M. Zaharia, M. Chowdhury, M. J. Franklin, S. Shenker, and I. Stoica. Spark: Cluster computing with working sets. HotCloud, 10(10-10):95, 2010.
3. T. Baer, P. Peltz, J. Yin, and E. Begoli. Integrating Apache Spark into PBS-Based HPC Environments. Proceedings of the 2015 XSEDE Conference on Scientific Advancements Enabled by Enhanced Cyberinfrastructure - XSEDE '15, pages 1–7, 2015.
4. Jacek Laskowski. Mastering Apache Spark 2. https://jaceklaskowski.gitbooks.io/mastering-apache-spark/
5. B. Chamberlain, D. Callahan, and H. Zima. Parallel Programmability and the Chapel Language. International Journal of High Performance Computing Applications, 21(3):291–312, 2007

## Acknowledgments

This material is based upon work supported by the National Science Foundation under Grants Nos. 1562659 and 1229282.
Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation.

We acknowledge the previous work and contributions of Mr. Jared Ramsey in his MS thesis at Auburn that motivated this work. Dr. Jonathan Dursi's blog [1] was a strong motivator for this work.