Map Reduce

Introducción

- **Parallel Computing.** Partir procesamiento en partes que pueden ser ejecutadas concurrentemente en múltiples cores.
- **Idea:**
  - Identificar **tareas** que pueden ejecutarse de forma concurrente,
  - Identificar **grupos de datos** que puedan ser procesados de forma concurrente.

Caso ideal (Master-Worker)

- **No existe dependencia** entre los datos.
- Datos partidos en **chunks del mismo tamaño**.
- C/ proceso puede trabajar con un **chunk/shard**.
- **Master.**
  - Parte la data en chunks,
  - Envía ubicación de los chunks a los workers,
  - Recibe ubicación de los resultados de todos los workers.
- **Workers.**
  - Recibe ubicación de los chunks,
  - Procesa el chunk,
  - Envia ubicación del resultado al Master.

Función Map

`Map: (input shard) -> intermediate(k:v pairs)`

- Función proporcionada por el usuario.
- Ejecutada en todos los chunks de data por `M` *map workers*.
- Usuario decide **criterio de filtrado**.
- Agrupa todos los valores asociados con una **misma key**.

Función Reduce

`Reduce: intermediate(k:v pairs) -> result files`

- Realiza una **agregación** de los datos.
- Llamada p/ c/ **Unique Key**.
- Función distribuida, particionando las keys en `R` *reduce workers*.

Algoritmo

1. **Partir datos** de entrada en **N chunks**.
2. **Fork de Procesos** en Cluster.
   - 1 master.
   - `M` mappers (tantos como chunks).
   - `R` reducers (configurado por el usuario)
3. **Map** de Shards en Mappers.
4. Creación de **Intermediate Files** (IFs).
   - Data particionada en `R` regiones.
   - **Función de Partición**.
     - Decide cuál de los workers va a trabajar con cada key.
     - Default: `hash(key) mod R`
     - Map workers `particionan data por Keys` con esta función.
5. **Reducers leen data** (por RPC) de los IF a procesar.

- Ordenan data por key.
- Agrupan ocurrencias de la misma key.

6. **Reducer aplica función** sobre c/ set de datos (agrupado por key).
7. Almacenamiento de resultados en **Output files**.