Worksheet #7 solution:
Associativity and Commutativity

Recap:
A binary function f is **associative** if \( f(f(x,y),z) = f(x,f(y,z)) \).
A binary function f is **commutative** if \( f(x,y) = f(y,x) \).

**Worksheet problems:**
1) Claim: a Finish Accumulator (FA) can only be used with operators that are **associative and commutative**. Why? What can go wrong with accumulators if the operator is non-associative or non-commutative?
You may get different answers in different executions if the operator is non-associative or non-commutative e.g., an accumulator can be implemented using one “partial accumulator” per processor core.
2) For each of the following functions, indicate if it is associative and/or commutative.
   a) \( f(x,y) = x+y \), for integers \( x, y \), is associative and commutative
   b) \( g(x,y) = (x+y)/2 \), for integers \( x, y \), is commutative but not associative
      ⇒ Incorrect answers found in some worksheets: Associative / Both / Neither
   c) \( h(s1,s2) = \text{concat}(s1, s2) \) for strings \( s1, s2 \), e.g., \( h(“ab”,”cd”) = “abcd” \), is associative but not commutative
      ⇒ Incorrect answers found in some worksheets: Commutative / Neither
Streaming data requirements have skyrocketed

- AT&T processes roughly 30 petabytes per day through its telecommunications network
- Google processed roughly 24 petabytes per day in 2009
- Facebook, Amazon, Twitter, etc., have comparable throughputs
- Two Sigma maintains over 100 teraflops of private computing power, continuously computing over 11 petabytes of quantitative data
- In comparison, the IBM Watson knowledge base stored roughly 4 terabytes of data when winning at Jeopardy

Parallelism enables processing of big data

- Continuously streaming data needs to be processed at least as fast as it is accumulated, or we will never catch up
- The bottleneck in processing very large data sets is dominated by the speed of disk access
- More processors accessing more disks enables faster processing
Parallelism enables “Cloud Computing” as a Utility

- Offers computing, storage, communication at pennies per hour
  - *Leverage Parallelism to Achieve Energy-Efficient High Performance*
- No premium to scale:
  
  \[
  \begin{align*}
  1000 \text{ computers} @ & 1 \text{ hour} \\
  = & 1 \text{ computer} @ 1000 \text{ hours}
  \end{align*}
  \]
- Illusion of infinite scalability to cloud user
  - As many computers as you can afford
- Leading examples: Amazon Web Services (AWS), Google App Engine, Microsoft Azure
  - Economies of scale pushed down datacenter costs by factors of 3-8X
  - Traditional data centers utilized 10% - 20%
  - Make profit offering pay-as-you-go use service at less than your costs for as many computers as you need
  - Strategic capability for company’s needs
- Challenge: portable and scalable parallelism at cloud scale
  - One solution: leverage functional programming with MapReduce pattern

---

MapReduce Pattern

- **Apply** Map function \( f \) to user supplied record of key-value pairs
- Compute set of intermediate key/value pairs
- **Apply** Reduce operation \( g \) to all values that share same key to combine derived data properly
  - Often produces smaller set of values
- User supplies Map and Reduce operations in functional model so that the system can parallelize them, and also re-execute them for fault tolerance
MapReduce: The Map Step

Input set of key-value pairs

Flattened intermediate set of key-value pairs

map

map

... 

Intermediate key-value pairs

Key-value groups

Output key-value pairs

group

reduce

reduce

reduce

... 

... 

... 

Map Reduce: Summary

- Input set is of the form \{(k_1, v_1), \ldots (k_n, v_n)\}, where \( (k_i, v_i) \) consists of a key, \( k_i \), and a value, \( v_i \).
  - Assume that the key and value objects are immutable, and that equality comparison is well defined on all key objects.
- Map function \( f \) generates sets of intermediate key-value pairs, \( f(k_i, v_i) = \{(k_1', v_1'), \ldots (k_m', v_m')\} \). The \( k_j' \) keys can be different from \( k_i \) key in the in of the map function.
  - Assume that a flatten operation is performed as a post-pass after the map operations, so as to avoid dealing with a set of sets.
- Reduce operation groups together intermediate key-value pairs, \( \{(k', v_i')\} \) with the same \( k' \), and generates a reduced key-value pair, \( (k', v'') \), for each such \( k' \), using reduce function \( g \).

Google Uses MapReduce For …

- **Web crawl**: Find outgoing links from HTML documents, aggregate by target document
- **Google Search**: Generating inverted index files using a compression scheme
- **Google Earth**: Stitching overlapping satellite images to remove seams and to select high-quality imagery
- **Google Maps**: Processing all road segments on Earth and render map tile images that display segments
- More than 10,000 MR programs at Google in 4 years, run 100,000 MR jobs per day (2008)
MapReduce: State of Practice

• **Apache Hadoop** now dominates use of the MapReduce framework

• Often, Hadoop map and reduce functions are *no longer written directly*
  
  • *Instead, a user writes a query in a very high level language and uses another tool to compile the query into map/reduce functions!*
    
    • **Hive** (another Apache project) compiles SQL queries into map/reduce
    
    • **Pig** (yet another Apache project) compiles direct relational algebra into map/reduce

MapReduce: State of Practice

• Eventually, users started realizing that a much larger class of algorithms could be expressed as an iterative sequence of map/reduce operations
  
  • Many machine learning algorithms fall into this category

• Tools started to emerge to enable easy expression of multiple map/reduce operations, along with smart scheduling

• **Apache Spark**: General purpose functional programming over a cluster
  
  • Caches results of map/reduce operations in memory so they can be used on subsequent iterations without accessing disk each time
  
  • Tends to be 10-100 times faster than Hadoop for many applications
MapReduce Execution

Fine granularity tasks: many more map tasks than machines

Bucket sort to get same keys together

2000 servers => ~ 200,000 Map Tasks, ~ 5,000 Reduce tasks

WordCount example

In: set of words
Out: set of (word, count) pairs

Algorithm:
1. For each in word W, emit (W, 1) as a key-value pair (map step).
2. Group together all key-value pairs with the same key (reduce step).
3. Perform a sum reduction on all values with the same key (reduce step).

- All map operations in step 1 can execute in parallel with only local data accesses
- Step 2 may involve a major reshuffle of data as all key-value pairs with the same key are grouped together.
- Step 3 performs a standard reduction algorithm for all values with the same key, and in parallel for different keys.
PseudoCode for WordCount

1. `<String, Integer>` map(String inKey, String inValue):
2.   // inKey: document name
3.   // inValue: document contents
4.   for each word w in inValue:
5.     emitIntermediate(w, 1) // Produce count of words
6. 
7. `<Integer>` reduce(String outKey, Iterator<Integer> values):
8.   // outKey: a word
9.   // values: a list of counts
10.  Integer result = 0
11.  for each v in values:
12.    result += v // the value from map was an integer
13.  emit(result)

Example Execution of WordCount Program

Distribute

that that is is that that is not is not that it it is
Map 1 Map 2 Map 3 Map 4
is 1, that 2 is 1, that 2 is 2, not 2 is 2, it 2, that 1

Shuffle

is 1,1,2,2 it 2
that 2,2,1 not 2
Reduce 1 Reduce 2
is 6; it 2 not 2; that 5

Collect

is 6; it 2; not 2; that 5