MapReduce Programming Framework

Basic Concepts

- What is “map”?
  - A function/procedure that is applied to every individual elements of a collection/list/array/…
    - int square(x) { return x*x;}
    - map square [1,2,3,4] -> [1,4,9,16]
- What is “reduce”?
  - A function/procedure that performs an operation on a list. This operation will “fold/reduce” this list into a single value (or a smaller subset)
    - reduce ([1,2,3,4]) using sum -> 10
    - reduce ([1,2,3,4]) using multiply -> 24

MPI: Perspective

- Call MPI_Scatter to distribute data
  - Knowledge about data size?
  - Bottleneck at root process
- Each process applies the “map” function on local data
  + Processes can interact
- Call MPI_Reduce to applies the “reduce” function on final data
  - Need to define MPI_OP for complex reduce operation
  - Eventual bottleneck at root process for non-aggregative operations

Word Count

- Count how many unique words there are in a file/multiple files
- The “Hello World” of Big Data/Data Intensive Computing

Word Count: MPI

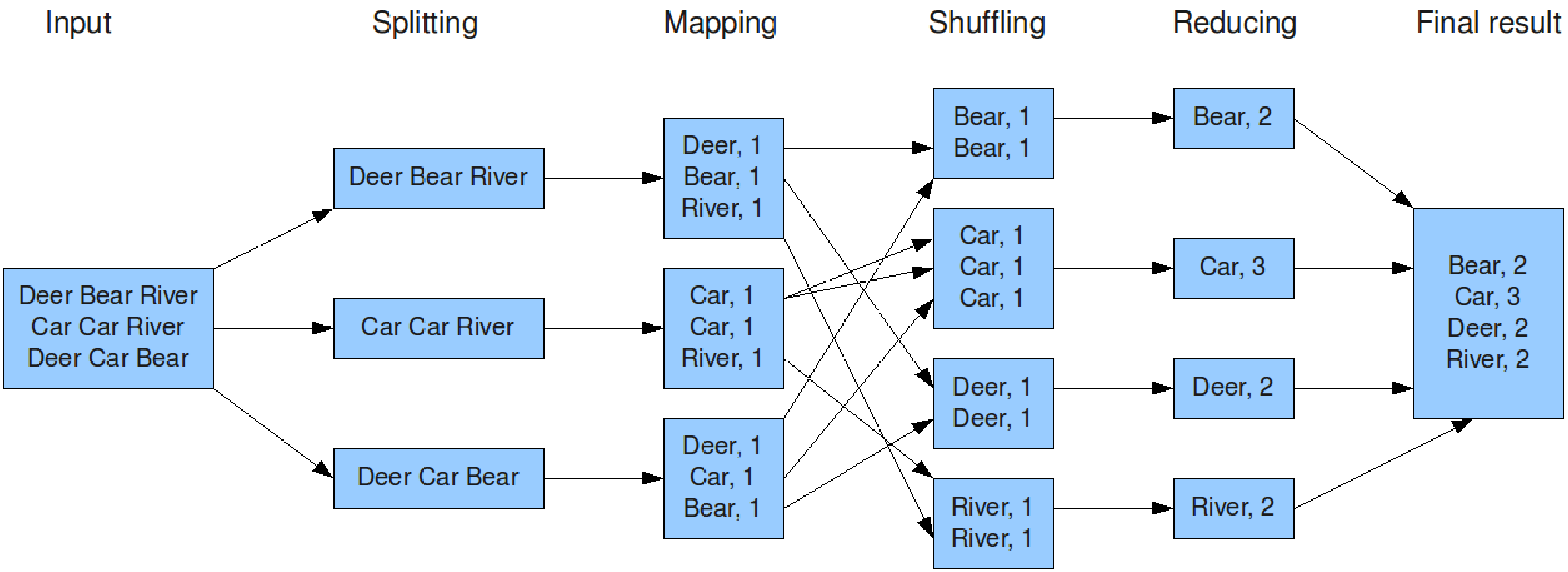
- MPI_Scatter:
  - Does data fit into memory?
  - If not, how do you set up MPI-IO or divide the files among the processes?
- Individual processes:
  - Unknown number of unique words in each workload: Linked list
  - struct {char* word, int count}
- MPI_Reduce
  - What happens if there are a lot of unique words?

Basic Programming Paradigm

- Programmers implement:
  - Map function:
    - Take in the input data and return a <key,value> pair
  - Reduce function:
    - Receive the <key,value> pairs from the mapper and provide a final output as a reduction operation on the pairs
- The MapReduce Framework (and HDFS) handles everything else

MapReduce WordCount Example

The overall MapReduce word count process



Input

Deer Bear River
Car Car River
Deer Car Bear

Splitting

Deer Bear River

Car Car River

Deer Car Bear

Mapping

Deer, 1
Bear, 1
River, 1

Car, 1
Car, 1
River, 1

Deer, 1
Car, 1
Bear, 1

Shuffling

Bear, 1
Bear, 1

Car, 1
Car, 1
Car, 1

Deer, 1
Deer, 1

River, 1
River, 1

Reducing

Bear, 2

Car, 3

Deer, 2

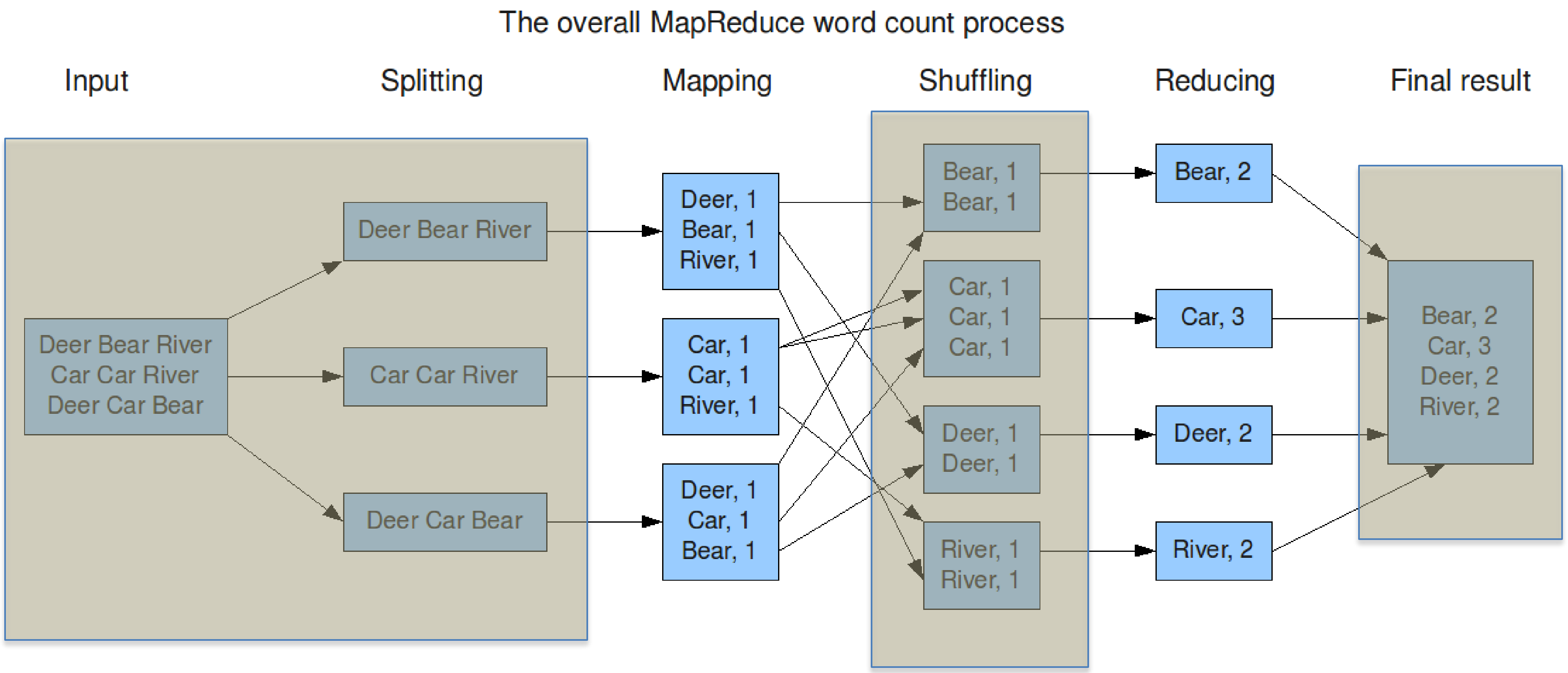
River, 2

Final result

Bear, 2
Car, 3
Deer, 2
River, 2

http://blog.jteam.nl/2009/08/04/introduction-to-hadoop/

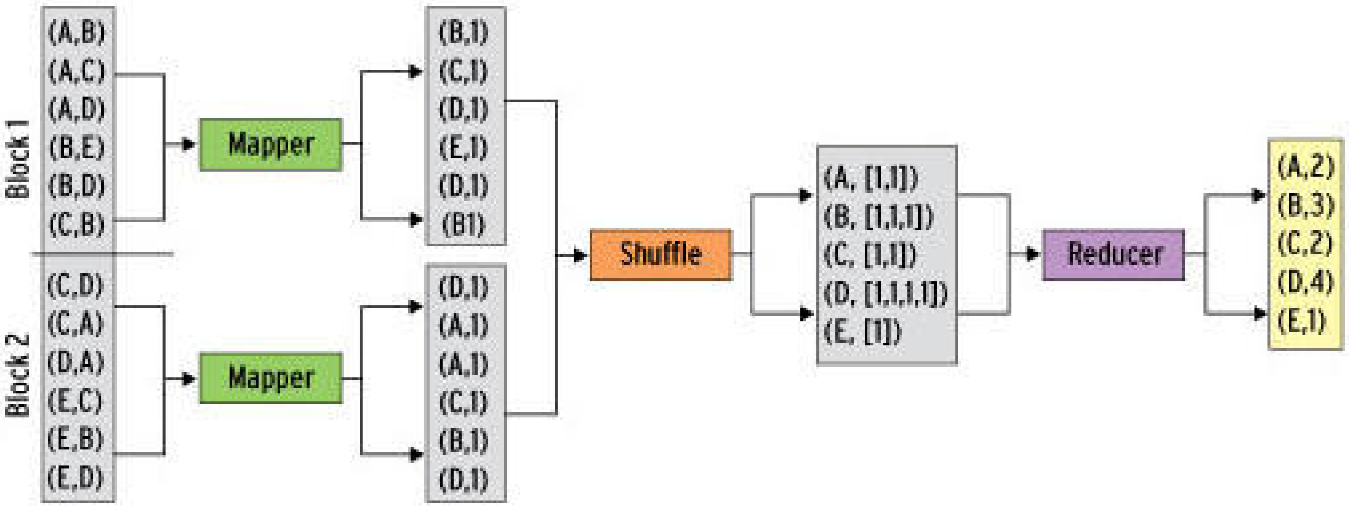
MapReduce WordCount Example



http://blog.jteam.nl/2009/08/04/introduction-to-hadoop/

MapReduce PageRank Example 1

(A,B): There is a referral (link) from site A to site B. Google looks at how many referrals site B has in order to determine the ranking of site B.



Block 1

(A,B)
(A,C)
(A,D)
(B,E)
(B,D)
(C,B)

Block 2

(C,D)
(C,A)
(D,A)
(E,C)
(E,B)
(E,D)

Mapper

(B,1)
(C,1)
(D,1)
(E,1)
(D,1)
(B1)

Mapper

(D,1)
(A,1)
(A,1)
(C,1)
(B,1)
(D,1)

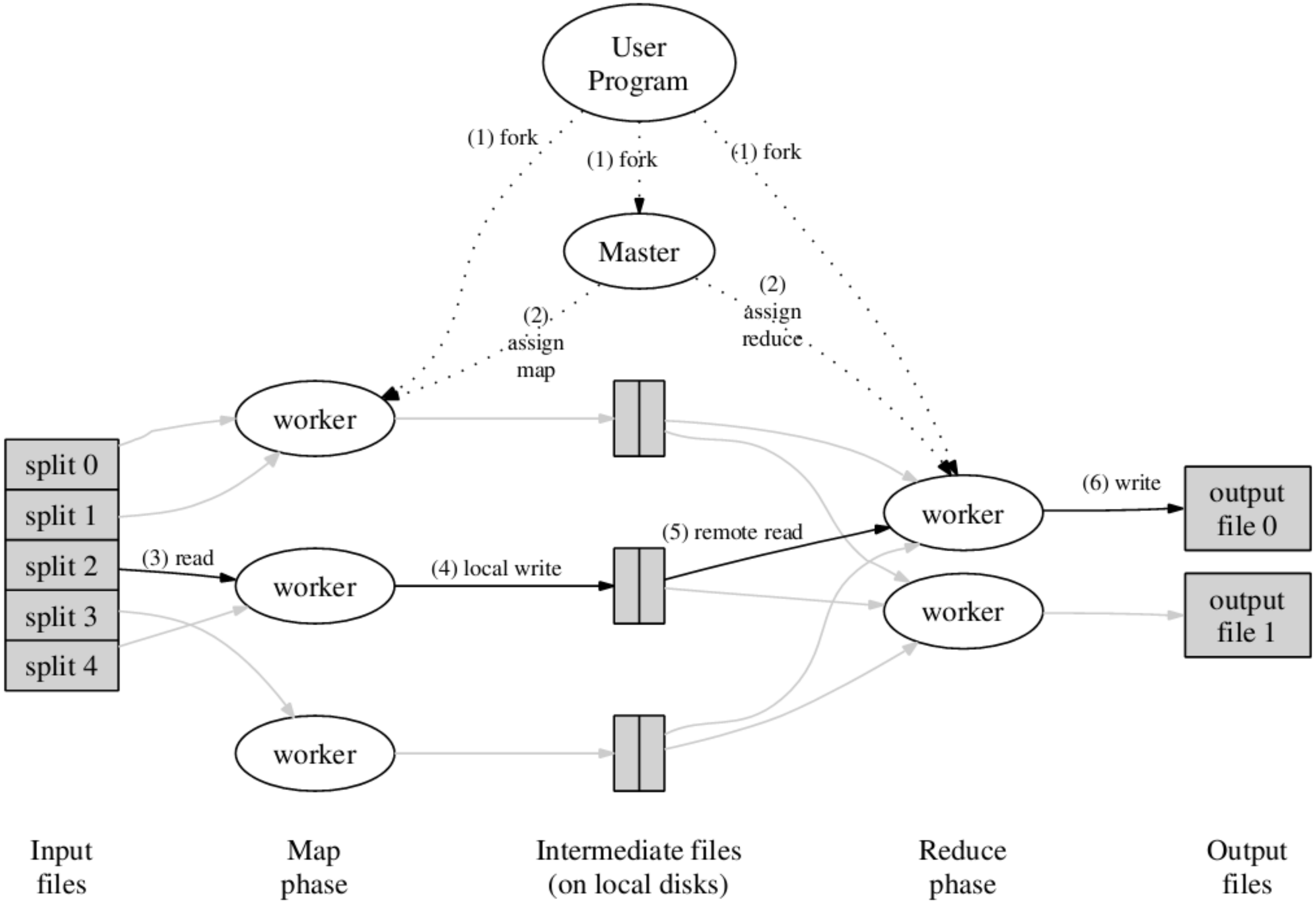
Shuffle

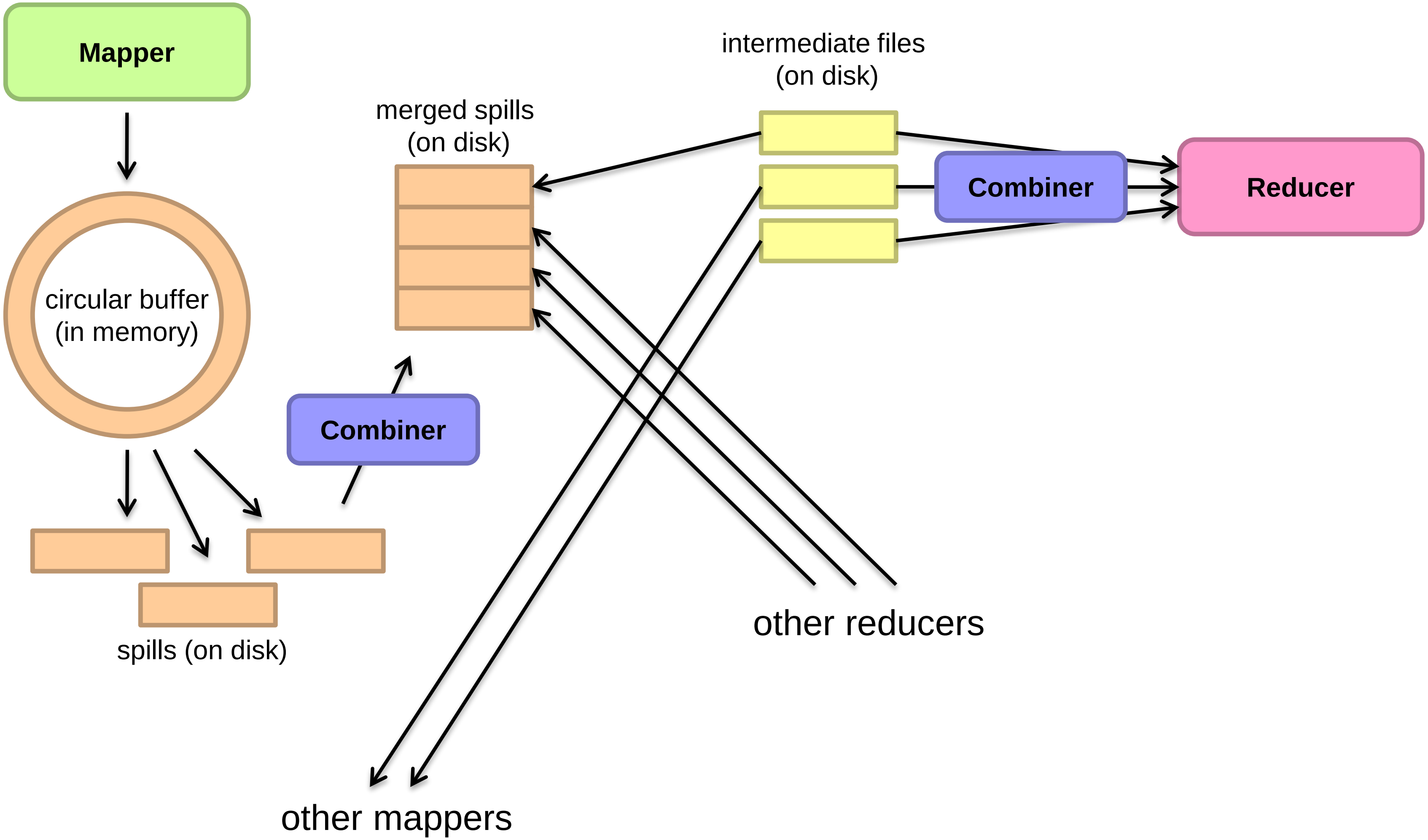
(A, [1,1])
(B, [1,1,1])
(C, [1,1])
(D, [1,1,1,1])
(E, [1])

Reducer

(A,2)
(B,3)
(C,2)
(D,4)
(E,1)

http://www.admin-magazine.com/HPC/Articles/MapReduce-and-Hadoop


User Program
(1) fork
(1) fork
(1) fork
Master
(2) assign map
(2) assign reduce
worker
worker
worker
split 0
split 1
split 2
split 3
split 4
(3) read
(4) local write
(5) remote read
worker
worker
(6) write
output file 0
output file 1
Input files
Map phase
Intermediate files (on local disks)
Reduce phase
Output files



Mapper

circular buffer
(in memory)

spills (on disk)

Combiner

merged spills
(on disk)

intermediate files
(on disk)

Combiner

Reducer

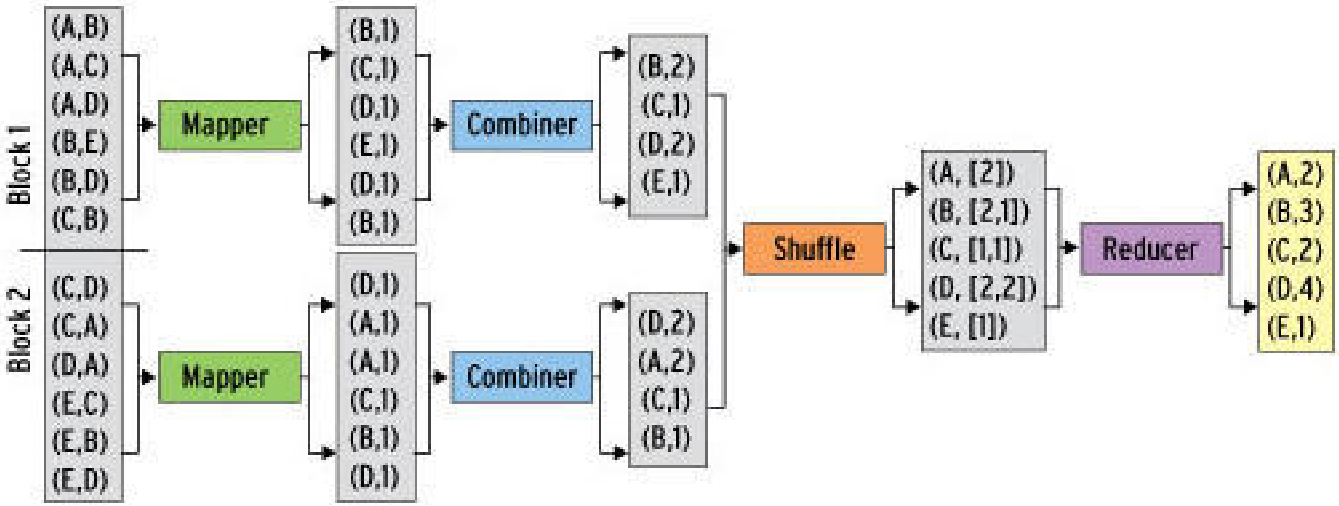
other reducers

other mappers

Basic Programming Paradigm

- Programmers implement:
  - Map function:
    - Take in the input data and return a <key,value> pair
  - Reduce function:
    - Receive the <key,value> pairs from the mapper and provide a final output as a reduction operation on the pairs
  - **Optional functions:**
    - **Partition function: determines the distribution of mappers' <key,value> pairs to the reducers**
    - **Combine functions: initial reduction on the mappers to reduce network traffics**
- The MapReduce Framework handles everything else

MapReduce PageRank Example 2



Block 1

(A,B)
(A,C)
(A,D)
(B,E)
(B,D)
(C,B)

Block 2

(C,D)
(C,A)
(D,A)
(E,C)
(E,B)
(E,D)

Mapper

(B,1)
(C,1)
(D,1)
(E,1)
(D,1)
(B,1)

Mapper

(D,1)
(A,1)
(A,1)
(C,1)
(B,1)
(D,1)

Combiner

(B,2)
(C,1)
(D,2)
(E,1)

Combiner

(D,2)
(A,2)
(C,1)
(B,1)

Shuffle

(A, [2])
(B, [2,1])
(C, [1,1])
(D, [2,2])
(E, [1])

Reducer

(A,2)
(B,3)
(C,2)
(D,4)
(E,1)

http://www.admin-magazine.com/HPC/Articles/MapReduce-and-Hadoop

What is “everything else”?

- “Everything else”
  - Scheduling
  - Data distribution
  - Synchronization
  - Error and Fault Handling
- Limited control over data and execution flow
  - All algorithms must be expressed as a combination of mapping, reducing, combining, and partitioning functions
- Extremely limited knowledge on
  - Location of mappers and reducers
  - Life cycle of individual mappers and reducers
  - Information about which mapper handles which data block
  - Information about which reducer handles which intermediate key

Challenges in working with MR

- All algorithms must be expressed as a combination of mapping, reducing, and possibly combining and partitioning, functions
- Large scale debugging is difficult
  - Functional errors are difficult to follow at large scale
  - Data-dependent errors are even more difficult to catch and fix

Applications of MapReduce

- Text tokenization, indexing, and search
  - Web access log stats
  - Inverted index construction
  - Term-vector per host
  - Distributed grep/sort
- Graph creation
  - Web link-graph reversal
  - Google's PageRank
- Data Mining and machine learning
  - Document clustering
  - Machine learning
  - Statistical machine translation