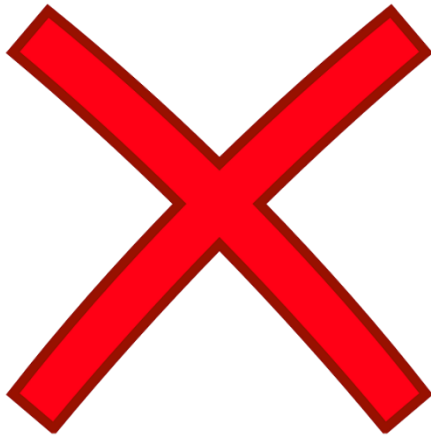
Quarto giorno: pomeriggio

Agenda 8/8



Do & Don’t

3 Ragioni per non usare i RDD
Sono obsoleti, più lenti e più complicati

Decisioni, decisioni, decisioni!
Numero di esecutori? N. di core per esecutore? memoria per esecutore?

Numero di partizioni
I blocchi nello shuffle non devono superare i 2GB
Il numero di partizioni condiziona le prestazioni.

Controllare lo shuffling

Esercizi di laboratorio
Esercizi di recapitolazione

Conclusioni

Recapitolazione e domande

Compilazione del questionario

Decisioni, decisioni, decisioni!!

Esempio:
Ho un cluster con 6 nodi
16 core ogni nodo
64 GB di RAM per ogni nodo

Devo decidere:

- Numero di esecutori (*--num-executors*)
- Numero di core per ogni esecutore (*--executor-cores*)
- Quanta memoria per ogni esecutore (*--executor-memory*)

Decisioni, decisioni, decisioni!!

Esempio:
Ho un cluster con 6 nodi
16 core ogni nodo
64 GB di RAM per ogni nodo

Devo decidere:

- Numero di esecutori (*--num-executors*)
- Numero di core per ogni esecutore (*--executor-cores*)
- Quanta memoria per ogni esecutore (*--executor-memory*)

Una scelta ragionevole:

- 5 core / esecutore
- 19GB / esecutore
- 17 esecutori

Criteri per la scelta

- Bisogna lasciare un po' di risorse per i demoni di sistema (OS, Hadoop, etc.).
  Almeno un core per nodo.
- Il numero di core/executor determina il numero di task concorrenti che un executor può eseguire. È opportuno che sia maggiore di 1 e si è visto che le prestazioni tendono a peggiorare se il numero è maggiore di 5. Quindi il numero ottimale è 5.
- Con i dati dell'esempio questo porta il totale a 18 esecutori. Però bisogna prevedere l'equivalente di un esecutore per Yarn (il cluster manager): ne Restano 17.
- Ogni esecutore sembrerebbe avere (64-1)/3 = 21 GB di RAM.
  Però c'è un overhead nell'uso della memoria (sempre dovuto a YARN).
  Questo overhead è *max(384, .07*spark.executor.memory)*
  Nel nostro caso è 1.47GB e porta la memoria realmente disponibile a 19GB.

Attenzione alla costruzione del DAG!!

- Alcuni metodi producono grandi concentrazioni di dati: ad es. è meglio usare *reduceByKey()* che *groupByKey()*.
- Evitare il più possibile gli *shuffles*.
- La quantità di memoria che viene scritta su disco (per ogni partizione) durante uno shuffle non può essere più grande di 2GB. (se necessario bisogna ridurre la grandezza delle partizioni : *coalesce()*)

Preferire DataFrame e Dataset agli RDD

Gli RDD vanno utilizzati solo se assolutamente necessario:

- Da un certo punto di vista sono obsoleti
- In molti casi sono più difficili da utilizzare
- Sono più lenti

Data skew

Le *data skew* sono asimmetrie nei dati.

Un job lento su una Join o un altro tipo di Shuffle può derivare da un data skew.

Una tecnica per limitare il fenomeno consiste nel «salare» le chiavi.

Consigli vari

- Ricordatevi di ridurre il numero di partizioni prima di scrivere su HDFS. HDFS non ama tanti file piccoli!
- Usate assennatamente *cache()* e *persist()*
- Nella messa a punto del programma chiamate spesso *explain*() per controllare quello che sta facendo l'ottimizzatore.
- Familiarizzatevi con la *Spark UI*. È uno strumento utile per rivelare possibili problemi e rallentamenti
- Limitate l'utilizzo di UDF (User Defined Function), specialmente in Python. Usate il più possibile le funzioni *built-in*.
- Controllate sempre l'utilizzo della memoria: limitate l'utilizzo dei join cartesiani, attenzione ai memory overflow sui singoli nodi, ecc.

Grazie

TIM Academy