Correction TD1
+
Suite du cours

Anagrammes

- Autre exemple: à partir d'une liste de mots, on cherche à déterminer lesquels sont des anagrammes.
- Données d'entrée:

```
crane
imaginer
surface
metropole
migraine
structure
ancre
```

- Là aussi, le choix de la clef est crucial.

Anagrammes

- *map*: renvoie un couple (clef;valeur), avec le mot pour valeur et les lettres du mot ordonnées dans l'ordre alphabétique pour clef.
- La clef constitue de fait un « point commun » aux anagrammes: les lettres elles-même, ordonnées. Le *shuffle* devrait donc regrouper les anagrammes ensemble.
- Aprés execution:

```
("acenr";"crane")
("aegiimnr";"imaginer")
("acefrsu";"surface")
("eelmooprt";"metropole")
("aegiimnr";"migraine")
("cerrsttuu";"structure")
("acenr";"ancre")
```

Anagrammes

- Aprés l'étape de *shuffle*:

```
("acenr";"crane"), ("acenr";"ancre")
("aegiimnr";"imaginer"), ("aegiimnr";"migraine")
("acefrsu";"surface")
("eelmooprt";"metropole")
("cerrsttuu";"structure")
```

Anagrammes

- ***reduce*: concatène toutes les valeurs d'entrée pour la clef unique; renvoie un couple (clef;valeur) avec la clef d'entrée pour clef et la chaîne ainsi concaténée pour valeur.**
- **Aprés execution:**

```
("acenr";"crane ancre")
("aegiimnr";"imaginer migraine")
("acefrsu";"surface")
("eelmooprt";"metropole")
("cerrsttuu";"structure")
```

Sentiment Client/Twitter

- **Clef: un descripteur de sentiment client (« satisfait », « insatisfait » ou « attente », « inconcluant »).**
- ***map*: génère un couple (clef;valeur) par sentiment client détecté (mot correspondant à une liste prédéfinie).**
- **Si deux sentiments contradictoires détectés: renvoyer inconcluant.**
- **On renvoie un couple (clef;valeur) pour chaque fragment des données d'entrée: chaque *tweet***

Sentiment Client/Twitter

Pseudo code:

```
WORDS_BAD = ["nul", "insatisfait", "bof", "incompétents", …]
WORDS_GOOD = ["satisfait", "super", "excellent", …]
BAD=0; GOOD=0
POUR MOT dans [TWEET], FAIRE:
  SI MOT PRESENT_DANS WORDS_BAD:
    BAD=1
  SINON SI MOT PRESENT_DANS WORDS_GOOD:
    GOOD=1
SI BAD==1 ET GOOD==0:
  RENVOYER("insatisfait",1)
SINON SI BAD==0 ET GOOD=1:
  RENVOYER("satisfait",1)
SINON:
  RENVOYER("inconcluant",1)
```

Sentiment Client/Twitter

- **Après exécution:**

```
"@acme Votre service client est nul"
"@acme 30min d'attente... très insatisfait"
"Très satisfait par un produit super !! @acme"
"merci d'avoir RT @acme"
"@acme produit déjà cassé, super insatisfait !"
```

↓

```
("insatisfait";1)
("insatisfait";1)
("satisfait";1)
("inconcluant";1)
("inconcluant";1)
```

Sentiment Client/Twitter

- ***reduce*: additionne les valeurs associées à la clef unique; renvoie le total pour valeur (identique au *reducer* du compteur d'occurences de mots).**
- **Après exécution:**

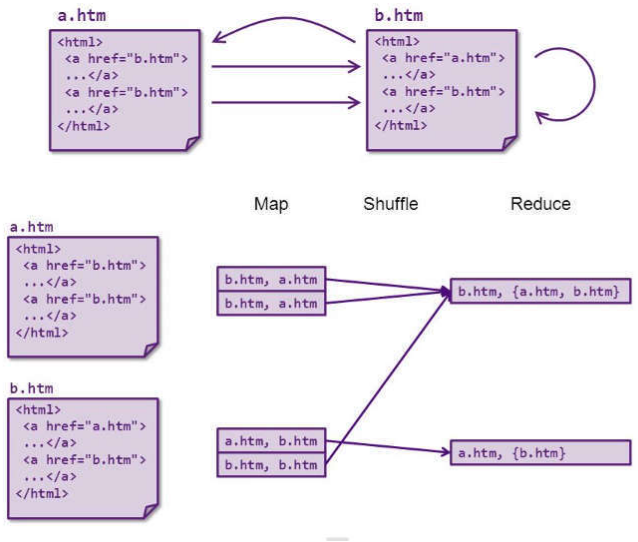
```
("insatisfait";2)
("satisfait";1)
("inconcluant";2)
```

- **Conclusion: lors de la dernière heure, 33% de *tweets* exprimant de la satisfaction.**
- **En important, heure par heure, les données au sein d'une base de données relationnelle, on pourra obtenir en permanence des statistiques sur la satisfaction à partir de *twitter*.**

Liens Web inverses

- Soit un très grand ensemble de pages web
- Pour chaque page p dans l'ensemble
  - Trouver l'ensemble des pages qui référencent p
- Ex. si dans les pages p_1 et p_2, il y a des liens vers la page q, alors nous avons:
  - Sources(q) : $\{p_1, p_2, \ldots\}$

Liens Web inverses



Liens Web inverses

- Map
  - For each URL linking to target, …
  - Output <target, source> pairs
- Reduce
  - Concatenate list of all source URLs
  - Outputs: <target, ***list*** (source)> pairs

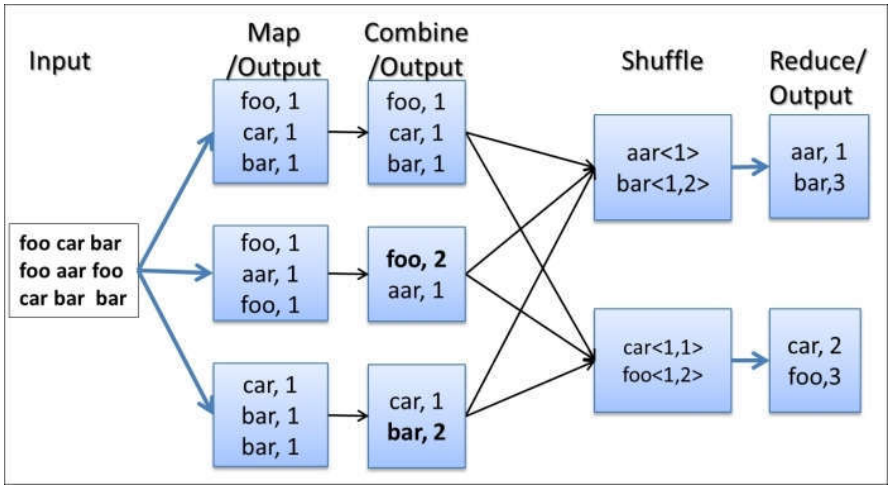
Liens Web inverses

```
map(key, value):
// key: un URL de page web; value: contenu de la page
// p: the given pattern
   for each link to a target t in value
        EmitIntermediate (t, {key});

reduce(key, values):
// key: un URL; values: une liste d'URLs référençant key
   src_set = {};
   for each value v in values
        if v ∉ src_set then
            src_set = src_set + v;
   Emit(key, src_set);
```

Suite du cours

Amélioration de WorCount: Utilisation du Combiner



Amélioration de WorCount: Utilisation du Combiner

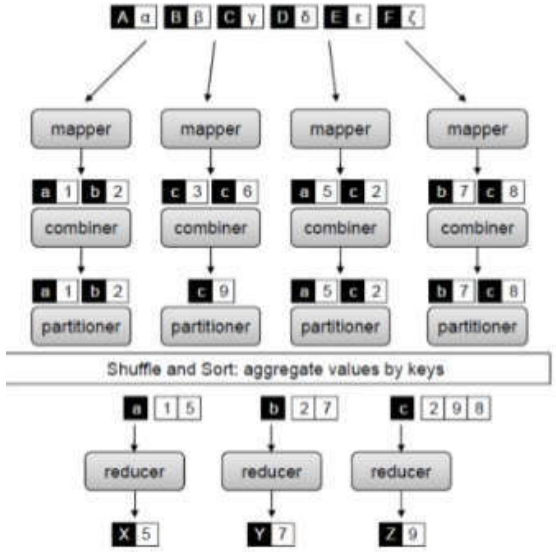
Without Combiner

Map input records	0	0	4.138.476
Map output records	0	0	4.138.476
Map output bytes	0	0	67.381.890
Input split bytes	0	0	420
Combine input records	0	0	0
Combine output records	0	0	0
Reduce input groups	0	0	103
Reduce shuffle bytes	0	0	75.658.866
Reduce input records	0	0	4.138.476
Reduce output records	0	0	103
Spilled Records	0	0	12.219.835
CPU time spent (ms)	0	0	43.420
Physical memory (bytes) snapshot	0	0	963.969.536
Virtual memory (bytes) snapshot	0	0	1.941.934.080
Total committed heap usage (bytes)	0	0	772.935.680
BYTES_READ	211.312.924	0	211.312.924

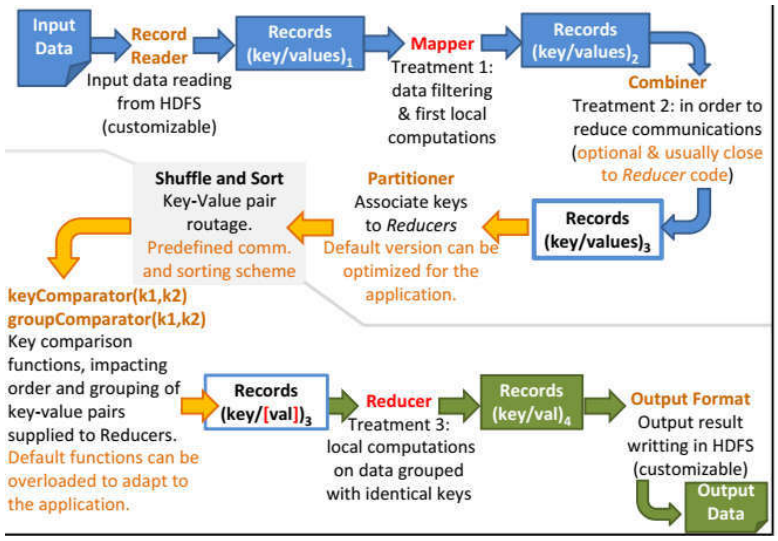
With Combiner

Map input records	0	0	4.138.476
Map output records	0	0	4.138.476
Map output bytes	0	0	67.381.890
Input split bytes	0	0	420
Combine input records	0	0	4.140.336
Combine output records	0	0	2.266
Reduce input groups	0	0	103
Reduce shuffle bytes	0	0	9.609
Reduce input records	0	0	412
Reduce output records	0	0	103
Spilled Records	0	0	2.678
CPU time spent (ms)	0	0	31.420
Physical memory (bytes) snapshot	0	0	798.273.536
Virtual memory (bytes) snapshot	0	0	1.944.887.296
Total committed heap usage (bytes)	0	0	657.997.824
BYTES_READ	211.312.924	0	211.312.924

Résumé: Chaine d'opérations MAP/REDUCE



Résumé: Chaine d'opérations MAP/REDUCE



Résumé: Chaine d'opérations MAP/REDUCE

– En entrée de la chaîne se trouve une classe de lecture des données, appelée *Record Reader*).

Cette classe lit le fichier HDFS d'entrée et le convertit en paires clé-valeur que traitera la fonction *map*. Par défaut elle lit ligne par ligne un flux d'entrée sensé être un fichier texte, et crée une paire dont la clé est l'offset (en octets) de cette ligne dans le fichier et dont la valeur est la ligne elle-même. L'ensemble du *Record Reader* est redéfinissable par l'utilisateur, ainsi que le type des données d'entrée (qui pourraient ne pas être du texte ASCII).

Résumé: Chaine d'opérations MAP/REDUCE

– La classe *Combiner* est un *Reducer* local et optionnel. Cette classe applique un traitement aux paires de sortie de la fonction *map* pour les combiner. En général elle fait la même chose ou presque que la classe *Reducer* sur un ensemble de paires de même clé, et permet de réduire le volume de données routées ensuite vers les *Reducer*. Sa définition n'est pas obligatoire, et si cette classe est définie, alors *Hadoop* décide librement à quels moments l'appeler pour combiner les sorties de chaque *Mapper*.
L'utilisation d'un *Combiner* permet d'améliorer les performances en réduisant le traffic lors du *Shuffle & Sort*, mais entraîne quelques contraintes algorithmiques.

Résumé: Chaine d'opérations MAP/REDUCE

– Le *Partitioner* est une classe optionelle qui permet de spécialiser la répartition des clés sur les différents *Reducer*. Toutes les paires de même clé sont envoyées à un même *Reducer*, mais un *Reducer* peut gérer plusieurs clés. Par défaut, *Hadoop* effectue une répartition aléatoire grossière des clés sur les *Reducer*. Il est possible de définir une politique de répartition de charge plus équilibrée si on a connaissance des clés possibles et de la distribution des volumes de valeurs associés.

Résumé: Chaine d'opérations MAP/REDUCE

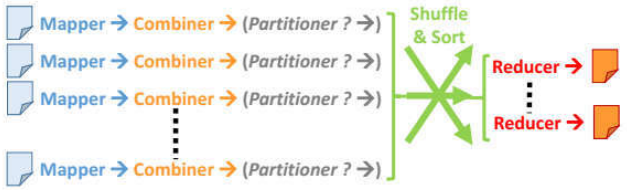
– L'étape de *shuffle & sort* n'est pas redéfinissable. Le schéma de communication est imposé et ressemble à un *all-to-all* orienté des *Mappers* vers les *Reducers* (pour emprunter le vocabulaire du HPC). Cependant, la stratégie de redistribution des clés vers les *Reducers* peut être redéfinie par le *Partitioner* présenté précédemment, ainsi que la stratégie de tri des clés et de regroupement des valeurs en entrée de chaque *Reducer* (voir ci-après).

Résumé: Chaine d'opérations MAP/REDUCE

– Deux fonctions de comparaison de clés à l'entrée des *Reducers* (*keyComparator* et *groupComparator*) permettent de contrôler l'ordre dans lequel les paires *clé - listes de valeurs* seront constituées et présentées aux *Reducers* (ce qui permet d'optimiser fortement certains algorithmes).

– Enfin, le format de sortie peut-être modifié dans l'interface *OutputFormat*, qui définit quels types de clés et valeurs de sortie sont attendues et comment les écrire sur disque. Par défaut, n'importe quel types de données *Writable* peut être écrit dans un fichier texte, et les paires de sortie des *Reducer* sont bien celles écrites sur disques.

La chaîne *Map-Reduce* d'*Hadoop* est donc optimisable ou spécialisable à plusieurs niveaux, mais fonctionne avec des comportements par défaut à toutes les étapes.



- Un *Mapper* par bloc de fichier d'entrée (beaucoup de *Mappers*)
- *Un Combiner* associé à chaque *Mapper* pour réduire le trafic dans le *Shuffle & Sort*
- Quelques *Reducers*, fonctions du nombre de termes attendus

- *Un Partitioner* pour améliorer l'équilibrage de charge entre *Reducers*
  … ssi connaissances pertinentes sur les résultats attendus!

Complément Hadoop MapReduce

Exercices: Max, moyenne, Combiners

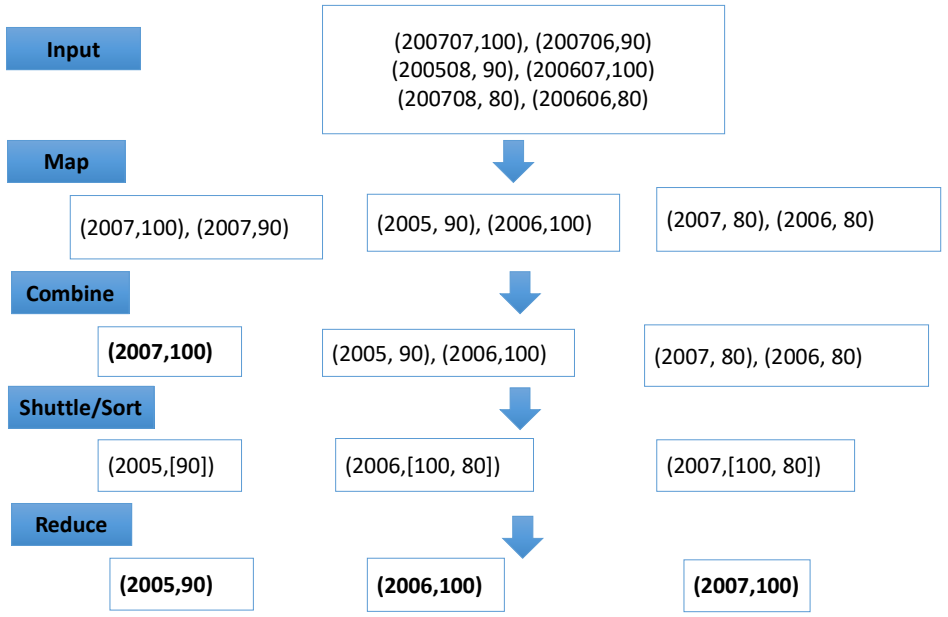
- **Exercice 1 : Calcul de la température maximale**
- On souhaite calculer la température mensuelle maximale pour chaque année à partir des bulletins météorologiques.
- Données en entrée : enregistrement de la forme :
- <Year/Month, Average Temperature of that month>
- **- (200707,100), (200706,90)**
- **- (200508, 90), (200607,100)**
- **- (200708, 80), (200606,80)**
- **On suppose que les données sont découpées par ligne.**
- **Donner les fonctions MAP et Reduce**
-

Mapper and Reducer of Max Temperature

- Map(key, value){
  // key: line number
  // value: tuples in a line
  for each tuple t in value:
  Emit(t->year, t->temperature);}
- Reduce(key, list of values){
  // key: year
  //list of values: a list of monthly temperature
  int max_temp = -100;
  for each v in values:
  max_temp= max(v, max_temp);
  Emit(key, max_temp);}

Combiner is the same as Reducer

MapReduce Example: Max Temperature



MapReduce In-class Exercise

- **Key-Value Pair of Map and Reduce:**
  - **Map:** (year, temperature)
  - **Reduce:** (year, maximum temperature of the year)

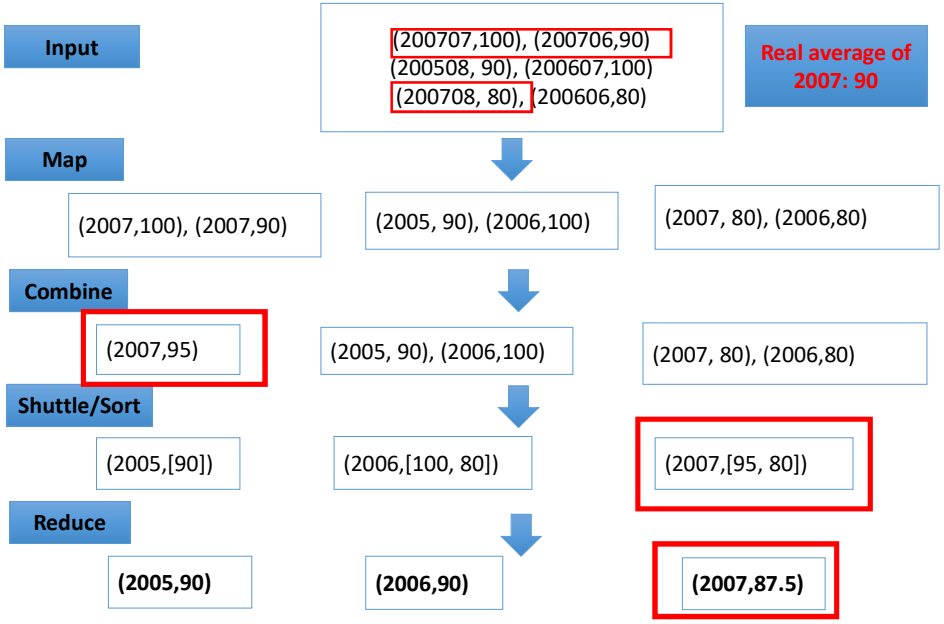
- **Question: How to use the above Map Reduce program *(that contains the combiner)* with slight changes to find the average monthly temperature of the year?**

Mapper and Reducer of Average Temperature

- Map(key, value){
  // key: line number
  // value: tuples in a line
  for each tuple t in value:
  Emit(t->year, t->temperature);}
- Reduce(key, list of values){
  // key: year
  // list of values: a list of monthly temperatures
  int total_temp = 0;
  for each v in values:
  total_temp= total_temp+v;
  Emit(key, total_temp/size_of(values));}

Combiner is the same as Reducer

MapReduce Example: Average Temperature

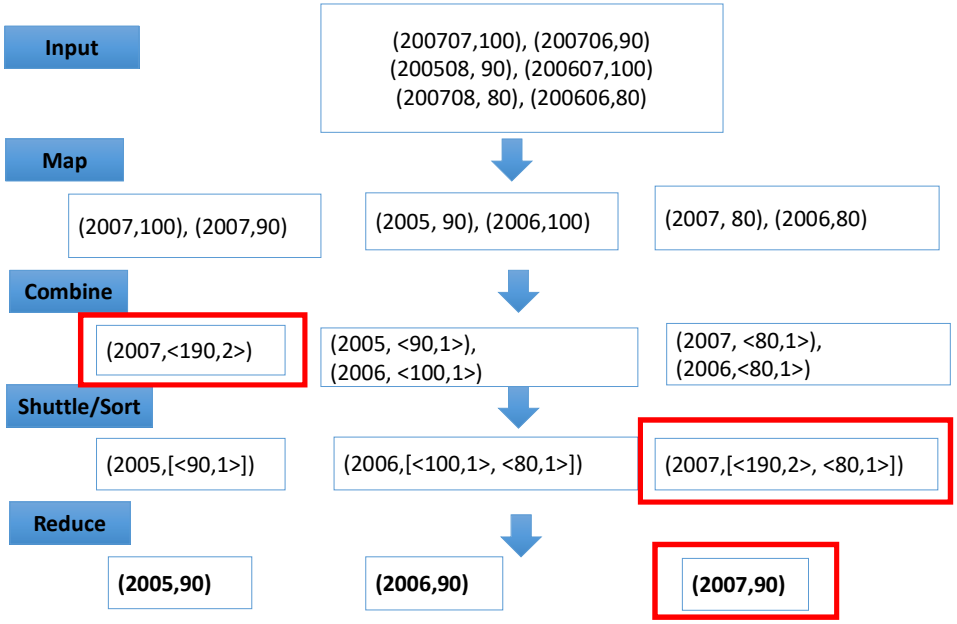


MapReduce In-class Exercise

- **The problem is with the combiner!**
- **Here is a simple counterexample:**
  - (2007, 100), (2007,90) -> (2007, 95)
    (2007,80)->(2007,80)
  - Average of the above is: (2007,87.5)
  - However, the real average is: (2007,90)

- **However, we can do a small trick to get around this**
  - Mapper: (2007, 100), (2007,90) -> (2007, <190,2>)
    (2007,80)->(2007,<80,1>)
  - Reducer: (2007,<270,3>)->(2007,90)

MapReduce Example: Average Temperature



Mapper and Reducer of Average Temperature

- Map(key, value){
  // key: line number
  // value: tuples in a line
  for each tuple t in value:
  Emit(t->year, t->temperature);}
- **Reduce (key, list of values)**{
  // key: year
  // list of values: a list of <temperature sums, counts> tuples
  int total_temp = 0;
  int total_count=0;
  for each v in values:
  total_temp= total_temp+v->sum;
  total_count=total_count+v->count;
  Emit(key,**total_temp/total_count**);}
- **Combine(key, list of values)**{
  // key: year
  // list of values: a list of monthly temperature
  int total_temp = 0;
  for each v in values:
  total_temp= total_temp+v;
  Emit(key,**<total_temp,size_of(values)>**);}

MapReduce In-class Exercise

- **Functions that can use combiner are called *distributive*:**
  - Distributive: Min/Max(), Sum(), Count(), TopK()
  - Non-distributive: Mean(), Median(), Rank()

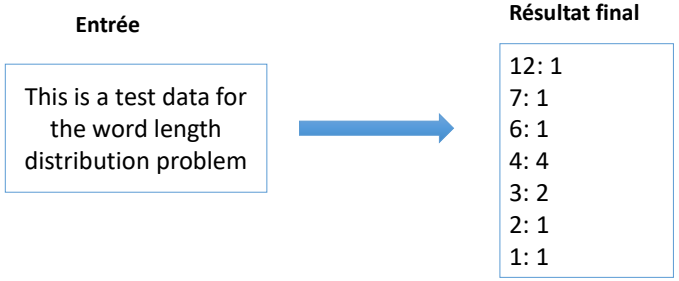
Gray, Jim*, et al. "Data cube: A relational aggregation operator generalizing group-by, cross-tab, and sub-totals." Data Mining and Knowledge Discovery 1.1 (1997): 29-53.

*Jim Gray received Turing Award in 1998

Map Reduce Problems Discussion

- **Problem 1:** Find Word Length Distribution
- **Statement:** Given a set of documents, use Map-Reduce to find the length distribution of all words contained in the documents
- **Question:**
  - What are the Mapper and Reducer Functions?

MapReduce



Mapper and Reducer of Word Length Distribution

- Map(key, value){
  // key: document name
  // value: words in a document
  for each word w in value:
  Emit(length(w), w);}
- Reduce(key, list of values){
  // key: length of a word
  // list of values: a list of words with the same length
  Emit(key, size_of(values));}

Map Reduce Problems Discussion

- **Problem 1:** Find Word Length Distribution
- **Mapper and Reducer:**
  - Mapper(document)
    { Emit (**Length(word), word**) }
  - Reducer(output of map)
    { Emit (Length(word), **Size of (List of words at a particular length))**}