

# Map-Reduce

## Алгоритмы

# WordCount

- **Описание задачи**
  - Есть коллекция документов
  - Каждый документ — это набор термов (слов)
  - Необходимо подсчитать кол-во вхождений каждого терма во всех документах
- **Дополнительно**
  - Функция может быть произвольной
  - Например, файл лога содержит время ответа  
Необходимо подсчитать среднее время

# WordCount: baseline

```
class Mapper
```

```
  method Map (docid id, doc d)
```

```
    for all term t in doc d do
```

```
      Emit(term t, count 1)
```

```
class Reducer
```

```
  method Reduce (term t, counts [c1, c2,...])
```

```
    sum = 0
```

```
    for all count c in [c1, c2,...] do
```

```
      sum = sum + c
```

```
    Emit(term t, count sum)
```

(!) Много лишних счетчиков от *Mapper*

```
class Mapper
```

```
    method Map (docid id, doc d)
```

```
        for all term t in doc d do
```

```
            Emit(term t, count 1)
```

```
class Combiner
```

```
    method Combine (term t, [c1, c2,...])
```

```
        sum = 0
```

```
        for all count c in [c1, c2,...] do
```

```
            sum = sum + c
```

```
        Emit(term t, count sum)
```

```
class Reducer
```

```
    method Reduce (term t, counts [c1, c2,...])
```

```
        sum = 0
```

```
        for all count c in [c1, c2,...] do
```

```
            sum = sum + c
```

```
        Emit(term t, count sum)
```

# WordCount: “In-mapper combining”, v.1

```
class Mapper  
  method Map (docid id, doc d)  
    H = new AssociativeArray  
    for all term t in doc d do  
      H{t} = H{t} + 1  
    for all term t in H do  
      Emit(term t, count H{t})
```

# WordCount: “In-mapper combining”, v.2

*class **Mapper***

*method **Initialize***

*$H = \text{new } \textbf{AssociativeArray}$*

*method **Map** (docid  $id$ , doc  $d$ )*

*for all term  $t$  in doc  $d$  do*

*$H\{t\} = H\{t\} + 1$*

*method **Close***

*for all term  $t$  in  $H$  do*

*$\text{Emit}(\text{term } t, \text{count } H\{t\})$*

# Среднее значение, v.1

```
class Mapper
```

```
method Map(string t, integer r)
```

```
Emit(string t, integer r)
```

```
class Reducer
```

```
method Reduce(string t, integers [r1, r2, ...])
```

```
sum = 0
```

```
cnt = 0
```

```
for all integers r in [r1, r2, ...] do
```

```
sum = sum + r
```

```
cnt = cnt + 1
```

```
avg = sum / cnt
```

```
Emit(string t, integer avg)
```

```
class Mapper
```

```
    method Map(string t, integer r)
```

```
        Emit(string t, integer r)
```

```
class Combiner
```

```
    method Combine(string t, integers [r1, r2, ...])
```

```
        sum = cnt = 0
```

```
        for all integers r in [r1, r2, ...] do
```

```
            sum = sum + r
```

```
            cnt = cnt + 1
```

```
        Emit(string t, pair(sum, cnt))
```

```
class Reducer
```

```
    method Reduce(string t, pairs[(s1,c1),(s2,c2) ...])
```

```
        sum = cnt = 0
```

```
        for all pairs p in [(s1,c1),(s2,c2) ...] do
```

```
            sum = sum + p.s
```

```
            cnt = cnt + p.c
```

```
        avg = sum / cnt
```

```
        Emit(string t, integer avg)
```



```
class Mapper
```

```
    method Map(string t, integer r)
```

```
        Emit(string t, pair (r,1))
```

```
class Combiner
```

```
    method Combine(string t pairs[(s1,c1),(s2,c2) ...])
```

```
        sum = cnt = 0
```

```
        for all pairs p in [(s1,c1),(s2,c2) ...] do
```

```
            sum = sum + p.s
```

```
            cnt = cnt + p.c
```

```
        Emit(string t, pair(sum, cnt))
```

```
class Reducer
```

```
    method Reduce(string t, pairs[(s1,c1),(s2,c2) ...])
```

```
        sum = cnt = 0
```

```
        for all pairs p in [(s1,c1),(s2,c2) ...] do
```

```
            sum = sum + p.s
```

```
            cnt = cnt + p.c
```

```
        avg = sum / cnt
```

```
        Emit(string t, pair (avg, cnt))
```

```
class Mapper
```

```
  method Initialize
```

```
     $S = \text{new } \mathbf{AssociativeArray}$ 
```

```
     $C = \text{new } \mathbf{AssociativeArray}$ 
```

```
  method Map (string  $t$ , integer  $r$ )
```

```
     $S\{t\} = S\{t\} + r$ 
```

```
     $C\{t\} = C\{t\} + 1$ 
```

```
  method Close
```

```
    for all term  $t$  in  $S$  do
```

```
       $\text{Emit}(\text{term } t, \text{pair}(S\{t\}, C\{t\}))$ 
```

# Distinct Values (Unique Items Counting)

## Описание проблемы

- Есть множество записей
- Каждая запись содержит поле  $F$  и производное число категорий  $G = \{G1, G2, \dots\}$

## Задача

- Подсчитать общее число уникальных значений поля  $F$  для каждой категории

*Record 1:  $F=1, G=\{a, b\}$*

*Record 2:  $F=2, G=\{a, d, e\}$*

*Record 3:  $F=1, G=\{b\}$*

*Record 4:  $F=3, G=\{a, b\}$*

*Result:*

*$a \rightarrow 3$  //  $F=1, F=2, F=3$*

*$b \rightarrow 2$  //  $F=1, F=3$*

*$d \rightarrow 1$  //  $F=2$*

*$e \rightarrow 1$  //  $F=2$*

# Distinct Values, v.1

- Решение в две фазы
- Первая фаза
  - *Mapper* пишет все уникальные пары  $[G, F]$
  - *Reducer* подсчитывает общее кол-во вхождений такой пары
  - Основная цель – гарантировать уникальность значений  $F$  для каждого значения  $G$
- Вторая фаза
  - Пары  $[G, F]$  группируются по  $G$  и затем считается общее кол-во элементов в каждой группе

# Distinct Values, v.1

Фаза 1:

```
class Mapper
```

```
    method Map(null, record [value f, categories [g1, g2,...]])
```

```
        for all category g in [g1, g2,...]
```

```
            Emit(record [g, f], count 1)
```

```
class Reducer
```

```
    method Reduce(record [g, f], counts [n1, n2, ...])
```

```
        Emit(record [g, f], null )
```

# Distinct Values, v.1

Фаза 2:

```
class Mapper
```

```
    method Map(record [f, g], null)
```

```
        Emit(value g, count 1)
```

```
class Reducer
```

```
    method Reduce(value g, counts [n1, n2,...])
```

```
        Emit(value g, sum( [n1, n2,...] ) )
```

# Distinct Values, v.2

Одна фаза MapReduce:

- *Mapper*

- Пишет значение и категории

- *Reducer*

- Исключает дубликаты из списка категорий для каждого значения
- Увеличивает счетчик для каждой категории
- В конце *Reducer* пишет общее количество для каждой категории

*class **Mapper***

*method **Map**(null, record [value f, categories [g1, g2,...] )*  
*for all category g in [g1, g2,...]*  
*Emit(value f, category g)*

*class **Reducer***

*method **Initialize***

*H = new AssociativeArray : category -> count*

*method **Reduce**(value f, categories [g1, g2,...])*

*[g1', g2',...] = ExcludeDuplicates( [g1, g2,...] )*

*for all category g in [g1', g2',...]*

*H{g} = H{g} + 1*

*method **Close***

*for all category g in H do*

*Emit(category g, count H{g})*



# Cross-Correlation

- Описание задачи
  - Есть множество кортежей объектов
  - Для каждой пары объектов посчитать число кортежей, где они встречаются вместе
  - Если число объектов  $N$ , то  $N*N$  объектов будет обработано
- Применение
  - Анализ текстов
    - Кортежи – предложения, объекты – слова
  - Маркетинг
    - Покупатели, кто покупает одни товары

# Cross-Correlation: Pairs

- Каждый *Mapper* генерит все пары соседних объектов
- Reducer суммирует количество для всех пар

```
class Mapper
```

```
    method Map(null, items [i1, i2,...] )
```

```
        for all item i in [i1, i2,...]
```

```
            for all item j in [i1, i2,...]
```

```
                Emit(pair [i j], count 1)
```

```
class Reducer
```

```
    method Reduce(pair [i j], counts [c1, c2,...])
```

```
        s = sum([c1, c2,...])
```

```
        Emit(pair[i j], count s)
```

# Cross-Correlation: Stripes

Mapper:  $(a, b) \rightarrow 1$   
 $(a, c) \rightarrow 2$   
 $(a, d) \rightarrow 5 \Rightarrow a \rightarrow \{ b: 1, c: 2, d: 5, e: 3, f: 2 \}$   
 $(a, e) \rightarrow 3$   
 $(a, f) \rightarrow 2$

Reducer:  $+$

$a \rightarrow \{ b: 1, \quad d: 5, e: 3 \}$
$a \rightarrow \{ b: 1, c: 2, d: 2, \quad f: 2 \}$
<hr/>
$a \rightarrow \{ b: 2, c: 2, d: 7, e: 3, f: 2 \}$

*class **Mapper***

*method **Map**(null, items [i1, i2,...] )*

*for all item i in [i1, i2,...]*

*H = new AssociativeArray : item -> counter*

*for all item j in [i1, i2,...]*

*H{j} = H{j} + 1*

*Emit(item i, stripe H)*

*class **Reducer***

*method **Reduce**(item i, stripes [H1, H2,...])*

*H = new AssociativeArray : item -> counter*

*H = merge-sum( [H1, H2,...] )*

*for all item j in H.keys()*

*Emit(pair [i j], H{j})*

# Cross-Correlation: Stripes

- Плюсы
  - Намного меньше операций сортировки и shuffle
  - Возможно, более эффективное использование *Combiner*
- Минусы
  - Более сложная реализация
  - Более “тяжелые” объекты для передаче данных
  - Ограничения на размеры используемой памяти для ассоциативных массивов
- Pairs vs Stripes
  - Обычно, подход со *stripes* быстрее, чем с *pairs*

# Реляционные паттерны MapReduce

# Выборка (Selection)

*class **Mapper***

*method **Map**(rowkey key, value t)*

*if t satisfies the predicate*

*Emit(value t, null)*

# Проекция (Projection)

```
class Mapper
```

```
method Map(rowkey key, value t)
```

```
value g = project(t) // выбрать необходимые поля в g
```

```
Emit(tuple g, null)
```

```
// используем Reducer для устранения дубликатов
```

```
class Reducer
```

```
method Reduce(value t, array n) // n - массив из nulls
```

```
Emit(value t, null)
```



# Объединение (Union)

*// на вход подаются элементы из двух множеств A и B*

*class **Mapper***

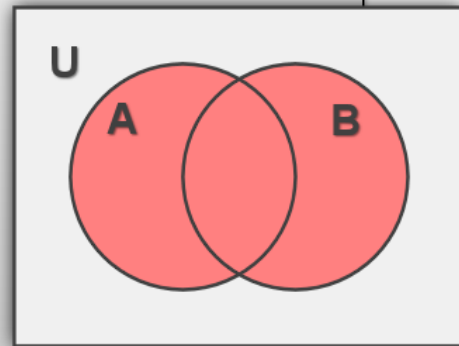
*method **Map**(rowkey key, value t)*

*Emit(value t, null)*

*class **Reducer***

*method **Reduce**(value t, array n) // n - массив из nulls*

*Emit(value t, null)*



# Пересечение (Intersection)

*// на вход подаются элементы из двух множеств A и B*

*class **Mapper***

*method **Map**(rowkey key, value t)*

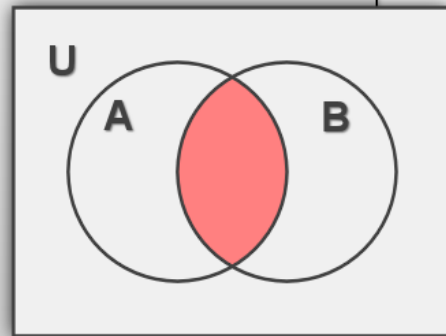
*Emit(value t, null)*

*class **Reducer***

*method **Reduce**(value t, array n) // n - массив из nulls*

*if n.size() = 2*

*Emit(value t, null)*



# Разность (Difference)

*// на вход подаются элементы из двух множеств A и B*

*class **Mapper***

*method **Map**(rowkey key, value t)*

*Emit(value t, string t.SetName) // t.SetName либо 'A' либо 'B'*

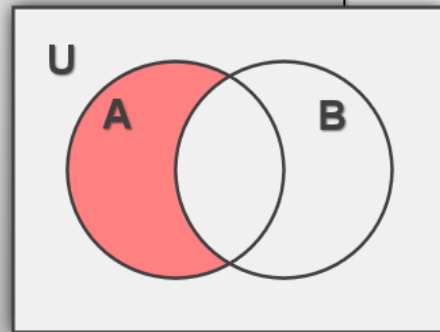
*class **Reducer***

*// массив n может быть ['A'], ['B'], ['A' 'B'] или ['B', 'A']*

*method **Reduce**(value t, array n)*

*if n.size() = 1 and n[1] = 'A'*

*Emit(value t, null)*



# Symmetric Difference

*// на вход подаются элементы из двух множеств A и B*

*class **Mapper***

*method **Map**(rowkey key, value t)*

*Emit(value t, string t.SetName) // t.SetName либо 'A' либо 'B'*

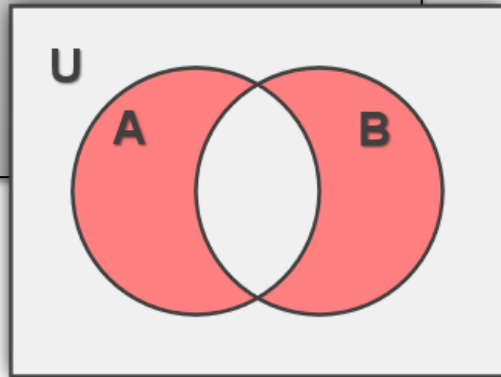
*class **Reducer***

*// массив n может быть ['A'], ['B'], ['A' 'B'] или ['A', 'B']*

*method **Reduce**(value t, array n)*

*if n.size() = 1 and (n[1] = 'A' or n[1] = 'B')*

*Emit(value t, null)*



# GroupBy и Aggregation

*class **Mapper***

*method **Map**(null, tuple [value GroupBy, value AggregateBy, value ...])  
    Emit(value GroupBy, value AggregateBy)*

*class **Reducer***

*method **Reduce**(value GroupBy, [v1, v2,...])  
    // aggregate() : sum(), max(),...  
    Emit(value GroupBy, aggregate( [v1, v2,...] ) )*

# Repartition Join

## Описание задачи

– Объединить два множества  $A$  и  $B$  по ключу  $k$

$id_1: a$		$id_1: 5$		$id_1: (a, 5)$
$id_2: b$	join	$id_2: 7$	=	$id_2: (b, 7)$
$id_2: c$		$id_2: 4$		$id_2: (b, 4)$
				$id_2: (c, 7)$
				$id_2: (c, 4)$

# Repartition Join

*class **Mapper***

*method **Map**(null, tuple [join\_key k, value v1, value v2,...])*

*Emit(join\_key k, tagged\_tuple [set\_name tag, values [v1, v2, ...] ] )*

*class **Reducer***

*method **Reduce**(join\_key k, tagged\_tuples [t1, t2,...])*

*H = new AssociativeArray : set\_name -> values*

*for all tagged\_tuple t in [t1, t2,...]    // separate values into 2 arrays*

*H{t.tag}.add(t.values)*

*for all values a in H{'A'}    // produce a cross-join of the two arrays*

*for all values b in H{'B'}*

*Emit(null, [k a b] )*

# Replicated Join

- Одно множество большое, другое – маленькое
- Храним маленькое в хеш-таблице с ключом  $k$
- *Mapper* объединяет элементы с данными из этой хеш-таблице



# Replicated Join

*class **Mapper***

*method **Initialize***

*$H = \text{new AssociativeArray} : \text{join\_key} \rightarrow \text{tuple from } A$*

*$A = \text{load}()$*

*for all [ join\_key  $k$ , tuple  $[a_1, a_2, \dots]$  ] in  $A$*

*$H\{k\} = H\{k\}.\text{append}( [a_1, a_2, \dots] )$*

*method **Map**(join\_key  $k$ , tuple  $B$ )*

*for all tuple  $a$  in  $H\{k\}$*

*$\text{Emit}(\text{null}, \text{tuple } [k \ a \ B] )$*

# TF-IDF на MapReduce

# TF-IDF

Term Frequency – Inverse Document Frequency

- Используется при работе с текстом
- В Information Retrieval

# TF

*Term Frequency* — отношение числа вхождения слова к общему количеству слов документа

$$tf(t, d) = \frac{n_i}{\sum_k n_k}$$

$n_i$  - число вхождений слова в документ

# IDF

*Inverse Document Frequency* — инверсия частоты, с которой слово встречается в документах коллекции

$$idf(t, D) = \log \frac{|D|}{|(d_i \supset t)|}$$

Где:

$|D|$  — количество документов в корпусе

$|(d_i \supset t)|$  — кол-во документов, содержащих  $t$

# TF-IDF

$$tf\_idf(t, d, D) = tf(t, d) \times idf(t, D)$$

# TF-IDF

Что нужно будет вычислить:

- Сколько раз слово  $T$  встречается в данном документе ( $tf$ )
- Сколько документов, в котором встречается данное слово  $T$  ( $n$ )
- Общее число документов ( $N$ )

# TF-IDF

- **Job 1:** Частота слова в документе
- *Mapper*
  - Input: (*docname*, *contents*)
  - Output: (*(word, docname), 1*)
- *Reducer*
  - Суммирует число слов в документе
  - Outputs: (*(word, docname), tf*)
- *Combiner* такой же как и *Reducer*



# TF-IDF

- **Job 2:** Кол-во документов для слова
- *Mapper*
  - Input:  $((word, docname), tf)$
  - Output:  $(word, (docname, tf, 1))$
- *Reducer*
  - Суммирует единицы чтобы посчитать  $n$
  - Output:  $((word, docname), (tf, n))$

# TF-IDF

- **Job 3:** Расчет TF-IDF
- *Mapper*
  - Input:  $((word, docname), (tf, n))$
  - Output:  $((word, docname), (TF * IDF))$
- *Reducer*
  - Не требуется