

CS 347:  
Distributed Databases and  
Transaction Processing

**Distributed  
Data Processing using MapReduce**

Hector Garcia-Molina  
Zoltan Gyongyi

CS 347 MapReduce 1

**Motivation: Building a Text Index**

Web page stream → LOADING → TOKENIZING → SORTING → FLUSHING → Intermediate runs → Disk

CS 347 MapReduce 2

**Motivation: Building a Text Index**

Intermediate runs → MERGE → Final index

CS 347 MapReduce 3

**Generalization: MapReduce**

Web page stream → LOADING → MAP → TOKENIZING → SORTING → FLUSHING → Intermediate runs → Disk

CS 347 MapReduce 4

**Generalization: MapReduce**

Intermediate runs → MERGE → REDUCE → Final index

CS 347 MapReduce 5

**MapReduce**

{ ... } = set  
{ { ... } } = bag

- Input
  - $R = \{ \{ r_1, r_2, \dots, r_n \} \}$
  - Functions  $M, R$
  - $M(r_i) \rightarrow \{ \{ [k_1, v_1], [k_2, v_2], \dots \} \}$
  - $R(k_i, \text{value bag}) \rightarrow \text{"new" value for } k_i$
- Let
  - $S = \{ \{ [k, v] \mid [k, v] \in M(r) \text{ for some } r \in R \} \}$
  - $K = \{ k \mid [k, v] \in S, \text{ for any } v \}$
  - $G(k) = \{ \{ v \mid [k, v] \in S \}$
- Output
  - $O = \{ [k, t] \mid k \in K, t = R(k, G(k)) \}$

CS 347 MapReduce 6

## Example: Counting Word Occurrences

- Map(string *key*, string *value*):  

```
// key is the document ID
// value is the document body
for each word w in value:
    EmitIntermediate(w, "1")
```
- Example: Map("29", "cat dog cat bat dog") emits [cat: 1], [dog: 1], [cat: 1], [bat: 1], [dog: 1]
- Why does Map() have two parameters?

CS 347

MapReduce

7

## Example: Counting Word Occurrences

- Reduce(string *key*, string iterator *values*):  

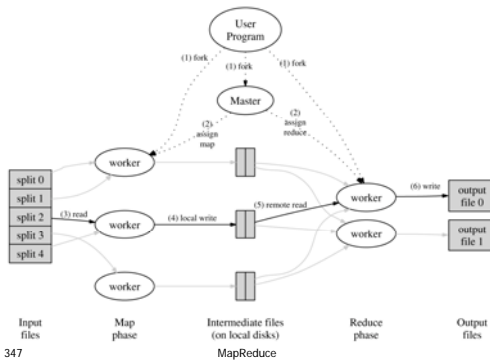
```
// key is a word
// values is a list of counts
int result = 0
for each value v in values:
    result += ParseInteger(v)
EmitFinal(ToString(result))
```
- Example: Reduce("dog", ["1", "1", "1", "1"]) emits "4"

CS 347

MapReduce

8

## Google MapReduce Architecture



CS 347

MapReduce

9

## Implementation Issues

- File system
- Data partitioning
- Combine functions
- Result ordering
- Failure handling
- Backup tasks

CS 347

MapReduce

10

## File system

- All data transfer between workers occurs through distributed file system
  - Support for split files
  - Workers perform local writes
  - Each **map** worker performs *local or remote read of one or more* input splits
  - Each **reduce** worker performs *remote read of multiple* intermediate splits
  - Output is left in as many splits as reduce workers

CS 347

MapReduce

11

## Data partitioning

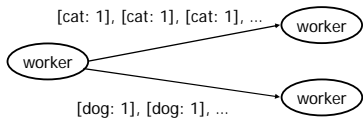
- Data partitioned (split) by hash on key
- Each worker responsible for certain hash bucket(s)
- How many workers/splits?
  - Best to have multiple splits per worker
    - Improves load balance
    - If worker fails, splits could be re-distributed across multiple other workers
  - Best to assign splits to "nearby" nearby
  - Rules apply to both map and reduce workers

CS 347

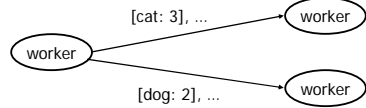
MapReduce

12

## Combine functions



Combine is like a local reduce applied (at map worker) before storing/distributing intermediate results:



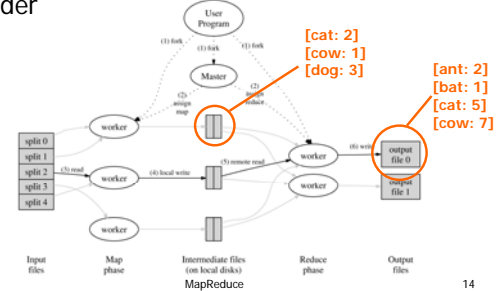
CS 347

MapReduce

13

## Result ordering

- Results produced by workers are in key order

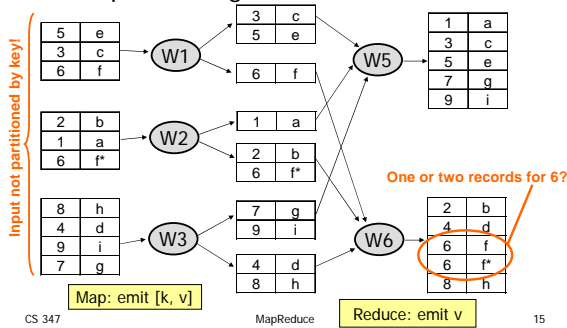


CS 347

14

## Result ordering

- Example: sorting records



CS 347

MapReduce

15

## Failure handling

- Worker failure
  - Detected by master through periodic pings
  - Handled via re-execution
    - Redo in-progress or completed map tasks
    - Redo in-progress reduce tasks
    - Map/reduce tasks committed through master
- Master failure
  - Not covered in original implementation
  - Could be detected by user program or monitor
  - Could recover persistent state from disk

CS 347

MapReduce

16

## Backup tasks

- Straggler: worker that takes unusually long to finish task
  - Possible causes include bad disks, network issues, overloaded machines
- Near the end of the map/reduce phase, master spawns backup copies of remaining tasks
  - Use workers that completed their task already
  - Whichever finishes first "wins"

CS 347

MapReduce

17

## Other Issues

- Handling bad records
  - Best is to debug and fix data/code
  - If master detects at least 2 task failures for a particular input record, skips record during 3<sup>rd</sup> attempt
- Debugging
  - Tricky in a distributed environment
  - Done through log messages and counters

CS 347

MapReduce

18

## MapReduce Advantages

- Easy to use
- General enough for expressing many practical problems
- Hides parallelization and fault recovery details
- Scales well, way beyond thousands of machines and terabytes of data

CS 347

MapReduce

19

## MapReduce Disadvantages

- One-input two-phase data flow rigid, hard to adapt
  - Does not allow for stateful multiple-step processing of records
- Procedural programming model requires (often repetitive) code for even the simplest operations (e.g., projection, filtering)
- Opaque nature of the map and reduce functions impedes optimization

CS 347

MapReduce

20

## Questions

- Could MapReduce be made more declarative?
- Could we perform joins?
- Could we perform grouping?
  - As done through GROUP BY in SQL

CS 347

MapReduce

21

## Pig & Pig Latin

- Layer on top of MapReduce (Hadoop)
  - Hadoop is an open-source implementation of MapReduce
  - Pig is the system
  - Pig Latin is the language, a hybrid between
    - A high-level declarative query language, such as SQL
    - A low-level procedural language, such as C++/Java/Python typically used to define Map() and Reduce()

CS 347

MapReduce

22

## Example: Average score per category

- Input table: pages(url, category, score)
- Problem: find, for each sufficiently large category, the average score of high-score web pages in that category
- SQL solution:

```
SELECT category, AVG(score)
FROM pages
WHERE score > 0.5
GROUP BY category HAVING COUNT(*) > 1M
```

CS 347

MapReduce

23

## Example: Average score per category

- SQL solution:

```
SELECT category, AVG(score)
FROM pages
WHERE score > 0.5
GROUP BY category HAVING COUNT(*) > 1M
```
- Pig Latin solution:

```
topPages = FILTER pages BY score > 0.5;
groups = GROUP topPages BY category;
largeGroups = FILTER groups
              BY COUNT(topPages) > 1M;
output = FOREACH largeGroups
          GENERATE category, AVG(topPages.score);
```

CS 347

MapReduce

24

### Example: Average score per category

topPages = FILTER pages BY score > 0.5;

pages:  
url, category, score

|            |      |     |
|------------|------|-----|
| z.cnn.com  | .com | 0.9 |
| y.yale.edu | .edu | 0.5 |
| w.uc.edu   | .edu | 0.1 |
| x.nyt.com  | .com | 0.8 |
| y.ut.edu   | .edu | 0.6 |
| w.wh.gov   | .gov | 0.7 |



topPages:  
url, category, score

|            |      |     |
|------------|------|-----|
| z.cnn.com  | .com | 0.9 |
| y.yale.edu | .edu | 0.5 |
| x.nyt.com  | .com | 0.8 |
| y.ut.edu   | .edu | 0.6 |
| w.wh.gov   | .gov | 0.7 |

CS 347

MapReduce

25

### Example: Average score per category

groups = GROUP topPages BY category;

topPages:  
url, category, score

|            |      |     |
|------------|------|-----|
| z.cnn.com  | .com | 0.9 |
| y.yale.edu | .edu | 0.5 |
| x.nyt.com  | .com | 0.8 |
| y.ut.edu   | .edu | 0.6 |
| w.wh.gov   | .gov | 0.7 |



groups:  
category, topPages

|      |   |
|------|---|
| .com | {{z.cnn.com .com 0.9} {x.nyt.com .com 0.8}} |
| .edu | {{y.yale.edu .edu 0.5} {y.ut.edu .edu 0.6}} |
| .gov | {{w.wh.gov .gov 0.7}}                       |

CS 347

MapReduce

26

### Example: Average score per category

largeGroups = FILTER groups BY COUNT(topPages) > 1;

groups:  
category, topPages

|      |   |
|------|---|
| .com | {{z.cnn.com .com 0.9} {x.nyt.com .com 0.8}} |
| .edu | {{y.yale.edu .edu 0.5} {y.ut.edu .edu 0.6}} |
| .gov | {{w.wh.gov .gov 0.7}}                       |



largeGroups:  
category, topPages

|      |   |
|------|---|
| .com | {{z.cnn.com .com 0.9} {x.nyt.com .com 0.8}} |
| .edu | {{y.yale.edu .edu 0.5} {y.ut.edu .edu 0.6}} |

CS 347

MapReduce

27

### Example: Average score per category

output = FOREACH largeGroups GENERATE category, AVG(topPages.score);

largeGroups:  
category, topPages

|      |   |
|------|---|
| .com | {{z.cnn.com .com 0.9} {x.nyt.com .com 0.8}} |
| .edu | {{y.yale.edu .edu 0.5} {y.ut.edu .edu 0.6}} |



output:  
category, topPages

|      |      |
|------|------|
| .com | 0.85 |
| .edu | 0.55 |

CS 347

MapReduce

28

## Pig (Latin) Features

- Similar to specifying a query execution plan (i.e., data flow graph)
  - Makes it easier for programmers to understand and control execution
- Flexible, fully nested data model
- Ability to operate over input files without schema information
- Debugging environment

CS 347

MapReduce

29

## Execution control: good or bad?

- Example:
  - spamPages = FILTER pages BY isSpam(url);
  - culpritPages = FILTER spamPages BY score > 0.8;
- Should system reorder filters?
  - Depends on selectivity

CS 347

MapReduce

30

## Data model

- **Atom**, e.g., 'alice'
- **Tuple**, e.g., ('alice', 'lakers')
- **Bag**, e.g.,  
{ ('alice', 'lakers') ('alice', ('iPod', 'apple')) }
- **Map**, e.g.,  
[ 'fan of' → { ('lakers') ('iPod') }  
'age' → 20 ]

CS 347

MapReduce

31

## Expressions

$t = ('alice', \{ ('lakers', 1), ('iPod', 2) \}, ['age' \rightarrow 20])$

Let fields of tuple  $t$  be called  $f1$ ,  $f2$ ,  $f3$

| Expression Type        | Example                  | Value for $t$                |
|------------------------|--------------------------|------------------------------|
| Constant               | 'bob'                    | Independent of $t$           |
| Field by position      | $f0$                     | 'alice'                      |
| Field by name          | $f3$                     | 'age' → 20                   |
| Projection             | $f2.\$0$                 | { ('lakers')<br>( 'iPod' ) } |
| Map Lookup             | $f3\#\text{'age'}$       | 20                           |
| Function Evaluation    | $SUM(f2.\$1)$            | $1 + 2 = 3$                  |
| Conditional Expression | $f3\#\text{'age'} > 18?$ | 'adult'                      |
| Flattening             | $FLATTEN(f2)$            | 'lakers', 1<br>'iPod', 2     |

CS 347

MapReduce

32

## Reading input

```
queries = LOAD 'query_log.txt'
          USING myLoad()
          AS (userId, queryString, timestamp);
```

Annotations: 'input file' points to 'query\_log.txt'; 'handle' points to 'queries'; 'custom deserializer' points to 'myLoad()'; 'schema' points to '(userId, queryString, timestamp)'; 'handle' also points to 'AS'.

CS 347

MapReduce

33

## For each

```
expandedQueries =
  FOREACH queries
  GENERATE userId, expandQuery(queryString);
```

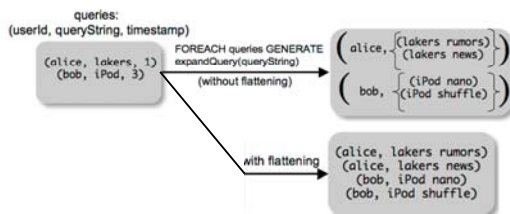
- Each tuple is processed independently ← good for parallelism
- Can flatten output to remove one level of nesting:  
 $expandedQueries = FOREACH queries$   
 $GENERATE userId,$   
 $FLATTEN(expandQuery(queryString));$

CS 347

MapReduce

34

## For each



CS 347

MapReduce

35

## Flattening example

| x  | a                          | b            | c |
|----|----------------------------|--------------|---|
| a1 | {(b1, b2), (b3, b4), (b5)} | {(c1), (c2)} |   |
| a2 | {(b6, (b7, b8))}           | {(c3), (c4)} |   |

$y = FOREACH x GENERATE a, FLATTEN(b), c;$

|  |  |  |
|--|--|--|
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |

CS 347

MapReduce

36

## Flattening example

| <i>x</i> | <i>a</i>                 | <i>b</i> | <i>c</i>    |
|----------|--------------------------|----------|-------------|
| a1       | {(b1, b2) (b3, b4) (b5)} |          | {(c1) (c2)} |
| a2       | {(b6, (b7,b8))}          |          | {(c3) (c4)} |

$y = \text{FOREACH } x \text{ GENERATE } a, \text{ FLATTEN}(b), c;$

|    |    |          |             |
|----|----|----------|-------------|
| a1 | b1 | b2       | {(c1) (c2)} |
| a1 | b3 | b4       | {(c1) (c2)} |
| a1 | b5 |          | {(c1) (c2)} |
| a2 | b6 | (b7, b8) | {(c3) (c4)} |

CS 347

MapReduce

37

## Flattening example

- Also flattening *c* (in addition to *b*) yields:
  - (a1, b1, b2, c1)
  - (a1, b1, b2, c2)
  - (a1, b3, b4, c1)
  - (a1, b3, b4, c2)
  - ...

CS 347

MapReduce

38

## Filter

$\text{realQueries} = \text{FILTER queries BY userId NEQ 'bot'}$ ;

$\text{realQueries} = \text{FILTER queries BY NOT isBot(userId)}$ ;

CS 347

MapReduce

39

## Co-group

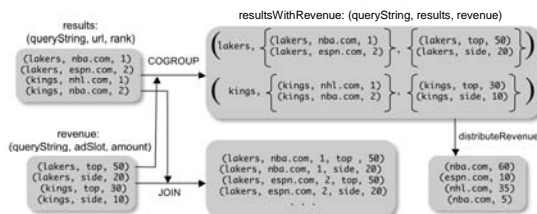
- Two input tables:
  - $\text{results}(\text{queryString}, \text{url}, \text{position})$
  - $\text{revenue}(\text{queryString}, \text{adSlot}, \text{amount})$
- $\text{resultsWithRevenue} = \text{COGROUP results BY queryString, revenue BY queryString}$ ;
- $\text{revenues} = \text{FOREACH resultsWithRevenue GENERATE FLATTEN}(\text{distributeRevenue}(\text{results}, \text{revenue}))$ ;
- More flexible than SQL joins

CS 347

MapReduce

40

## Co-group



CS 347

MapReduce

41

## Group

- Simplified co-group (single input)
  - $\text{groupedRevenue} = \text{GROUP revenue BY queryString}$ ;
  - $\text{queryRevenues} = \text{FOREACH groupedRevenue GENERATE queryString, SUM}(\text{revenue.amount}) \text{ AS total}$ ;

CS 347

MapReduce

42

### Co-group example 1

| <i>x</i> | <i>a</i> | <i>b</i> | <i>c</i> |
|----------|----------|----------|----------|
| 1        | 1        | 1        | c1       |
| 1        | 1        | 1        | c2       |
| 2        | 2        | 2        | c3       |
| 2        | 2        | 2        | c4       |

| <i>y</i> | <i>a</i> | <i>b</i> | <i>d</i> |
|----------|----------|----------|----------|
| 1        | 1        | 1        | d1       |
| 1        | 2        | 2        | d2       |
| 2        | 2        | 1        | d3       |
| 2        | 2        | 2        | d4       |

$s = \text{GROUP } x \text{ BY } a;$

| <i>s</i> | <i>a</i> | <i>x</i> |
|----------|----------|----------|
|          |          |          |
|          |          |          |

CS 347

MapReduce

43

### Co-group example 1

| <i>x</i> | <i>a</i> | <i>b</i> | <i>c</i> |
|----------|----------|----------|----------|
| 1        | 1        | 1        | c1       |
| 1        | 1        | 1        | c2       |
| 2        | 2        | 2        | c3       |
| 2        | 2        | 2        | c4       |

| <i>y</i> | <i>a</i> | <i>b</i> | <i>d</i> |
|----------|----------|----------|----------|
| 1        | 1        | 1        | d1       |
| 1        | 2        | 2        | d2       |
| 2        | 2        | 1        | d3       |
| 2        | 2        | 2        | d4       |

$s = \text{GROUP } x \text{ BY } a;$

| <i>s</i> | <i>a</i> | <i>x</i>                |
|----------|----------|-------------------------|
| 1        |          | {(1, 1, c1) (1, 1, c2)} |
| 2        |          | {(2, 2, c3) (2, 2, c4)} |

CS 347

MapReduce

44

### Co-group example 2

| <i>x</i> | <i>a</i> | <i>b</i> | <i>c</i> |
|----------|----------|----------|----------|
| 1        | 1        | 1        | c1       |
| 1        | 1        | 1        | c2       |
| 2        | 2        | 2        | c3       |
| 2        | 2        | 2        | c4       |

| <i>y</i> | <i>a</i> | <i>b</i> | <i>d</i> |
|----------|----------|----------|----------|
| 1        | 1        | 1        | d1       |
| 1        | 2        | 2        | d2       |
| 2        | 2        | 1        | d3       |
| 2        | 2        | 2        | d4       |

$t = \text{GROUP } x \text{ BY } (a, b);$

| <i>t</i> | <i>a/b</i> | <i>x</i> |
|----------|------------|----------|
|          |            |          |
|          |            |          |

CS 347

MapReduce

45

### Co-group example 2

| <i>x</i> | <i>a</i> | <i>b</i> | <i>c</i> |
|----------|----------|----------|----------|
| 1        | 1        | 1        | c1       |
| 1        | 1        | 1        | c2       |
| 2        | 2        | 2        | c3       |
| 2        | 2        | 2        | c4       |

| <i>y</i> | <i>a</i> | <i>b</i> | <i>d</i> |
|----------|----------|----------|----------|
| 1        | 1        | 1        | d1       |
| 1        | 2        | 2        | d2       |
| 2        | 2        | 1        | d3       |
| 2        | 2        | 2        | d4       |

$t = \text{GROUP } x \text{ BY } (a, b);$

| <i>t</i> | <i>a/b</i> | <i>x</i>                |
|----------|------------|-------------------------|
| (1, 1)   |            | {(1, 1, c1) (1, 1, c2)} |
| (2, 2)   |            | {(2, 2, c3) (2, 2, c4)} |

CS 347

MapReduce

46

### Co-group example 3

| <i>x</i> | <i>a</i> | <i>b</i> | <i>c</i> |
|----------|----------|----------|----------|
| 1        | 1        | 1        | c1       |
| 1        | 1        | 1        | c2       |
| 2        | 2        | 2        | c3       |
| 2        | 2        | 2        | c4       |

| <i>y</i> | <i>a</i> | <i>b</i> | <i>d</i> |
|----------|----------|----------|----------|
| 1        | 1        | 1        | d1       |
| 1        | 2        | 2        | d2       |
| 2        | 2        | 1        | d3       |
| 2        | 2        | 2        | d4       |

$u = \text{COGROUP } x \text{ BY } a, y \text{ BY } a;$

| <i>u</i> | <i>a</i> | <i>x</i> | <i>y</i> |
|----------|----------|----------|----------|
|          |          |          |          |
|          |          |          |          |

CS 347

MapReduce

47

### Co-group example 3

| <i>x</i> | <i>a</i> | <i>b</i> | <i>c</i> |
|----------|----------|----------|----------|
| 1        | 1        | 1        | c1       |
| 1        | 1        | 1        | c2       |
| 2        | 2        | 2        | c3       |
| 2        | 2        | 2        | c4       |

| <i>y</i> | <i>a</i> | <i>b</i> | <i>d</i> |
|----------|----------|----------|----------|
| 1        | 1        | 1        | d1       |
| 1        | 2        | 2        | d2       |
| 2        | 2        | 1        | d3       |
| 2        | 2        | 2        | d4       |

$u = \text{COGROUP } x \text{ BY } a, y \text{ BY } a;$

| <i>u</i> | <i>a</i> | <i>x</i>                | <i>y</i>                |
|----------|----------|-------------------------|-------------------------|
| 1        |          | {(1, 1, c1) (1, 1, c2)} | {(1, 1, d1) (1, 2, d2)} |
| 2        |          | {(2, 2, c3) (2, 2, c4)} | {(2, 1, d3) (2, 2, d4)} |

CS 347

MapReduce

48

### Co-group example 4

| <i>x</i> | <i>a</i> | <i>b</i> | <i>c</i> |
|----------|----------|----------|----------|
| 1        | 1        | 1        | c1       |
| 1        | 1        | 1        | c2       |
| 2        | 2        | 2        | c3       |
| 2        | 2        | 2        | c4       |

| <i>y</i> | <i>a</i> | <i>b</i> | <i>d</i> |
|----------|----------|----------|----------|
| 1        | 1        | 1        | d1       |
| 1        | 2        | 1        | d2       |
| 2        | 2        | 1        | d3       |
| 2        | 2        | 2        | d4       |

$v = \text{COGROUP } x \text{ BY } a, y \text{ BY } b;$

| <i>v</i> | <i>a/b</i> | <i>x</i> | <i>y</i> |
|----------|------------|----------|----------|
|          |            |          |          |
|          |            |          |          |

CS 347

MapReduce

49

### Co-group example 4

| <i>x</i> | <i>a</i> | <i>b</i> | <i>c</i> |
|----------|----------|----------|----------|
| 1        | 1        | 1        | c1       |
| 1        | 1        | 1        | c2       |
| 2        | 2        | 2        | c3       |
| 2        | 2        | 2        | c4       |

| <i>y</i> | <i>a</i> | <i>b</i> | <i>d</i> |
|----------|----------|----------|----------|
| 1        | 1        | 1        | d1       |
| 1        | 2        | 1        | d2       |
| 2        | 2        | 1        | d3       |
| 2        | 2        | 2        | d4       |

$v = \text{COGROUP } x \text{ BY } a, y \text{ BY } b;$

| <i>v</i> | <i>a/b</i> | <i>x</i>                | <i>y</i>                |
|----------|------------|-------------------------|-------------------------|
| 1        |            | {(1, 1, c1) (1, 1, c2)} | {(1, 1, d1) (2, 1, d3)} |
| 2        |            | {(2, 2, c3) (2, 2, c4)} | {(1, 2, d2) (2, 2, d4)} |

CS 347

MapReduce

50

### Join

- Syntax:  
`joinedResults = JOIN results BY queryString,  
revenue BY queryString;`
- Shorthand for:  
`temp = COGROUP results BY queryString,  
revenue BY queryString;  
joinedResults = FOREACH temp GENERATE  
FLATTEN(results), FLATTEN(revenue);`

CS 347

MapReduce

51

### MapReduce in Pig Latin

```
mapResult = FOREACH input GENERATE
FLATTEN(map(*));
keyGroups = GROUP mapResult BY $0;
output = FOREACH keyGroups
GENERATE reduce(*);
```

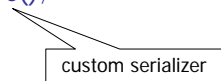
CS 347

MapReduce

52

### Storing output

```
STORE queryRevenues INTO 'output.txt'
USING myStore();
```



CS 347

MapReduce

53

### Pig on Top of MapReduce

- Pig Latin program can be “compiled” into a sequence of mapreductions
- Load, for each, filter: can be implemented as map functions
- Group, store: can be implemented as reduce functions (given proper intermediate data)
- Cogroup and join: special map functions that handle multiple inputs split using the same hash function
- Depending on sequence of operations, include identity mapper and reducer phases as needed

CS 347

MapReduce

54

## References

- MapReduce: Simplified Data Processing on Large Clusters (Dean and Ghemawat)  
<http://labs.google.com/papers/mapreduce.html>
- Pig Latin: A Not-so-foreign Language for Data Processing (Olston *et al.*)  
<http://wiki.apache.org/pig/>
- Interpreting the Data: Parallel Analysis with Sawzall (Pike *et al.*)  
<http://labs.google.com/papers/sawzall.html>

Another MapReduce wrapper

CS 347

MapReduce

55

## Summary

- MapReduce
  - Two phases: map and reduce
  - Transparent distribution, fault tolerance, and scaling
- Pig and Pig Latin
  - Semi-declarative layer on top of MapReduce
  - Programs expressed as sequences of simple SQL-like “queries”

CS 347

MapReduce

56