# Apache Hadoop and MapReduce Part 2

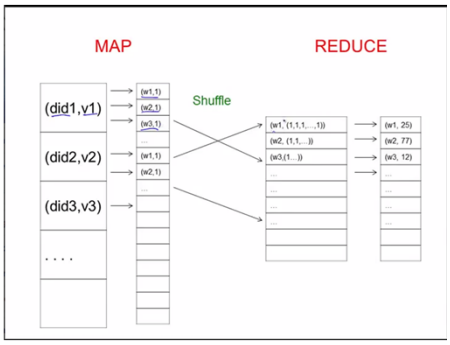



Figure 2.2 suggests this computation.

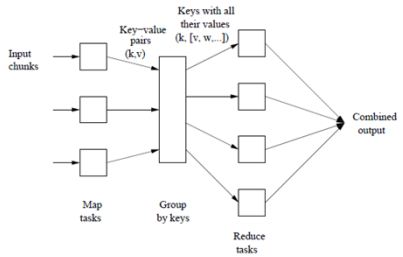



Figure 2.2: Schematic of a MapReduce computation

## Example

- Create an inverted index of words in tweets

DATA: <key=tweetID, value=tweetText>

Map?

<tweetID, tweetText> ➔ <word, tweetID> for each word

Reduce?

<word,[tweetID, tweetID, tweetID…]> ➔
<word,[tweetID, tweetID, tweetID…]>

## Example

- Word Length Histogram

Data: <DocID, line of text >

Task: create histogram of word lengths

Map?

< DocID, line of text > ➔ < length, 1 > for each word

Combiner?

< length,[1,1,1,1] > ➔ < length, sum of ones >

Reduce?

< length,[sum, sum, sum] > ➔ < length, sum of sums >

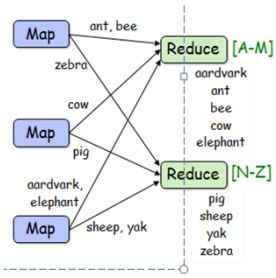
## Sorting?

- Can we sort using MapReduce?
- Yes – assuming that the shuffle/partitioning phase is done carefully.
  - Need a hash function that guarantees that
    if $key_i < key_j$ then $h(key_i) < h(key_j)$
  - -- Need to spread the sort key values uniformly over the reducers

- **Input:** (key, value) records
- **Output:** same records, sorted by key

- **Map:** identity function
- **Reduce:** identify function

- Pick partitioning function h such that $k_1<k_2 \Rightarrow h(k_1)<h(k_2)$




http://googleblog.blogspot.com/2008/11/sorting-1pb-with-mapreduce.html

**Sorting 1PB with MapReduce**

Posted: Friday, November 21, 2008

At Google we are fanatical about organizing the world's information. As a result, we spend a lot of time finding better ways to sort information using MapReduce, a key component of our software infrastructure that allows us to run multiple processes simultaneously. MapReduce is a perfect solution for many of the computations we run daily, due in large part to its simplicity, applicability to a wide range of real-world computing tasks, and natural translation to highly scalable distributed implementations that harness the power of thousands of computers.

In our sorting experiments we have followed the rules of a standard terabyte (TB) sort benchmark. Standardized experiments help us understand and compare the benefits of various technologies and also add a competitive spirit. You can think of it as an Olympic event for computations. By pushing the boundaries of these types of programs, we learn about the limitations of current technologies as well as the lessons useful in designing next generation computing platforms. This, in turn, should help everyone have faster access to higher-quality information.

We are excited to announce we were able to sort 1TB (stored on the Google File System as 10 billion 100-byte records in uncompressed text files) on 1,000 computers in 68 seconds. By comparison, the previous 1TB sorting record is 209 seconds on 910 computers.

Sometimes you need to sort more than a terabyte, so we were curious to find out what happens when you sort more and gave one petabyte (PB) a try. One petabyte is a thousand terabytes, or, to put this amount in perspective, it is 12 times the amount of archived web data in the U.S. Library of Congress as of May 2008. In comparison, consider that the aggregate size of data processed by all instances of MapReduce at Google was on average 20PB per day in January 2008.

It took six hours and two minutes to sort 1PB (10 trillion 100-byte records) on 4,000 computers. We're not aware of any other sorting experiment at this scale and are obviously very excited to be able to process so much data so quickly.

An interesting question came up while running experiments at such a scale: Where do you put 1PB of sorted data? We were writing it to 48,000 hard drives (we did not use the full capacity of these disks, though), and every time we ran our sort, at least one of our disks managed to break (this is not surprising at all given the duration of the test, the number of disks involved, and the expected lifetime of hard disks). To make sure we kept our sorted petabyte safe, we asked the Google File System to write three copies of each file to three different disks.

## Example

- Web logs

DATA: <userID, URL, timestamp, additional-info>
Task: Count number of accesses to each domain (domain is in URL)

Map?
< userID, URL, timestamp, additional-info > ➔
< domain, 1 > for each data element

Combiner?
< domain, [1, 1, 1,…] > ➔ < domain, count >

Reduce?
< domain, [count, count, count, …] > ➔
< domain, sum of counts >

## Example

- Web logs

DATA: <userID, URL, timestamp, additional-info>

Task: Count "value" of access to each web page; could be revenue generated (from additional-info)

Map?

< userID, URL, timestamp, additional-info > ➔
< URL, value >

Combiner?

< URL, [value, value, value,…] > ➔ < URL, sum of values >

Reduce?

< URL, [sum, sum, sum, …] > ➔
< URL, sum of sums >

## Example

- Web logs

DATA: <userID, URL, timestamp, additional-info>

Task: Count "value" of access to each web page by users who are age 18 to 24; assume revenue generated is in additional-info but user age is not

This calls for retrieving additional data based on the userID. In a relational database we would perform a join with some other table or data source. Can we do this within the MR paradigm?

## Joins and MapReduce

- Consider newT = R(A,B) |X| S(B,C)

Map?

< R, (a,b) > ➔ < b, (R, a) >

Or

< S, (b,c) > ➔ < b, (S, c ) >

Reduce?

< b-value, [ (R,a),(S,c),…(S,c),…(R,a),…]> ➔
< newT, [(a,b,c),(a,b,c),(a,b,c) … ] > "the join tuples"

NOTE: These key-value pairs could be the input for another MapReduce cycle.

Questions?

## Social Networks

For each individual, how many people are they following?

| Input | | Desired Output |
|---|---|---|
| Jim, Sue<br>Sue, Jim<br>Lin, Joe<br>Joe, Lin<br>Jim, Kai<br>Kai, Jim<br>Jim, Lin<br>Lin, Jim | REDUCE | Jim, 3<br>Lin, 2<br>Sue, 1<br>Kai, 1<br>Joe, 1 |

Alternately, what if we just want, for each person, a list of the people they follow?

… including the people that those people follow

… and the people those people follow

… and the people those people follow

…

## Sparse Matrix Multiplication

- Probably the single most commonly executed operation using MapReduce techniques.
- Several implementations, some involve one map-reduce pass, some require a map-reduce-map-reduce sequence
- We'll look at one single-pass technique today
- (on the board)