Unidad 2: Modelo de mapeo y reducción

Teoría de la complejidad para el modelo de mapeo y reducción

Blanca Vázquez

Febrero 2020

Resumen: MapReduce

- Modelo de programación bajo el enfoque de procesamiento paralelo
- Reto: computación distribuida masiva
- Características: balanceo de carga, tolerante a fallos y escalamiento

Cálculos en MapReduce

- **Paso de mapeo:**
  Aplica una función μ para cada valor x_i para producir un conjunto finito de pares llave - valor (k, v)
- **Paso de agrupar:**
  Coleciona todos los pares (k, v) y genera una lista ordenada $L_k = (k; v_1, v_2, ...)$
- **Paso de reducción:**
  Aplica una función ρ para cada lista $L_k = (k; v_1, v_2, ...)$ para producir una conjunto de valores y_1, y_2

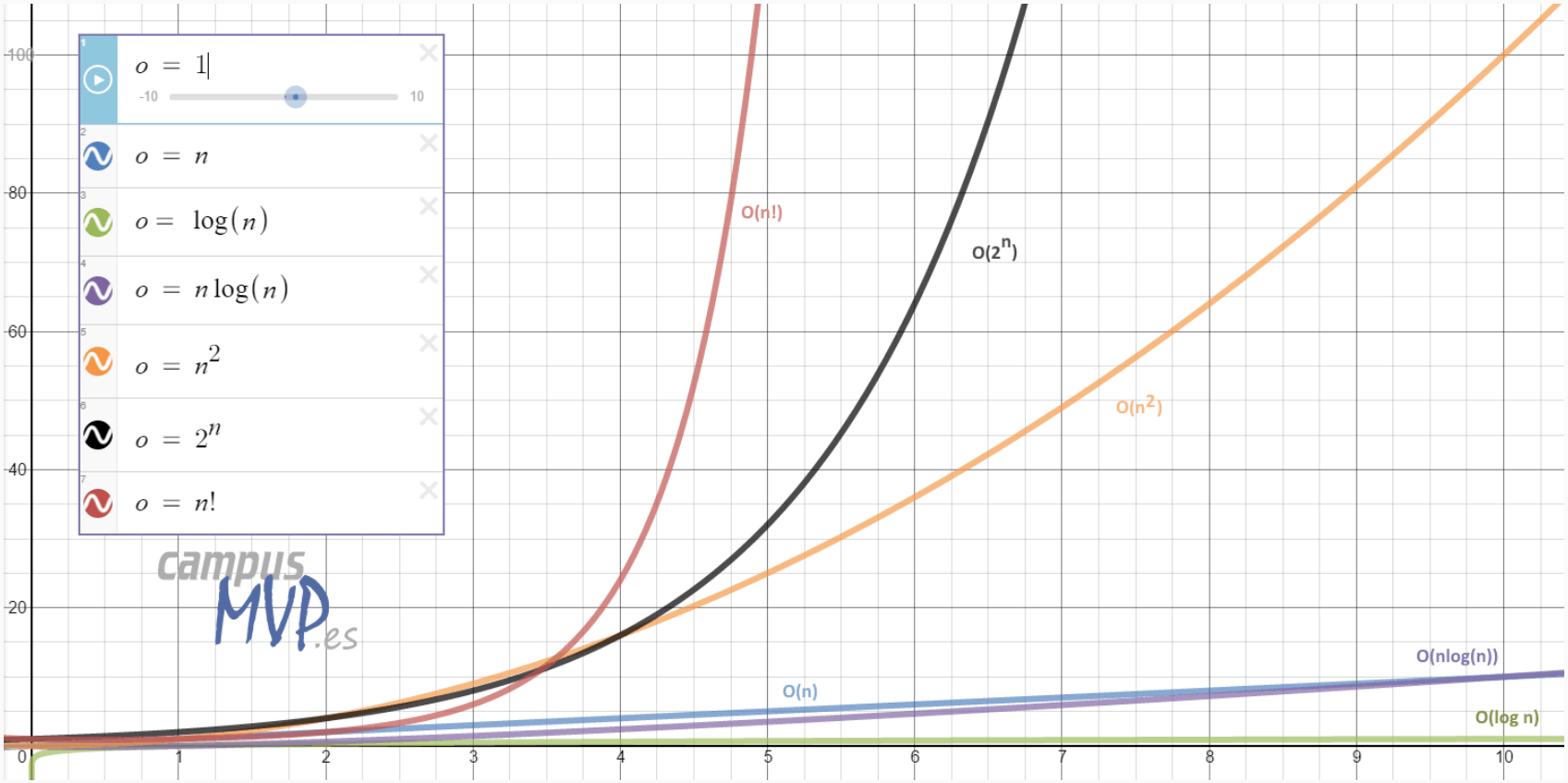
Cálculos en MapReduce

- En MapReduce, el **paralelismo** se produce debido a que cada tarea de mapeo y reducción se ejecuta en nodos diferentes.
- El programador define las funciones de μ y ρ.
- El modelo se encarga de **planificar** los pasos de mapeo, agrupación, reducción, **rutas de los datos** y la **tolerancia a fallos**.

Rounds en MapReduce

- La salida del paso de reducción puede ser la entrada hacia otro round **(mapear - agrupar - reducir)**
- MapReduce en **aplicaciones reales** es una secuencia de pasos que llevan a cabo una acción a través de una serie de rounds.
- Valor deseado: que el número de rounds sea un valor constante

Complejidad de los algoritmos



Conteo de palabras

- Entrada: conjunto de documentos
- **Función de mapeo:**
  1. Lee un documento y los separa en secuencia de palabras
  2. Genera los pares (k, v): $(w1, 1), (w2, 1), ..., (w_n, 1)$
- Agrupar:
  1. Agrupa todas las llaves (k, v)
  2. Se asignan las llaves a los nodos de reducción (usando una función hash)
- **Función de reducción:**
  1. Combina todos los valores asociados a una llave dada
  2. Conteo de palabras (añadiendo todos valores asociados con una palabra)

!Un solo round!

Round en MapReduce

- En el ejemplo anterior, solo se necesitó un round para calcular el número de veces que aparece cada palabra.
- Sin embargo, no siempre el número de rounds es igual a 1.

Ejemplo: una consulta triangular

- Input: three tables
  R(X, Y), S(Y, Z), T(Z, X)

  size(R)+size(S)+size(T)=M

- Output: compute

  $Q(x,y,z) = R(x,y) \bowtie S(y,z) \bowtie T(z,x)$

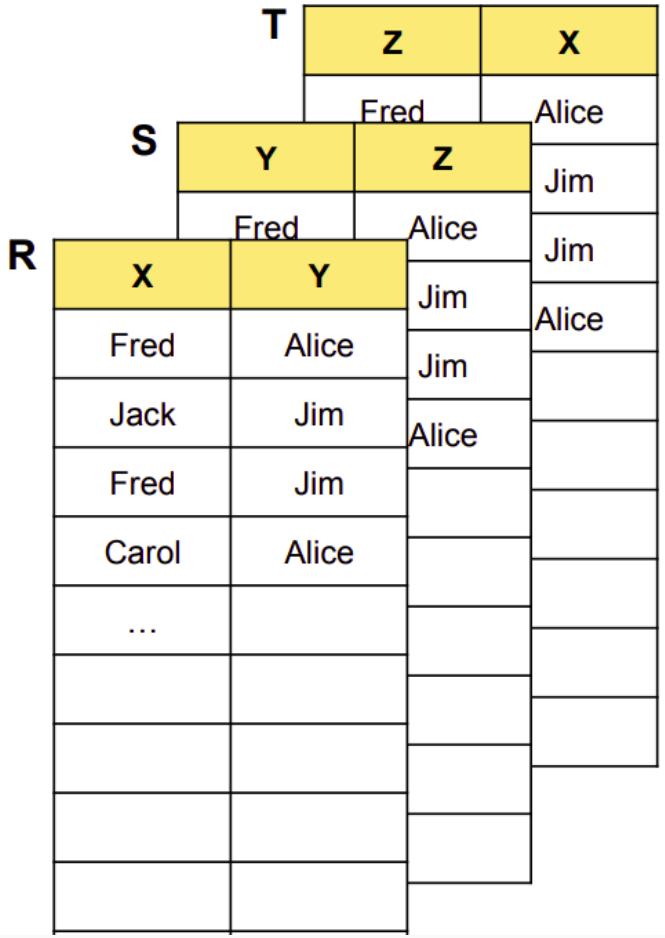


Imagen tomada de Dan Suciu, University of Washington

M = tamaño de datos de entrada en bits

Haciendo el join en dos pasos

- Paso 1: calcular un join (almacenarlo en temp)
  $temp(X, Y, Z) = R(X, Y) \bowtie S(Y, Z)$
- Paso 2: calcular un segundo join
  $Q(X, Y, Z) = temp(X, Y, Z) \bowtie T(Z, X)$

Midiendo la eficiencia de MapReduce

Para medir la **eficiencia del algoritmo MapReduce** sobre el transcurso de su ejecución debemos obtener:

- R: número de rounds que el algoritmo usa
- Cr: la complejidad de comunicación del round r
- t_r: tiempo de ejecución interno

Complejidad de comunicación

Cálculo de la complejidad de comunicación *Cr* del round *r*:

- Denotamos $n_{r,i}$ como el tamaño de las entradas y salidas de las tareas de mapeo y reducción *i* en el round *r*.
  Ejemplo: sea $n_{r,1}$,$n_{r,2}$ el tamaño de las I/O de las tareas de mapeo y reducción para el round r

$$C_r = \sum_i n_{r,i}$$ complejidad de comunicación del round r

$$C = \sum_{r=0}^{R-1} C_r$$ complejidad de comunicación para todo el algoritmo

Tiempo de ejecución

Cálculo del tiempo de ejecución interno t_r para cada round r

- Denotamos como t_r como el tiempo máximo de ejecución interno para la tarea de mapeo o reducción en el round r, asumiendo que

$t_r \geq max_i\{n_{r,i}\}$ donde la tarea de M o R tienen un tiempo de ejecución que es al menos el tamaño de las I/O

$t = \sum_{r=0}^{R-1} t_r$ tiempo total de ejecución interno

Parámetros adicionales

Para calcular la eficiencia total de MapReduce debemos considerar también:

- L (latencia de la red): se refiere al número de pasos que la tarea de mapeo o reducción tienen que esperar para recibir su 1era entrada en una ronda.
- B (ancho de banda): es el número de elementos que puede ser entregado por la red en una unidad de tiempo

Tiempo total de ejecución

El tiempo total de ejecución para la implementación de un algoritmo en MapReduce se define como:

$$T = \Omega(\sum_{r=0}^{R-1}(t_r + L + C_r/B)) = \Omega(t + RL + C/B)$$

Ω representa el conjunto de todas las funciones de complejidad para un algoritmo (cota inferior)

El número de rounds es clave

En un algoritmo MapReduce el **número de rounds** está proporcionalmente relacionado con el tiempo total de comunicación y complejidad del algoritmo.

Menos rounds y ¿el paralelismo?

- Si nos enfocamos únicamente en el número de rounds tendríamos un algoritmo ineficiente.
- Por ejemplo: si todos los algoritmos tuvieran un solo round:
  - Siempre se mapearía todas las entradas a una sola llave
  - Y la tarea de reducción haría un algoritmo secuencial para resolver el problema
- ¿y el paralelismo?

Retos en MapReduce

- Uno de los principales retos es la **localidad de los datos**.
- Como ya sabemos, cada nodo de reducción tiene acceso a una **porción limitada de los datos.**
- Existe una **estricta modularización** que prohíbe que los reductores se comuniquen dentro de una ronda
- Toda la comunicación pasa a través de los nodos de mapeo (se ven limitados).
- Por eso conseguimos tener varios rounds.

Paralelismo en MapReduce

Paradigmas existentes que buscan optimizar la complejidad del número de rounds y hacer paralelismo:

1. Limitar el tamaño del reductor
2. Limitar la tasa de replicación
3. Comparar la complejidad de MapReduce con las máquinas de Turing.
4. Grafos

Paralelismo en MapReduce

Paradigmas existentes que buscan optimizar la complejidad del número de rounds y hacer paralelismo:

1. Model MRC de Karloff [1]
2. Algoritmo MUD de Feldman [2]
3. Modelo BSP de Valiant [3]
4. La clase MPC de Beame [4]
5. Muchas extensiones y generalizaciones de estos [5]

Referencias

[1] Howard Karloff, Siddharth Suri, and Sergei Vassilvitskii. A model of computation for mapreduce. In SODA '10, pages 938–948, Philadelphia, PA, USA, 2010. Society for Industrial and Applied Mathematics.

[2] Jon Feldman, S. Muthukrishnan, Anastasios Sidiropoulos, Clifford Stein, and Zoya Svitkina. On distributing symmetric streaming computations. ACM Transactions on Algorithms, 6(4), 2010.

[3] Leslie G. Valiant. A bridging model for parallel computation. Commun. ACM, 33(8):103–111, 1990.

Referencias

[4] Paul Beame, Paraschos Koutris, and Dan Suciu. Communication steps for parallel query processing. In PODS, pages 273–284, 2013.

[5]Michael T. Goodrich, Nodari Sitchinava, and Qin Zhang. Sorting, searching, and simulation in the mapreduce framework. In ISAAC, pages 374–383, 2011.

[6] On the Computational Complexity of MapReduce `http://homepages.math.uic.edu/~bfish3/mapreduce_DISC_2015.pdf`

Referencias

[7] Sorting, Searching, and Simulation in the MapReduce Framework https://arxiv.org/pdf/1101.1902.pdf

[8] MapReduce https://www.cs.utah.edu/~hari/teaching/bigdata/03_MapReduce.pdf

[9] Model for Massively Parallel Computation http://grigory.us/blog/mapreduce-model/

[10] Complexity Theory for Map-Reduce https://shonan.nii.ac.jp/archives/seminar/011/files/2012/01/ullman.pdf