Introducción a Google Map Reduce

Dr. Víctor J. Sosa

¿Qué es MapReduce?

- Un modelo de programación y su implementación asociada
- Procesa una cantidad importante de datos
- Explota un grupo grande de computadoras
- Ejecuta los procesos de manera distribuida
- Ofrece un alto grado de transparencia
- Antecedentes:
  - Idea originada en Google
  - Utilizado para el cálculo del PageRank

Motivación

- La cantidad de datos de entrada es grande.
  - Toda la Web, millones de páginas.
- Muchas computadoras disponibles
  - Se trata de usarlas eficientemente.
- Contexto:
  - Muchos archivos o archivos enormes (100's de GB o TB)
  - Los datos no se actualizan en el lugar
  - Son más usuales las operaciones de read/append

Modelo de Programación

- El programador define dos funciones:

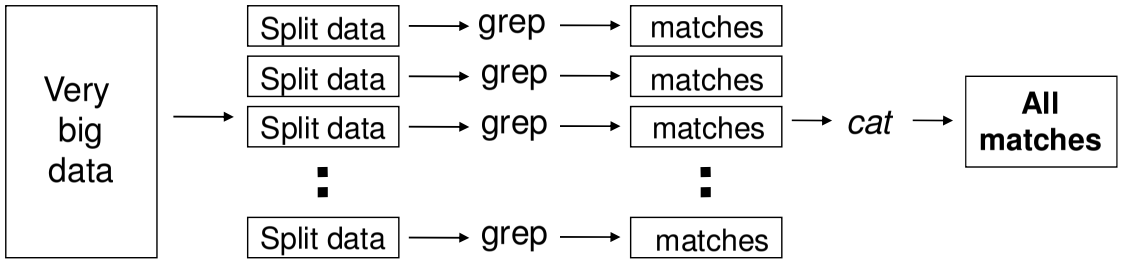
map (in_key, in_value) ➔ list(out_key, intermediate_value)

- Los procesos reciben como entrada pares key/value
- Producen un conjunto de pares key/lista de valores intermedios

reduce (out_key, list(intermediate_value)) ➔ list(out_value)

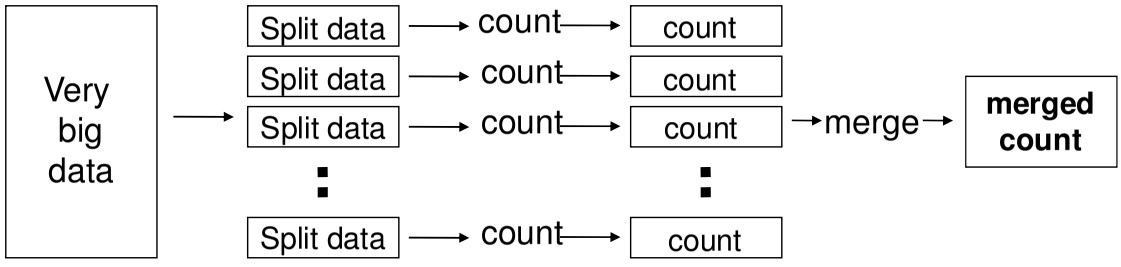
- Combina todo los valores intermedios para una clave (key) en particular (se podría hacer esto como un módulo previo)
- Produce un conjunto fusionado de valores como salida (usualmente sólo uno)

Ejemplo: Grep distribuido



Ejemplo: Conteo distribuido

- Frecuencia de palabras en páginas web (word, frequency)
- Conteo de URLs origen que hacen referencia a URLs destino (target, source)



Ejemplo: Conteo de palabras

Page 1: the weather is good
Page 2: today is good
Page 3: good weather is good.

Map: Salida

- Proceso 1:
  - (the 1), (weather 1), (is 1), (good 1).
- Proceso 2:
  - (today 1), (is 1), (good 1).
- Proceso 3:
  - (good 1), (weather 1), (is 1), (good 1).

Ejemplo: Conteo de palabras

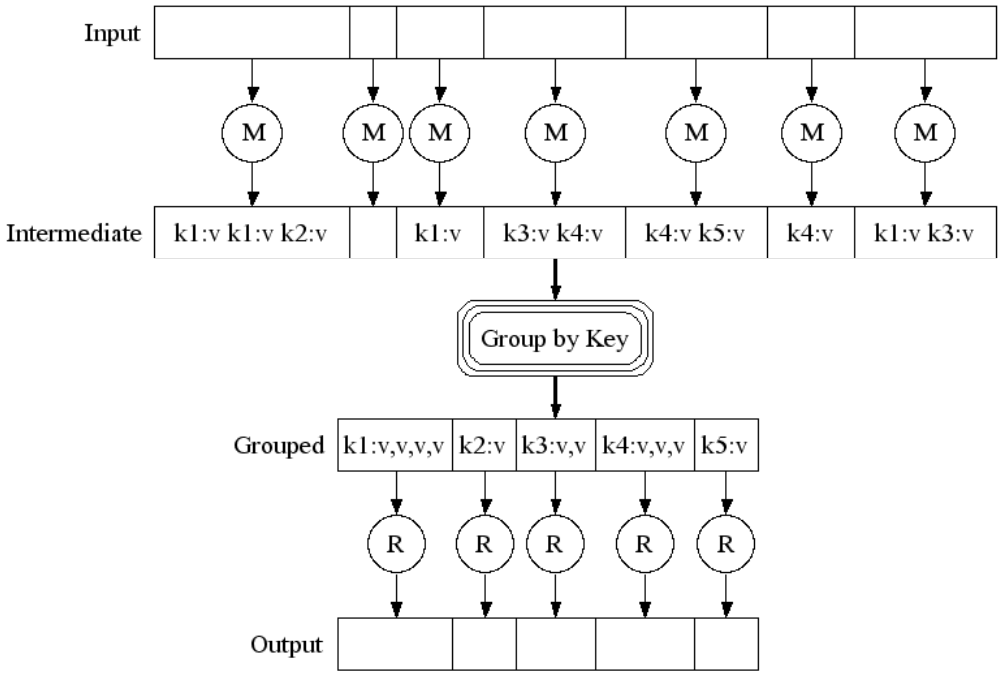
Reduce: Entrada

- Proceso 1:
  - (the 1)
- Proceso 2:
  - (is 1), (is 1), (is 1)
- Proceso 3:
  - (weather 1), (weather 1)
- Proceso 4:
  - (today 1)
- Proceso 5:
  - (good 1), (good 1), (good 1), (good 1)

Reduce: Salida

- Proceso 1:
  - (the 1)
- Proceso 2:
  - (is 3)
- Proceso 3:
  - (weather 2)
- Proceso 4:
  - (today 1)
- Proceso 5:
  - (good 4)

Función de particionamiento



M: Map
R:Reduce

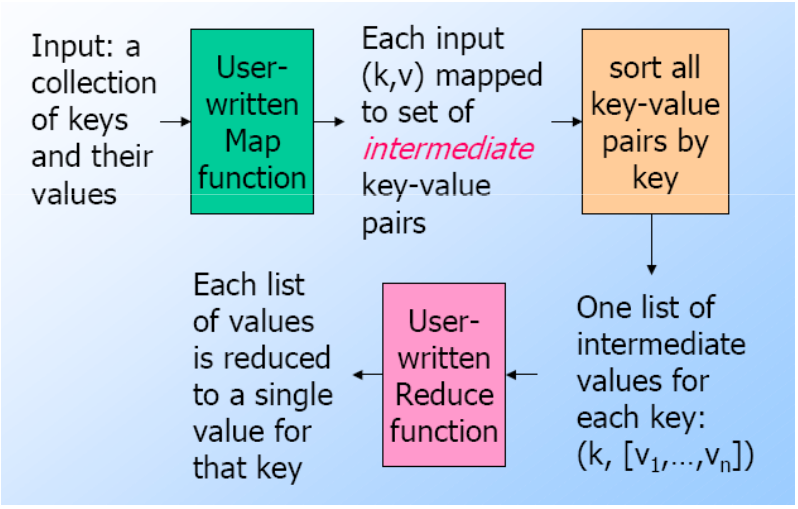
Función de Particionamiento

- Default : (`hash(key) mod R`)
- Garantiza:
  - Particiones relativamente bien balanceadas
  - Orden dentro de las particiones (con base en su llave)
- Clasificación distribuida
  - Map:
    `emit(key,value)`
  - Reduce (con R=1):
    `emit(key,value)`

Implementación

- Un programa crea (*fork*) un proceso master y varios procesos worker.
- La entrada es particionada en un número determinado de splits.
- Los procesos worker son asignados tanto para ejecutar **Map** en una de las particiones (*splits*) o **Reduce** para algún conjunto de claves intermedias

Map-Reduce Framework

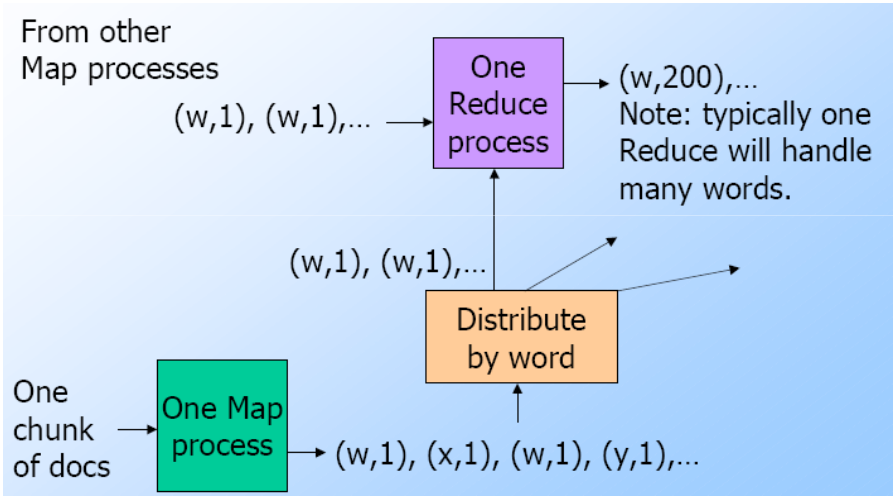


Conteo de palabras utilizando Map-Reduce

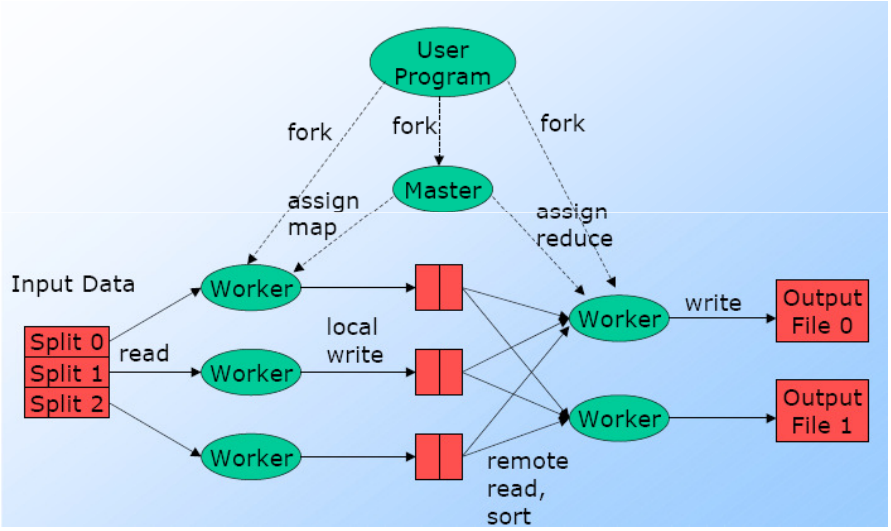
```
map(key, value):
// key: doc name;      value: text of doc
for each word w in value: emit(w, 1);


reduce(key, values):
// key: a word; values: list of counts
result = 0;
for (each count v in values)
result += v;
emit(key, result);
```

Ejecución de MapReduce



Ejecución distribuida



Responsabilidad del Master

1. Asignar tareas de Map y Reduce a los *workers*
2. Verificar que ningún *worker* haya muerto (quizás debido a que su procesador falló).
3. Comunicar los resultados de la tarea Map a la tarea Reduce.

Comunicación de Map a Reduce

- Seleccionar un número R de tareas Reduce.
- Dividir las llaves intermedias en R grupos, e.g. usando hashing
- Cada tarea Map crea, en su procesador, R archivos con pares key-value intermedios, uno para cada tarea Reduce

Combinadores

- A menudo una tarea Map producirá pares (k,v1), (k,v2), … para la misma clave k.
- Ejemplo: En el proceso de conteo de palabras, lo produce para casi todas excepto las muy raras.
- Ahorra tiempo mediante ir pre-agregando en la tarea Map.
- Esto es, utiliza la función Reduced sobre la lista parcial de valores producidos por una tarea Map
- Funciona bien sólo si la función Reduce es conmutativa y asociativa
- Ejemplo: Funciona bien para el caso del conteo de las palabras

PageRank y Map-Reduce

- Splits = conjunto de páginas (keys), su PageRank actual
- Map: Produce un conjunto de pares intermedios

  (p, r) = las páginas p en el *Split* expresan su PageRank r.
- Reduce: suma las contribuciones del PageRank

MapReduce: Transparencia

- Google (Distributed) File System (GFS)
  - Paralelización
  - Tolerancia a fallos
  - Busca optimizar la ubicación de los datos
    - Almacenar los datos donde se van a utilizar o al menos "cerca", en el mismo switch (Locality optimization)
  - Balanceo de carga

Tolerancia a Fallos

- Falla en el proceso "worker":
  - Se detectan los fallos vía un "heartbeat" periódico
  - Re-ejecuta las tareas *Map* completadas y en progreso
  - Re-ejecuta las tareas *Reduce* en progreso
  - La finalización de la tarea se compromete a través del proceso master
- Falla en el proceso "master":
  - Pudiera manejarse, pero aún no es posible (las fallas en el maestro son menos probables)

Mejoras

- Diferentes funciones de particionamiento
- Incluir función "Combiner".
- Tipos diferentes de input/output.
- Ignora registros inapropiados (bad records).
- Ejecución local.
- Manejo de información de estado.
- Contadores.

MapReduce: desde fuera de Google

- Hadoop (Java)
  - Emula MapReduce y GFS
- La arquitectura de Hadoop MapReduce y DFS es master/slave

	Master	Slave
MapReduce	jobtracker	tasktracker
DFS	namenode	datanode

Example Word Count (1)

- Map

```
public static class MapClass extends MapReduceBase
implements Mapper {
  private final static IntWritable one = new IntWritable(1);
  private Text word = new Text();

  public void map(WritableComparable key, Writable value,
  OutputCollector output, Reporter reporter)
  throws IOException {
    String line = ((Text)value).toString();
    StringTokenizer itr = new StringTokenizer(line);
    while (itr.hasMoreTokens()) {
      word.set(itr.nextToken());
      output.collect(word, one);
    }
  }
}
```

Example Word Count (2)

- Reduce

```
public static class Reduce extends MapReduceBase implements
Reducer {
  public void reduce(WritableComparable key, Iterator
  values, OutputCollector output, Reporter reporter)
  throws IOException {
    int sum = 0;
    while (values.hasNext()) {
      sum += ((IntWritable) values.next()).get();
    }
    output.collect(key, new IntWritable(sum));
  }
}
```

Example Word Count (3)

- Main

```
public static void main(String[] args) throws IOException {
  //checking goes here
  JobConf conf = new JobConf();

  conf.setOutputKeyClass(Text.class);
  conf.setOutputValueClass(IntWritable.class);

  conf.setMapperClass(MapClass.class);
  conf.setCombinerClass(Reduce.class);
  conf.setReducerClass(Reduce.class);

  conf.setInputPath(new Path(args[0]));
  conf.setOutputPath(new Path(args[1]));

  JobClient.runJob(conf);
}
```

Inicialización

- Inicializar `hadoop-site.xml` y `slaves`
- Iniciar namenode
- Correr Hadoop MapReduce y DFS
- Subir los datos al DFS
- Correr tu proceso…
- Descargar los datos del DFS

Conclusiones

- MapReduce ha demostrado ser una abstracción útil
- Ha simplificado en Google el procesamiento a gran escala.
- Se enfoca en el problema y deja a la librería los detalles complejos

Referencias

- Artículo original:
  http://labs.google.com/papers/mapreduce.html
- En Wikipedia: http://en.wikipedia.org/wiki/MapReduce
- Framework en Java: Hadoop – MapReduce
  http://lucene.apache.org/hadoop/
- MapReduce en Ruby: Starfish http://rufy.com/starfish/