Report on Big Data project: Director-Actor-Job

Federico Naldini - Mat. 0000852918

June 10, 2019

Contents

1 Introduzione **3**
1.1 Descrizione del dataset 3
1.2 Descrizione dei files 3

2 Preparazione dei files **4**
2.1 Posizione degli eseguibili, files di input/output e configurazioni necessarie 4
2.2 Preprocessing dei dati 4

3 Jobs **4**
3.1 Job: Trovare i tre attori/attrici diretti più di frequente per ogni regista, ordinando i registi per numero di film girati 4
3.1.1 MapReduce implementation 5
3.1.2 SparkSQL implementation 7

4 Miscellaneous **8**
4.1 Problematiche col Dataset 8

1 Introduzione

1.1 Descrizione del dataset

Il dataset che ho scelto di utilizzare per questo elaborato di progetto è fornito da *IMDb*, uno dei maggiori siti per la gestione di informazioni legate al mondo cinematografico.
Il dataset in questione è disponibile all'indirizzo `https://www.imdb.com/interfaces/` e si presenta diviso in sette files salvati in formato *TSV* e formattati in UTF-8. Ciascun file contiene nella prima riga la descrizione delle colonne in cui sono suddivisi i dati; altra caratteristica la presenza di diversi valori mancanti all'interno delle varie tabelle del dataset, indicati con il valore '/N'.

1.2 Descrizione dei files

Il dataset è composto dai seguenti file:

- `title.akas.tsv.gz`: contiene, per ogni titolo, i dati riguardanti le trasposizioni dell'opera in paesi differenti da quello di origine, come ad esempio il titolo nel singolo paese, la lingua in cui è stato tradotto...
  Link per il download

- `title.basics.tsv.gz`: Questo file modella l'entità titolo all'interno del dataset, tenendo traccia di tutte le informazioni necessarie, tra cui la tipologia di pellicola(film, documentario, episodio di serie TV,...), l'anno di pubblicazione, la durata, i generi a cui appartiene la pellicola e altro ancora.
  Link per il download

- `title.crew.tsv.gz`: In questo file sono contenuti i principali registi e scrittori per ogni titolo.
  Link per il download

- `title.episode.tsv.gz`:Questo file contiene le informazioni per ciascun episodio di una serie tv, tra cui la serie madre, la stagione e il numero di episodio.
  Link per il download

- `title.principals.tsv.gz`: Questo file modella la relazione tra un titolo e le persone che vi prendono parte, contiene infatti dati quali gli identificatori di titoli e persone, il ruolo che le persone hanno avuto all'interno della produzione del titolo e il personaggio eventualmente interpretato.
  Link per il download

- `title.ratings.tsv.gz`: Per ogni titolo, mantiene l'elenco delle valutazioni espresse dagli utenti e la loro media.
  Link per il download

- `name.basics.tsv.gz`: Contiene tutte le informazioni rigurardati attori, registi e scrittori tra cui il nome, l'anno di nascita, quello di morte e i titoli per cui è più famoso(se presenti).
  Link per il download

2 Preparazione dei files

2.1 Posizione degli eseguibili, files di input/output e configurazioni necessarie

La cartella *exam* contenente i fatjar eseguibili si trova all'indirizzo `isi-vclust7/ /home/fnaldini`, al suo interno sono contenuti due jar eseguibili(contenenti uno l'implementazione del job mediante*Map-Reduce*, l'altro tramite *SparkSQL*).
Per quanto riguarda invece i singoli file di input-output, essi sono memorizzati su filesystem distribuito hdfs: tutti gli input condividono un path comune, ovvero `hdfs:///user/fnaldini/bigdata/dataset/`, all'interno di quest'ultima cartella,sono state create tante cartelle quanti i file di input rilevanti per la risoluzione del job assegnato, aventi ciascuna lo stesso nome del file contenuto.
Trattando dell'output infine, i risultati parziali dell'implementazione *Map-Reduce* sono salvati all'interno di diverse cartelle all'indirizzo: `hdfs:///user/fnaldini/mapreduce/`, i risultati finali sono disponibili dentro la sottocartella `output`; il job implementato con *SparkSQL* produce come output una tabella che viene salvata sulla piattaforma *HIVE* all'indirizzo
`fnaldini_director_actors_db.actor_director_table_definitive`

2.2 Preprocessing dei dati

Come accenato nella sezione 2.1, alcuni dati all'interno del dataset risultavano incompleti, tuttavia queste mancanze riguardavano principalmente dati di importanza secondaria per il job da realizzare(ad esempio, sono assenti diverse date di nascita e di morte di molti attori, attrici e registi vissuti all'inizio del 900), mentre al contrario i dati necessari al corretto funzionamento del job non risultavano lacunosi nella loro espressività.

3 Jobs

3.1 Job: Trovare i tre attori/attrici diretti più di frequente per ogni regista, ordinando i registi per numero di film girati

Il job concordato coi docenti consisteva nella realizzazione di una ricerca che per ogni regista presente all'interno del dataset trovasse i tre attori/attrici con cui questi avesse lavorato più frequentemente, ordinando poi i registi per numero di film diretti.
Il primo passo da me compiuto è stato lo studio dei vari files e la modellazione

dello schema relazionale del database, da ciò ho dedotto che, sui sette files disponibili, solo tre erano necessari per la realizzazione del job: `title.basics`, `title.principals` e infine `name.basics`, `title.crew` al contrario di quanto si potesse evincere dalla descrizione presente sul sito, conteneva solamente informazioni ridondanti rispetto ai tre files sopracitati, di conseguenza ho scelto di non utilizzarlo. Il piano di esecuzione di massima che ho pensato è il seguente: estrarre i tutti i film da `title.basics` e tutti i registi da `title.principals`, eseguire un passo di join e raggruppare i film per regista, in modo da poter contare il numero film diretti; successivamente il risultato di questa operazione sarebbe stato messo in join con tutti gli attori/attrici, ottenuti filtrando il file `title.principals`, sulla base dell'identificatore del titolo; poi sarebbe stato il momento della ricerca degli attori con il maggior numero di collaborazioni per ogni regista; una volta terminato ciò, il passo successivo sarebbe stato un join tra l'output della fase precedente e `name.basics` così da sostituire i nomi di attori e registi ai loro identificatori alfanumerici, infine l'ultima operazione avrebbe ordinato i risultati sulla base del numero di film calcolato nei passi precedenti.
Durante la fase di pianificazione ho cercato di anticipare il più possibile i passi che includevano un filtraggio e/o una riduzione della dimensione del dataset, mentre ho posticipato il più possibile quelli che richiedevano una sua espansione(come ad esempio le operazioni di join)

3.1.1 MapReduce implementation

- *Comando per eseguire il Job*: hadoop jar director-actor-job-x.x.x-mr.jar, non è richiesto nessun parametro di configurazione in entrata.
- *Link all'esecuzione su YARN*:
  - `http://isi-vclust0.csr.unibo.it:19888/jobhistory/job/job_1552648127930_3634`;
  - `http://isi-vclust0.csr.unibo.it:19888/jobhistory/job/job_1552648127930_3635`;
  - `http://isi-vclust0.csr.unibo.it:19888/jobhistory/job/job_1552648127930_3636`;
  - `http://isi-vclust0.csr.unibo.it:19888/jobhistory/job/job_1552648127930_3637`;
  - `http://isi-vclust0.csr.unibo.it:19888/jobhistory/job/job_1552648127930_3638`;
  - `http://isi-vclust0.csr.unibo.it:19888/jobhistory/job/job_1552648127930_3639`;
  - `http://isi-vclust0.csr.unibo.it:19888/jobhistory/job/job_1552648127930_3746`;
- *Files/ Tabelle di Input*:`title.basics`, `title.principals`, `name.basics`.

- *Files/Tabelle di Output*: Viene generato un output per ogni passo di Map-Reduce che viene concluso, l'output finale è nella forma: `DirectorName(DirectedMoviesCount) ActorName1(MoviesCount1);...; ActorNameN(MoviesCountN)`. Dove MoviesCount rappresenta il numero di film in cui attore e regista hanno collaborato.

- *Descrizione dell'implementazione*: Il job è stato realizzato con sette passi di `Map-Reduce`, cercando di aggregare il più possibile operazioni compatibili all'interno di uno stesso passo. Tutti i job, fatta eccezione per l'ultimo, leggono e producono dati formattati secondo il paradigma *Key-Value*, utilizzando la classe `Text` come tipo di dato, visto la preponderanza di valori testuali all'interno dei dati da manipolare.
  I sette job in gioco sono i seguenti:

  - `Join between title.principals and title.basics`: questo job filtra in fase di mapping title.basics estraendo solo i film e title.principals estraendo solo i registi, dopodiché emette in output delle tuple aventi come chiave l'ID di un titolo, come valore l'ID del regista che ha diretto quel film e un valore numerico posto a uno per facilitare le fasi successive;
  - `Aggregation job for the directors`: esegue un aggregazione associando a ogni regista il numero di film diretti, l'output è una tupla nella forma IDFilm, Regista;FilmDiretti;
  - `Join between Actors and Directors`: Esegue un join tra l'output della fase precente e tutti gli attori presenti in title.basics, restituisce delle tuple nella forma IDRegista:IDAttore, FilmDiretti.
  - `Find for each director the three actors`: Questo è il job più importante tra tutti,durante la fase di Mapping proietta i dati utilizzando come chiave l'ID di un regista e in fase di reduce colleziona per ogni attore il numero di volte che compare, utilizzando una mappa come struttura di appoggio; dopodiché estrae dalla mappa i tre attori con frequenza maggiore e li scrive in output.
  - `Join between Names and Actors`: proiettando l'output della fase precedente sull'ID degli attori, esegue il join con la tabella name.basics, così da sostituire l'ID dei ogni attore col proprio nome.
  - `Join between Names and Directors`: esegue un operazione analoga alla precedente, ma considerando l'ID dei registi, inoltre riduce a uno il numero di tuple per ogni regista, combinando tra loro le varie tuple corrispondenti a uno stesso regista.
  - `Sort Job`: Utilizzando come chiave il numero di film diretti da ogni regista, ordina globalmente il risultato finale.

- *Considerazioni sulle performance*: Ogni job di quelli elencati sopra ha un tempo di esecuzione compreso tra un minimo di 30 s per i più veloci e un massimo di un minuto per quelli più lenti, rendendo così il tempo di

esecuzione complessivo attorno ai 6 minuti.
Sicuramente la presenza di due operazioni di Join consecutive, prima sulla base degli ID degli attori e in secondo luogo sull'ID dei registi può essere motivo di dubbio, tuttavia, nonostante sia in teoria possibile ottenere lo stesso risultato dei due join con un solo passo, non è conveniente farlo: infatti bisognerebbe anticipare tale join a prima dell'operazione di filtraggio dei tre attori per ogni regista, operazione che riduce considerevolmente le dimensioni dei dati in gioco e rende di conseguenza più leggeri i due join successivi.
Per come ho implementato i diversi job, mi è stato possibile applicare un combiner per ridurre i tempi di calcolo nel job che si occupava di trovare per ogni regista i tre attori: in quella situazione era infatti possibile aggregare parzialmente il numero di collaborazioni tra attore e regista a livello di mapper senza inficiare il risultato; ho pensato anche a se fosse possibile applicare un *Combiner* anche nella fase di aggregazione del numero di film diretti per ogni regista, tuttavia avendo bisogno di mantenere anche l'ID di ogni film, l'aggreagazione avrebbe dovuto calcolare il numero di film e allo stesso tempo conservare il numero di tuple. Ho provato infine a eseguire un alcune operazioni di tuning sul numero dei reducer da disporre in gioco, partendo da un numero massimo di 20(impostato di default da Hadoop) fino a giungere a numeri piuttosto bassi, come 2 o 3: quello che ho notato è che riducendo il numero di reducers, le prestazioni globali rimanevano più o meno costanti, ma al calare del numero di questi ultimi, i job che si occupavano di join tendevano a aumentare i loro tempi, mentre gli altri tendevano a ridurli.

3.1.2 SparkSQL implementation

- *Comando per eseguire il Job*: spark2-shell director-actor-job-x.x.x-spark.jar, si può specificare il numero di `executors` con l'argomento `--executors=` e il numero di tasks per ogni executor con il flag `--taskForExceutor=`.
- *Link all'esecuzione su YARN*: `http://isi-vclust0.csr.unibo.it:19888/history/application_1552648127930_3663/jobs/`;
- *Files/ Tabelle di Input*:`title.basics`, `title.principals`, `name.basics`.
- *Files/Tabelle di Output*: L'output consiste in una tabella a due colonne, DirectorName e ActorName, salvata su *Hive* all'indirizzo `fnaldini_director_actors_db.actor_director_table_definitive`
- *Descrizione dell'implementazione*: Il job è stato realizzato sfruttando appieno le potenzialità di SparkSQL, in particolare concentrandosi sull'utilizzo dei *Dataframe* e l'utilizzo del motore di query SQL messo a disposizione da SparkSQL. Lo schema di risoluzione del problema rimane praticamente lo stesso descritto nella sezione 3.1; tali passaggi sono però stati effettuati utilizzando l'espressività del linguaggio SQL, analogamente a quanto visto

a lezione durante sia il laboratorio su SparkSQL sia durante l'analisi del caso di studio.
Per quanto l'obiettivo fissato fosse la risoluzione del problema utilizzando solo query e costrutti SQL, così da mostrare appieno le potenzialità di SparkSQL, non è stato possibile raggiungere tale risultato: questo a causa di un limite noto di SparkSQL che non permette di richiamare campi appartenenti a una tabella esterna all'interno di una query annidata, di conseguenza il passo di ricerca di tre attori per ogni regista risultava non realizzabile con primitive SQL.
Per risolvere questo problema ho pensato così di utilizzare le primitive di Spark, trasformando l'input della fase in un `RDD` e ritrasformandolo in un `DF` alla fine della computazione, così da poter rimanere il più fedele possibile all'obiettivo sopracitato.

- *Considerazioni sulle performance*: Pur non stravolgendo il piano di esecuzione presentato nella sezione 3.1, con alcuni accorgimenti sono riuscito a migliorare considerevolmente il tempo di esecuzione, che da 6-7 minuti è calato a circa 4.

  - *Numero di tasks per stage*: abbassando il numero di task per stage da 200 a 8(numero calcolato considerando 4 cores per 2 executors) il tempo di esecuzione si è abbassato considerevolmente.
  - *Caching dei dataframe*: Salvando in cache alcuni tra i dataframe utilizzati più volte all'interno del programma, le performance sono ulteriormente migliorate,in particolare sono stati salvati il dataframe costruito filtrando `title.principals` sulla categoria registi, il dataframe `title.principals` e infine il dataframe ottenuto da `name.basics`
  - *Considerazioni sui join*: Come nella realizzazione dell'implementazione in map reduce, anche qui mi sono domandato se potesse esserci un modo per velocizzare la doppia operazione di join: tra le tabelle in gioco, name.basics compare in entrambi i passi di join e non necessita di particolari operazioni di processing prima di essere utilizzata, se tramite un join broadcast si potesse inviare una copia della tabella a ogni nodo e riuscire a riutilizzarla successivamente, allora si avrebbe un notevole speedup delle performance; purtroppo al momento non c'è modo di eseguire l'operazione sopra descritta(fonte: `https://issues.apache.org/jira/browse/SPARK-3863`)

4 Miscellaneous

4.1 Problematiche col Dataset

A metà del lavoro circa, ho scoperto un problema con il dataset, in particolare con la tabella `title.principals`: invece di contenere tutte le associazioni tra opere e attori/registi, contiene solamente tre o quattro attori per ogni pellicola,

questo implica che il risultato della query finale venga compromesso, poiché il job tiene conto solo degli attori che hanno recitato con ruoli da protagonisti/ coprotagonisti all'interno dei film.
Ad esempio per il regista Christopher Nolan l'attore con cui risultano più collaborazioni risulta Christian Bale, con cui risultano 4 film assieme(che sono il numero di film diretti da Nolan in cui Bale ha ruolo da protagonista), mentre invece Micheal Caine, attore con molte più collaborazioni di Bale con Nolan ma spesso come interprete di personaggi secondari, risulta solo terzo