Introduction to MapReduce

Today's Topics

- Functional programming
- MapReduce
- Distributed file system

Functional Programming

- MapReduce = functional programming meets distributed processing on steroids
  - Not a new idea… dates back to the 50's (or even 30's)
- What is functional programming?
  - Computation as application of functions
  - Theoretical foundation provided by lambda calculus
- How is it different?
  - Traditional notions of "data" and "instructions" are not applicable
  - Data flows are implicit in program
  - Different orders of execution are possible
- Exemplified by LISP and ML

Overview of Lisp

- Lisp ≠ Lost In Silly Parentheses
- We'll focus on particular a dialect: "Scheme"
- Lists are primitive data types
- ```
  '(1 2 3 4 5)
  '((a 1) (b 2) (c 3))
  ```
- Functions written in prefix notation

```
(+ 1 2) → 3
(* 3 4) → 12
(sqrt (+ (* 3 3) (* 4 4))) → 5
(define x 3) → x
(* x 5) → 15
```
```

Functions

- Functions = lambda expressions bound to variables
-
```
(define foo
  (lambda (x y)
  — (sqrt (+ (* x x) (* y y)))))
  —
```
- Syntactic sugar for defining functions
  - Above expressions is equivalent to:

```
(define (foo x y)
 _(sqrt (+ (* x x) (* y y))))
 —
```

- Once defined, function can be applied:
- `(foo 3 4)` → `5`

Other Features

- In Scheme, everything is an s-expression
  - No distinction between “data” and “code”
  - Easy to write self-modifying code
- Higher-order functions
  - Functions that take other functions as arguments

```
(define (bar f x) (f (f x)))
```

Doesn’t matter what *f* is, just apply it twice.

```
(define (baz x) (* x x))
(bar baz 2) → 16
```

Recursion is your friend

- Simple factorial example

```
(define (factorial n)
  (if (= n 1)
      1
      (* n (factorial (- n 1)))))
(factorial 6) → 720
```

- Even iteration is written with recursive calls!

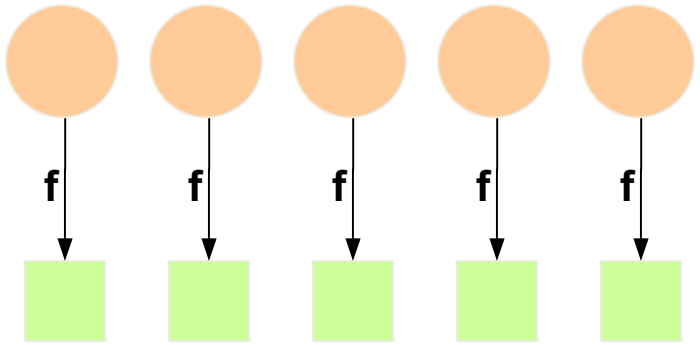
```
(define (factorial-iter n)
  (define (aux n top product)
    (if (= n top)
        (* n product)
        (aux (+ n 1) top (* n product))))
  (aux 1 n 1))
(factorial-iter 6) → 720
```

Lisp → MapReduce?

- What does this have to do with MapReduce?
- After all, Lisp is about processing *lists*
- Two important concepts in functional programming
  - Map: do something to everything in a list
  - Fold: combine results of a list in some way

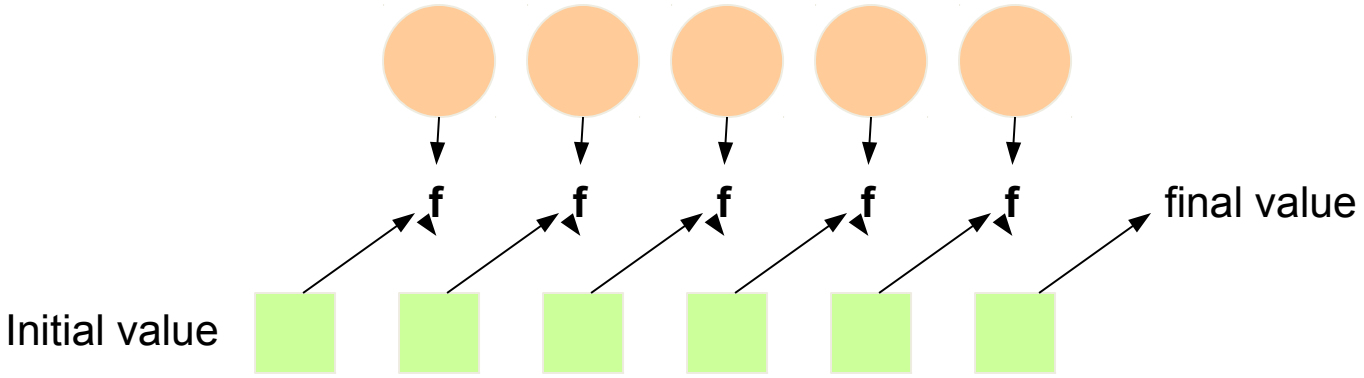
Map

- Map is a higher-order function
- How map works:
  - Function is applied to every element in a list
  - Result is a new list



Fold

- Fold is also a higher-order function
- How fold works:
  - Accumulator set to initial value
  - Function applied to list element and the accumulator
  - Result stored in the accumulator
  - Repeated for every item in the list
  - Result is the final value in the accumulator



Map/Fold in Action

- Simple map example:

```
(map (lambda (x) (* x x))
     '(1 2 3 4 5))
 → '(1 4 9 16 25)
```

- Fold examples:

```
(fold + 0 '(1 2 3 4 5)) → 15
(fold * 1 '(1 2 3 4 5)) → 120
```

- Sum of squares:

```
(define (sum-of-squares v)
  (fold + 0 (map (lambda (x) (* x x)) v)))
(sum-of-squares '(1 2 3 4 5)) → 55
```

Lisp → MapReduce

- Let's assume a long list of records: imagine if...
  - We can parallelize map operations
  - We have a mechanism for bringing map results back together in the fold operation
- That's MapReduce! (and Hadoop)
- Observations:
  - No limit to map parallelization since maps are indepedent
  - We can reorder folding if the fold function is commutative and associative

Typical Problem

- Iterate over a large number of records
- Extract something of interest from each
- Shuffle and sort intermediate results
- Aggregate intermediate results
- Generate final output

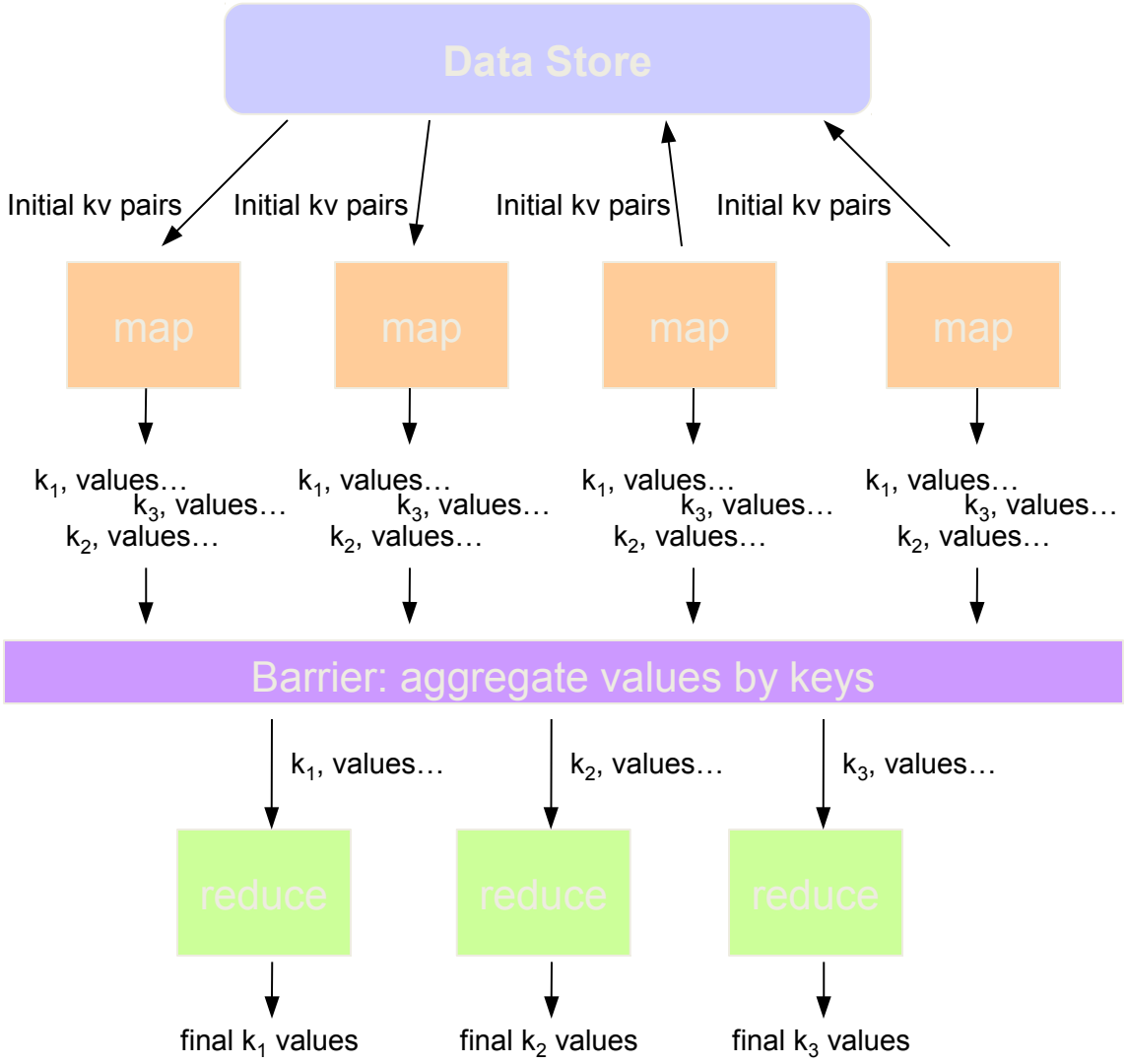
Map

Reduce

MapReduce

- Programmers specify two functions:
  map (k, v) → <k', v'>*
  reduce (k', v') → <k', v'>*
  - All v' with the same k' are reduced together
- Usually, programmers also specify:
  partition (k', number of partitions) → partition for k'
  - Often a simple hash of the key, e.g. hash(k') mod n
  - Allows reduce operations for different keys in parallel
- Implementations:
  - Google has a proprietary implementation in C++
  - Hadoop is an open source implementation in Java (lead by Yahoo)

It’s just divide and conquer!



Recall these problems?

- How do we assign work units to workers?
- What if we have more work units than workers?
- What if workers need to share partial results?
- How do we aggregate partial results?
- How do we know all the workers have finished?
- What if workers die?