Universidad Internacional de Valencia

Procesamiento de Datos Masivos
03MBID

Tema 2: Big Data y MapReduce
MapReduce: un modelo de programación paralela para Big Data

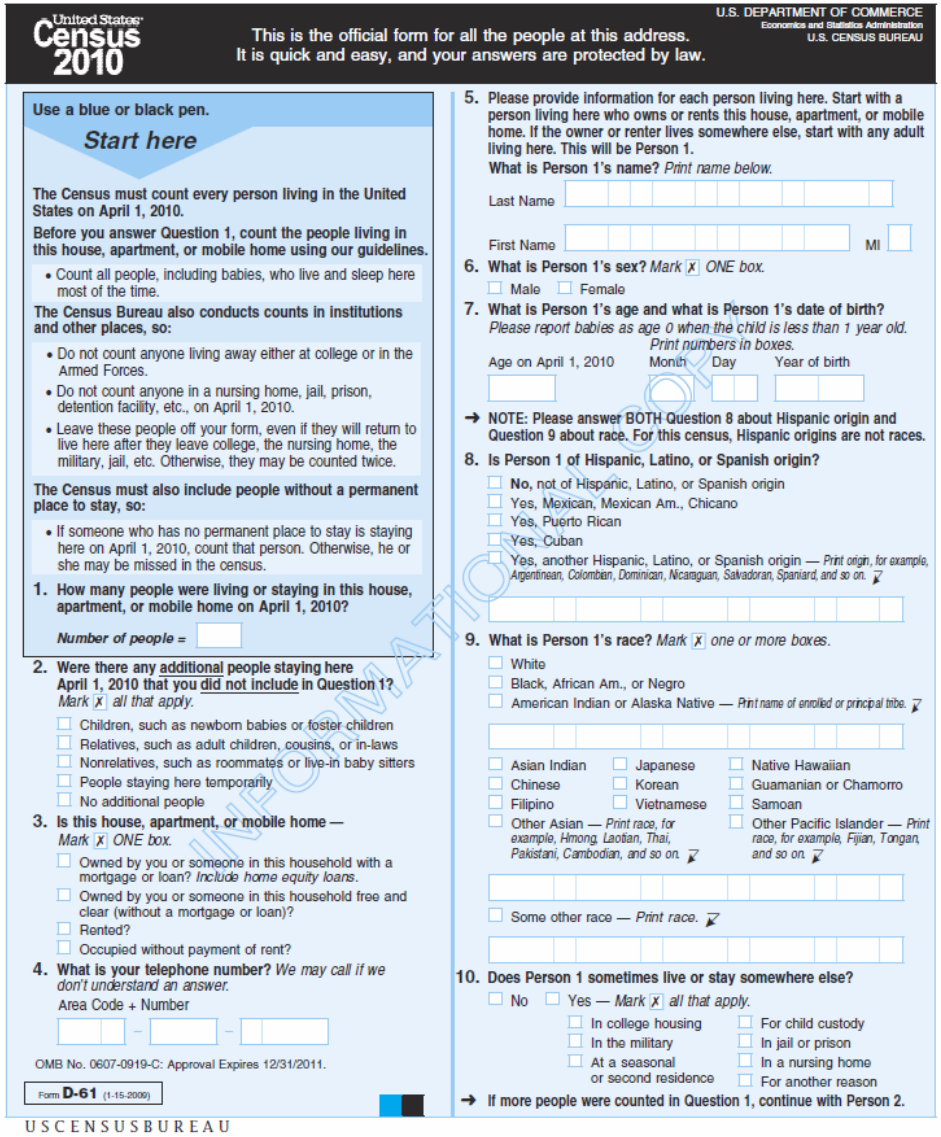
Yudith Cardinale

Diciembre 2022

Analogía: Censo Nacional

- Supongan que tenemos 10,000 empleados, cuyo trabajo es recopilar los formularios de un censo y determinar cuántas personas viven en cada ciudad

- ¿Cómo organizarías estas tareas?



United States Census 2010

This is the official form for all the people at this address.
It is quick and easy, and your answers are protected by law.

U.S. DEPARTMENT OF COMMERCE
Economics and Statistics Administration
U.S. CENSUS BUREAU

Use a blue or black pen.

Start here

The Census must count every person living in the United States on April 1, 2010.

Before you answer Question 1, count the people living in this house, apartment, or mobile home using our guidelines.

- Count all people, including babies, who live and sleep here most of the time.

The Census Bureau also conducts counts in institutions and other places, so:

- Do not count anyone living away either at college or in the Armed Forces.
- Do not count anyone in a nursing home, jail, prison, detention facility, etc., on April 1, 2010.
- Leave these people off your form, even if they will return to live here after they leave college, the nursing home, the military, jail, etc. Otherwise, they may be counted twice.

The Census must also include people without a permanent place to stay, so:

- If someone who has no permanent place to stay is staying here on April 1, 2010, count that person. Otherwise, he or she may be missed in the census.

1. How many people were living or staying in this house, apartment, or mobile home on April 1, 2010?
   Number of people =

2. Were there any additional people staying here April 1, 2010 that you did not include in Question 1?
   Mark [X] all that apply.
   - [ ] Children, such as newborn babies or foster children
   - [ ] Relatives, such as adult children, cousins, or in-laws
   - [ ] Nonrelatives, such as roommates or live-in baby sitters
   - [ ] People staying here temporarily
   - [ ] No additional people

3. Is this house, apartment, or mobile home —
   Mark [X] ONE box.
   - [ ] Owned by you or someone in this household with a mortgage or loan? *Include home equity loans.*
   - [ ] Owned by you or someone in this household free and clear (without a mortgage or loan)?
   - [ ] Rented?
   - [ ] Occupied without payment of rent?

4. What is your telephone number? *We may call if we don't understand an answer.*
   Area Code + Number

OMB No. 0607-0919-C: Approval Expires 12/31/2011.

Form D-61 (1-15-2009)

5. Please provide information for each person living here. Start with a person living here who owns or rents this house, apartment, or mobile home. If the owner or renter lives somewhere else, start with any adult living here. This will be Person 1.
   What is Person 1's name? *Print name below.*
   Last Name
   First Name MI

6. What is Person 1's sex? *Mark [X] ONE box.*
   - [ ] Male
   - [ ] Female

7. What is Person 1's age and what is Person 1's date of birth?
   Please report babies as age 0 when the child is less than 1 year old.
   Print numbers in boxes.
   Age on April 1, 2010 | Month | Day | Year of birth

→ NOTE: Please answer BOTH Question 8 about Hispanic origin and Question 9 about race. For this census, Hispanic origins are not races.

8. Is Person 1 of Hispanic, Latino, or Spanish origin?
   - [ ] No, not of Hispanic, Latino, or Spanish origin
   - [ ] Yes, Mexican, Mexican Am., Chicano
   - [ ] Yes, Puerto Rican
   - [ ] Yes, Cuban
   - [ ] Yes, another Hispanic, Latino, or Spanish origin — *Print origin, for example, Argentinean, Colombian, Dominican, Nicaraguan, Salvadoran, Spaniard, and so on.*

9. What is Person 1's race? *Mark [X] one or more boxes.*
   - [ ] White
   - [ ] Black, African Am., or Negro
   - [ ] American Indian or Alaska Native — *Print name of enrolled or principal tribe.*
   - [ ] Asian Indian
   - [ ] Chinese
   - [ ] Filipino
   - [ ] Japanese
   - [ ] Korean
   - [ ] Vietnamese
   - [ ] Other Asian — *Print race, for example, Hmong, Laotian, Thai, Pakistani, Cambodian, and so on.*
   - [ ] Native Hawaiian
   - [ ] Guamanian or Chamorro
   - [ ] Samoan
   - [ ] Other Pacific Islander — *Print race, for example, Fijian, Tongan, and so on.*
   - [ ] Some other race — *Print race.*

10. Does Person 1 sometimes live or stay somewhere else?
    - [ ] No
    - [ ] Yes — *Mark [X] all that apply.*
      - [ ] In college housing
      - [ ] In the military
      - [ ] At a seasonal or second residence
      - [ ] For child custody
      - [ ] In jail or prison
      - [ ] In a nursing home
      - [ ] For another reason

→ If more people were counted in Question 1, continue with Person 2.

USCENSUSBUREAU

La situación puede ser más complicada

- Supongamos que los trabajadores se van de vacaciones, se enferman, trabajan a diferentes ritmos
- Supongamos que algunos formularios se completan incorrectamente y requieren correcciones o deben desecharse
- ¿Qué pasa si el supervisor se enferma?
- ¿Qué tan grandes deben ser las pilas de trabajo?
- ¿Cómo se puede monitorear el progreso?
- ...

Un poco de instrospección

- ¿Cuál es el principal desafío?
  - ¿Son las tareas individuales complicadas?
  - Si no es así, ¿qué es lo que lo hace complicado?
- ¿Qué tan resistente es nuestra solución?
- ¿Qué tan bien balancedo es el trabajo entre los empleados?
  - ¿Qué factores afectan esto?
- ¿Qué tan general es el conjunto de técnicas usadas?

¡No queremos lidiar con todo esto!

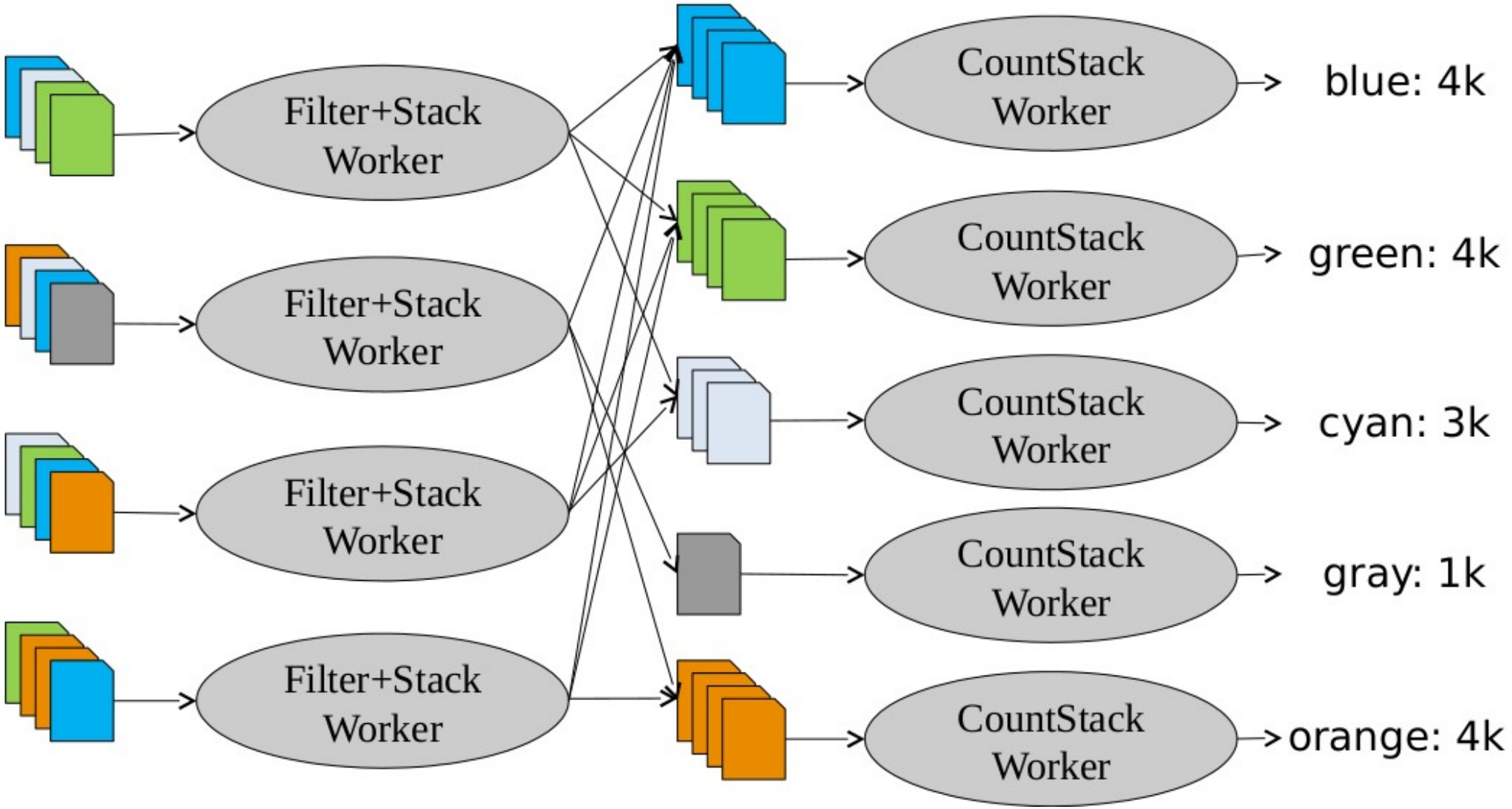
- ¿No sería mejor si hubiera algún sistema que se encargara de todos estos detalles?
- Idealmente, solo le diríamos al sistema lo que debe hacerse
- Esto es lo que pretende MapReduce.

¿Qué necesitamos?

- Escalabilidad con grandes volúmenes de datos.
  - 1000’s de máquinas.
  - 10,000’s de discos
- Equipos y redes de bajo coste.
  - poco fiables.
  - elevadas latencias
- Facilidad de programación
- Tolerancia a fallos automática

Abstracción en un flujo de datos digitales

viu Universidad Internacional de Valencia



Un poco más de abstracción

- Hay dos tipos de trabajadores:
  - Aquellos que toman elementos de datos de entrada y producen elementos de salida para las "pilas"
  - Aquellos que toman las pilas y agregan los resultados para producir salidas por pila
- Podemos llamarlos:
  - map: toma (item_key, value), produce uno o más pares (stack_key, value')
  - reduce: toma (stack_key, {set of value'}), produce uno o más resultados de salida – típicamente (stack_key, agg_value)
  - stack_key constituye reduce key

¿Qué es MapReduce?

- Modelo de programación ***data-parallel*** diseñado para escalabilidad y tolerancia a fallos en grandes sistemas de commodity hardware
- Combina operaciones Map y Reduce con una implementación asociada
- Usado para el procesamiento y generación de grandes conjuntos de datos
- Propuesto inicialmente por Google (2004)
  - Usado en múltiples operaciones
  - Procesa varios PB de datos por día
- Popularizado por la implementación open-source del proyecto Apache Hadoop
  - Usado por múltiples organizaciones como Facebook, Twitter, eBay, LinkedIn, Rackspace, Yahoo!, AWS, etc.

¿Para qué se usa?

- **En Google:**
  - Construcción de índices para el buscador (page rank).
  - Clustering de artículos en Google News.
  - Búsqueda de rutas en Google Maps.
  - Traducción estadística
- **En Facebook**:
  - Minería de datos.
  - Optimización de ads.
  - Detección de spam.
  - Gestión de logs
- **En Yahoo!**:
  - Construcción de índices para el buscador (Yahoo! Search)
  - Detección de SPAM (Yahoo! mail)

¿Para qué se usa?

- **En investigación**::
  - Análisis de conflictos en Wikipedia (PARC)
  - Procesamiento de lenguage natural (CMU)
  - Bioinformática (Maryland)
  - Física de partículas (Nebraska)
  - Simulación de clima oceánico (Washington)
  - Mantenimiento predictivo (Airbus, France)
  - <Your application here>

Big Ideas detrás de MapReduce

- **Scale “out”, no “up”**
  - Gran número de clusters de servidores commodity
  - ¿Se requiere más potencia? – scale-out es fácil
- **Escalabilidad perfecta**
  - Scaling “out” mejora el desempeño de un algoritmo sin realizarle modificaciones
- **Asumir que las fallas son comunes**
  - Los nodos baratos fallan, especialmente si hay muchos
  - Tiempo promedio de falla (MTBF) para 1 nodo = 3 años
  - MTBF para 1000 nodos = 1 día
  - MTBF para 10000 nodos, 10 fallas por día
  - **Solución**: implementar tolerancia a fallos en el sistema

Big Ideas detrás de MapReduce

- **Mover el procesamiento a los datos**
  - Commodity network = bajo ancho de banda
  - Tomar ventaja de la localidad de datos y evitar la transferencia de grandes datasets a través de la red
  - **Solución**: Llevar el cómputo a los datos

- **Esconder detalles del nivel del sistema a los desarrolladores de aplicaciones**
  - La programación de sistemas distribuidos es difícil
  - Los desarrolladores se quieren enfocar en sus problemas en lugar de lidiar con los aspectos de programación distribuida
  - **Solución**: Los usuarios escriben funciones data-parallel "`map`" y "`reduce`", el sistema maneja la distribución del trabajo y los fallos

El modelo de programación MapReduce

- Primitivas de programación funcional distribuida simple (en java, C++, Python, bash, ...)
- Modelado a partir de primitivas Lisp:
  - `map` (la función se aplica a todos los items en una colección) y
  - `reduce` (la función se aplica al conjuno de items con una clave común)
- Se comienza con:
  - Una función definida-por-usuario que se aplica a todos los datos,
    `map`: (key,value) → (key, value)
  - Otra operación definida-por-usuario
    `reduce`: (key, {set of values}) → result
  - Un conjunto de *n* nodos, cada uno con datos
- Todos los nodos ejecutan `map` sobre todos sus datos, produciendo nuevos datos con claves nuevas
  - Estos datos se coleccionan por key, luego se particionan (shuffled), y finalmente `reduced`
  - El dataflow es a través de ficheros temporales en GFS (Google File System)

Abstracción Map

- Map function:

(K_{input}, V_{input}) ➡ $list(K_{inter}, V_{inter})$

- Recibe un par key-value como input
  - `key` es una referencia al valor del input y
  - `value` es el conjunto de datos sobre el cual se opera

- Evaluación
  - Una función definida-por-usuario
  - Aplicada a cada valor plied to every `value`
    - Podría requerir “parsear” el input

- Produce una nueva lista de pares key-value
  - Puede ser diferente del par de entrada

Abstracción Reduce

- Reduce function:

$(K_{inter}, list(V_{inter})) \rightarrow list(K_{out}, V_{out})$

- Comiena con un gran número de pares intermedios key-value
  - Los pares del input se ordenan por by key
- Evaluación
  - Una función definida-por-usuario
  - El iterador proporciona los valores de una determinada key a la función reduce
- Produce una lista final de pocos pares key-value

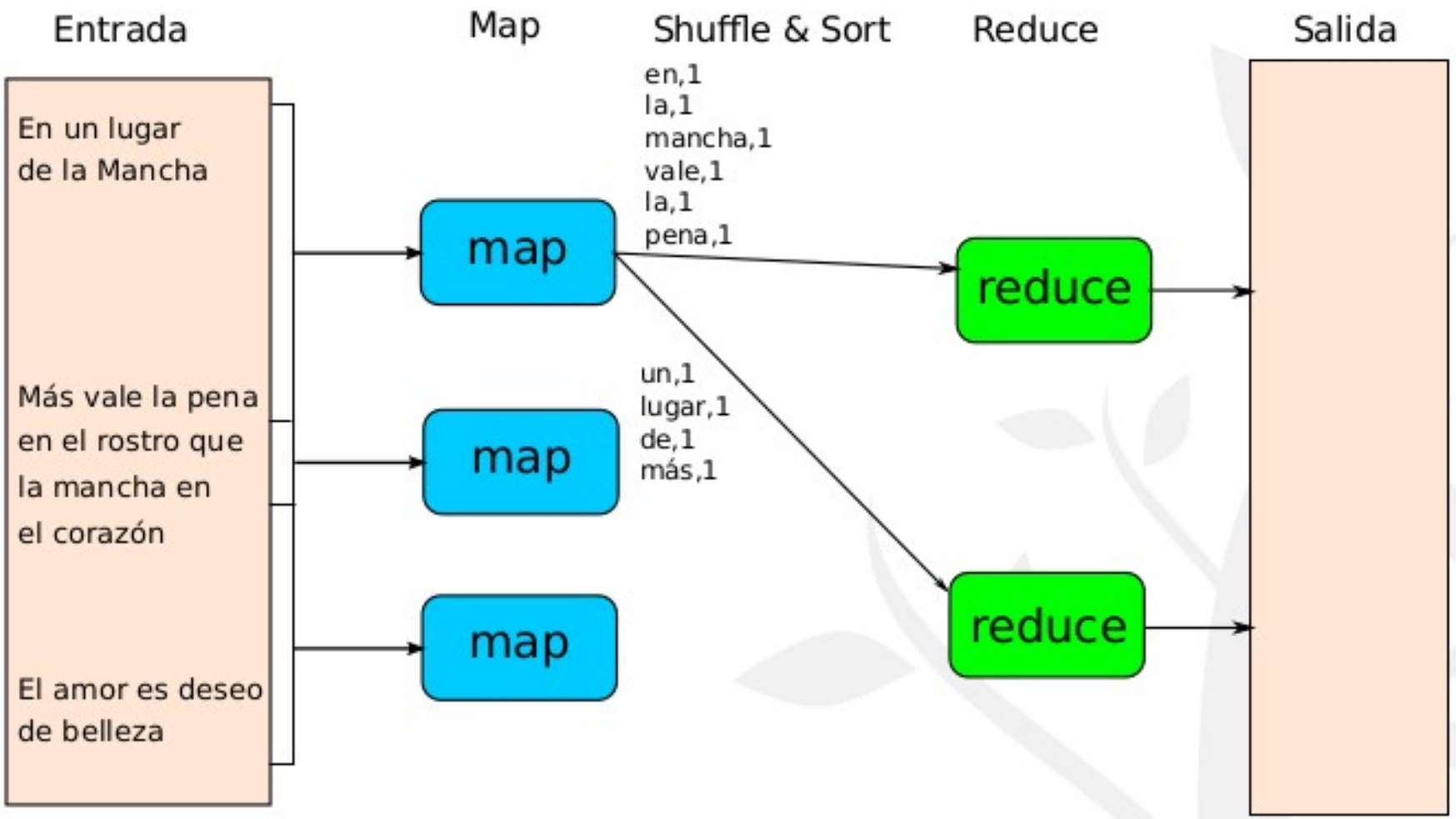
Ejemplo simple: Word count

```
map(String key, String value) {
   // key: document name, line no
   // value: contents of line
    for each word w in value:
       emit(w, "1")
}
```

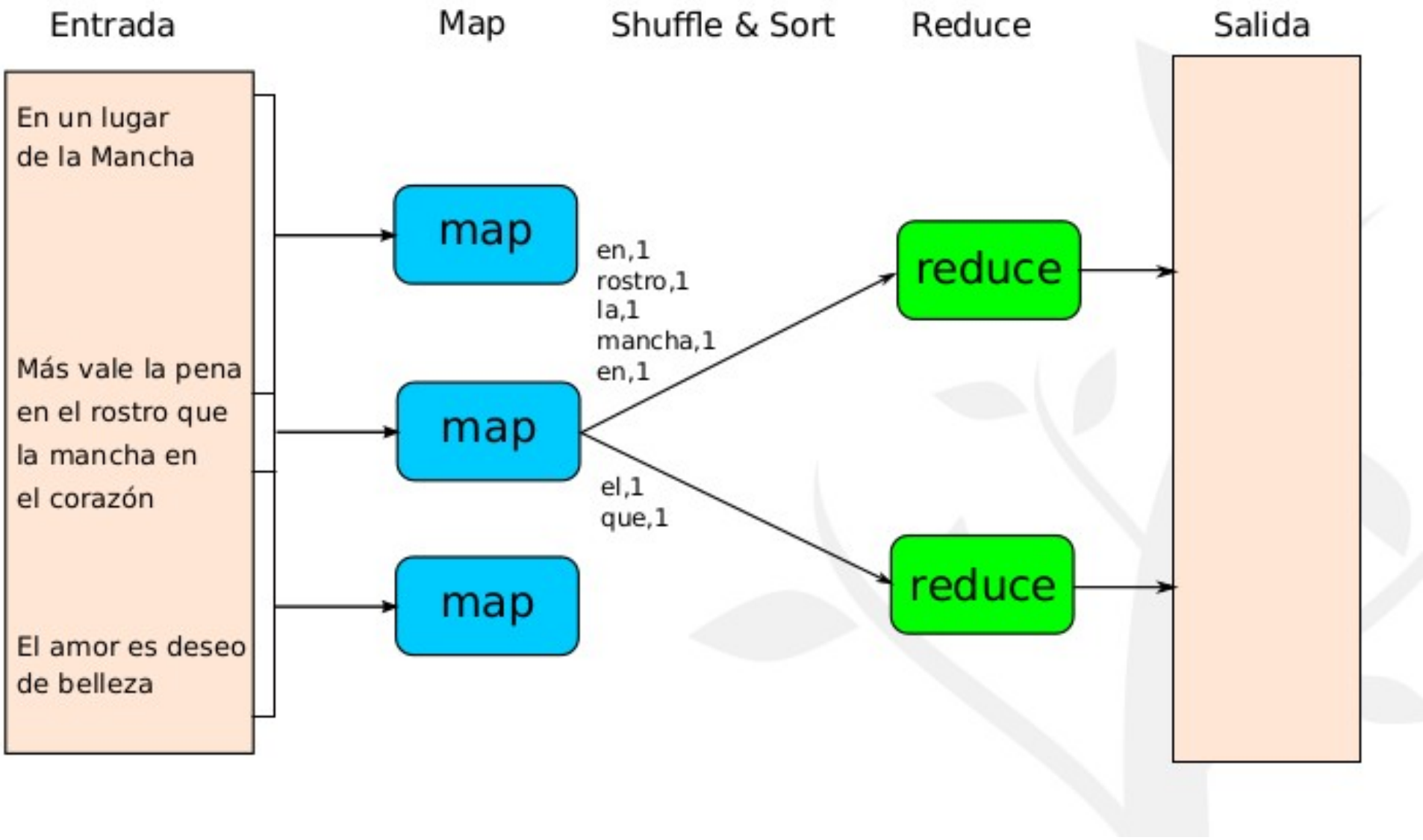
```
reduce(String key, Iterator values) {
  // key: a word
  // values: a list of counts
  int result = 0;
  for each v in values:
    result += ParseInt(v);
  emit(key, result)
}
```

- Objetivo: dado un conjunto de documentos, contar la frecuencia de cada palabra
  - Input: Key-value pairs (document:lineNumber, text)
  - Output: Key-value pairs (word, #occurrences)
  - What should be the intermediate key-value pairs?

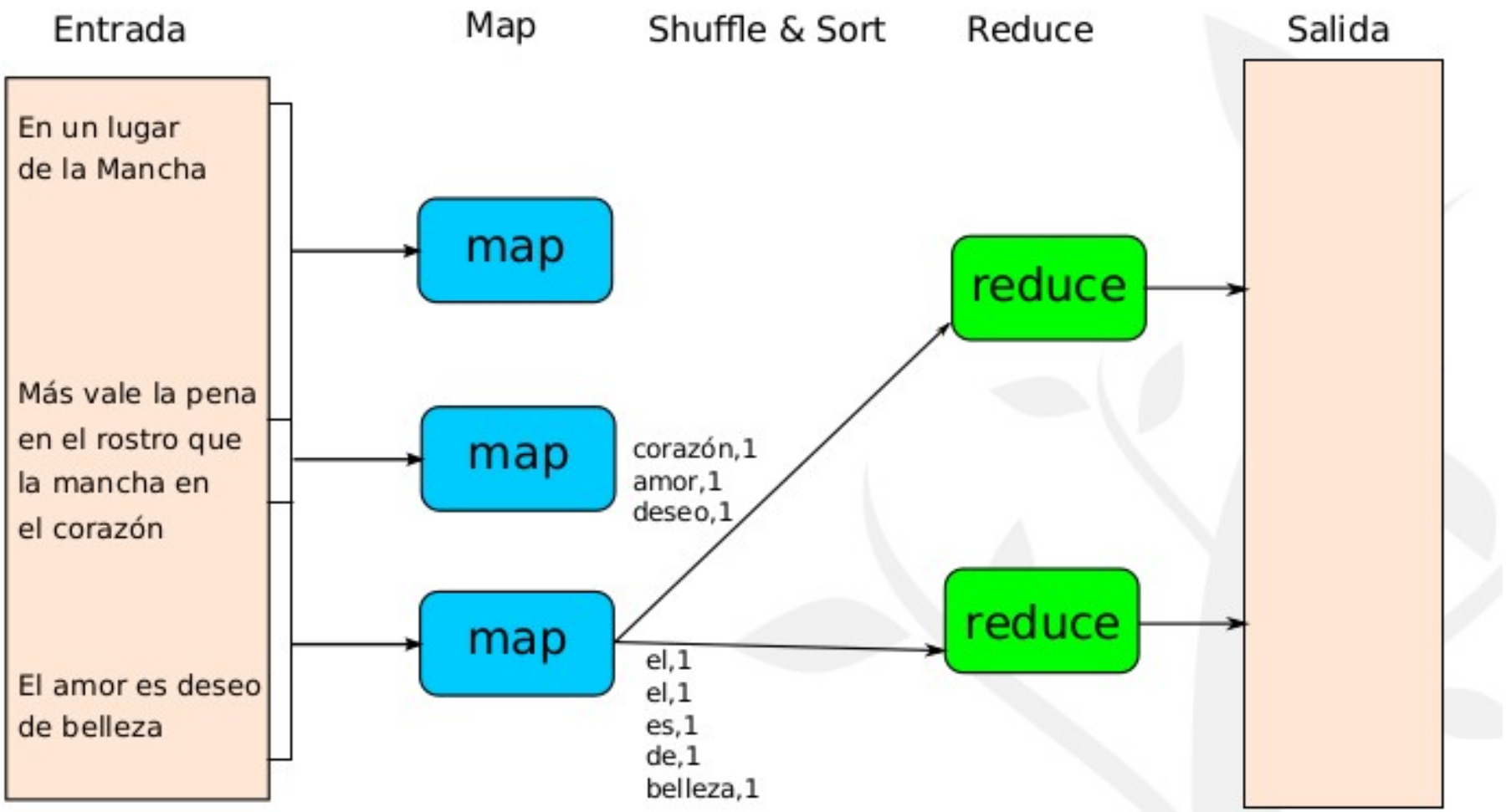
Ejemplo simple: Word count



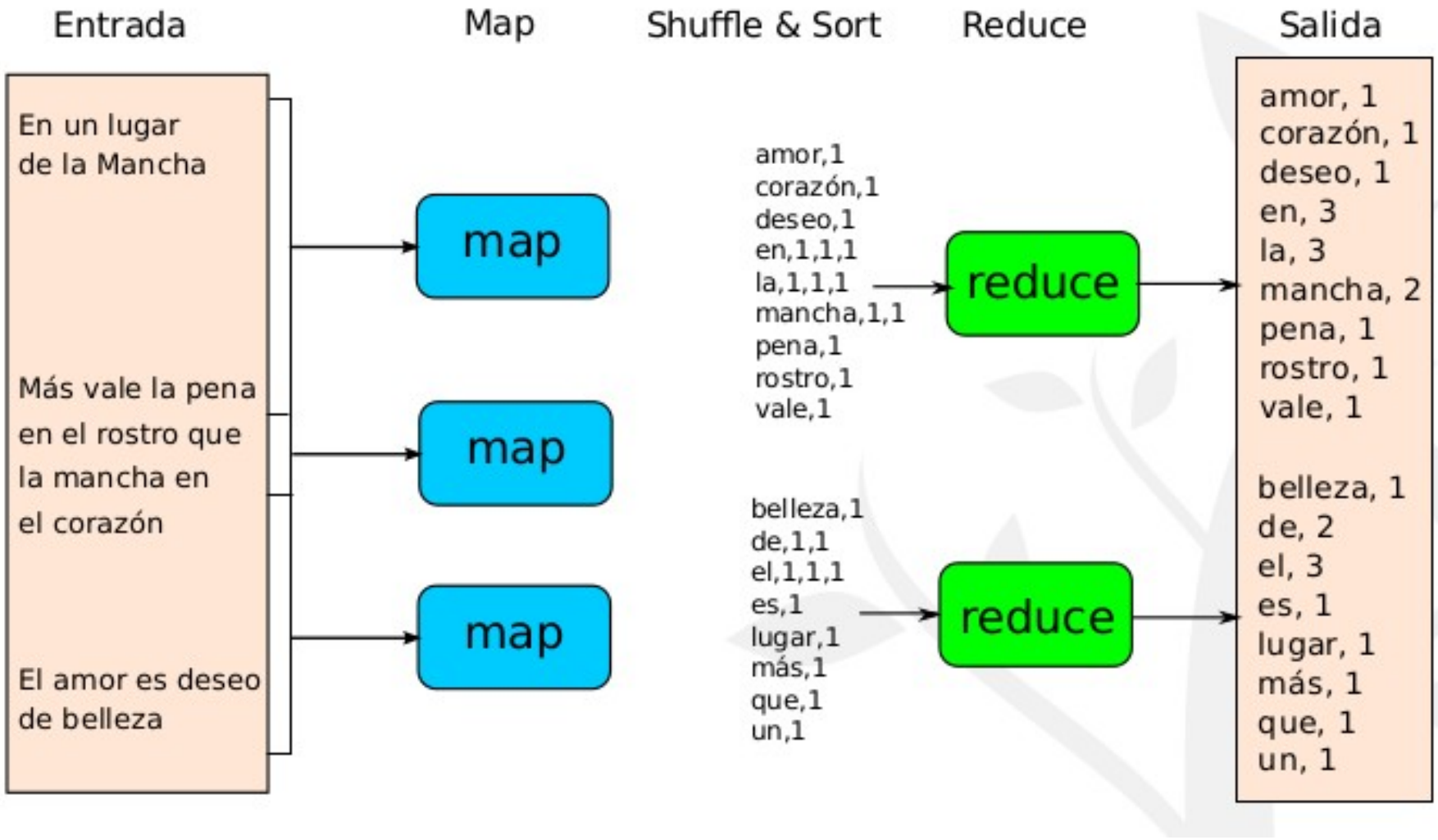
Ejemplo simple: Word count



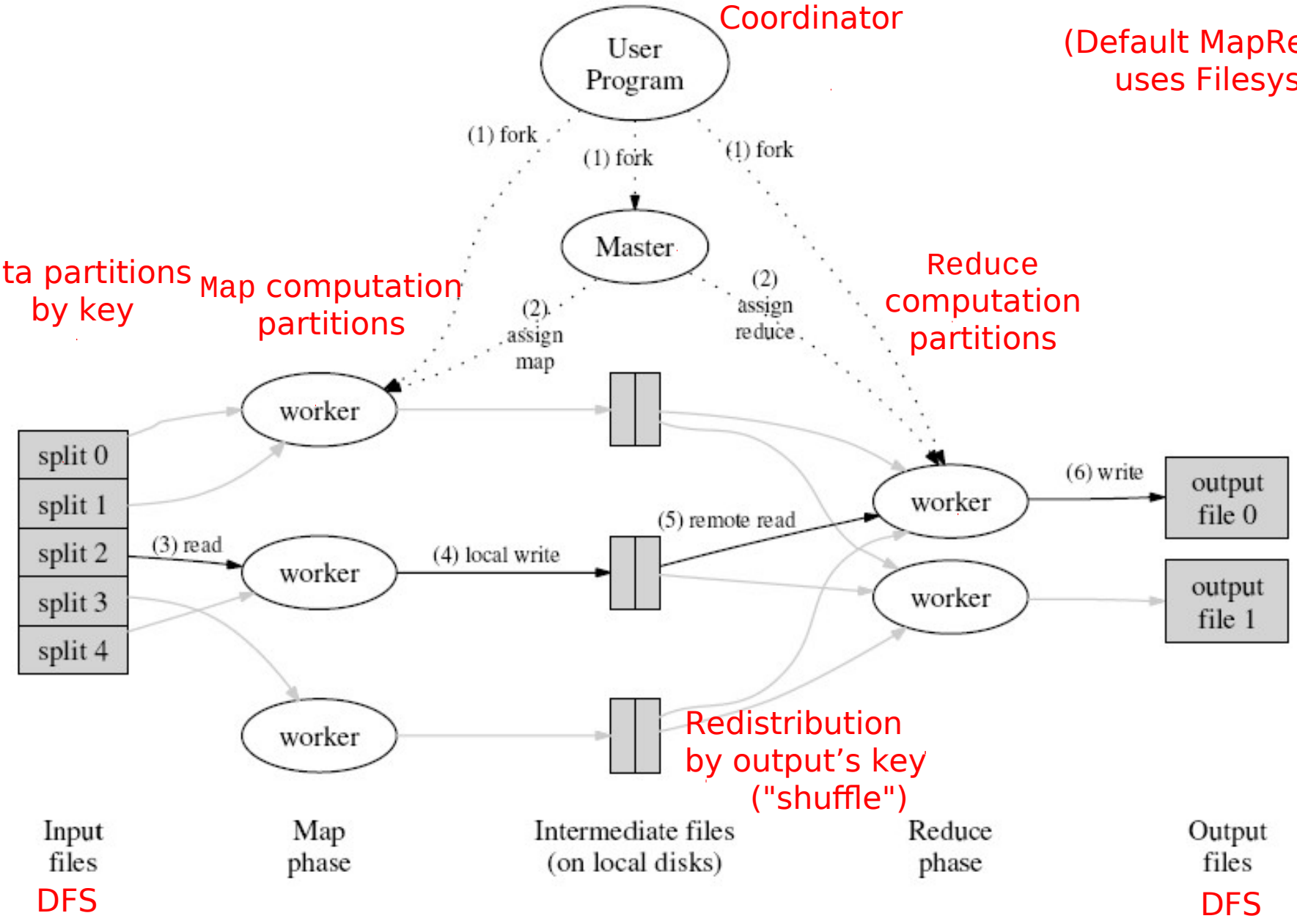
Ejemplo simple: Word count



Ejemplo simple: Word count



Detalles de ejecución MapReduce: data flow



Más detalles de ejecución de MapReduce

- Sistema de ficheros distribuido para gestionar los datos
  - GFS (Google File System),
  - HDFS (Hadoop Distributed FS).
  - Ficheros divididos en bloques grandes (64-128 MB en HDFS), con replicación
- Arquitectura *master-worker*
- Mappers procesan splits de datos
  - preferiblemente se colocan en el mismo nodo o rack donde estén sus datos de entrada
  - Mover el cómputo a los datos, minimiza el uso de la red
- Mappers salvan sus salidas a discos locales ante de enviárselas a los reducers
  - Permite tener más reducers que nodos
  - Permite la recuperación si un reducer falla
- Reducers graban su salida en el sistema distribuido.
  - Replicación en los resultados.
  - Un fichero de salida por cada reduce

Más detalles de ejecución de MapReduce

Coordinación de tareas:

- El Master se encarga de coordinar las diferentes tareas.
  - Estado de cada tarea: *idle*, *in-progress*, *completed*.
  - Las tareas *idle* se planifican a medida que van quedando workers libres

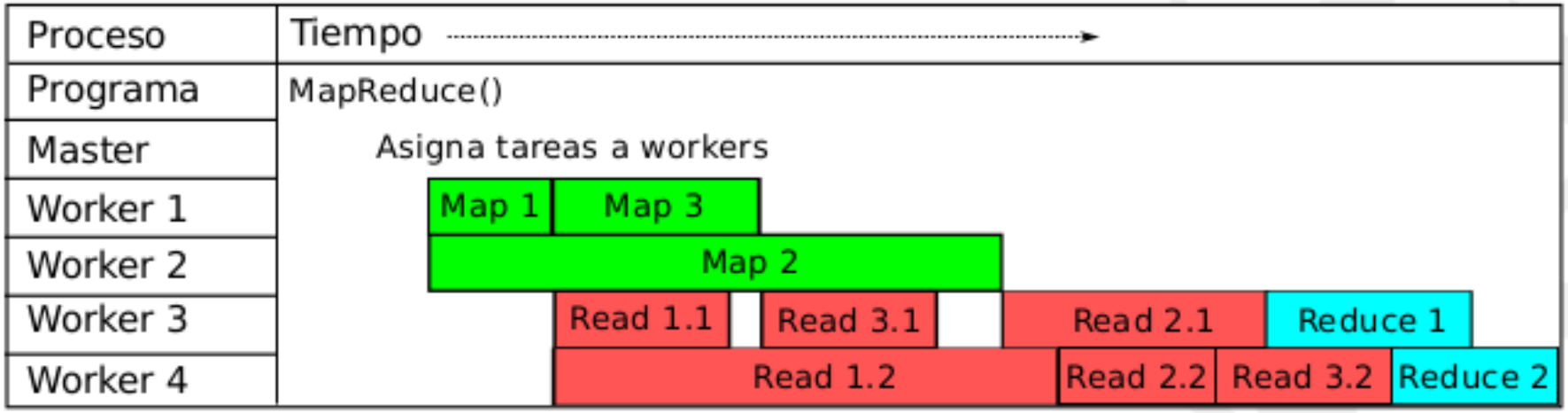
- Cuando una tarea map acaba, le envía al Master la localización y tamaño de sus ficheros intermedios, uno por tarea reduce

- El Master envía esta información a las tareas reduce

Más detalles de ejecución de MapReduce

Solapamiento de tareas:

- La fase de reduce no puede comenzar hasta que se complete la de map
- Se puede simultanear la ordenación/mezcla con la ejecución del map
  - Mejora el balance de carga dinámico



Tolerancia a fallos

- Mediante *heartbeats*, el Master detecta fallos en los workers
- Si falla una tarea map o reduce
  - La tarea se reintenta en otro nodo
    - Correcto para los map porque no tienen dependencias
    - Correcto para los reduce porque las salidas de los maps están en disco
    - Necesario que las tareas no tengan efectos colaterales.
  - Si la misma tarea falla repetidamente, el trabajo MapReduce se aborta y se notifica al usuario (ajustable)
- Si falla un nodo completo
  - Relanzar sus tareas en curso en otros nodos.
  - Reejecutar cualquier map que se hubiera ejecutado en el nodo
    - Necesario pues las salidas de los map se perdieron

Tolerancia a fallos

- Si una tarea no progresa (*straggled* o rezagada)
  - Se lanza una segunda copia de la tarea en otro nodo (ejecución especulativa).
  - Se toma la salida de la tarea que acabe antes, y se mata a la otra.
  - Situación bastante común en clusters grandes
    - Debidos a errores hardware, bugs software, fallos de configuración, etc.
    - Una tarea rezagada puede relentizar de forma importante un traba.

- Si falla el Master
  - Se intenta relanzar de nuevo
    - Las tareas en proceso o acabadas durante el reinicio, se relanza
  - Si continúa fallando, el trabajo se aborta y se le notifica al usuario

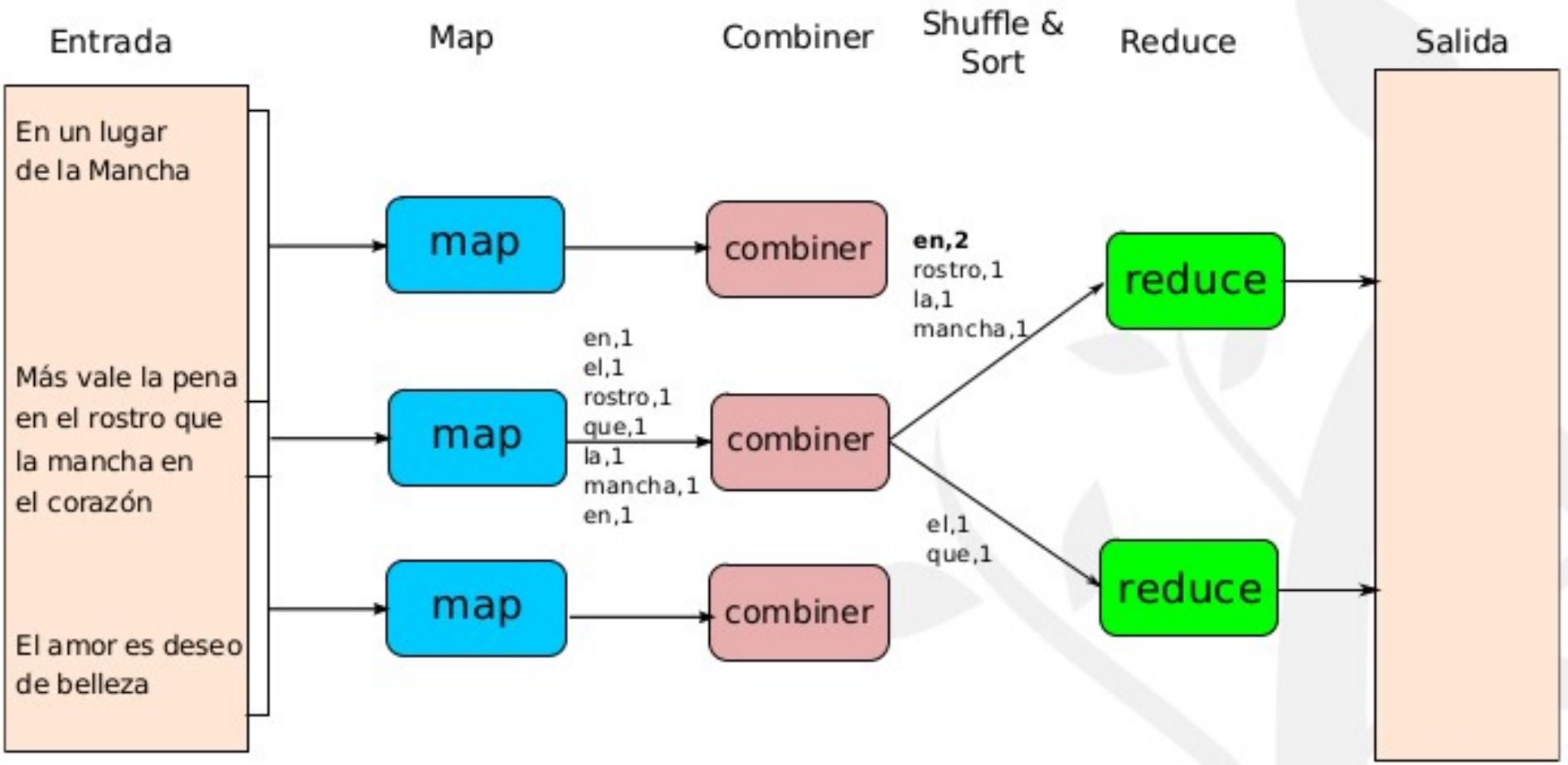
Combinador

- Un combinador (*Combiner*) es una función de agregación local para las claves repetidas de cada map
- Puede ahorrar ancho de banda al reducir el tamaño de los datos intermedios del map

$$combiner(K^m_2, list(V^m_2)) \rightarrow (K^m_3, list(V^m_3))$$

- Habitualmente, misma función que el reducer
- Se ejecuta en el mismo nodo que el map
- Solo puede utilizarse si la función reduce es commutativa y asociativa

Combinador


Entrada
Map
Combiner
Shuffle & Sort
Reduce
Salida
En un lugar de la Mancha
Más vale la pena en el rostro que la mancha en el corazón
El amor es deseo de belleza
map
map
map
combiner
combiner
combiner
en,1
el,1
rostro,1
que,1
la,1
mancha,1
en,1
en,2
rostro,1
la,1
mancha,1
el,1
que,1
reduce
reduce

Particionador

- El particionador (*shuffler*) por defecto es un hash de las claves:
  - `hash(K) mod R`
- Garantiza:
  - Claves iguales van al mismo reducer
  - Carga de los reducers relativamente bien balanceada (en muchos casos)
- Hay situaciones en las que puede interesar cambiarlo:
  - Ejemplo: `hash(hostame(URL))modR`.
  - Todas las URLs de cada host se juntan en el mismo fichero de salida

Implementaciones

Implementaciones *open source* y en la nube:

- Hadoop: Implementación open source de MapReduce
- Amazon Elastic MapReduce: ejecución simple de Apache Hadoop, Spark, HBase, Hive, y otras aplicaciones Big Data
- Microsoft Azure HDInsight: servicio en la nube totalmente administrado para el procesamiento Big Data
- Google Cloud Dataproc: Apache Hadoop y Apache Spark nativos en la nube

Conclusiones

- El modelo de programación MapReduce oculta la complejidad de la distribución del trabajo y la tolerancia a fallos
- Principales aspectos de diseño:
  - Altamente escalable, maneja los fallos hardware.
  - Reduce los costes del hardware, programación y administración
- No es adecuado para todos los problemas, pero cuando funciona puede ahorrar mucho tiempo
- Muy apropiado para la ejecución en la nube