MapReduce

En el mundo científico siempre aparecen nuevos y más grandes problemas. La modelación y resolución de estos problemas requieren de una cantidad, cada vez más alta, de memoria y de poder de cómputo. Es normal que rápidamente una máquina se quede sin memoria al hacer determinado cálculo. Además de la memoria, el tiempo que se demora el algoritmo es un aspecto a optimizar también. Una solución a este problema puede ser escalar horizontalmente aprovechando los recursos de varias computadoras.

Especificaciones

En un sistema MapReduce existen dos partes fundamentales:
1. **Map** (Filtrar y mappear ;-))
2. **Reduce** (Agrupar por llaves y calcular).

Se quiere que dada una especificación de un algoritmo utilizando estos elementos y los datos, el sistema sea capaz de ejecutar este código de manera distribuida haciendo uso de los recursos disponibles.

Map

```
def map ( in_key : InKey , in_value : Val ) ->list [( OutKey , InterVal )]:
    pass
```

La función recibe una llave y un valor y devuelve una lista de llaves con un valor intermedio. Esta función es la encargada de filtrar y precalcular las llaves y valores que se utilizarán en el proceso posterior.

Reduce

```
def reduce ( out_key : OutKey , values : list [ InterVal ]) → list [ OutVal ]:
    pass
```

Luego que la operación map termina, se agrupa por las outkey y se instancia cada *reducer* encargado de cada llave. Normalmente la lista que resultante tiene un solo valor.

Datos

Para realizar los cálculos se necesitan datos. Estos normalmente se representan en forma de archivos o tablas (más estructurados). Su sistema debe ser capaz de analizar al menos estos dos tipos de datos. El tipo de dato utilizado no debe alterar la implementación de map ni reduce. La forma en que los datos “organizados” en el archivo puede ser indicada por el usuario. Por ejemplo, en el caso de tipo de dato “tuple” puede asumirse que cada línea es de la forma *llave, valor*.

Los datos de entrada del algoritmo deben ser previamente almacenados de manera distribuida. Para esto se debe implementar algún tipo de sistema que almacene tanto la entrada como salida del algoritmo de manera distribuida.

Ejecución

Primeramente es necesario que los datos se añadan al sistema. Luego deben ejecutarse los mappers y luego los reducers. Nótese que el hecho de que uno de estos procesos falle no debe provocar una falla total de la tarea. El cliente, que es el que define la tarea a realizar, se conecta al sistema y brinda toda la información necesaria para la ejecución. Este debe ser informado del estado del proceso.

Ejemplo

Un caso de uso ampliamente utilizado para explicar el proceso es el de contar las palabras de un documento. La función map se encarga de detectar cada una de las palabras que se encuentran en el documento. Luego la función reduce se encarga de sumar todos los valores para la llave word.

```
def map ( doc_line : int , doc_line_text : str ):
        res = []
        for word in doc_line_text . split ():
                res . append (( word , 1))
        return res

def reduce ( word : str , vals : list ):
        count = 0
        for v in vals :
                count += v
        return count
```

Recomendaciones

Sobre los aspectos que siempre se tendrán en cuenta en la revisión son:

- tolerancia a fallas
- replicación.

El soporte de estas debe ser, preferentemente, parametrizado, para que el servicio ofrecido pueda escalar sin dificultad.

Nota

Cualquier enriquecimiento del proyecto es válido y se tendrá en cuenta en la evaluación del mismo. En caso de modificar la orden del proyecto debe consultarse a los profesores con anterioridad.