MapReduce

February 21, 2019
Data Science CSCI 1951A
Brown University
Instructor: Ellie Pavlick
HTAs: Wennie Zhang, Maulik Dang, Gurnaaz Kaur
Announcements

• Next assignment released today—due March 7
• First project checkin will be assigned next week, so this will be happening…in parallel (← that was a hilarious pun, FYI)
• The HTAs are complaintless
• Questions? Concerns? Anything?
Announcements

• Next assignment released today—due March 7
Announcements

• Next assignment released today—due March 7
  • First project checkin will be assigned next week, so this will be happening…in parallel (← that was a hilarious pun, FYI)
Announcements

• Next assignment released today—due March 7

  • First project checkin will be assigned next week, so this will be happening…in parallel (← that was a hilarious pun, FYI)

• The HTAs are complaintless 👍
Announcements

• Next assignment released today—due March 7

  • First project checkin will be assigned next week, so this will be happening…in parallel (← that was a hilarious pun, FYI)

• The HTAs are complaintless 👍

• Questions? Concerns? Anything?
Today

I have no idea what I'm doing

but I'm doing it at scale
MapReduce
MapReduce

• Functional-programming paradigm (inspired by LISP and friends)

https://research.google.com/archive/mapreduce-osdi04-slides
MapReduce

- Functional-programming paradigm (inspired by LISP and friends)

- Two functions:
MapReduce

• Functional-programming paradigm (inspired by LISP and friends)

• Two functions:
  
  • Map: (in_key, in_value) -> list_of(out_key, intermediate_value)

https://research.google.com/archive/mapreduce-osdi04-slides
MapReduce

- Functional-programming paradigm (inspired by LISP and friends)

- Two functions:
  - Map: (in_key, in_value) -> list_of(out_key, intermediate_value)
  - Reduce: (out_key, list_of(intermediate_value)) -> list_of(out_value)

https://research.google.com/archive/mapreduce-osdi04-slides
MapReduce

- Functional-programming paradigm (inspired by LISP and friends)

- Two functions:
  - Map: (in_key, in_value) -> list_of(out_key, intermediate_value)
  - Reduce: (out_key, list_of(intermediate_value)) -> list_of(out_value)

https://research.google.com/archive/mapreduce-osdi04-slides
MapReduce

distributed grep
distributed sort
web link-graph reversal
web access log stats
inverted index construction
document clustering
machine learning
statistical machine translation
...

https://research.google.com/archive/mapreduce-osdi04-slides
Map Reduce

- One “master” scheduler which assigns tasks (mapping or reducing) to machines
Map Reduce

- One “master” scheduler which assigns tasks (mapping or reducing) to machines

- No shared state between machines—massively parallelizable
Map Reduce

• One “master” scheduler which assigns tasks (mapping or reducing) to machines

• No shared state between machines—massively parallelizable

• Assume very high failure rates on workers
Counting Words

Documents

- hello world
- oh hi there world
- why hello there , world
- world ! how the hell are ya ?
Counting Words

Documents

- hello world
- oh hi there world
- why hello there, world
- world! how the hell are ya?

Counts for each word

- hello 2
- world 4
- oh 1
- hi 1
- there 2
- why 1
- ! 1
- how 1
- ...
hello world

oh hi there world

why hello there, world

world! how the hell are ya?
hello world
oh hi there world
why hello there, world
world! how the hell are ya?

Mapper 1
Mapper 2
Mapper 3
Mapper 4
hello world
oh hi there world
why hello there, world
world! how the hell are ya?

Mapper 1
Mapper 2
Mapper 3
Mapper 4

Reducer 1
Reducer 2
Reducer 3
Reducer 4
Reducer 5
hello world

oh hi there, world

why hello there, world

world! how the hell are ya?

Mapper 1

Mapper 2

Mapper 3

Mapper 4

Reducer 1

Reducer 2

Reducer 3

Reducer 4

Reducer 5

Input

26
hello world

oh hi there, world

why hello there, world

world! how the hell are ya?

Mapper 1

Mapper 2

Mapper 3

Mapper 4

Reducer 1

Reducer 2

Reducer 3

Reducer 4

Reducer 5

Input

Map Phase

Reducer 1

Reducer 2

Reducer 3

Reducer 4

Reducer 5
Input

Map Phase

Shuffle Phase ("Group By")

Reducer 1 (hello, 2)
Reducer 2 (world, 4)
Reducer 3 (oh, 28)
Reducer 4 (hi, 1)
Reducer 5 (there, 2)
Input

Mapper 1
- (hello, 1)
- (world, 1)

Mapper 2
- (oh, 1)
- (hi, 1)

Mapper 3
- (why, 1)
- (hello, 1)
- (there, 1)

Mapper 4
- (world, 1)
- (!, 1)
- (how, 1)
- (the, 1)
- (hell, 1)
- (are, 1)
- (ya, 1)

Map Phase

Shuffle Phase ("Group By")

Reducer 1
- (hello, 2)

Reducer 2
- (world, 4)

Reducer 3
- (oh, 1)

Reducer 4
- (hi, 1)

Reducer 5
- (there, 2)
hello world
oh hi there world
why hello there, world
world! how the hell are ya?

Mapper 1
Mapper 2
Mapper 3
Mapper 4

Reducer 1
Reducer 2
Reducer 3
Reducer 4
Reducer 5

(hello, 1)
(world, 1)
(oh, 1)
(hi, 1)
(there, 1)
(world, 1)
(why, 1)
(hello, 1)
(there, 1)
(,, 1)
(world, 1)
(world, 1)
(!, 1)
(how, 1)
(the, 1)
(hell, 1)
(are, 1)
(ya, 1)
(hello, 1)
(hello, 1)
(world, 1)
(world, 1)
(world, 1)
(world, 1)
(oh, 1)
(hi, 1)
(there, 1)
(there, 1)

(hello, 2)
(world, 4)
(oh, 1)
(hi, 1)
(there, 1)
hello world

oh hi there world

why hello there, world

world! how the hell are ya?

Mapper 1

Mapper 2

Mapper 3

Mapper 4

Reducer 1

Reducer 2

Reducer 3

Reducer 4

Reducer 5

(hello, 1)

(world, 1)

(oh, 1)

(hi, 1)

(there, 1)

(world, 1)

(why, 1)

(hello, 1)

(there, 1)

(,, 1)

(world, 1)

(world, 1)

(oh, 1)

(hi, 1)

(there, 1)

(there, 1)

(hi, 1)

(there, 1)
<table>
<thead>
<tr>
<th>Mapper 1</th>
<th>Mapper 2</th>
<th>Mapper 3</th>
<th>Mapper 4</th>
<th>Mapper 5</th>
<th>Mapper 6</th>
<th>Mapper 7</th>
<th>Mapper 7</th>
</tr>
</thead>
<tbody>
<tr>
<td>(hello, 1)</td>
<td>(oh, 1)</td>
<td>(there, 1)</td>
<td>(why, 1)</td>
<td>(there, 1)</td>
<td>(world, 1)</td>
<td>(the, 1)</td>
<td>(ya, 1)</td>
</tr>
<tr>
<td>(world, 1)</td>
<td>(hi, 1)</td>
<td>(world, 1)</td>
<td>(hello, 1)</td>
<td>(there, 1)</td>
<td>(world, 1)</td>
<td>(hel, 1)</td>
<td>(are, 1)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>(,, 1)</td>
<td>(!, 1)</td>
<td>(how, 1)</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Reducer 1</th>
<th>Reducer 2</th>
<th>Reducer 3</th>
<th>Reducer 4</th>
<th>Reducer 5</th>
</tr>
</thead>
<tbody>
<tr>
<td>(hello, 2)</td>
<td>(world, 4)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

|                   |                   |                   | (hi, 1)           | (there, 1)        |
|                   |                   |                   |                   |                   |
|                   |                   |                   | (oh, 1)           |                   |
|                   |                   |                   |                   |                   |
|                   |                   |                   |                   | (there, 2)        |
Map Reduce

//define your mapper function(s)
def MapFn: (String, String) -> (String, Int) { 
  TODO;
}

//define your reduce function(s)
def ReduceFn: (String, Int) -> (String, Int) {
  TODO;
}

//define your pipeline
Table<String, String> table = read(table_path)
Table<String, Int> output =
  table.MapFn().ReduceFn();
write(output)
Map Reduce

// define your mapper function(s)
def MapFn: (String, String) -> (String, Int)
{
    TODO;
}

// define your reduce function(s)
def ReduceFn: (String, Int) -> (String, Int)
{
    TODO;
}

// define your pipeline
Table<String, String> table = read(table_path)
Table<String, Int> output =
    table.MapFn().ReduceFn();
write(output)
//define your mapper function(s)
def MapFn: (String, String) -> (String, Int) {
    TODO;
}

//define your reduce function(s)
def ReduceFn: (String, Int) -> (String, Int) {
    TODO;
}

//define your pipeline
Table<String, String> table = read(table_path)
Table<String, Int> output =
    table.MapFn().ReduceFn();
write(output)
Map Reduce

//define your mapper function(s)
def MapFn: (String, String) -> (String, Int) { TODO; }

//define your reduce function(s)
def ReduceFn:(String, Int) -> (String, Int){ TODO; }

//define your pipeline
Table<String, String> table = read(table_path)
Table<String, Int> output =
    table.MapFn().ReduceFn();
write(output)
Map Reduce

// enumerate occurrences of each word, with count of 1
def MapFn: (String, String) -> (String, Int) {
    for w in input.value().split()
        emit(w, 1);

}
// enumerate occurrences of each word, with count of 1
def MapFn: (String, String) -> (String, Int) {
    for w in input.value().split(){
        emit(w, 1);
    }
}
Map Reduce

// sum the total counts of each word
def ReduceFn:(String, Int) -> (String, Int) {
    sum = 0;
    for c in input.value() {
        sum += c;
    }
    emit(input.key(), sum);
}
Map Reduce

// sum the total counts of each word
def ReduceFn:(String, Int) -> (String, Int) {
  sum = 0;
  for c in input.value() {
    sum += c;
  }
  emit(input.key(), sum);
}

list of ints (counts)
// sum the total counts of each word
def ReduceFn:(String, Int) -> (String, Int){
    sum = 0;
    for c in input.value()
        sum += c;
    }
    emit(input.key(), sum);
}
Find the number of occurrences of each word?

// enumerate occurrences of each word
// with count of 1
def MapFn: (String, String) -> (String, Int) {
    for w in input.split(){
        emit(w, 1);
    }
}

// sum the total counts of each word
// sum = 0;
// for (w, c) in input{ sum += c; } emit(w, sum);

// define your pipeline
// define your pipeline
def main() {
    Table<String, String> table = read(table_path)
    Table<String, Int> output =
        table.MapFn().ReduceFn();
    write(output)
}
(non)Clicker Question!

Find the number of unique documents that each word occurs in?
// enumerate occurrences of each word
// with count of 1
def MapFn1: String -> (String, Int) {
    ???
}

// sum the total counts of each word
// define your pipeline
// sum the total counts of each word
def ReduceFn1: String -> (String, Int) {
    ???
}
def ReduceFn2:(String, Int) -> (String, Int){
    ???
}
def main() {
    Table<String, String> table = read(table_path)
    Table<String, Int> output =
        table.MapFn1().ReduceFn1().ReduceFn2();
    write(output)
}
(non)Clicker Question!

Find the *number of unique documents* that each word occurs in?

```java
// enumerate occurrences of each word
// with count of 1
def MapFn1: String -> (String, Int) {
    ???
}
def ReduceFn1: String -> (String, Int) {
    ???
}
// sum the total counts of each word
def ReduceFn2:(String, Int) -> (String, Int) {
    ???
}
// define your pipeline
def main() {
    Table<String, String> table = read(table_path)
    Table<String, Int> output =
        table.MapFn1().ReduceFn1().ReduceFn2();
    write(output)
}
```
(non)Clicker Question!

Find the *number of unique documents* that each word occurs in?

```
// enumerate occurrences of each word
// with count of 1
def MapFn1: String -> (String, Int) {
    ???
}
def ReduceFn1: String -> (String, Int) {
    ???
}
// sum the total counts of each word
def ReduceFn2:(String, Int) -> (String, Int){
    ???
}
// define your pipeline
def main() {
    Table<String, String> table = read(table_path)
    Table<String, Int> output =
        table.MapFn1().ReduceFn1().ReduceFn2();
    write(output)
}
```
D1
hello world,
just saying
hello

D2
oh hi, hi
there world

D3
why hello
there, world

D4
world! how
the hell are
ya???
hello world, just saying hello

oh hi, hi there world

why hello there, world

world! how the hell are ya???

((D1, hello), 1)
((D1, world), 1)
...
((D1, hello), 1)

....

....

....

((D4, world), 1)
...
((D4, ?), 1)
((D4, ?), 1)
hello world,
just saying hello

oh hi, hi there world

why hello there, world

world! how the hell are ya???

((D1, hello), 1)
((D1, world), 1)
...
((D1, hello), 1)

((D2, world), 1)

((D3, ?), 1)
((D3, ?), 1)

((D4, world), 1)

((D4, ?), 1)
((D4, ?), 1)
hello world,
just saying
hello

D1

Mapper

((D1, hello), 1)
((D1, world), 1)
((D1, hello), 1)

Reducer 1

(hello, 1)

oh hi, hi
there world

D2

Mapper

((D1, world), 1)

Reducer 2

(world, 1)

why hello
there ,
world

D3

Mapper

((D4, world), 1)

Reducer 3

(world, 1)

world ! how
the hell are
ya ?? ?

D4

Mapper

((D4, world), 1)

Reducer 4

(world, 1)

(?, 1)
hello world,
just saying
hello

oh hi, hi
there world

why hello
there, world

world! how
the hell are
ya???

Mapper

Mapper

Mapper

Mapper

Reducer 1

Reducer 2

Reducer 3

Reducer 4

((D1, hello), 1)

((D1, world), 1)

... 

((D1, hello), 1)

Reducer 1

Reducer 2

Reducer 3

Reducer 4

(hello, 1)

(world, 1)

(world, 1)

(?, 1)
hello world,
just saying
hello

oh hi, hi
there world

why hello
there, world

world! how
the hell are
ya???

Mapper

Reducer 1
((D1, hello), 1)
((D1, world), 1)
... 
((D1, hello), 1)

Reducer 2

Reducer 3

Reducer 4
((D4, world), 1)
((D4, ?), 1)
((D4, ?), 1)

Reducer 1
(hello, 1)

Reducer 2
(world, 1)

Reducer 3
(world, 1)

Reducer 4
(?, 1)

Reducer 1
(hello, 2)

Reducer 2
(world, 4)

Reducer 3
(?, 1)
// enumerate occurrences of each word
// with count of 1
def MapFn1: String -> (String, Int) {
    ???
}
def ReduceFn1: String -> (String, Int) {
    ???
}
// sum the total counts of each word
def ReduceFn2:(String, Int) -> (String, Int){
    ???
}
// define your pipeline
def main() {
Table<String, String> table = read(table_path)
Table<String, Int> output =
    table.MapFn1().ReduceFn1().ReduceFn2();
write(output)
}
// enumerate occurrences of each word
// with count of 1
def MapFn1: (String, String) -> ((String, String), Int) {
    for w in input.value().split() {
        emit((input.key(), w), 1)
    }
}
def ReduceFn1: ((String, String), Int) -> (String, Int) {
    emit(input.key()[1], 1)
}
// sum the total counts of each word
def ReduceFn2: (String, Int) -> (String, Int) {
    sum = 0;
    for (w, c) in input {
        sum += c;
    }
    emit(w, sum);
}
// define your pipeline
def main() {
    Table<String, String> table = read(table_path)
    Table<String, Int> output =
        table.MapFn1().MapFn2().ReduceFn();
    write(output)
}
// enumerate occurrences of each word
// with count of 1
def MapFn1: (String, String) -> ((String, String), Int) {
    for w in input.value().split(){
        emit((input.key(), w), 1)
    }
}

def ReduceFn1: ((String, String), Int) -> (String, Int) {
    emit(input.key()[1], 1)
}

// sum the total counts of each word
def ReduceFn2:(String, Int) -> (String, Int){
    sum = 0;
    for (w, c) in input{ sum += c; }  
    emit(w, sum);
}

// define your pipeline
def main() {
    Table<String, String> table = read(table_path)
    Table<String, Int> output =
        table.MapFn1().MapFn2().ReduceFn();
    write(output)
}
Find the *number of unique documents* that each word occurs in?

```python
// enumerate occurrences
// of each word with count of 1
def MapFn1: {
    for w in input.value().split():
        emit((input.key(), w), 1)
}
def ReduceFn1: {
    emit(input.key()[1], 1)
}
// sum the total counts
// of each word
def ReduceFn2: {
    sum = 0;
    for (w, c) in input:
        sum += c;
    emit(w, sum);
}
```
Find the *number of unique documents* that each word occurs in?

```python
// enumerate occurrences
// of each word with count of 1
def MapFn1: {
    for w in input.value().split()
        emit((input.key(), w), 1)
}

def ReduceFn1: {
    emit(input.key()[1], 1)
}

// sum the total counts
// of each word
def ReduceFn2:
    sum = 0;
    for (w, c) in input
        sum += c;
    emit(w, sum);
}
```

```python
// enumerate occurrences
// of each word with count of 1
def MapFn1: {
    for w in input.value().split()
        emit(input.key(), w)
}

def ReduceFn1: {
    emit(input.key(), w)
}

def ReduceFn2: {
    sum = 0;
    for (w, c) in input
        sum += c;
    emit(w, sum);
}
```
(non)Clicker Question!

Find the number of unique documents that each word occurs in?

```python
// enumerate occurrences
// of each word with count of 1
def MapFn1: {
    for w in input.value().split(){
        emit((input.key(), w), 1)
    }
}
def ReduceFn1: {
    emit(input.key()[1], 1)
}
// sum the total counts
// of each word
def ReduceFn2:{
    sum = 0;
    for (w, c) in input{ sum += c; }
    emit(w, sum);
}
```

```python
// enumerate occurrences
// of each word with count of 1
def MapFn1: {
    for w in input.value().split(){
        emit(input.key(), w)
    }
}
def ReduceFn1: {
    emit(input.key()[1], 1)
}
// sum the total counts
// of each word
def ReduceFn2:{
    sum = 0;
    for (w, c) in input{ sum += c; }
    emit(w, sum);
}
```

Do these produce the same output?

(a) Yes  (b) No
Clicker Question!

Input K: V
Doc1: here are some words
Doc2: words words words
Doc3: here are words

```
def MapFn1: (S, S) -> (S, S) {
    for w in input.value().split()
        emit(input.key(), w)
}
```

```
def ReduceFn1: (S, S) -> (S, I) {
    for w in input.value()
        emit(w, 1)
}
```

```
def ReduceFn2: (S, I) -> (S, I) {
    sum = 0;
    for (w, c) in input
        sum += c;
    emit(w, sum);
}
```

What will this produce?

(a) here: 2, are: 2, some: 1, words: 3
(b) here: 2, are: 2, some: 1, words: 5
(c) here: 1, are: 1, some: 1, words: 1
What will this produce?

(a) here: 2, are: 2, some: 1, words: 3
(b) here: 2, are: 1, some: 1, words: 5
(c) here: 1, are: 1, some: 1, words: 1
def MapFn1: (S, S) -> (S, S) {
    for w in input.value().split()
        emit(input.key(), w)
}
def ReduceFn1: (S, S) -> (S, I) {
    for w in input.value()
        emit(w, 1)
}
def ReduceFn2: (S, I) -> (S, I) {
    sum = 0;
    for (w, c) in input
        sum += c;
    emit(w, sum);
}

Input K: V
Doc1 : here are some words
Doc2: words words words words
Doc3: here are words

Reducer is by DocId only

What will this produce?
(a) here:2, are:2, some:1, words:3
(b) here:2, are:1, some:1, words:5
(c) here:1, are:1, some:1, words:1
Other MapReduce Functions

• Sort
• Unique
• Sample
• First
• Filter
• Join
Other MapReduce Functions

- Sort
- Unique
- Sample
- First
- Filter
- Join
Joins

• Joins are usually computed “under the hood” by most MR implementations (like in SQL).
• But you can imagine having to do them yourself…
• …or, if you aren’t the imaginative type, you can just look at the homework (sry).
Joins

- Joins are usually computed “under the hood” by most MR implementations (like in SQL)
Joins

- Joins are usually computed “under the hood” by most MR implementations (like in SQL)
- But you can imagine having to do them yourself…
Hacky Joins

• Joins are usually computed “under the hood” by most MR implementations (like in SQL)

• But you can imagine having to do them yourself…

• …or, if you aren’t that imaginative type, you can just look at the homework
Hacky Joins

• Joins are usually computed “under the hood” by most MR implementations (like in SQL)

• But you can imagine having to do them yourself…

• …or, if you aren’t that imaginative type, you can just look at the homework

• (sry)
Real Life Application
Real Life Application

Is Charles Mingus a composer?
Is Charles Mingus a composer?

“Mingus is a composer”
Real Life Application

Is Charles Mingus a composer?

“Mingus is a composer”

Visions of Jazz: The First Century - Page 452 - Google Books Result
https://books.google.com/books?isbn=0199879532
Gary Giddins - 1998 - Music
If Mingus is a composer worthy of our attention, it must be because his melodies are one with his voicings and scaffolding. Set adrift among Harry Partch’s globes …

Jazz: There's a Mingus a-Monk us, in The Abstract Truth - Daily Kos
www.dailykos.com/story/.../-Jazz-There-s-a-Mingus-a-Monk-us-in-The-Abstract-Trut... ▼
Mar 9, 2014 - Mingus is a composer and arranger. In fact a big band has been established which performs in Manhattan every week in NYC that just plays …
Real Life Application

Is Charles Mingus a **1950s American jazz composer**?

"Mingus is a **1950s American jazz composer**"

No results found for "mingus is a 1950s american jazz composer".
Is Charles Mingus a 1950s American jazz composer?
Is Charles Mingus a 1950s American jazz composer?

... if Mingus is a composer worthy of our attention, it must be because...

**Mingus dominated the scene back in the 1950s and 1960s.**

**Mingus was truly a product of America** in all its historic complexities...

A virtuoso bassist and composer, **Mingus** irrevocably **changed the face of jazz**...
ComposerX dominated the scene back in the 1950s and 1960s.

ComposerX is a 1950s composer.
Real Life Application

<table>
<thead>
<tr>
<th>Subject</th>
<th>Predicate</th>
<th>Object</th>
</tr>
</thead>
<tbody>
<tr>
<td>Barack Obama</td>
<td>won</td>
<td>the electoral vote</td>
</tr>
<tr>
<td>Kamala Lopez</td>
<td>wrote</td>
<td>an op-ed for HuffPo</td>
</tr>
<tr>
<td>Charles Mingus</td>
<td>wrote</td>
<td>jazz</td>
</tr>
<tr>
<td>Barack Obama</td>
<td>opposed</td>
<td>the appropriations bill</td>
</tr>
<tr>
<td>Barack Obama</td>
<td>listens to</td>
<td>jazz</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Category</th>
<th>Entity</th>
</tr>
</thead>
<tbody>
<tr>
<td>Person</td>
<td>Barack Obama</td>
</tr>
<tr>
<td>Person</td>
<td>Kamala Lopez</td>
</tr>
<tr>
<td>Person</td>
<td>Charles Mingus</td>
</tr>
<tr>
<td>Huffington Post Columnists</td>
<td>Barack Obama</td>
</tr>
<tr>
<td>Huffington Post Columnists</td>
<td>Kamala Lopez</td>
</tr>
<tr>
<td>US Presidents</td>
<td>Barack Obama</td>
</tr>
<tr>
<td>Jazz Composers</td>
<td>Charles Mingus</td>
</tr>
<tr>
<td>Subject</td>
<td>Predicate</td>
</tr>
<tr>
<td>------------------</td>
<td>-----------</td>
</tr>
<tr>
<td>Barack Obama</td>
<td>won</td>
</tr>
<tr>
<td>Kamala Lopez</td>
<td>wrote</td>
</tr>
<tr>
<td>Charles Mingus</td>
<td>wrote</td>
</tr>
<tr>
<td>Barack Obama</td>
<td>opposed</td>
</tr>
<tr>
<td>Barack Obama</td>
<td>listens to</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Category</th>
<th>Entity</th>
</tr>
</thead>
<tbody>
<tr>
<td>Person</td>
<td>Barack Obama</td>
</tr>
<tr>
<td>Person</td>
<td>Kamala Lopez</td>
</tr>
<tr>
<td>Person</td>
<td>Charles Mingus</td>
</tr>
<tr>
<td>Huffington Post Columnists</td>
<td>Barack Obama</td>
</tr>
<tr>
<td>Huffington Post Columnists</td>
<td>Kamala Lopez</td>
</tr>
<tr>
<td>US Presidents</td>
<td>Barack Obama</td>
</tr>
<tr>
<td>Jazz Composers</td>
<td>Charles Mingus</td>
</tr>
</tbody>
</table>

Desired output:

<table>
<thead>
<tr>
<th>Subject</th>
<th>Predicate</th>
<th>Object</th>
<th>Categories</th>
</tr>
</thead>
<tbody>
<tr>
<td>Barack Obama</td>
<td>won</td>
<td>the electoral vote</td>
<td>Person, US_Presidents, Huffington_Post_Columnists</td>
</tr>
<tr>
<td>Kamala Lopez</td>
<td>wrote</td>
<td>an op-ed for HuffPo</td>
<td>Person, Huffington_Post_Columnists</td>
</tr>
</tbody>
</table>
### Joins

<table>
<thead>
<tr>
<th>Subject</th>
<th>Predicate</th>
<th>Object</th>
<th>Category</th>
<th>Entity</th>
</tr>
</thead>
<tbody>
<tr>
<td>Barack Obama</td>
<td>won</td>
<td>the electoral vote</td>
<td>Person, US_Presidents, Huffington_Post_Columnists</td>
<td>Barack Obama</td>
</tr>
<tr>
<td>Kamala Lopez</td>
<td>wrote</td>
<td>an op-ed for HuffPo</td>
<td>Huffington_Post_Columnists, Person</td>
<td>Kamala Lopez</td>
</tr>
<tr>
<td>Charles Mingus</td>
<td>wrote</td>
<td>jazz</td>
<td>Huffington_Post_Columnists, Jazz_Composers</td>
<td>Charles Mingus</td>
</tr>
<tr>
<td>Barack Obama</td>
<td>opposed</td>
<td>the appropriations bill</td>
<td>Huffington_Post_Columnists, US_Presidents</td>
<td>Barack Obama, Kamala Lopez</td>
</tr>
<tr>
<td>Barack Obama</td>
<td>listens to</td>
<td>jazz</td>
<td>Huffington_Post_Columnists, Jazz_Composers</td>
<td>Charles Mingus</td>
</tr>
</tbody>
</table>

### Desired output:

<table>
<thead>
<tr>
<th>Subject</th>
<th>Predicate</th>
<th>Object</th>
<th>Categories</th>
</tr>
</thead>
<tbody>
<tr>
<td>Barack Obama</td>
<td>won</td>
<td>the electoral vote</td>
<td>Person, US_Presidents, Huffington_Post_Columnists</td>
</tr>
<tr>
<td>Kamala Lopez</td>
<td>wrote</td>
<td>an op-ed for HuffPo</td>
<td>Huffington_Post_Columnists, Person, Huffington_Post_Columnists</td>
</tr>
</tbody>
</table>

80
## Joins

### Facts

<table>
<thead>
<tr>
<th>Subject</th>
<th>Predicate</th>
<th>Object</th>
</tr>
</thead>
<tbody>
<tr>
<td>Barack Obama</td>
<td>won</td>
<td>the electoral vote</td>
</tr>
<tr>
<td>Kamala Lopez</td>
<td>wrote</td>
<td>an op-ed for HuffPo</td>
</tr>
<tr>
<td>Charles Mingus</td>
<td>wrote</td>
<td>jazz</td>
</tr>
<tr>
<td>Barack Obama</td>
<td>opposed</td>
<td>the appropriations bill</td>
</tr>
<tr>
<td>Barack Obama</td>
<td>listens to</td>
<td>jazz</td>
</tr>
</tbody>
</table>

### Categories

<table>
<thead>
<tr>
<th>Category</th>
<th>Entity</th>
</tr>
</thead>
<tbody>
<tr>
<td>Person</td>
<td>Barack Obama</td>
</tr>
<tr>
<td>Person</td>
<td>Kamala Lopez</td>
</tr>
<tr>
<td>Person</td>
<td>Charles Mingus</td>
</tr>
<tr>
<td>Huffington Post Columnists</td>
<td>Barack Obama</td>
</tr>
<tr>
<td>Huffington Post Columnists</td>
<td>Kamala Lopez</td>
</tr>
<tr>
<td>US Presidents</td>
<td>Barack Obama</td>
</tr>
<tr>
<td>Jazz Composers</td>
<td>Charles Mingus</td>
</tr>
</tbody>
</table>

Select * from Facts, Categories
Where Subject == Entity
### Facts

<table>
<thead>
<tr>
<th>Subject</th>
<th>Predicate</th>
<th>Object</th>
</tr>
</thead>
<tbody>
<tr>
<td>Barack Obama</td>
<td>won</td>
<td>the electoral vote</td>
</tr>
<tr>
<td>Kamala Lopez</td>
<td>wrote</td>
<td>an op-ed for HuffPo</td>
</tr>
<tr>
<td>Charles Mingus</td>
<td>wrote</td>
<td>jazz</td>
</tr>
<tr>
<td>Barack Obama</td>
<td>opposed</td>
<td>the appropriations bill</td>
</tr>
<tr>
<td>Barack Obama</td>
<td>listens to</td>
<td>jazz</td>
</tr>
</tbody>
</table>

### Categories

<table>
<thead>
<tr>
<th>Category</th>
<th>Entity</th>
</tr>
</thead>
<tbody>
<tr>
<td>Person</td>
<td>Barack Obama</td>
</tr>
<tr>
<td>Person</td>
<td>Kamala Lopez</td>
</tr>
<tr>
<td>Person</td>
<td>Charles Mingus</td>
</tr>
<tr>
<td>Huffington Post Columnists</td>
<td>Barack Obama</td>
</tr>
<tr>
<td>Huffington Post Columnists</td>
<td>Kamala Lopez</td>
</tr>
<tr>
<td>US Presidents</td>
<td>Barack Obama</td>
</tr>
<tr>
<td>Jazz Composers</td>
<td>Charles Mingus</td>
</tr>
</tbody>
</table>

Select * from Facts, Categories
Where Subject == Entity
GroupBy Subject
Joins

### Facts

<table>
<thead>
<tr>
<th>Subject</th>
<th>Predicate</th>
<th>Object</th>
</tr>
</thead>
<tbody>
<tr>
<td>Barack Obama</td>
<td>won</td>
<td>the electoral vote</td>
</tr>
<tr>
<td>Kamala Lopez</td>
<td>wrote</td>
<td>an op-ed for HuffPo</td>
</tr>
<tr>
<td>Charles Mingus</td>
<td>wrote</td>
<td>jazz</td>
</tr>
<tr>
<td>Barack Obama</td>
<td>opposed</td>
<td>the appropriations bill</td>
</tr>
<tr>
<td>Barack Obama</td>
<td>listens to</td>
<td>jazz</td>
</tr>
</tbody>
</table>

### Categories

<table>
<thead>
<tr>
<th>Category</th>
<th>Entity</th>
</tr>
</thead>
<tbody>
<tr>
<td>Person</td>
<td>Barack Obama</td>
</tr>
<tr>
<td>Person</td>
<td>Kamala Lopez</td>
</tr>
<tr>
<td>Person</td>
<td>Charles Mingus</td>
</tr>
<tr>
<td>Huffington Post Columnists</td>
<td>Barack Obama</td>
</tr>
<tr>
<td>Huffington Post Columnists</td>
<td>Kamala Lopez</td>
</tr>
<tr>
<td>US Presidents</td>
<td>Barack Obama</td>
</tr>
<tr>
<td>Jazz Composers</td>
<td>Charles Mingus</td>
</tr>
</tbody>
</table>

Select * from Facts, Categories

Where Subject == Entity

GroupBy Subject

Key: String

Value: (list_of((String, String, String), list_of((String, String)))
DIY Joins

def MapFn: (String, Obj) -> (String, Obj)

v = input.value()
if (typeof(v) == Fact) {
  emit(v.Subject, v)
} else {
  emit(v.Entity, v)
}

def ReduceFn: (String, Obj) -> (Fact, List(String))

all_cats = []
all_facts = []
for v in input.value:
  if (typeof(v) == Fact) {
    all_facts.append(v)
  } else {
    all_cats.append(v.Category)
  }

for f in all_facts { emit(f, all_cats); }

Categories

<table>
<thead>
<tr>
<th>Category</th>
<th>Entity</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Facts

<table>
<thead>
<tr>
<th>Subject</th>
<th>Predicate</th>
<th>Object</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
def MapFn: (String, Obj) -> (String, Obj){
    v = input.value()
    if (typeof(v) == Fact) {
        emit(v.Subject, v)
    } else {
        emit(v.Entity, v)
    }
}

def ReduceFn: (String, Obj) -> (Fact, List(String)){
    all_cats = []; all_facts = []
    for v in input.value(
        if (typeof(v) == Fact) {
            all_facts.append(v)
        } else {
            all_cats.append(v.Category)
        }
    for f in all_facts { emit(f, all_cats); }
}
def MapFn: (String, Obj) -> (String, Obj) {
    v = input.value()
    if (typeof(v) == Fact) {
        emit(v.Subject, v)
    } else {
        emit(v.Entity, v)
    }
}

def ReduceFn: (String, Obj) -> (Fact, List(String)) {
    all_cats = []
    all_facts = []
    for v in input.value() {
        if (typeof(v) == Fact) {
            all_facts.append(v)
        } else {
            all_cats.append(v.Category)
        }
    }
    for f in all_facts {
        emit(f, all_cats)
    }
}
def MapFn: (String, Obj) -> (String, Obj){
    v = input.value()
    if (typeof(v) == Fact) {
        emit(v.Subject, v)
    } else {
        emit(v.Entity, v)
    }
}

def ReduceFn: (String, Obj) -> (Fact, List(String)){
    all_cats = []; all_facts = []
    for v in input.value()
        if (typeof(v) == Fact) {
            all_facts.append(v)
            emit(v.Subject, v)
        } else {
            all_cats.append(v.Category)
        }
    }
}
def MapFn: (String, Obj) -> (String, Obj){
    v = input.value()
    if (typeof(v) == Fact) {
        emit(v.Subject, v)
    } else {
        emit(v.Entity, v)
    }
}

def ReduceFn: (String, Obj) -> (Fact, List(String)){
    all_cats = []; all_facts = []
    for v in input.value()
        if (typeof(v) == Fact) {
            all_facts.append(v)
        } else {
            all_cats.append(v.Category)
        }
    }
    for f in all_facts { emit(f, all_cats); }
}
def MapFn: (String, Obj) -> (String, Obj){
    v = input.value()
    if (typeof(v) == Fact) {
        emit(v.Subject, v)
    } else {
        emit(v.Entity, v)
    }
}

def ReduceFn: (String, Obj) -> (Fact, List(String)){
    all_cats = []; all_facts = []
    for v in input.value()
    if (typeof(v) == Fact) {
        all_facts.append(v)
    } else {
        all_cats.append(v.Category)
    }
    for f in all_facts { emit(f, all_cats); }
}
def MapFn: (String, Tuple) -> (String, Tuple){
    v = input.value()
    if (len(v) == 3) {
        emit(v[0], v)
    } else {
        emit(v[1], v)
    }
}

def ReduceFn: (String, Tuple) -> (Tuple, List(String)){
    all_cats = []; all_facts = []
    for v in input.value()
    {
        if (len(v) == 3) {
            all_facts.append(v)
        } else {
            all_cats.append(v[0])
        }
    }
    for f in all_facts { emit(f, all_cats); }
Hacky Joins

Facts

<table>
<thead>
<tr>
<th>Subject</th>
<th>Predicate</th>
<th>Object</th>
</tr>
</thead>
</table>

Categories

<table>
<thead>
<tr>
<th>Category</th>
<th>Entity</th>
</tr>
</thead>
</table>

def MapFn: (String, Tuple) -> (String, Tuple)

v = input.value()
if (len(v) == 3) {
    emit(v[0], v)
} else {
    emit(v[1], v)
}

This is not a thing you ever actually do! Please do not do this!

But do it for the homework, and I will never tell, and we will never speak of it, and if someone asks you in an interview if you are the kind of person who would do this in a map reduce, you will deny deny deny. Agreed?

def ReduceFn: (String, Tuple) -> (Tuple, List(String))

all_cats = []; all_facts = []
for v in input.value()
    if (len(v) == 3) {
        all_facts.append(v)
    } else {
        all_cats.append(v[0])
    }

for f in all_facts
    emit(f, all_cats);
Hacky Joins

```python
def MapFn: (String, Tuple) -> (String, Tuple){
    v = input.value()
    if (len(v) == 3) {
        emit(v[0], v)
    } else {
        emit(v[1], v)
    }
}
```

This is not a thing you ever actually do! Please do not do this!

(But do it for the homework, and I will never tell, and we will never speak of it, and if someone asks you in an interview if you are the kind of person who would do this in a map reduce, you will deny deny deny. Agreed?)

```python
def ReduceFn: (String, Tuple) -> (Tuple, List(String)){
    all_cats = []; all_facts = []
    for v in input.value(
        if (len(v) == 3) {
            all_facts.append(v)
        } else {
            all_cats.append(v[0])
        } 
    }
    for f in all_facts { emit(f, all_cats); }
}
Bottlenecks!
Mappers: (DocID, Doc) -> (DocID, Sent)

Mappers: (DocID, Sent) -> (Word, Count)

Reducers: (Word, Count) -> Word, sum(Count)
Clicker Question!

In the best-case scenario, how much parallelization could we get here (maximum number of mappers)?

(a) $N$
(b) $\log(N)$
(c) As many as we can afford.
Clicker Question!

In the best-case scenario, how much parallelization could we get here (maximum number of mappers)?

(a) $N$
(b) $\log(N)$
(c) As many as we can afford.
Mappers: (DocID, Doc) -> (DocID, Sent)

Mappers: (DocID, Sent) -> (Word, Count)

Reducers: (Word, Count) -> Word, sum(Count)

Clicker Question!
How about here?

(a) N
(b) M
(c) N*M
Mappers: (DocID, Doc) -> (DocID, Sent)

Mappers: (DocID, Sent) -> (Word, Count)

Reducers: (Word, Count) -> Word, sum(Count)

Clicker Question!
How about here?

(a) \(N\)
(b) \(M\)
(c) \(N \times M\)
Mappers: (DocID, Doc) -> (DocID, Sent)

Sent1

Sent2

SentM

Mappers: (DocID, Sent) -> (Word, Count)

Word1

WordK

Clicker Question!
How about here?

(a) N
(b) M
(c) N*M

Mapping doesn’t require the same keys to route to the same machine.

Reducers: (Word, Count) -> (Word, sum(Count))
Clicker Question!
Which is (likely to be) faster?

(a) Mapper1: (DocID, Doc) -> (DocID, Sent)
  ↓
  Mapper2: (DocID, Sent) -> (Word, Count)
  ↓
  Reducer: (Word, Count) -> Word, sum(Count)

(b) Mapper: (DocID, Doc) -> (Word, Count)
  ↓
  Reducer: (Word, Count) -> Word, sum(Count)

(c) They are the same
Mapper1: 
(DocID, Doc) -> (DocID, Sent)

Mapper2: 
(DocID, Sent) -> (Word, Count)

Reducer: 
(Word, Count) -> Word, sum(Count)

(a)

Mapper: 
(DocID, Doc) -> (Word, Count)

Reducer: 
(Word, Count) -> Word, sum(Count)

(b)

(c) They are the same

Clicker Question!
Which is (likely to be) faster?
Clicker Question!
Which is (likely to be) faster?

(a) Mapper1:
(DocID, Doc) -> (DocID, Sent)

Mapper2:
(DocID, Sent) -> (Word, Count)

Reducer:
(Word, Count) -> Word, sum(Count)

(b) Mapper:
(DocID, Doc) -> (Word, Count)

Reducer:
(Word, Count) -> Word, sum(Count)

(c) They are the same
Clicker Question!
Which is (likely to be) faster?

(a) Mapper1: (DocID, Doc) -> (DocID, Sent)
(b) Mapper: (DocID, Doc) -> (Word, Count)

Smaller jobs = more dynamic load balancing and faster recovery from failure

Reducer: (Word, Count) -> Word, sum(Count)

(c) They are the same
Clicker Question!
Which is (likely to be) faster?

(a) Mapper1:
(DocID, Doc) -> (DocID, Sent)

Reducer:
(Word, Count) -> Word, sum(Count)

(b) Mapper:
(DocID, Doc) -> (Word, Count)

for sentence in doc:
    for word in sentence:
        blah blah

Reducer:
(Word, Count) -> Word, sum(Count)

(c) They are the same

In general, nested loops should be refactored into multiple mappers.
(non)Clicker Question!

What might be bad here?

Mappers: (DocID, Doc) -> (Sent, 1)

Mappers: (Sent, 1) -> (Word, Count)

Reducers: (Word, Count) -> Word, sum(Count)
(non)Clicker Question!

What might be bad here?

Skewed Key Distributions!
(Need all values with the same key to be together, so can’t automatically load balance)

Reducers: (Word, Count) -> Word, sum(Count)
Zipf’s Law

https://en.wikipedia.org/wiki/Zipf%27s_law
Zipf’s Law

“The frequency of any word is inversely proportional to its rank in the frequency table” (Wikipedia)

https://en.wikipedia.org/wiki/Zipf%27s_law
“The frequency of any word is inversely proportional to its rank in the frequency table” (Wikipedia)

https://en.wikipedia.org/wiki/Zipf%27s_law
Zipf’s Law

“The frequency of any word is inversely proportional to its rank in the frequency table” (Wikipedia)
Zipf’s Law

The most frequent 0.2% of words make up 50% of occurrences.
# Real Life Application

<table>
<thead>
<tr>
<th>Subject</th>
<th>Predicate</th>
<th>Object</th>
<th>Categories</th>
</tr>
</thead>
<tbody>
<tr>
<td>Barack Obama</td>
<td>won</td>
<td>the electoral vote</td>
<td>Person, US_Presidents, Huffington_Post_Columnists</td>
</tr>
<tr>
<td>Kamala Lopez</td>
<td>wrote</td>
<td>an op-ed for HuffPo</td>
<td>Person, Huffington_Post_Columnists, Actor</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Predicate</th>
<th>Object</th>
<th>Category</th>
<th>Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>won</td>
<td>the electoral vote</td>
<td>US_Presidents</td>
<td>0.92</td>
</tr>
<tr>
<td>won</td>
<td>the electoral vote</td>
<td>Person</td>
<td>0.89</td>
</tr>
<tr>
<td>won</td>
<td>the electoral vote</td>
<td>Huffington Post Columnists</td>
<td>0.23</td>
</tr>
<tr>
<td>wrote</td>
<td>an op-ed for HuffPo</td>
<td>Huffington Post Columnists</td>
<td>0.99</td>
</tr>
<tr>
<td>wrote</td>
<td>an op-ed for HuffPo</td>
<td>Person</td>
<td>0.91</td>
</tr>
</tbody>
</table>
# Real Life Application

<table>
<thead>
<tr>
<th>Subject</th>
<th>Predicate</th>
<th>Object</th>
<th>Categories</th>
</tr>
</thead>
<tbody>
<tr>
<td>Barack Obama</td>
<td>won</td>
<td>the electoral vote</td>
<td>Person, US_Presidents, Huffington_Post_Columnists</td>
</tr>
<tr>
<td>Kamala Lopez</td>
<td>wrote</td>
<td>an op-ed for HuffPo</td>
<td>Person, Huffington_Post_Columnists, Actor</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Predicate</th>
<th>Object</th>
<th>Category</th>
<th>Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>won</td>
<td>the electoral vote</td>
<td>US_Presidents</td>
<td>702,345</td>
</tr>
<tr>
<td>won</td>
<td>the electoral vote</td>
<td>Person</td>
<td>812,485</td>
</tr>
<tr>
<td>won</td>
<td>the electoral vote</td>
<td>Huffington Post Columnists</td>
<td>24,571</td>
</tr>
<tr>
<td>wrote</td>
<td>an op-ed for HuffPo</td>
<td>Huffington Post Columnists</td>
<td>134,213</td>
</tr>
<tr>
<td>wrote</td>
<td>an op-ed for HuffPo</td>
<td>Person</td>
<td>136,091</td>
</tr>
</tbody>
</table>
First Attempt

Mapper1:
(subject, predicate, object), list_of(categories) ->
category, (predicate, object)

Reducer1: ...
(category, list_of(predicate, object)) ->
(category, predicate, object), 1

Reducer2:
(category, predicate, object), list_of(count) ->
(category, predicate, object), total

114
First Attempt

Mapper1:
(subject, predicate, object), list_of(categories) ->
category, (predicate, object)

Reducer1:
(category, list_of(predicate, object)) ->
(category, predicate, object), 1

Reducer2:
(category, predicate, object), list_of(count) ->
(category, predicate, object), total

✔

115
First Attempt

Mapper1:  
(subject, predicate, object), list_of(categories) ->
category, (predicate, object)

Reducer1:  ...
category, list_of(predicate, object) ->
(category, predicate, object), 1

Every tuple involving a single category (e.g. “Person”) has to go through the same reducer...
First Attempt

Mapper1:
(subject, predicate, object), list_of(categories) ->
category, (predicate, object)

Reducer1:
(category, list_of(predicate, object)) ->
(category, predicate, object), 1

Reducer2:
(category, predicate, object), list_of(count) ->
(category, predicate, object), total

✔
First Attempt

Mapper1:
(subject, predicate, object), list_of(categories) ->
category, (predicate, object)

Reducer1:
(category, list_of(predicate, object)) ->
(category, predicate, object), 1

Reducer2:
(category, predicate, object), list_of(count) ->
(category, predicate, object), total

✔
So much better!

Mapper1:
(subject, predicate, object), list_of(categories) ->
(category, predicate, object), 1

Reducer2:
(category, predicate, object), list_of(count) ->
(category, predicate, object), total

119
Alright, scram.