Aprendizaje Automático sobre Grandes Volúmenes de Datos

Clase 12

Pablo Ariel Duboue, PhD

Universidad Nacional de Córdoba,
Facultad de Matemática, Astronomía y Física

Material de lectura

- Clase pasada:
  - Capítulo 6 del Owen et al. (2012)
  - http://www.cs.utah.edu/~jeffp/teaching/cs7960/L17-MR-Matrix+DB
  - HAMA: An efficient matrix computation with the mapreduce framework por Seo, Yoon, Kim, Jin, Kim, Maeng
    - CloudCom, 2010
- Ésta clase:
  - Scalable Scientific Computing Algorithms Using MapReduce por Xiang Jingen, Master of Mathematics UWaterloo '13
    - https://uwspace.uwaterloo.ca/bitstream/handle/10012/7830/Xiang_Jingen.pdf
  - Design and Evaluation of Parallel Block Algorithms: LU Factorization on an IBM 3090 VF/600J, por Dackland, Elmroth, Kågström, Van Loan.
    - 5th SIAM Conf. on Parallel Processing for Scientific Computing, 1991
  - Parallelized stochastic gradient descent por Zinkevich, Weimer, Li, Smola (NIPS 2010)

Preguntas

- Preguntas sobre LU (hoy)
- Representaciones estándar
  - HIVE
- Ejemplo de Map/Reduce
  - Próxima clase
- Material de lectura previo
  - Ahora en Twitter / Lista
- Metodología de evaluación

Recordatorio

- El sitio Web de la materia es http://aprendizajengrande.net
  - Allí está el material del curso (filminas, audio)
- Leer la cuenta de Twitter https://twitter.com/aprendengrande es obligatorio antes de venir a clase
  - Allí encontrarán anuncios como cambios de aula, etc
  - No necesitan tener cuenta de Twitter para ver los anuncios, simplemente visiten la página
- Suscribirse a la lista de mail en aprendizajengrande@librelist.com es optativo
  - Si están suscriptos a la lista no necesitan ver Twitter
- Feedback para alumnos de posgrado es obligatorio y firmado, incluyan si son alumnos de grado, posgrado u oyentes
  - El "resúmen" de la clase puede ser tan sencillo como un listado del título de los temas tratados

Distribución de Matrices Dispersas

- Según el tipo de operación, distribuimos filas o columnas
- Si una fila o columna no entra en un solo nodo, distribuimos franjas de filas o columnas

Multiplicación de una matriz por un vector

$$Ax = \begin{bmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \vdots & \vdots & \vdots & \vdots \\ a_{m1} & a_{m2} & \dots & a_{mn} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} =$$

$$\begin{bmatrix} a_{11}x_1 + a_{12}x_2 + \cdots + a_{1n}x_n \\ a_{21}x_1 + a_{22}x_2 + \cdots + a_{2n}x_n \\ \vdots \\ a_{m1}x_1 + a_{m2}x_2 + \cdots + a_{mn}x_n \end{bmatrix}$$

http://mathinsight.org/matrix_vector_multiplication

Matriz por vector en MR

- Entrada: Matriz $M = n \times n$, vector $V = n \times 1$
- Salida: Vector $X = M * V$
  - $x_i = \sum_{j=1}^{n} m_{ij} * v_j$
- Map(i, <fila i de M, segmento de V entre b_s y b_e>):
  - $(i, \sum_{j=b_s}^{b_e} m_{ij} * v_j)$
- Reduce($i, \sum_{j=b_s}^{b_e} m_{ij} * v_j$):
  - $x_i = \sum_{j=1}^{n} m_{ij} v_j$

Multiplicación de Matriz por Matriz

$$\begin{bmatrix} b_{11} & b_{12} & \dots & b_{1p} \\ b_{21} & b_{22} & \dots & b_{2p} \\ \vdots & \vdots & \vdots & \vdots \\ b_{n1} & b_{n2} & \dots & b_{np} \end{bmatrix} = \left[\begin{bmatrix} b_{11} \\ b_{21} \\ \vdots \\ b_{n1} \end{bmatrix} \begin{bmatrix} b_{12} \\ b_{22} \\ \vdots \\ b_{n2} \end{bmatrix} \dots \begin{bmatrix} b_{1p} \\ b_{2p} \\ \vdots \\ b_{np} \end{bmatrix} \right]$$

http://mathinsight.org/matrix_vector_multiplication

Matriz por Matriz dispersa

- Una concatenación de los vectores obtenidos de multiplicar las columnas de la segunda matriz por la primera
- La clave recibida en el mapper es una clave compuesta y recibe la fila y la columna sobre la que se está operando
- Map((i,k), <fila i de M, columna k de$B$ >):
  - $((i,k), \sum_{j=1}^{n} m_{ij} * n_{jk})$
  - el índice de la columna final es el mismo que el de la columna en al segunda matriz

Solución de $Ax = b$

- Dados una matriz A simétrica y definida positiva y un vector B, buscamos un vector x tal que $Ax = b$
- Método del gradiente conjugado:
  - Definimos $f(x) = \frac{1}{2}x^T Ax - b^T x + c$
  - Entonces $f'(x) = \frac{1}{2}A^T x + \frac{1}{2}Ax - b = Ax - b$
  - La ecuación $Ax = b$ tiene un cero en los puntos críticos de la ecuación de arriba
- Usamos el gradiente conjugado para decidir la dirección de búsqueda y una búsqueda lineal para optimizar el tamaño del paso en esa dirección

Temas Claves

- Identificación de la máxima tarea que puede hacerse por nodo
- Descomposición de la tarea normal en tareas por nodo
- Tareas globales, realizadas en un nodo (central) y tareas paralelas
- Comunicación entre tareas mediante HDFS y archivos bandera
- Tareas que sólo hacen Map
- Cálculos intermedios

Inversión Matricial

- La inversa de una matriz $\mathbf{A}$ es otra matriz $\mathbf{B}$ tal que $\mathbf{AB} = \mathbf{BA} = \mathbf{I}_n$ donde $\mathbf{I}_n$ es la matriz identidad.
  - La inversa de $\mathbf{A}$ se denota por $\mathbf{A}^{-1}$

Algoritmos clásicos para encontrar la matriz inversa

- Gauss-Jordan
  - Realizar operaciones en órden
- SVD, calcular $\mathbf{U}$, $\mathbf{V}$ y $\mathbf{W}$ tales que
  - $\mathbf{A} = \mathbf{U}\mathbf{W}\mathbf{V}^T$ con
    - $\mathbf{U}\mathbf{U}^T = \mathbf{V}\mathbf{V}^T = \mathbf{I}_n$
    - $\mathbf{W}$ diagonal
  - Requiere intercambios de filas
- Decomposición QR
  - $\mathbf{A} = \mathbf{Q}\mathbf{R}$ con
    - $\mathbf{Q}\mathbf{Q}^T = \mathbf{I}_n$
    - $\mathbf{R}$ triangular superior
  - Usa Gram-Schmidt para calcular vectores en órden
- Descomposición LU

Descomposición LU

- $A = LU$
  - Para mejorar la estabilidad numérica se suele usar una permutación P de A
  - L es triangular inferior, U es triangular superior
- Las matrices triangulares son fáciles de invertir

Descomposición LU

$$\begin{pmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ a_{31} & a_{32} & \dots & a_{3n} \\ \dots & & & \\ a_{n1} & a_{n2} & \dots & a_{nn} \end{pmatrix} = \begin{pmatrix} l_{11} & & & \\ l_{12} & l_{22} & & \\ l_{13} & l_{23} & l_{33} & \\ \dots & & & \\ l_{n1} & l_{2n} & \dots & l_{nn} \end{pmatrix} \begin{pmatrix} u_{11} & u_{12} & \dots & u_{1n} \\ & u_{22} & \dots & u_{2n} \\ & & u_{33} & \dots \\ & & & \dots \\ & & & u_{nn} \end{pmatrix}$$

(adaptado de Xiang Jingen, 2013)

Descomposición LU

$$u_{ij} = a_{ij} - \sum_{k=1}^{j-1} l_{jk} u_{kj},$$

$$l_{ij} = \frac{1}{u_{jj}} (a_{ij} - \sum_{k=1}^{j-1} l_{ik} u_{kj}),$$

(adaptado de Xiang Jingen, 2013)

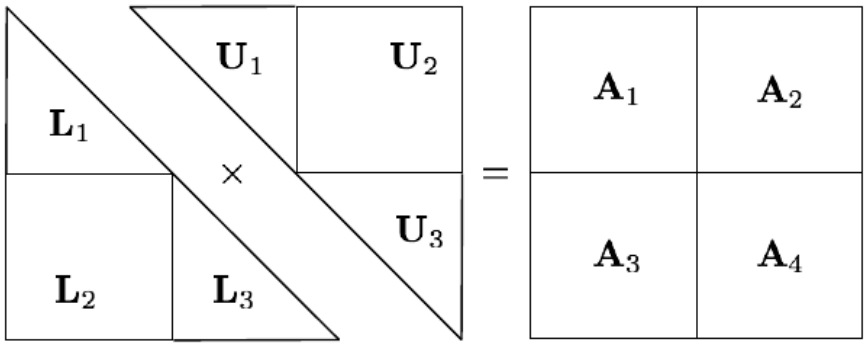
Descomposición LU en una máquina

Algorithm 5 LU decomposition on a single node.

```
1:  function LUDecomposition(A)
2:  for i = 1 to n do
3:      j = {j | [A]_ji = max([A]_ii, [A]_i+1i, ..., [A]_ni)}
4:      Add j to P
5:      Swap i-th row with j-th row if i ≠ j
6:      for j = i + 1 to n do
7:          [A]_ji = [A]_ji/[A]_ii
8:      end for
9:      for j = i + 1 to n do
10:         for k = i + 1 to n do
11:             [A]_jk = [A]_jk − [A]_ji × [A]_ik
12:         end for
13:     end for
14: end for
15: return (A, P)      /* i.e., return (L, U, P) */
```

(adaptado de Xiang Jingen, 2013)

Descomposición LU en MR



(adaptado de Xiang Jingen, 2013)

Descomposición LU en MR

$$
\begin{aligned}
\mathbf{L}_1\mathbf{U}_1 &= \mathbf{P}_1\mathbf{A}_1, \\
\mathbf{L}_1\mathbf{U}_2 &= \mathbf{P}_1\mathbf{A}_2, \\
\mathbf{L}_2^{'}\mathbf{U}_1 &= \mathbf{A}_3, \\
\mathbf{L}_3\mathbf{U}_3 &= \mathbf{P}_2(\mathbf{A}_4 - \mathbf{L}_2^{'}\mathbf{U}_2), \\
\mathbf{L}_2 &= \mathbf{P}_2\mathbf{L}_2^{'},
\end{aligned}
$$

(adaptado de Xiang Jingen, 2013)

El algoritmo a vuelo de pájaro

- Dividir la matriz original recursivamente hasta que $\mathbf{L}_1\mathbf{U}_1 = \mathbf{P}_1\mathbf{A}_1$ sea soluble en una máquina
- Descomponer $\mathbf{A}_1$
- Calcular $\mathbf{U}_2$
- Calcular la permutación de $\mathbf{L}_2$, $\mathbf{L}'_2$
- Calcular $\mathbf{B} = \mathbf{A}_4 - \mathbf{L}'_2\mathbf{U}_2$
- Descomponer $\mathbf{B}$
- Armar la salida total a partir las partes

El Algoritmo

Algorithm 6 Block LU decomposition.

1: **function** BlockLUDecom($\mathbf{A}$)
2: **if** A is small enough **then**
3: $(\mathbf{L}, \mathbf{U}, \mathbf{P}) = \text{LUDecompoistion}(\mathbf{A})$
4: **else**
5: Partition $\mathbf{A}$ into $\mathbf{A}_1, \mathbf{A}_2, \mathbf{A}_3, \mathbf{A}_4$
6: $(\mathbf{L}_1, \mathbf{U}_1, \mathbf{P}_1) = \text{BlockLUDecom}(\mathbf{A_1})$
7: Compute $\mathbf{U}_2$ from $\mathbf{A}_2$, $\mathbf{U}_1$ and $\mathbf{P}_1$
8: Compute $\mathbf{L}_2^{'}$ from $\mathbf{A}_3$ and $\mathbf{U}_1$
9: Compute $\mathbf{B} = \mathbf{A}_4 - \mathbf{L}_2^{'}\mathbf{U}_2$
10: $(\mathbf{L}_3, \mathbf{U}_3, \mathbf{P}_2) = \text{BlockLUDecom}(\mathbf{B})$
11: $\mathbf{P}$ = Combination of $\mathbf{P}_1$ and $\mathbf{P}_2$
12: $\mathbf{L}$ = Combination of $\mathbf{L}_1$, $\mathbf{L}_2^{'}$, $\mathbf{L}_3$, and $\mathbf{P}_2$
13: $\mathbf{U}$ = Combination of $\mathbf{U}_1$, $\mathbf{U}_2$ and $\mathbf{U}_3$
14: **end if**
15: **return** $(\mathbf{L}, \mathbf{U}, \mathbf{P})$

(adaptado de Xiang Jingen, 2013)

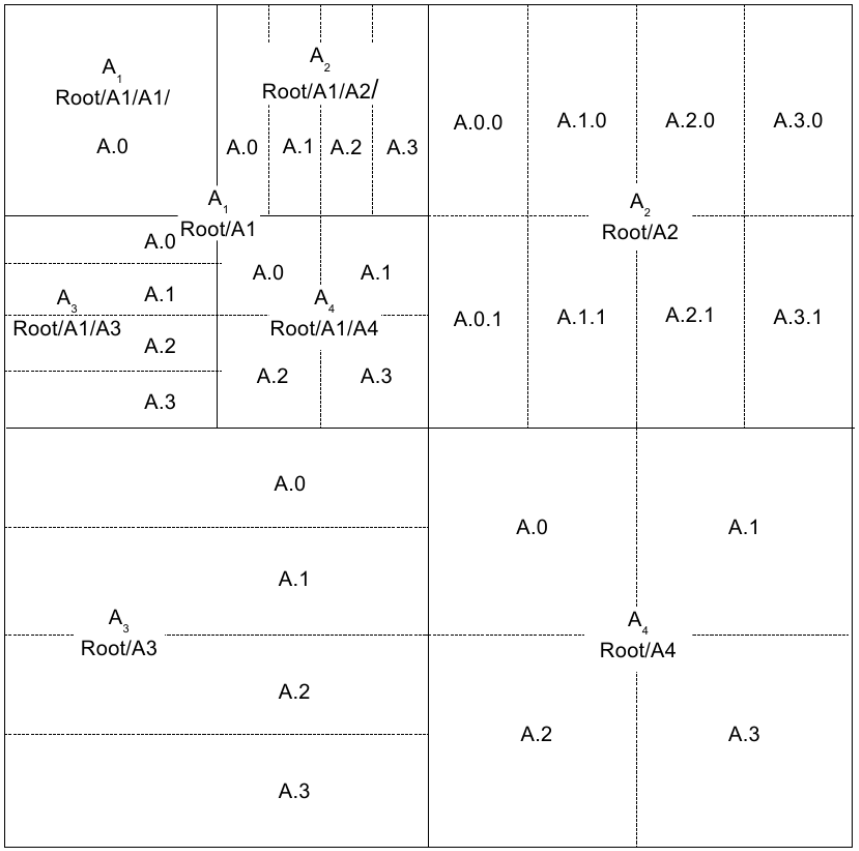
Implementación en MapReduce

- Datos en archivos separados
- Ciertas partes de la matriz se guarda por filas, ciertas partes por columnas
- Los resultados intermedios se concretizan en disco
  - Mejor tolerancia a fallas
  - Enlentece el algoritmo innecesariamente
  - Restricción de Hadoop (Spark no tendría ese problema)
- Primera pasada recursiva genera archivos de control
- El resto del sistema opera a partir de esos archivos de control
- Tareas de sólo map

Implementación en MapReduce

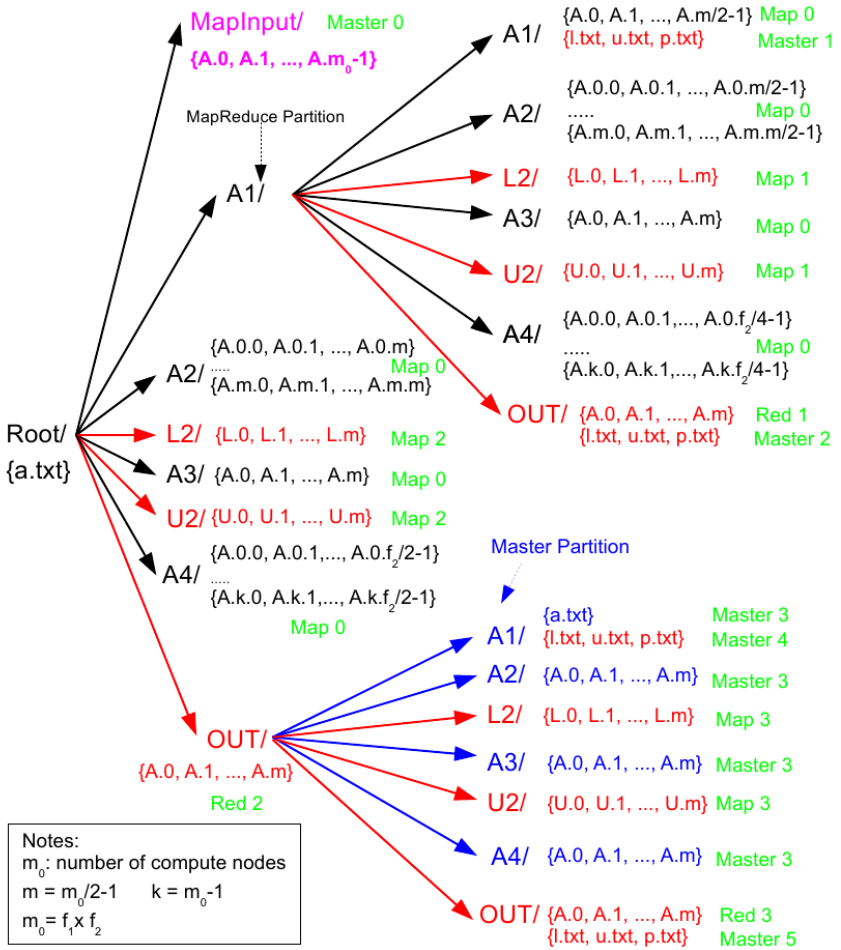
1. Nodo master crea archivos de control que son usados como entrada a las tareas map subsecuentes
   - Falsas entradas
2. Una tarea MapReduce subdivide $\mathbf{A}$ de manera recursiva
3. Se envian tareas MapReduce para $\mathbf{L}'_2$, $\mathbf{U}_2$ y $\mathbf{B}$
   - $\mathbf{L}_1$ y $\mathbf{U}_1$ son ejecutadas en el master node si $\mathbf{A}_1$es lo suficientemente pequeña

Datos



(adaptado de Xiang Jingen, 2013)

Ejecución



(adaptado de Xiang Jingen, 2013)

Zinkevich et al., NIPS 2010

- Distribuir los datos al azar entre nodos
- Realizar descenso por el gradiente en cada nodo, por separado
- Unificar los resultados en un nodo central

Algoritmo en nodo worker

Algorithm 1 SGD($\{c^1, \ldots, c^m\}, T, \eta, w_0$)

for $t = 1$ **to** T **do**
  Draw $j \in \{1 \ldots m\}$ uniformly at random.
  $w_t \leftarrow w_{t-1} - \eta \partial_w c^j(w_{t-1})$.
end for
return w_T.

(adaptado de Zinkevich et al, 2010)

Algoritmo en nodo central

Algorithm 2 ParallelSGD$(\{c^1, \dots c^m\}, T, \eta, w_0, k)$

for all $i \in \{1, \dots k\}$ **parallel do**
  $v_i = \text{SGD}(\{c^1, \dots c^m\}, T, \eta, w_0)$ on client
end for
Aggregate from all computers $v = \frac{1}{k} \sum_{i=1}^{k} v_i$ and **return** v

(adaptado de Zinkevich et al, 2010)