Adatelemzési platformok, Május 2., 9. elméleti óra

Big data logika folytatás

Adatelemzős világban van egy pár nem szokványos megoldás programozási szempontból.

MapReduce függvények formalizálása

k, v: kulcs-érték párok

step	input	output
map	<k1, v1>	list(<k2, v2>)
reduce	<k2, list(v2)>	list(<k3, v3>)

- A k2-höz tartozó összs kulcsnak el kell férnie ugyanazon a gépen, és ezt maga az algoritmus nem tudja automatizálni. Emiatt is fontos, hogy tudjuk, hogy mit csináljunk (e.g. minden tranzakció joinolása saját magával katasztrofális tud lenni, MapReduce-szal együtt is).

Példa 1

'Meta-kód'

step	input	output
map	<None, txt>	list(<ch, 1>)
reduce	<ch, list(1)>	<ch, count>

Példa 2

txt line(rowid, year, month, day, city, temp)

Minden napra próbáljuk megmondani, hogy mi a maximális hőmérséklet.

step	input	output
map	<rowid, txt>	list(<year+month+day, temp>)
reduce	<y+m+d, list(temp)>	<y+m+d, maxtemp>

A külön lekérdezések (pl csak Budapesti adatokat szeretnénk) könnyen beépíthetőek a kalkulációba.

Példa 3

Városok átlagos hőmérsélkete

step	input	output
map	<rowid, txt>	list(<city+month, temp>)
reduce	<city+month, list(temp)>	<city+month, avgtemp>

Ezek eddig szonylag egyszerűek voltak.

Példa 4

Egyszerű weblog: (date, time, session, url, referrer). Akár már egy közepes méretű oldal esetében is big data problémáról beszélhetünk.

Kérdés: napi egyedi látogató(session) megállapítása.

Ezt nem lehet egy MapReduce-szal megoldani, hanem legalább kettővel.

I.

step	input	output
map	<None, txt>	list(<date+session, 1>)
reduce	<date+session, list(1)>	<date+session, 1>

II.

step	input	output
map	<date+session, 1>	list(<date, 1>)
reduce	<date, list(1)>	<date, count>

Ilyen jellegű problémáknál MapReduce sorozatokra van szükség, emiatt az ilyen redszerek nagy része I/O műveletekből áll.

Példa 5

A példa 4-beli session-ökhüz szeretnénk még joinolni egy user_id-t is.

Reduce-side join

step	input	output

step	input	output
map	<rowid, txt>	list(<session, R/L + txt>)
reduce	<session, list(R/L + txt)>	<session, txt+user_id>

Map-side join

Ha a bemeneti adat befér a memóriába

step	input	output
map	<rowid, txt>	list(<session, join_txt>)

Itt nincs reduce.

Alkalmazás-történet

Ezeket a MapReduce-okat főleg Java-ban kellett megírni, elég bonyolultak voltak, a 'végfelhasználó' SQL query-ket tudott lefuttatni.

- Mai SQL feldolgozó technológiák:
  - Impala
  - vmi másik?
- Ma már zippelt txt-ben tárolják ezeket az adatokat
- Oszlop alapú tárolást hajtanak végre (e.g. parquet?), és az oszlopokhoz nagyon gyorsan tudnak statisztikai lekérdezéseket gyártani (sőt ezeket előre le is tárolják sokszor).
- Update szinte soha nincs támogatva, erős az adattárház szemlélet.

Spark

MapReduce problémái

1. Az iteratív lekérdzésekben nagyon rossz
2. Nem lehetett egymásra épülő feladatokat hatékonyan végezni benne
3. Alapvetően batch feldolgozásra találták ki, és ezért a nagyon gyors kis lekérdezéseket is nagyon lassan csinálta

A Spark megoldásai

- Lehet a MapReduce felett és attól függetlenül is futtatni
- Memória-alapú: sokkal több adatot tudsz feldolgozni, mint ami befér a memóriába, de ennek a folyamatát ő menedzseli

    - Emiatt viszont nem lehet debuggolni
    - Nem tudjuk, hogy milyen sorrendben hajtja végre a feladatokat
- Van SQL interfésze
- Van streaming
- Van Machine Learning (ez főleg marketing, illetve arra jó, hogy ne kelljen környezetet váltani)
- Gráf-feldolgozó
- Java, Scala, Python (PySpark) (esetleg R-ben is)

Kitérő: Big data és machine learning közötti viszony

Data preparation	Machine Learning
small	small
big	small (sima ügyfél-szintű adat)
big	big
?	deep learning (nem big data féle párhuzamosítás)

RDD: Resilient Distributed Dataset

- Egy objektum **bármilyen** egyöntetű dolog lehet.
- Bármilyne hosszú lehet (pl lehet akár egy Hadoop adatforrás is)
- Nem lehet átírni benne semmit, helyette:
    - "Transformation" (RDD -> RDD): Gyárt egy új RDD-t
        - `filter`
        - `map()` : minden egyes objektumból egy darab objektumot csinál
        - `flatmap()` : több értékpárt is tud gyáratani egy objektumból
        - `reduceByKey()` : nem az egész adathalmazból próbál dolgozni, hanem előbb két sort von össze, aztán azok eredményeit használja, stb, v1, v2 -> v3
        - Az RDD nem létezik fizikailag, hanem egy logikai érték
    - "Action" (RDD -> más formátum):
        - pl: `toTXT` , `take(5)` , `collect()`
        - Addig nem történik semmi, amíg le nem futtatunk egy Action-t

Köv héten ezeket gyakoroljuk.