Software for cluster computing

Steve Simon, P.Mean Consulting

Abstract

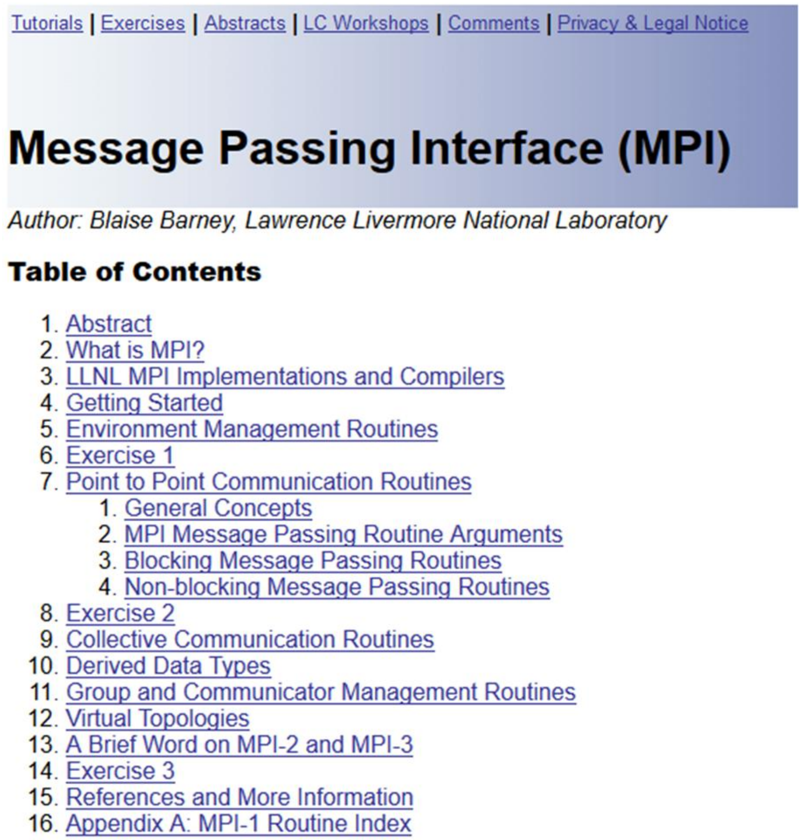
This presentation will cover some of the software systems for cluster computing with an emphasis on R libraries that can work with these systems.

Common features of software systems for cluster computing

All of software systems described in this presentation are

- distributed under open source licenses,
- have lots of free documentation, and
- run on any reasonable Unix based system.

Message Passing Interface (MPI)



Tutorials | Exercises | Abstracts | LC Workshops | Comments | Privacy & Legal Notice

Message Passing Interface (MPI)

Author: Blaise Barney, Lawrence Livermore National Laboratory

Table of Contents

Screenshot from Blaise Barney's MPI tutorial

MPI is a routine library

MPI Routines used for communicating between nodes

The Message Passing Interface is a standard used for message passing. Typically it is used in conjunction with a C or C++ program to farm out computation to the nodes of a cluster. The implementation of MPI used in this project was the open-source MPICH library. Two types of MPI operations were used in this project, collective and non-collective operations. Only two non-collective operations were used. *MPI_Send* is used to send data from one node to another, *MPI_Recv* is used to receive data from a particular node. Both these operations are blocking, meaning that the node which calls the operation pauses until the operation is complete.

```
MPI_Send(void *buf, int count, MPI_Datatype datatype, int dest, int tag,
         MPI_Comm comm)
MPI_Recv(void *buf, int count, MPI_Datatype datatype, int source, int tag,
         MPI_Comm comm, MPI_Status *status)
```

The other operations used are all collective. The *MPI_Bcast* operation broadcasts a message from the root node to all other nodes/processes in the specified group. This is used to broadcast the dimension of the matrix to all nodes, and also to broadcast an "exit" matrix to each node.

```
MPI_Bcast(void *buffer, int count, MPI_Datatype datatype, int root,
          MPI_Comm comm)
```

When the group of nodes that are to work on the matrix-vector multiplication has been set up, the root node must give out a portion of the matrix to each node. This can be achieve with *MPI_Send*, but it is much more efficient to use the *MPI_Scatter* operation. This operation farms out pieces of an array to different node. Thus, the decomposition of the matrix can be achieved in just one command!

```
MPI_Scatter(void *sendbuf, int sendcnt, MPI_Datatype sendtype, void *recvbuf,
            int recvcnt, MPI_Datatype recvtype, int root, MPI_Comm comm)
```

There is also a function called *MPI_Gather* that implements the opposite function of *MPI_Scatter*. When called on the root node, it gathers in data of a fixed size from all the nodes in the specified group, into an array. This is used to gather in the newly calculated qubit vector from the nodes, when the calculation is finished.

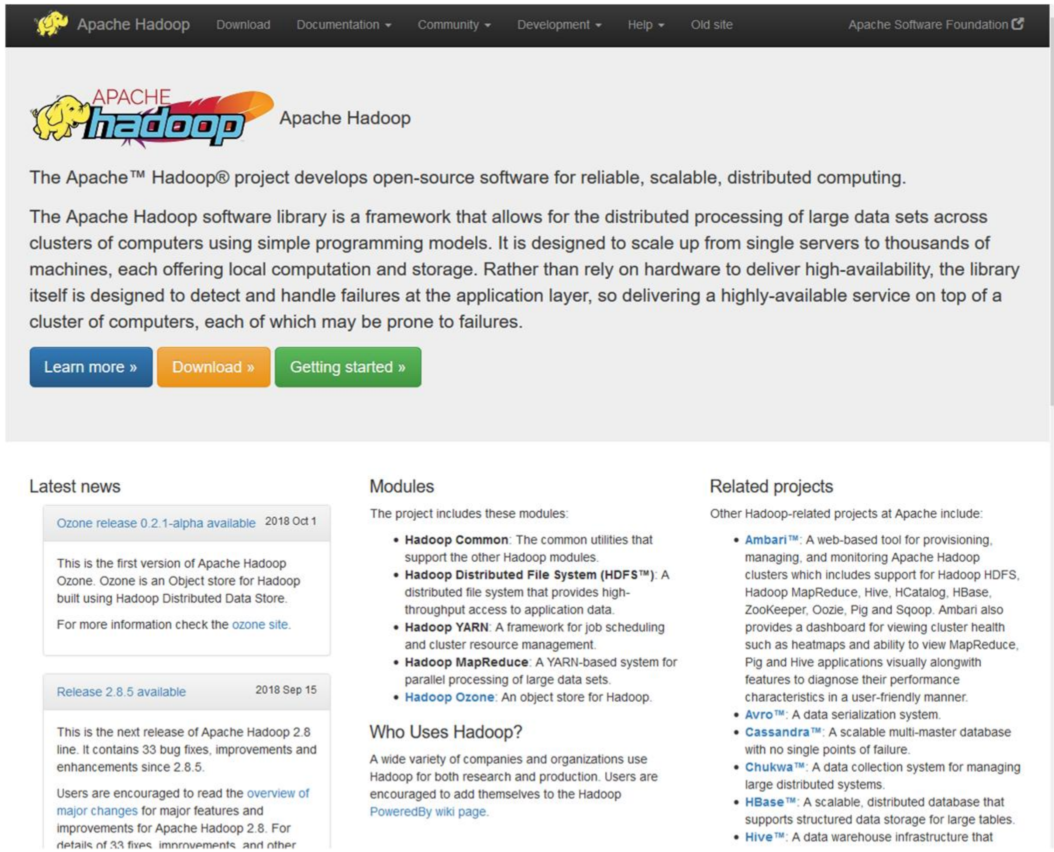
```
MPI_Gather(void *sendbuf, int sendcnt, MPI_Datatype sendtype, void *recvbuf,
           int recvcount, MPI_Datatype recvtype, int root, MPI_Comm comm)
```

None of the collective operations detailed above are blocking, even though they must operate at the same time on each node. To synchronize all the nodes, the *MPI_Barrier* operation is called after a collective function. This ensures that all nodes in the group are operating in the correct place.

```
MPI_Barrier(MPI_Comm comm)
```

Colm Ó hÉigeartaigh;'s website listing of MPI routines

Hadoop

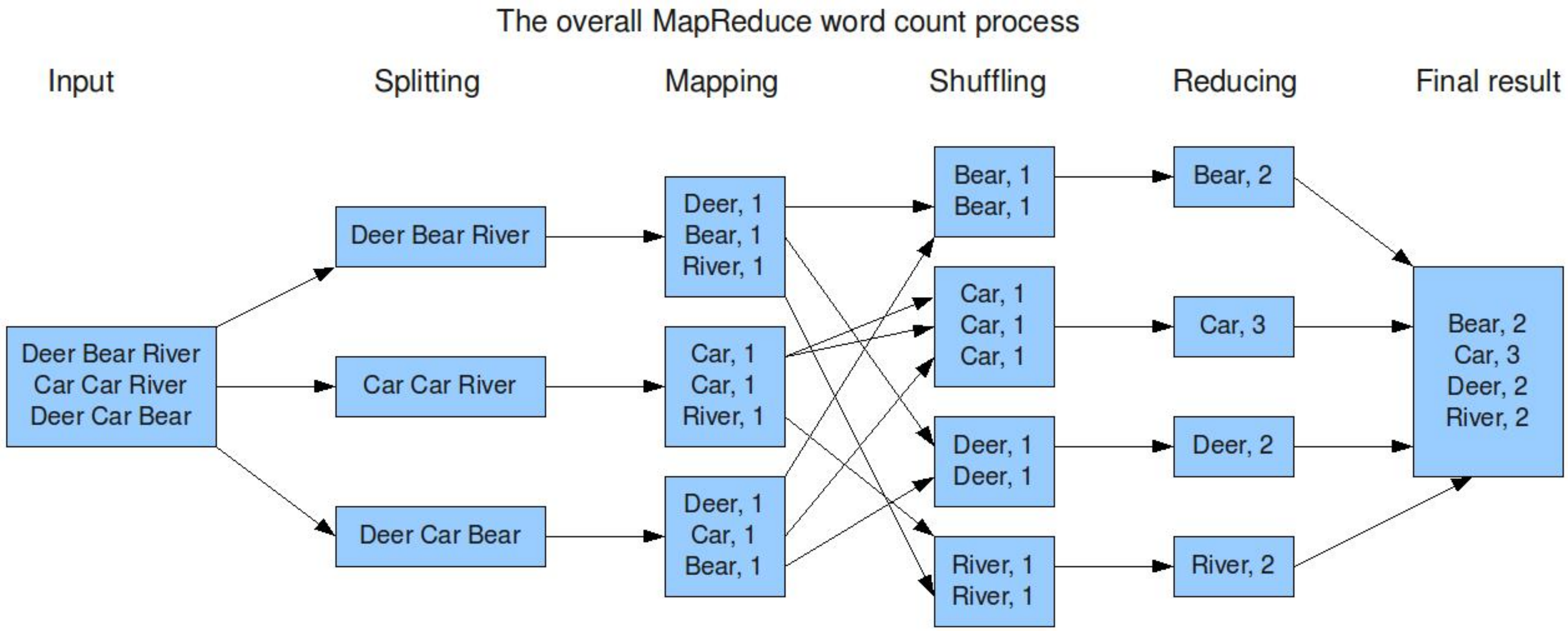


Screenshot from Apache Hadoop website

Components of Hadoop

– MapReduce
– Hadoop Distributed File System (HDFS)
– Hive
– Pig

MapReduce



Mapreduce applied to a simple word count example

Hadoop Distributed File System (HDFS)

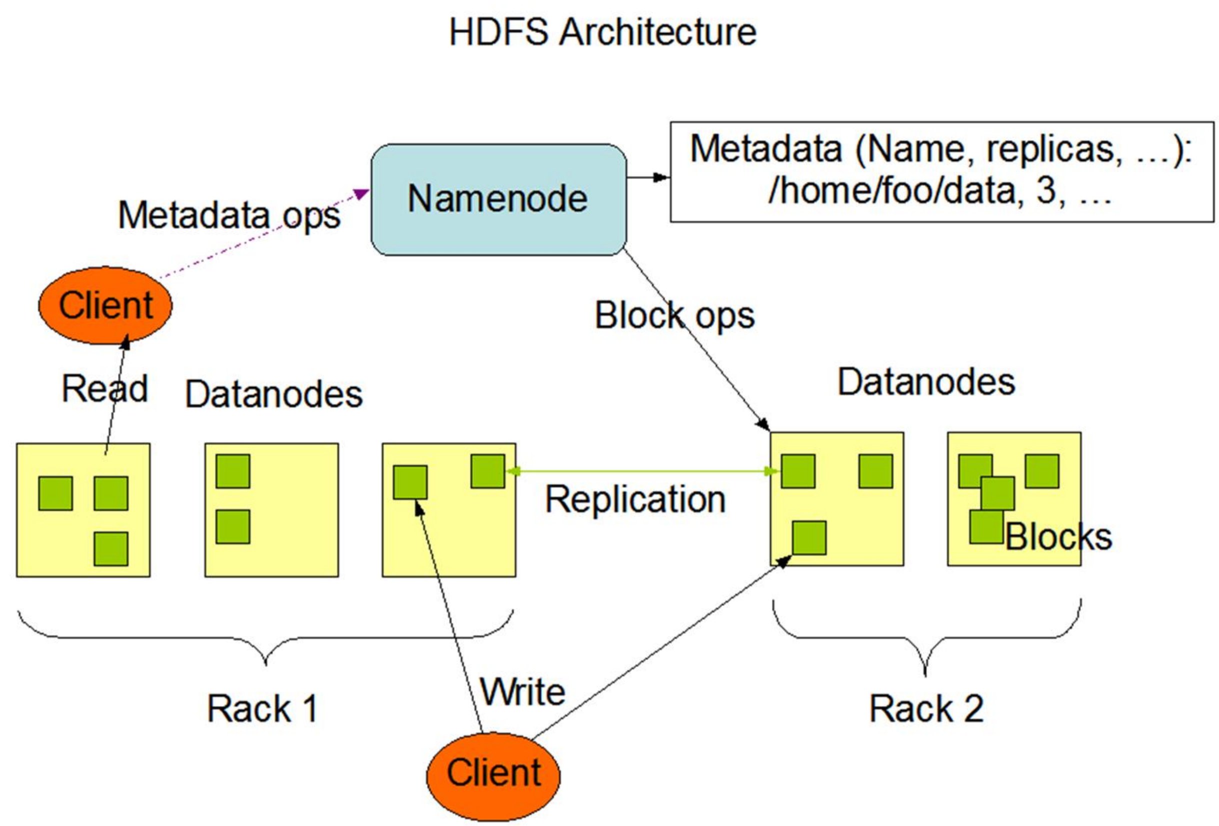


Illustration of HDFS architecture

Pig

Hive

Spark

Screenshot of Apache Spark main web page

Conclusion

This talk has covered

- Message Passing Interface
- Hadoop
  - MapReduce
  - HDFS
  - Pig
  - Hive
- Spark