Aprendizaje Automático sobre Grandes Volúmenes de Datos

Clase 10

Pablo Ariel Duboue, PhD

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

Material de lectura

- Clase pasada:
  - MapReduce: Simplified Data Processing on Large Clusters por J Dean, S Ghemawat
    - OSDI'04, 2004
  - The Google File System por S Ghemawat, H Gobioff, ST Leung
    - 19th ACM Symposium on Operating Systems Principles, 2003.
- Ésta clase:
  - Brewer's conjecture and the feasibility of consistent, available, partition-tolerant web services por S Gilbert, N Lynch
    - ACM SIGACT News, 2002
  - CAP twelve years later: How the" rules" have changed por E Brewer
    - Computer, 2012
  - http://en.wikipedia.org/wiki/Fallacies_of_Distributed_Computing

Preguntas

- Localidad de datos en Hadoop
- Comentario: MR en Google no Hadoop
- Creación de modelos vs. aplicación
  - Se usa cómputo distribuido en ambos casos
- MapReduce y bases de datos
  - Hive
  - MongoDB

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

Revisión Map Reduce

- Simplificar el acceso al cómputo de gran volúmen de datos para programadores sin experiencia en cómputo distribuido
- Simplificar la tolerancia a fallas
- Simplificar la alocación de recursos (máquinas, disco y red)

Modelo de Programación

- Computa una función $f(\{(k_{in}, v_{in})\}) \rightarrow \{(k_{out}, list(v_{out})\}$
  - $map(k_{in}, v_{in}) \rightarrow list(k_{out}, v_{int})$
    - Para cada par (clave, valor) de entrada, produce una lista de pares de otros claves y valores intermedios
  - $reduce(k_{out}, list(v_{int})) \rightarrow list(v_{out})$
    - Acumular los valores intermedios según su clave de salida
    - Generar la salida combinada

Tolerancia a Fallas

- Si el master muere, el sistema cae
- Si un worker muere el sistema se da cuenta via *heartbeats*
  - Re-ejecución de tareas fallidas
  - Re-ejecución de tareas en ejecución
    - Mejora el peor caso (ejecución redundante)
- Semántica en caso de fallas:
  - No presenta problemas para tareas determinísticas

Ciclo de vida de una aplicación MR/ML

1. Ingesta de datos
2. Creación de modelos
3. Aplicación de modelos

Las 8 falacias del cómputo distribuido

- Todo programador comete alguno de estos errores cuando empieza cómputo distribuido (L. Peter Deutsch y otros):
  1. La red es confiable
  2. Hay cero latencia
  3. El ancho de banda es infinito
  4. La red es segura
  5. La topología no cambia
  6. Hay un sólo administrador
  7. El costo de transporte es cero
  8. La red es homogénea

Costo de la Paralelización

- Con más N veces hardware no podemos ejectuar necesariamente N veces más rápido
- El énfasis es en atacar problemas más complejos, intratables en una sola máquina
- Hay un costo alto en la solución distribuida en términos de overhead comunicacional
- En algún momento ya no es posible de aumentar la velocidad agregando más máquinas
  - Hay tareas que no pueden paralelizarse
- La ley de Amdahl vs. la ley de Gustafson

Modelo ACID

- En el contexto de bases de datos, este modelo garantiza que todos los cambios al modelo de datos se realizan de forma:
  - Atómica: las transacciones se ejecutan completamente o fallan de manera completa
  - Consistente: las transacciones ejecutadas son visibles para todas las transacciones futuras y respetan todas las restricciones sobre los datos (claves únicas, etc)
  - Aislada (*Isolation*): las transacciones en ejecución están separadas entre sí
  - Durable: las transacciones ejecutadas son permanentes (en disco)
- El teorema CAP habla de términos similares pero es más general que sólo el sistema de datos

Teorema CAP

- Cuando estamos en un ambiente distribuido, de estas tres características sólo podes escoger dos:
  - Consistencia
  - Disponibilidad (*Availability*)
  - Tolerancia a particiones de la red (*Partition-tolerance*)

Consistencia

- Consistencia:
  - los datos se acceden de manera atómica
  - todos los nodos tienen el mismo valor de cada dato
- Es equivalente a que todas las operaciones se ejecuten en un mismo nodo

Disponibilidad

- Disponiblidad:
  - Los datos y servicios del sistema están disponibles a todo momento
    - Énfasis en **cambios de estado** (*updates*)
- Todo pedido hecho a un nodo que no se encuentra fallado debe tener una respuesta
  - El algoritmo debe terminar
  - Sin embargo, esta definición no restringe el tiempo para responder

Tolerancia a particiones de la red

- Es posible que ciertos nodos no estén disponibles por periodos de tiempo
  - Ya sea por problemas de red
  - O por problemas físicos del nodo en cuestión
- En general se considera el caso de que la red se particiona en uno o más componentes conexos
- La partición se representa como perdida arbitraria de mensajes entre dos nodos

Modelo de cómputo asíncrono

- Los nodos pueden mandar mensajes entre ellos
- No existe un concepto de reloj o paso del tiempo

Teorema CAP asíncrono

Dado el modelo de cómputo asíncrono, no es posible garantizar Consistencia y Disponibilidad

Demostración:

- por el absurdo, se construye una cadena de ejecución que actualiza un valor v en un nodo y se pierden los mensajes de actualización en el otro
- cuando se pide el valor de v en el otro nodo, por Disponibilidad el otro nodo tiene que responder y responde con el valor equivocado

Corolario:

- Tampoco se puede garantizar Consistencia y Disponibilidad aún cuando no se pierdan mensajes (pero se puedan demorar lo suficiente como para que se de la construcción por el absurdo del teorema).

Construcción de Pares

- Consistencia sin Disponibilidad
  - Esperar que la red vuelva a estar sin partición antes de responder
- Disponibilidad sin Consistencia
  - Devolver el valor desactualizado en el cache local
- Sin errorer de red
  - Asignar valores a nodos, sólo se accede al valor a través de dicho nodo (sistema centralizado)

Modelo de cómputo parcialmente asíncrono

- Igual que el modelo asíncrono pero
  - Los nodos tienen un reloj local que les permite calcular el paso del tiempo
  - Eso permite la implementación de time-outs

Teorema CAP parcialmente asíncrono

En el modelo parcialmente asíncrono tampoco es posible garantizar Consistencia y Disponiblidad en el caso que se pierdan mensajes.

- La demostración es similar, eligiendo el momento de escritura y lectura de forma tal que los time-outs se excedan

Diferencia importante con el model totalmente asíncrono: si los mensajes no se pierden, es posible garantizar Consistencia y Disponibilidad

Consistencia Eventual

- Una forma de paliar el problema de falta de consistencia en caso de ruptura de la red es poner cotas en la cantidad de tiempo que los nodos tienen un valor incorrecto
- Cuando la situación de ruptura de red se restablece, los nodos con valores en cache inválidos deben actualizarse con el paso del tiempo
- Delayed-t Consistent

Modelo de sistema centralizado

- Un nodo central se encarga que el mismo valor se sepa en todos los otros nodos
  - Los otros nodos usan un sistema de time-out para darse cuenta si la operación fue exitosa o no
  - Con el tiempo, se re-establece la comunicación con el nodo central y se re-adquiere consistencia
    - Cuando se pierden mensajes, se pierde atomicidad (en un nodo se ejecutó una escritura que es desconocida por el nodo central)

Teorema CAP revisitado

- Formalización clásica ignora el concepto de latencia, pero es clave.
  - Durante un time-out, el programa debe decidir:
    1. cancelar la operación (afecta Disponibilidad)
    2. continuar la operación (afecta Consistencia)
  - Continuar re-intentando es elegir Consistencia en vez de Disponiblidad
- También está la cuestión práctica... es posible realmente elegir Consistencia+Disponibilidad? Tarde o temprano la red fallará
  - Interpretación probabilística de C, A y P
- Nuevo énfasis en recuperación después de particiones

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

- 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, V >):
  - $(j, m_{ij} * v_j)$
- Reduce($j, m_{ij} * v_j$):
  - $x_i = \sum_{j=1}^{n} m_{ij} v_j$
- Si V no entra en un mapper, distribuir franjas de V a cada mapper