CSE 4/587
Data Intensive Computing

Dr. Eric Mikida
epmikida@buffalo.edu
208 Capen Hall

Dr. Shamshad Parvin
shamsadp@buffalo.edu
313 Davis Hall

MapReduce

Announcements

- Regrade requests

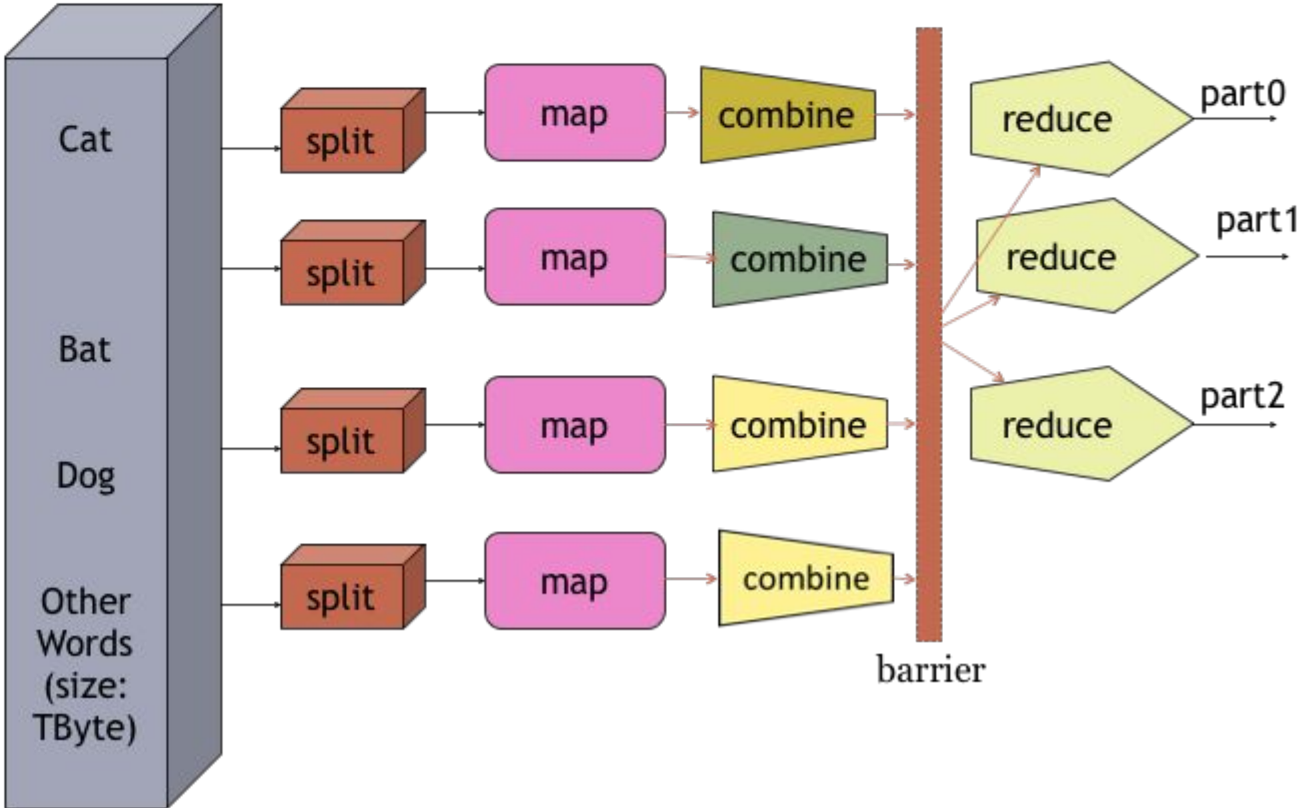
Additional Reference for MapReduce

Data-Intensive Text Processing with MapReduce, Jimmy Lin and Chris Dyer, Synthesis Lectures on Human Language Technologies, 2010, Vol. 3, No. 1, Pages 1-177, (doi: 10.2200/S00274ED1V01Y201006HLT007).

An online version of this text is also available through UB Libraries since UB subscribes to Morgan and Claypool Publishers.

Online version available at:
http://lintool.github.com/MapReduceAlgorithms/index.html

Recap of MR and Word Count



Word Count Problem Revisited

This is a cat
Cat sits on a roof

The roof is a tin roof
There is a tin can on the roof

Cat kicks the can
It rolls on the roof and falls on the next roof

The cat rolls too
It sits on the can

Word Count Problem: Mappers

This is a cat
Cat sits on a roof
<this 1> <is 1> <a 1> <cat 1> <cat 1> <sits 1> <on 1> <a 1> <roof 1>

The roof is a tin roof
There is a tin can on the roof
<the 1> <roof 1> <is 1> <a 1> <tin 1> <roof 1> <there 1> <is 1> <a 1> <can 1> <on 1> <the 1> <roof 1>

Cat kicks the can
It rolls on the roof and falls on the next roof
<cat 1> <kicks 1> <the 1> <can 1> <it 1> <rolls 1> <on 1> <the 1> <roof 1> <and 1> <falls 1> <on 1> <the 1> <next 1> <roof 1>

The cat rolls too
It sits on the can
<the 1> <cat 1> <rolls 1> <too 1> <it 1> <sits 1> <on 1> <the 1> <can 1>

Word Count Problem: Shuffle to Reducers

Output of Mappers:
<this 1> <is 1> <a 1> <cat 1> <cat 1> <sits 1> <on 1> <a 1> <roof 1> <the 1> <roof 1> <is 1> <a 1> <tin 1> <roof 1> <there 1> <is 1> <a 1> <can 1> <on 1> <the 1> <roof 1> <cat 1> <kicks 1> <the 1> <can 1> <it 1> <rolls 1> <on 1> <the 1> <roof 1> <and 1> <falls 1> <on 1> <the 1> <next 1> <roof 1> <the 1> <cat 1> <rolls 1> <too 1> <it 1> <sits 1> <on 1> <the 1> <can 1>

Word Count Problem: Shuffle to Reducers

Output of Mappers:
<this 1> <is 1> <a 1> <cat 1> <cat 1> <sits 1> <on 1> <a 1> <roof 1> <the 1> <roof 1> <is 1> <a 1> <tin 1> <roof 1> <there 1> <is 1> <a 1> <can 1> <on 1> <the 1> <roof 1> <cat 1> <kicks 1> <the 1> <can 1> <it 1> <rolls 1> <on 1> <the 1> <roof 1> <and 1> <falls 1> <on 1> <the 1> <next 1> <roof 1> <the 1> <cat 1> <rolls 1> <too 1> <it 1> <sits 1> <on 1> <the 1> <can 1>

Input to the Reducers: delivered sorted, by key
…
<can <1,1>>
<cat <1,1,1,1>>
…
<roof <1,1,1,1,1,1>>
…

Word Count Problem: Reduce

Reduce (sum in this case) the values:

…

<can 2>

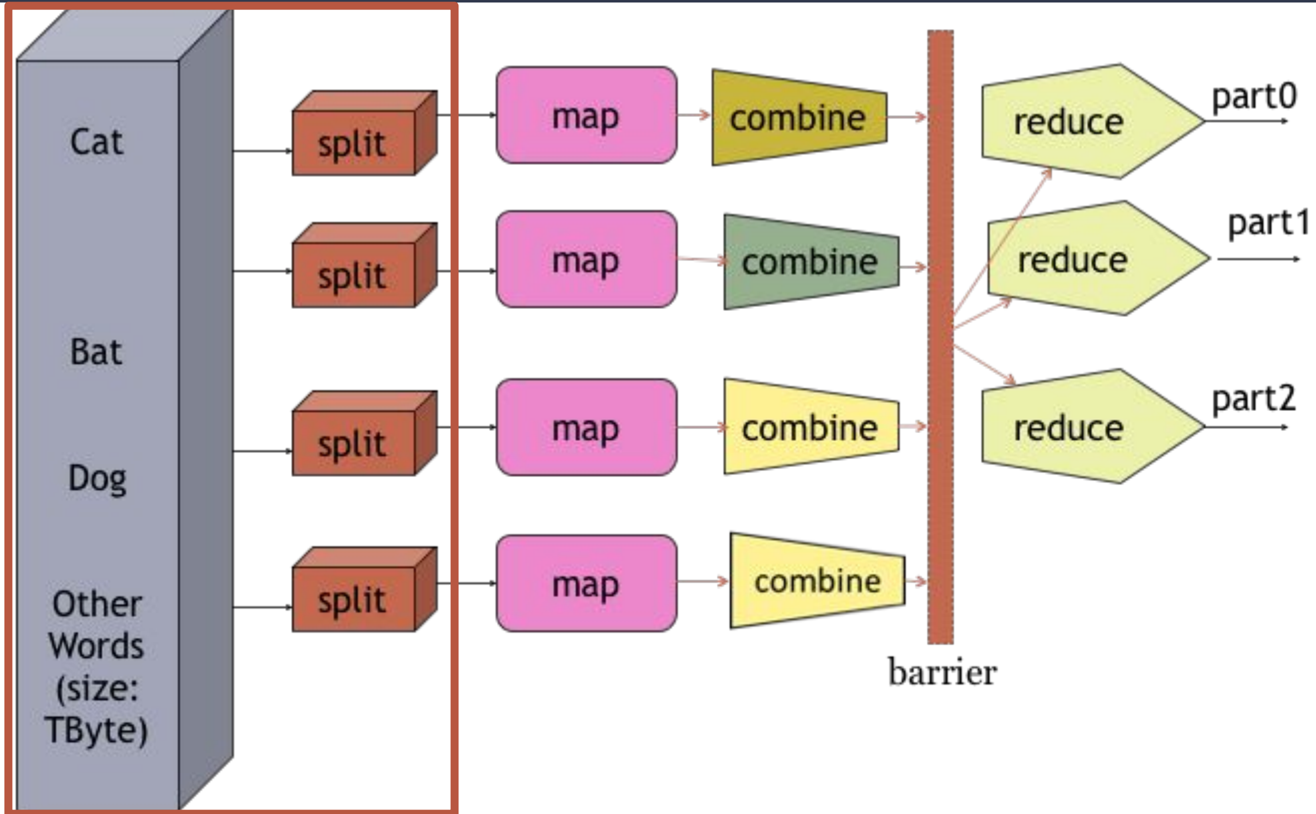
<cat 4>

…

<roof 6>

…

Step-by-Step Analysis: The Split



Step-by-Step Analysis: The Split

- The data that is input to MapReduce is divided into multiple "splits"
- This is a given based on the fact that we are running on Hadoop
  - Our large files are already automatically split into blocks
  - These blocks are then used as input to our MR program

Word Count Problem: Split

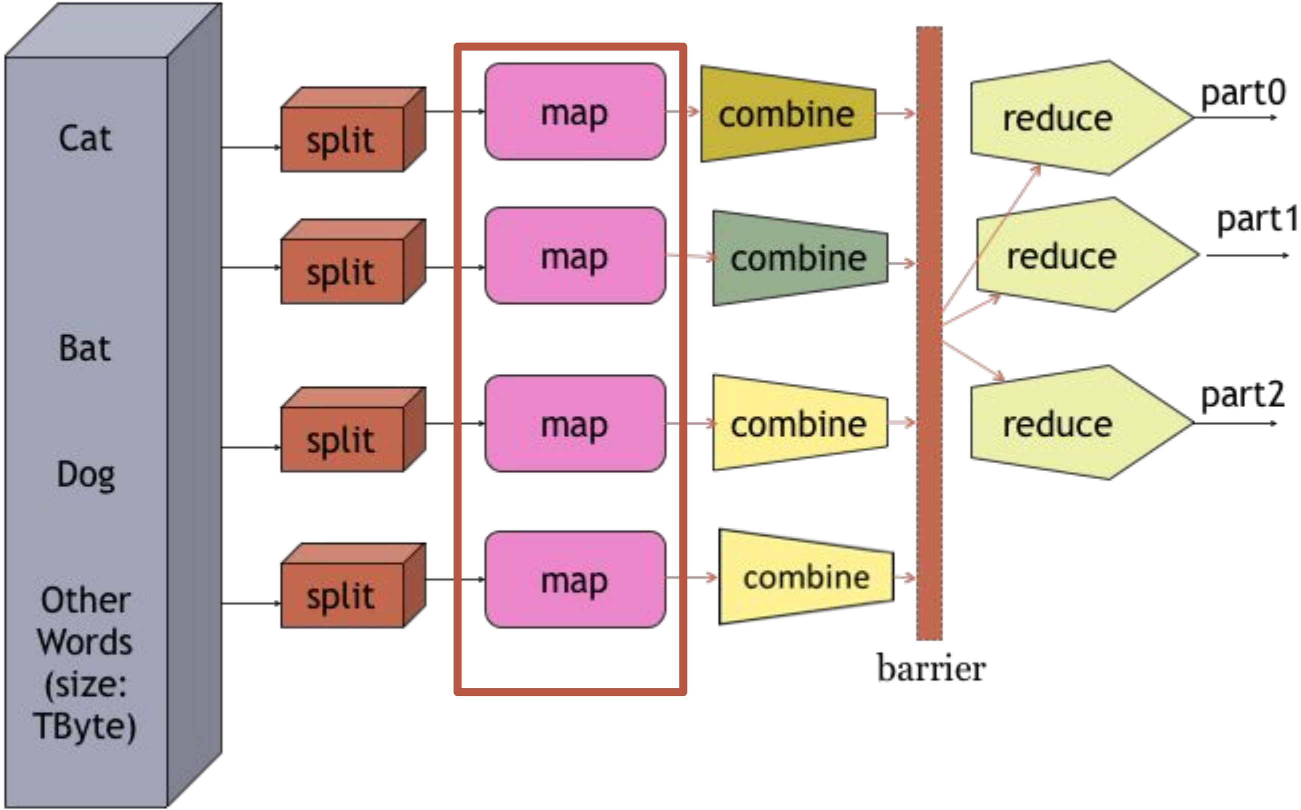
This is a cat
Cat sits on a roof

The roof is a tin roof
There is a tin can on the roof

Cat kicks the can
It rolls on the roof and falls on the next roof

The cat rolls too
It sits on the can

Step-by-Step Analysis: Mappers



Step-by-Step Analysis: Mappers

- Once we start up our MapReduce application, the first part of the computation is the mapping phase
- Mappers run local to their input data
  - They do not communicate with other mappers/across the network
  - Data is WORM so all of this work can be done in parallel
- They take the input data (whatever it happens to be) and maps it to a series of key-value pairs

Step-by-Step Analysis: Mappers

- The MapReduce framework will spawn one mapper per split
  - Usually this is one mapper per block
  - Mappers are expensive to startup – choosing granularity is important
    - Bigger blocks mean less parallelism, but more work per mapper/less overhead
- Output of the mapper step is series of <key, value> pairs

Keyspace

- The set of keys output by the mappers is referred to as the keyspace
- The size of the keyspace will have a direct effect on the number of reducers used, and the mapping of keys to reducers
- For word count, keyspace would be { set of words in our input }
  - Some applications may ignore some potential keys, ie stop words. This reduces the keyspace, because our mappers output <k,v> pairs for fewer keys

Word Count Problem: Mappers

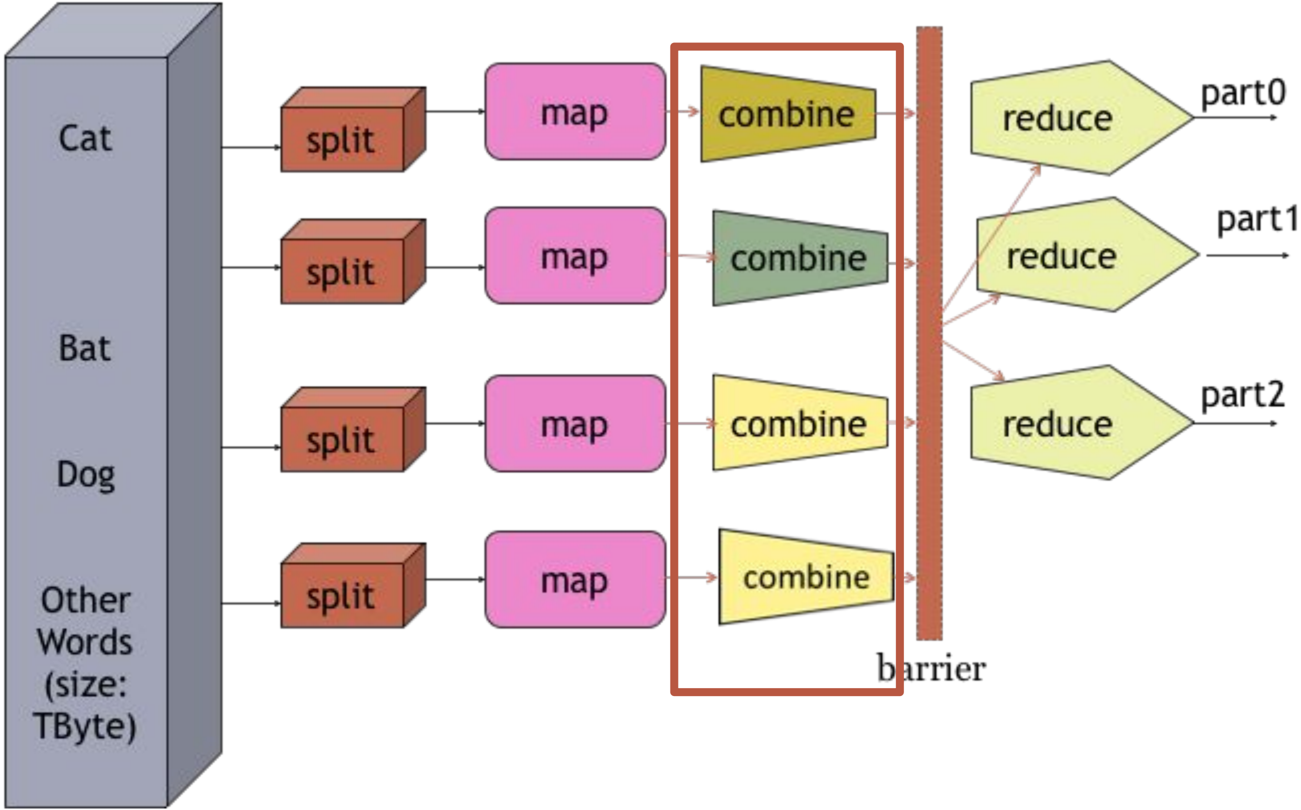
This is a cat
Cat sits on a roof
<this 1> <is 1> <a 1> <cat 1> <cat 1> <sits 1> <on 1> <a 1> <roof 1>

The roof is a tin roof
There is a tin can on the roof
<the 1> <roof 1> <is 1> <a 1> <tin 1> <roof 1> <there 1> <is 1> <a 1> <can 1> <on 1> <the 1> <roof 1>

Cat kicks the can
It rolls on the roof and falls on the next roof
<cat 1> <kicks 1> <the 1> <can 1> <it 1> <rolls 1> <on 1> <the 1> <roof 1> <and 1> <falls 1> <on 1> <the 1> <next 1> <roof 1>

The cat rolls too
It sits on the can
<the 1> <cat 1> <rolls 1> <too 1> <it 1> <sits 1> <on 1> <the 1> <can 1>

Step-by-Step Analysis: Combiners



Combiners

- Combiners are an optional optimization that can be used to reduce the amount of data output from the mapping step
  - Later we will look at this in more detail, as well as other ways to reduce intermediate data size
- There is one combiner per mapper, running on the same compute node as that mapper
  - No network communication required between Mapper and Combiner, or Combiner and other Combiners (this is a LOCAL optimization)

Combiners

- Combiners take the output of the local mapper, and do a partial reduction to reduce the number of <key,value> pairs
  - Often times the Combiner functionality is identical to the Reducer, but not always
- Combiners are strictly an optimization, **they cannot affect the semantics of your application**
  - There is no guarantee about when a combiner will be run, or even the number of times it is run, so your application cannot rely on it

Word Count Problem: Combiners

<this 1> <is 1> <a 1> <cat 1> <cat 1> <sits 1> <on 1> <a 1> <roof 1> → **<this 1> <is 1> <a 2> <cat 2> <sits 1> <on 1> <roof 1>**

<the 1> <roof 1> <is 1> <a 1> <tin 1> <roof 1> <there 1> <is 1> <a 1> <can 1> <on 1> <the 1> <roof 1> → **<the 2> <roof 3> <is 2> <a 2> <tin 1> <there 1> <can 1> <on 1>**

<cat 1> <kicks 1> <the 1> <can 1> <it 1> <rolls 1> <on 1> <the 1> <roof 1> <and 1> <falls 1> <on 1> <the 1> <next 1> <roof 1> → **<cat 1> <kicks 1> <the 3> <can 1> <it 1> <rolls 1> <on 2> <roof 2> <and 1> <falls 1> <next 1>**

<the 1> <cat 1> <rolls 1> <too 1> <it 1> <sits 1> <on 1> <the 1> <can 1> → **<the 2> <cat 1> <rolls 1> <too 1> <it 1> <sits 1> <on 1> <can 1>**

Word Count Problem: Combiners

<this 1> <is 1> <a 1> <cat 1> <cat 1> <sits 1> <on 1> <a 1> <roof 1> →
<this 1> <is 1> <a 2> <cat 2> <sits 1> <on 1> <roof 1>

<the 1> <roof 1> < … s 1> <a 1> <can
1> <on 1> <the 1> … **> <tin 1> <there**
1> <can 1> <on 1>

How does this Combiner affect the keyspace?

<cat 1> <kicks 1> <the 1> <can 1> <it 1> <rolls 1> <on 1> <the 1> <roof 1> <and 1> <falls 1> <on 1> <the 1> <next 1> <roof 1> → **<cat 1> <kicks 1> <the 3> <can 1> <it 1> <rolls 1> <on 2> <roof 2> <and 1> <falls 1> <next 1>**

<the 1> <cat 1> <rolls 1> <too 1> <it 1> <sits 1> <on 1> <the 1> <can 1> →
<the 2> <cat 1> <rolls 1> <too 1> <it 1> <sits 1> <on 1> <can 1>

Word Count Problem: Combiners

<this 1> <is 1> <a 1> <cat 1> <cat 1> <sits 1> <on 1> <a 1> <roof 1> →
<this 1> <is 1> <a 2> <cat 2> <sits 1> <on 1> <roof 1>

<the 1> <roof 1> < … s 1> <a 1> <can
1> <on 1> <the 1> … **> <tin 1> <there**
1> <can 1> <on 1

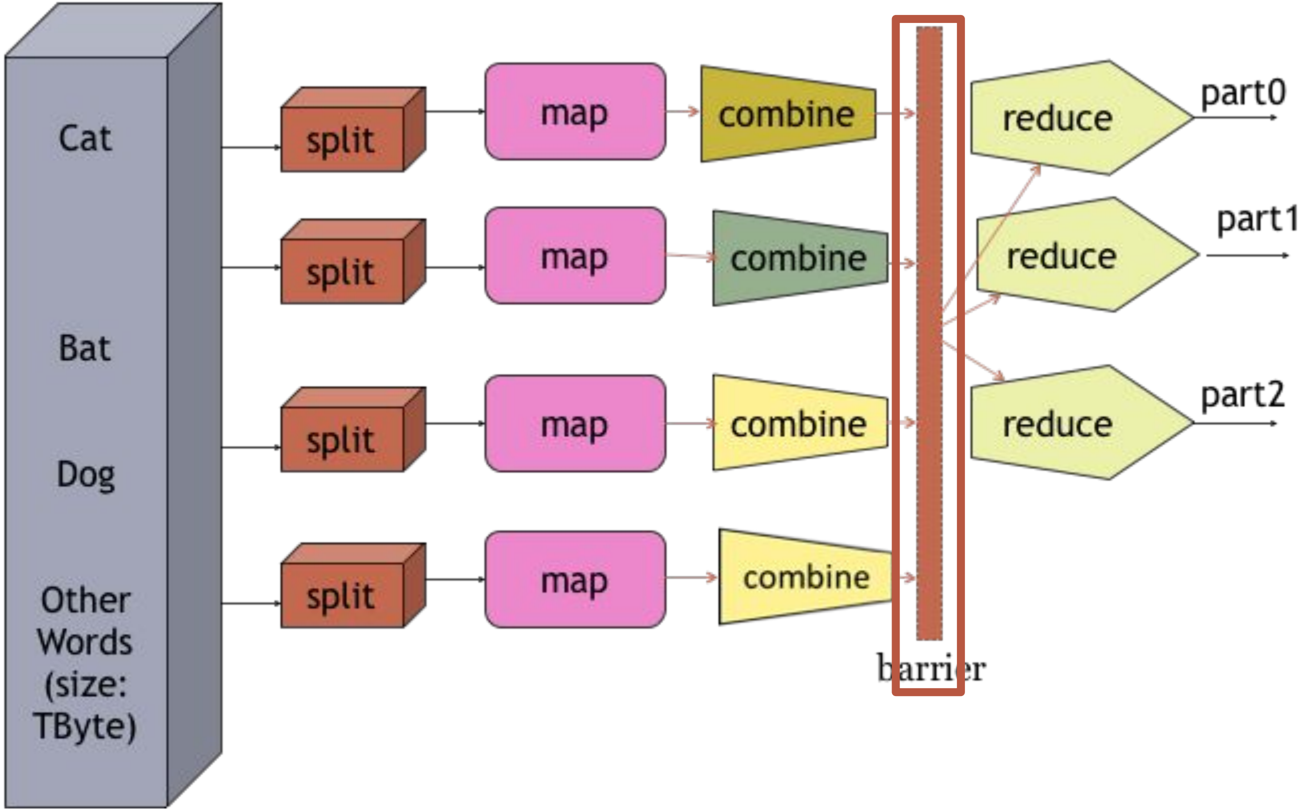
<cat 1> <kicks 1> … the 1> <roof 1>
<and 1> <falls 1> … **t 1> <kicks 1>**
<the 3> <can 1> < … falls 1> <next 1>

<the 1> <cat 1> <rolls 1> <too 1> <it 1> <sits 1> <on 1> <the 1> <can 1> →
<the 2> <cat 1> <rolls 1> <too 1> <it 1> <sits 1> <on 1> <can 1>

How does this Combiner affect the keyspace?

It doesn't. We still have the same exact set of keys, just fewer <key,value> pairs.

Step-by-Step Analysis: Shuffle and Sort



Step-by-Step Analysis: Shuffle and Sort

- Barrier between Map and Reduce
- Here is where the synchronization and communication occurs
  - All <key,value> pairs from the map step must be sorted and shuffled
  - This involves all-to-all communication
- How much data are we communicating and shuffling?
  - Depends on the output of the mappers…

Step-by-Step Analysis: Shuffle and Sort

How much data are we shuffling?

Let's assume the combiners are run on the entire output of their mappers, fully minimizing the number of key value pairs to the smallest number

If we have ***M*** mappers, and our keyspace contains ***K*** keys, then each mapper will have at most one <key,value> pair per key (due to combining), so we will have ***M*** x ***K*** <key, value> pairs to shuffle and sort

Without combining, this number is larger…AKA more communication required, and more data being transferred

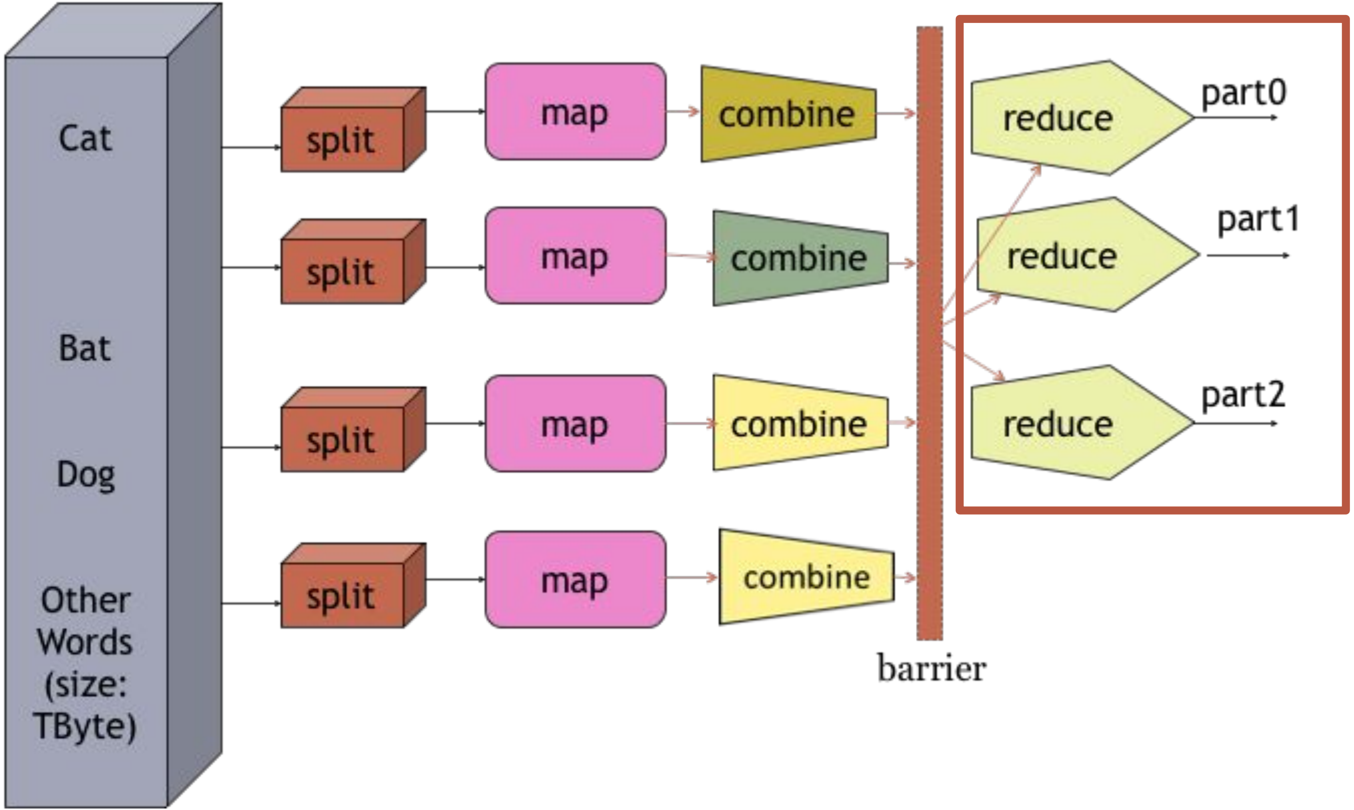
Word Count Problem: Shuffle to Reducers

Output of Mappers/Combiners:
<this 1> <is 1> <a 2> <cat 2> <sits 1> <on 1> <roof 1> <the 2> <roof 3> <is 2> <a 2> <tin 1> <there 1> <can 1> <on 1> <the 2> <roof 3> <is 2> <a 2> <tin 1> <there 1> <can 1> <on 1> <the 2> <cat 1> <rolls 1> <too 1> <it 1> <sits 1> <on 1> <can 1> <cat 1> <kicks 1> <the 3> <can 1> <it 1> <rolls 1> <on 2> <roof 2> <and 1> <falls 1> <next 1>

Input to the Reducers: delivered sorted, by key

…
<can <1,1,1>>
<cat <2,1,1,1>>
…
<roof <1,3,2>>
…

Step-by-Step Analysis: Reducers



Step-by-Step Analysis: Reducers

- The output of the shuffle and sort step is partitioned to the reducers
    - How many reducers you have is part of the Job configuration
    - How the keys are divided amongst reducers is decided by the Partitioner – this is also something you can configure to optimize your MR applications
- Each reducer will receive the keys one at a time, along with the list of values associated with that key, and emit a single <key, value> pair for that key

Word Count Problem: Reduce

Reduce (sum in this case) the values:

…

<can 2>

<cat 4>

…

<roof 6>

…

Word Count Problem: Reduce

Reduce (sum in this case) the values:

…

<can 2>

<cat 4>

…

<roof 6>

…

How many <key, value> pairs are in our output after the reduce step?

Word Count Problem: Reduce

Reduce (sum in this case) the values:

…

<can 2>

<cat 4>

…

<roof 6>

…

How many <key, value> pairs are in our output after the reduce step?

One per key in the keyspace.