Comparing MPI and MapReduce and Implementations of MapReduce

Nitya Dhanushkodi

MPI

Distributed memory

Centralized (master-worker) or decentralized coordination

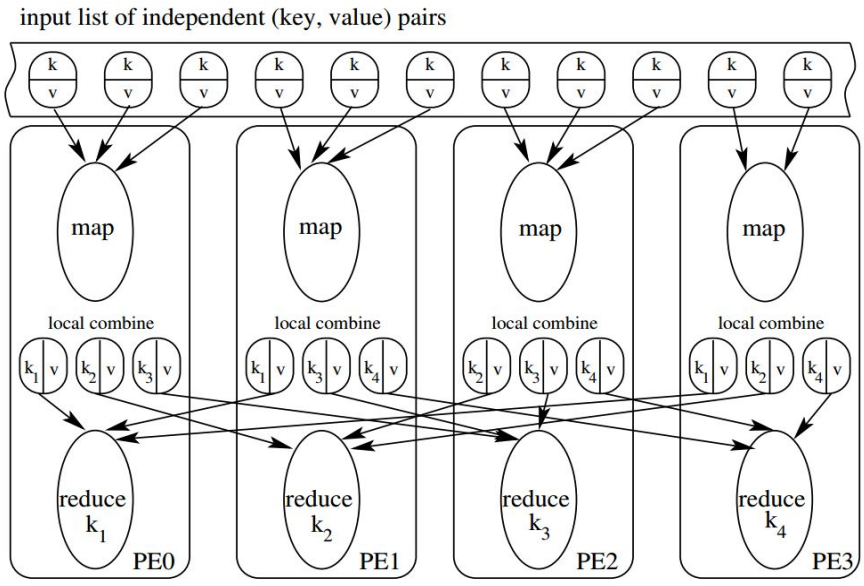
User must know what every process has to do

Whether process 3 is talking to process 5, etc.

MapReduce

User defines map and reduce function

MapReduce Communication



Towards Efficient MapReduce Using MPI. Hoefler et al.

When it gets to the reduce function, it is already grouped by key. Shuffling happens automatic!

Matrix Multiplication: MPI vs. MapReduce

P.S. Keep in mind differences in thinking for each algo :)

As an example to illustrate different kind of thinking to frame a problem in both of these paradigms, let's do matrix multiplication

(Review) MPI Matrix Multiplication

A x B = C, where cell (i, j) of C needs row i from A and col j from B.

Partition: compute row of C

Communication: Master divides tasks, sends row(s) of A to each worker, broadcast B. Worker receives broadcast, and where their offset in A/C is, plus data in A.

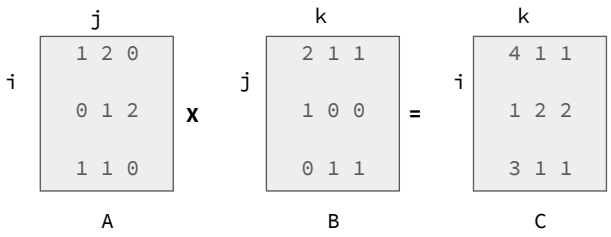
Compute: Workers compute their row(s) of C. Barrier until all workers are done.

Communication: Workers send row(s) of C and offset to master. Master receives row(s) of C and puts it into one big matrix at the correct offsets. Master can print C!

For review
Co

MR Matrix Multiplication

Matrix Multiplication with MapReduce. Maruf Aytekin



value

```
A, 0, 0, 1
A, 0, 1, 2
A, 0, 2, 0
A, 1, 0, 0
A, 1, 1, 1
…
B, 0, 0, 2
B, 0, 1, 1
B, 0, 2, 1
```

Input

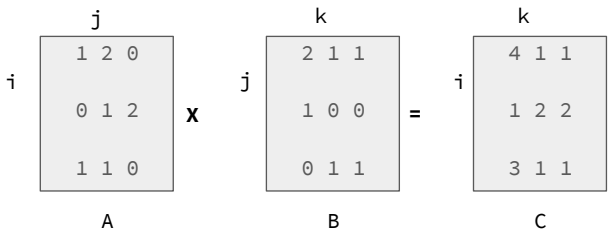
```
reduce (key, value) {
  if(A){
    for 0->k{
      nkey=value.row,k
      emit(nkey,value)
    }
  }
  if(B){
    for 0->i{
      nkey=i,value.col
      emit(nkey,value)
    }
  }
}
```

```
0,0 - A, 0, 0, 1
0,1 - A, 0, 0, 1
0,2 - A, 0, 0, 1
0,0 - A, 0, 1, 2
0,1 - A, 0, 1, 2
0,2 - A, 0, 1, 2
...1st row A to 0,0
0,0 - B, 0, 0, 2
1,0 - B, 0, 0, 2
…
0,0 - B, 1, 0, 1
```

Emitted by map function

MR Matrix Multiplication

Matrix Multiplication with MapReduce. Maruf Aytekin




```
0,0 -
<(A,0,0,1),
(A,0,1,2),
(A,0,2,0),
(B,0,0,2),
(B,1,0,1),
(B,2,0,0)>
```

Input

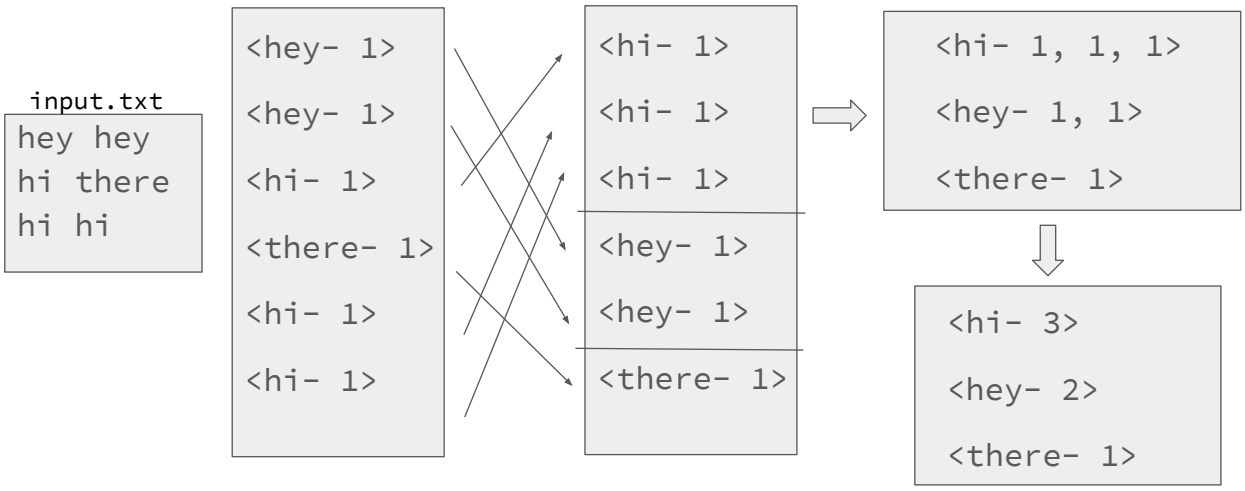
```
reduce (key, values) {
  int[] row
  int[] col
  for val:values ->{
    if(A){
      row[index]
    }
    if(B){
      col[index]
    }
  }
emit(doMatrixMult(row,col))
}
```

```
0,0 - 4
```

Emitted by reduce function

Some things are SUPER easy with MapReduce

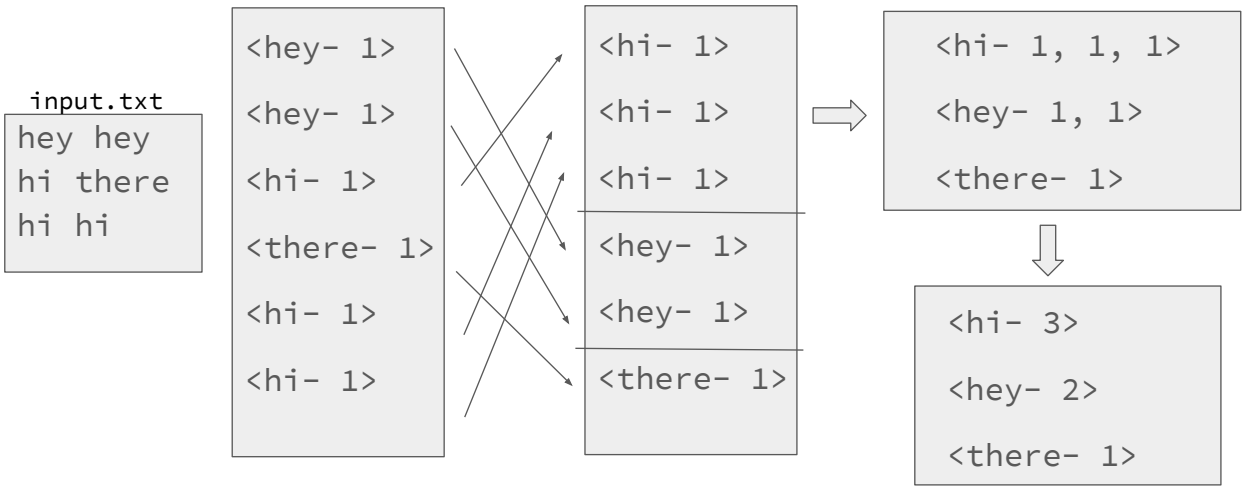
Wordcount



Matrix Mult is a little hefty with mapreduce, and requires a little bit of change in thinking, but there are some programs that are really really easy to do with mapReduce

Each word is a value, key is byte offset in file, unused. Map emits word, 1. Then the framework does the shuffling around til you have lists with keys. Reduce just says for every value, add it to some total, once done, return that value

Word Count Data Flow In MapReduce



Matrix Mult is a little hefty with mapreduce, and requires a little bit of change in thinking, but there are some programs that are really really easy to do with mapReduce

Each word is a value, key is byte offset in file, unused. Map emits word, 1. Then the framework does the shuffling around til you have lists with keys. Reduce just says for every value, add it to some total, once done, return that value

Differences between MPI and MapReduce?

- Communication?
  - How?
- Flexibility?
- What can go wrong?

Got any thoughts?

MPI

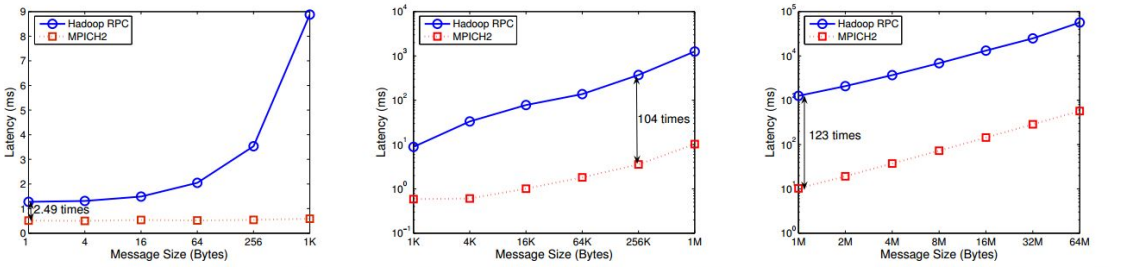
- Communication explicitly specified
- Low-level
- Sends (receives) data to (from) a node's memory
- Little to no fault tolerance support

MapReduce

- Communication is implicit
- High-level
- Communication is via expensive, shared distributed disk (distributed file system)
- Strong fault tolerance support

Pull out the couple of key strengths/weaknesses

Latency Comparison: MapReduce (Hadoop) and MPI (MPICH2)



(a) Small Messages (Size from 1 B to 1KB) (b) Medium Messages (Size from 1 KB to 1 MB) (c) Large Messages (Size from 1 MB to 64 MB)

Also, with thinking/application differences, there are latency differences that an interesting paper talked about
MPICH2 is impl of MPI
Hadoop is impl of MapReduce
This compares those 2.
Cool to think about, implementation wise, their protocols are pretty different so the communication is faster/slower depending on that

while sending large amounts of data between threads over a network can be expensive, it is nothing compared to the cost of reading from a hard drive

Implementation of MapReduce?

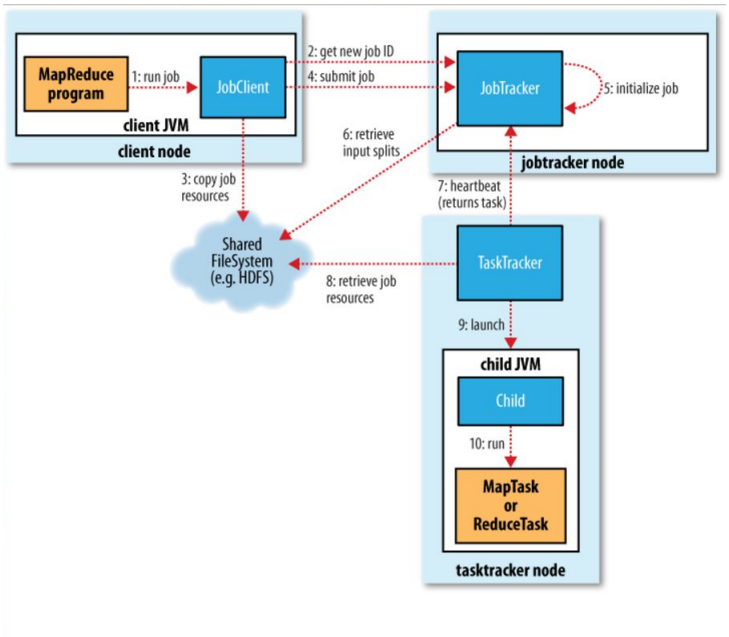
Can it be faster tho!?!?!

Zoom zoom zoom

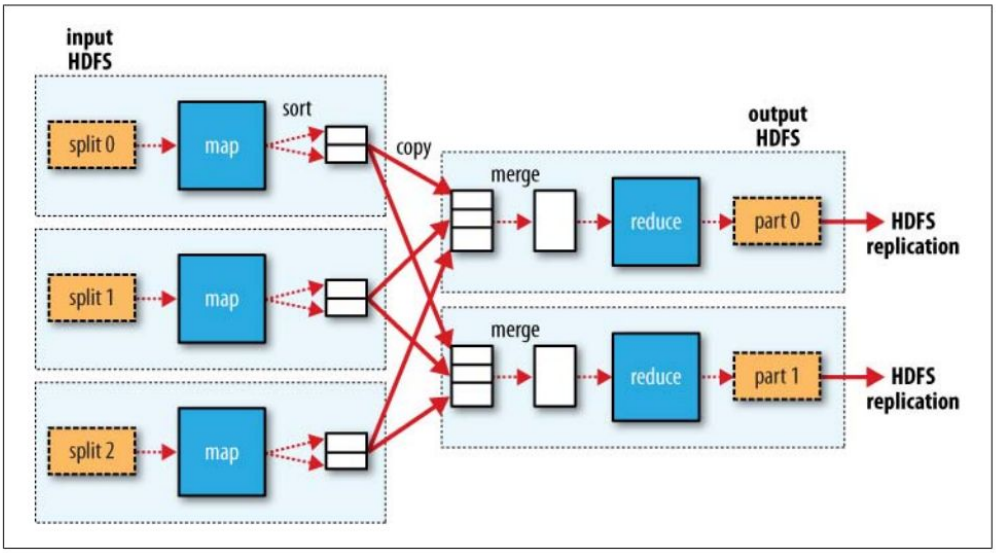
So theoretically, you can come to the same end goal of implementing parallel programs in either one. And we pulled out a few key strengths and weaknesses.

Most clearly, MapReduce is more fault tolerant (christine), and MPI's communication speed is faster.
For certain applications, depending on how efficiently you are communicating, etc. you could get faster times with either.

But if you improved the speed of MapReduce itself, then apps that do fall really elegantly in the MapReduce paradigm, like word count, could be made even faster.



Quick actual hadoop impl



Where all the communication happens

Implement MapReduce with MPI

1. Master splits up and assigns workers all the map tasks.
2. Master splits and assigns the reduce tasks to the workers.
3. Each reduce task has key(s) and function to carry out on all values associated with each key. Each reduce task works on 1 key at a time. For each reduce task, the worker queries all other workers from the map process for the values associated with that key.
4. Then the worker carries out the reduce task on all of these values associated with that particular key.

Implement with mpi simple

Problems?

Unbalanced and not optimized point to point communications

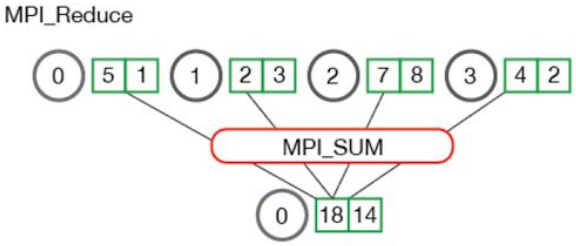
Slightly More Robust Implementation

Use collective functions!

But…no key?

```
int MPI_Reduce(const void *sendbuf, void *recvbuf, int count, MPI_Datatype datatype,
               MPI_Op op, int root, MPI_Comm comm)
```

Workaround



1. Reduce must be associative
2. Number of keys must be known
3. If there is no value for a given key, there must be some identity value

Limitations

Constraints on Reduce function

Data structures are different for input data

Fault tolerance

But…

Potential to be faster

Constraints on reduce make hard to adopt for general MapReduce fram

DFS allow non contiguous, variable sized data. MPI programs usually operate on contiguous/fixed size data,At least that is easy. The programmer would have to figure out how to manage non-contiguous data with extra effort.

Error handling in Mapreduce is to restart a task. Default error handling in MPI is to abort the entire job. If the user does change the default to handle errors, it will work ok for point to point comms, but collective communications would be much harder to handle errors from. MPI would need to internally support rebuilding communicators or restarting communications if they fail


Cool
POP 2,520
ELEV 1,518