Report del Progetto di Big Data: StackLite – Domande e tag di Stack Overflow

Martina Magnani – Mat. 0000855897
Mattia Vandi – Mat. 0000850806

September 5, 2019

Contents

1 Introduzione

1.1 Descrizione del Dataset

Il *dataset* che abbiamo scelto è *Stacklite*[1]; è un *dataset* di StackOverflow contenente domande di programmazione e i relativi tag. Questo set di dati è stato estratto il 2016-10-13 18:09:48 UTC e contiene domande fino al 2016-10-12. Questo include 12583347 domande non cancellate e 3654954 domande cancellate. Si tratta di tutti i dati pubblici all'interno dello *Stack Exchange Data Dump*, che è molto più completo (in quanto include anche i testi delle domande e delle risposte), ma richiederebbe un maggiore overhead computazionale per il download e l'elaborazione. Questo dataset è progettato per essere letto e analizzato facilmente. Allo stesso modo, questi dati possono essere esaminati all'interno di *Stack Exchange Data Explorer*, ma questo offre agli analisti la possibilità di lavorare localmente usando un *tool* a loro scelta.
Il link al repository è il seguente:
`https://github.com/dgrtwo/StackLite`.
Mentre, il link usato per scaricare il *dataset* (previa registrazione al portale) è il seguente: `https://www.kaggle.com/stackoverflow/stacklite/downloads/stacklite.zip`.
I link per il download diretto dei file sono disponibili (previa registrazione al portale) ai seguenti URLs: `https://www.kaggle.com/stackoverflow/stacklite/downloads/questions.csv` and `https://www.kaggle.com/stackoverflow/stacklite/downloads/question_tags.csv`.

1.1.1 Descrizione dei File

I file che costituiscono il dataset sono:

- *questions.csv*: il file contiene le domande fatte su Stack Overflow. Per ogni domanda sono disponibili le seguenti informazioni:
  - `ID`: identificativo della domanda.
  - `CreationDate`: data di creazione della domanda.
  - `ClosedDate`, data di chiusura della domanda, se disponibile.
  - `DeletionDate`, data di cancellazione della domanda, se disponibile.
  - `Score`: il punteggio assegnato alla domanda.
  - `OwnerUserID`: l'identificativo dell'utente che ha formulato la domanda, se disponibile.
  - `AnswerCount`: il numero di risposte fornite alla domanda, se disponibile.
- *question_tags.csv*: il file contiene i tag associati alle domande. Per ogni record sono disponibili:

[1] `https://www.kaggle.com/stackoverflow/stacklite`

  - `ID`: identificativo della domanda in cui è stato utilizzato il tag.
  - `Tag`: tag associato alla domanda.

I campi utilizzati per l'analisi sono `CreationDate`, `ClosedDate`, `Score` e `AnswerCount`. In particolare, i campi indicati sono stati utili per:

1. Mostrare in ordine descrescente i primi cinque tag che hanno ricevuto il punteggio più alto (`Score`) per ogni coppia mese-anno (`CreationDate`).
2. Calcolare il tasso di chiusura (`ClosedDate`) e la partecipazione media per ogni tag (`AnswerCount`) e discretizzala in tre intervalli: *alta*, *media*, *bassa*.

2 Preparazione dei Dati

- Utente di riferimento: `mvandi`.
- Nome della macchina (o indirizzo IP): `http://isi-vclust9.csr.unibo.it` (`http://137.204.72.242`).
- I percorsi ad ogni file su HDFS sono: `/user/mvandi/exam/data/questions.csv` e `/user/mvandi/exam/data/question_tags.csv`

3 Jobs

In questa sezione mostriamo, nel dettaglio, i job creati. Lo script da lanciare per eseguire un Job (che si trova al percorso `/user/mvandi/exam`), è il seguente: `./run.sh [options] <output_path>`.
Opzioni:

- `--mapreduce`: per eseguire il Job in Apache Hadoop MapReduce.
- `--spark`: per eseguire il Job in Apache Spark 2.
- `--job <job_n>`: numero del Job da eseguire (1 o 2).
- `--no-save`: per mantenere l'output del job su HDFS senza spostarlo nella cartella locale da cui è stato eseguiro il comando.

3.1 Job #1: breve descrizione

L'obiettivo di questo Job è determinare per ogni coppia mese-anno i cinque tag che hanno ricevuto il punteggio più alto in ordine decrescente.
Attraverso questa interrogazione è possibile osservare quali tag sono di maggiore interesse per gli utenti di StackOverflow e come gli interessi degli stessi cambiano nel tempo.

3.1.1 Implementazione in MapReduce

Il comando per eseguire il Job è il seguente:
`./run.sh --mapreduce --job 1 [options] <output_path>`.
È possibile specificare l'opzione `--no-save` per mantenere l'output del job su HDFS senza spostarlo nella cartella locale da cui è stato eseguiro il comando.

- Link diretti alla cronologia dell'applicazione su YARN:
  1. `http://isi-vclust0.csr.unibo.it:8088/proxy/application_1552648127930_3238`
  2. `http://isi-vclust0.csr.unibo.it:8088/proxy/application_1552648127930_3240/`
- File di Input:
  - `/user/mvandi/exam/data/questions.csv`.
  - `/user/mvandi/exam/data/question_tags.csv`.
- File di Output: il *path* del file di Output è fornito dall'utente che esegue lo script.

Per soddisfare questa interrogazione sono stati realizzati due Job:

- `Join`; per effettuare il join tra il file contenente le domande e il file contenente i tag relativi alle domande.
- `HighestScoreTags`; per calcolare i tag più votati per ogni coppia mese-anno.

Join L'obiettivo della fase di join è quello di ottenere un file con le seguenti coppie chiave-valore: `<<year, month>,<tag, score>>`.
Per realizzare tale obiettivo, durante la fase di mapping, ogni record viene trasformato in una coppia `<ID, Question>` nel caso si stia elaborando il file delle domande, mentre in caso si stia elaborando il file dei tag ogni record viene trasformato in una coppia `<ID, QuestionTag>`.
Durante la fase di reduce per ogni coppia `<Question, QuestionTag>` viene estratta ed utilizzata come chiave la coppia `<year, month>`, mentre la coppia `<tag, score>` viene utilizzata come valore.

HighestScoreTags L'obiettivo di questa fase è quello di ottenere una lista dei primi cinque tag che hanno ricevuto il punteggio più alto per ogni coppia `<year, month>`.
Per realizzare tale obiettivo, durante la fase di combining per ogni coppia `<year, month>` vengono sommati i punteggi di ogni tag ottenuti come output della fase di join. L'utilizzo del combiner ha permesso di risparmiare una parte di lavoro al reducer.
Durante la fase di reduce vengono ulteriormente sommati i punteggi di ogni tag, ordinati per punteggio in ordine decrescente e scelti i primi cinque.

3.1.2 Implementazione in Spark

Il comando per eseguire il Job è il seguente:
`./run.sh --spark --job 1 [options] <output_path>`
È possibile specificare l'opzione `--no-save` per mantenere l'output del job su HDFS senza spostarlo nella cartella locale da cui è stato eseguiro il comando.

- Link diretto alla cronologia dell'applicazione su YARN: `http://isi-vclust0.csr.unibo.it:18089/history/application_1552648127930_3187/jobs/`.
- File di Input:
  - `/user/mvandi/exam/data/questions.csv` e
  - `/user/mvandi/exam/data/question_tags.csv`.
- File di Output: il *path* del file di Output è fornito dall'utente che esegue lo script.

Dapprima vengono creati due RDD, il primo contenente tutte le domande e il secondo contenente tutti i tag realtivi alle domande formulate; di seguito viene applicato un filtro al primo RDD (quello contenente le domande) per selezionare le sole domande che sono state formulate dal 1 Gennaio 2012 al 31 Dicembre 2014; di seguito, per entrambi gli RDD, vengono raggruppati gli elementi per l'identificatore della domanda.
A questo punto è possibile effettuare il join per chiave tra i due RDD, di seguito gli elementi dell'RDD vengono trasformati in delle coppie `<<year, month>, <tag, score>>`, dove la coppia `<year, month>` rappresenta la coppia anno-mese in cui è stata formulata la domanda e la coppia `<tag, score>` rappresenta la coppia tag utilizzato per la domanda e relativo punteggio ottenuto. Di seguito viene effettuato il partizionamento dei valori utilizzando un `HashPartitioner` e vengono raggruppati per chiave (coppia `<year, month>`).
I valori vengono raggruppati per chiave (`score`) e viene calcolata la somma dei punteggi ottenuti, i valori vengono ordinati per punteggio in ordine decrescente (dal più grande al più piccolo) e vengono selezionati i primi cinque elementi.
Infine, vengono trasformati gli elementi in stringa e salvati in un file di testo al percorso di output specificato dall'utente.

3.1.3 Conclusioni

L'utilizzo di Apache Spark ha agevolato la scrittura del job in questione, oltre a migliorare la qualità del codice grazie al suo stile funzionale. A differenza dell'implementazione in Apache Hadoop MapReduce, la quale, pur non avendo una complessità elevata, ha reso necessaria la scrittura di molto codice.
Nonostante questo, il tempo di esecuzione in Apache Spark è stato maggiore rispetto a quello della sua controparte in Apache Hadoop MapReduce.

3.2 Job #2: breve descrizione

Questo Job ha l'obiettivo di calcolare, per ogni tag, il tasso di chiusura e mostrare la partecipazione media discretizzandola in: alta, media, bassa.
Con *tasso di chiusura* si intende il numero di domande aperte rispetto al totale.
Con *partecipazione media* si intende il rapporto tra il numero di risposte, e il numero totale di domande.

Attraverso questa interrogazione, è possibile mettere in relazione i due valori calcolati e valutare, per esempio, se un certo tag si riferisce ad argomenti di nicchia o argomenti difficili.
Infatti, un tag con **basso** tasso di chiusura e:

- partecipazione **medio/bassa**, potrebbe significare che il tag è relativo ad argomenti poco conosciuti.
- partecipazione **alta**, potrebbe significare che il tag è relativo ad un argomento ostico da risolvere per la comunità di StackOverflow.

Per quanto riguarda la **discretizzazione** della *partecipazione media*, abbiamo pensato di normalizzare la partecipazione media p_i, in un intervallo $[0, 1]$; applicando la seguente formula:

$$n_i = \text{normalize}\,(p_i) = \frac{p_i - \min(P)}{\max(P) - \min(P)} \tag{1}$$

dove:

- p_i è la partecipazione media per il tag i-esimo.
- P è l'insieme delle partecipazioni medie.

A questo punto, è possibile identificare le tre fasce di partecipazione, come segue:

1. **partecipazione bassa** se $n_i < \frac{1}{3}$;
2. **partecipazione media** se $\frac{1}{3} \leq n_i \leq \frac{2}{3}$;
3. **partecipazione alta** se $n_i > \frac{2}{3}$.

3.2.1 Implementazione in MapReduce

Il comando per eseguire il Job è il seguente:
`./run.sh --mapreduce --job 2 [options] <output_path>`.
È possibile specificare l'opzione `--no-save` per mantenere l'output del job su HDFS senza spostarlo nella cartella locale da cui è stato eseguito il comando.

- Link diretti alla cronologia dell'applicazione su YARN:
    1. `http://isi-vclust0.csr.unibo.it:8088/proxy/application_1552648127930_3239/`

2. `http://isi-vclust0.csr.unibo.it:8088/proxy/application_1552648127930_3241/`
3. `http://isi-vclust0.csr.unibo.it:8088/proxy/application_1552648127930_3242/`
4. `http://isi-vclust0.csr.unibo.it:8088/proxy/application_1552648127930_3243/`

- File di Input:
  - `/user/mvandi/exam/data/questions.csv` e
  - `/user/mvandi/exam/data/question_tags.csv`.
- File di Output: Il *path* del file di Output è fornito dall'utente che esegue lo script.

Con il paradigma *MapReduce*, per soddisfare l'interrogazione sopra descritta, è stato necessario realizzare più job di map-reduce.
In particolare, si sono resi necessari due job per l'estrazione della partecipazione media minima e della partecipazione media massima. Infatti, questi due valori non potevano essere calcolati direttamente nel Job `OpeningRateWithPaticipation`, in quanto in tale Job il raggruppamento è stato fatto per tag. Quindi i `Reducer` finali sono a conoscenza della sola partecipazione media relativa al tag (chiave) a loro assegnato; di conseguenza non possono effettuare confronti con le partecipazioni medie degli altri tag.

Di seguito, i Job implementati:

- `Join`; per effettuare il join tra i due file in input; quello contenente le domande e quello contenente i tag relativi alle domande.
- `AverageParticipationByTag`; per calcolare la partecipazione media di ogni tag.
- `MinMax`; per calcolare la partecipazione media minima e la partecipazione media massima tra tutti i tag.
- `OpeningRateWithPaticipation`; per calcolare il tasso di chiusura e discretizzare la partecipazione media per ogni tag.

Join Il Job seguente è stato necessario per unire i due file di input, in un unico file di output, contenente le seguenti coppie chiave-valore: `<<tag>,<questions>>`.
Per realizzare tale obiettivo, durante la fase di **mapping**, ogni record viene trasformato in una coppia `<ID, Question>` nel caso si stia elaborando il **file delle domande**, mentre in caso si stia elaborando il **file dei tag** ogni record viene trasformato in una coppia `<ID, QuestionTag>`.
Durante la fase di **reduce** per ogni coppia `<Question, QuestionTag>` viene estratto ed utilizzato come chiave il `tag`, mentre la domanda `question` viene utilizzata come valore.

AverageParticipationByTag Questo Job ha il compito di calcolare la partecipazione media di ogni tag (rapporto tra il numero di risposte e il numero totale di domande). Il file di input contiene coppie della forma: `<tag,questions>`.
Mentre il file di output deve contenere le seguenti coppie chiave-valore:
`<tag,averageParticipation>`
In questo caso, nella fase di **mapping**, per ogni domanda viene estratto il numero di risposte al fine di passare al `Combiner`, record nella forma:
`<tag,<questionCount,totalAnswers>>`.
Se il valore dell'`answerCount` è `null` vuol dire che la domanda in esame non ha ricevuto risposta e quindi ritorniamo `0`, altrimenti ritorniamo l'`answerCount`. Il `questionCount` è necessario al `Reducer` per calcolare la partecipazione media.
Nella fase di **combining**, vengono aggregati i valori attraverso una funzione di somma.
Anche nella fase di **reduce** viene aggregato il valore attraverso la funzione di somma e infine calcolata la partecipazione media. Il risultato è un file contenente record nella forma: `<tag,averageParticipation>`

MinMax Questo Job, dato il file di input con le coppie
`<tag,averageParticipation>`, ha il compito di estrarre la partecipazione media minima e la partecipazione media massima tra tutti i tag.
Per realizzare tale obiettivo, nella fase di **mapping** viene scritto un file con record nella forma:
`<0,<averageParticipation><averageParticipation>>`. Viene settata la stessa chiave (0) per tutti i record, perché è necessario trovare la partecipazione media minima e massima tra tutti i record e quindi il confronto deve essere fatto tra tutti i valori; di conseguenza è necessaria un'unica chiave.
Nella fase di **combing** viene fatta un primo confronto delle `averageParticipation`, per trovare il minimo e massimo locale al nodo.
Nella fase di **reduce** vengono estratte la partecipazione media minima e la partecipazione media massima finali.
In questo Job non viene scritto (`context.write`) nessun file di output, ma vengono settate due Java `Properties`, per memorizzare i valori di minimo e di massimo. La classe `Properties` rappresenta un insieme di proprietà, nella forma chiave-valore, che possono essere salvate in uno *stream* di dati e/o caricate da uno *stream* di dati. Ogni chiave e il suo valore corrispondente nell'elenco delle proprietà è una stringa. Abbiamo usato i metodi `load` e `store` per caricare e memorizzare le proprietà da/verso lo *stream* di input/output di HDFS.
È stato scelto di utilizzare le properties Java per evitare un ulteriore join tra il risultato della fase di join e il risultato della fase di estrazione del minimo e del massimo.

OpeningRateWithPaticipation Il Job in esame è quello finale; infatti ha l'obiettivo di calcolare, per ogni tag, il tasso di apertura e di discretizzare la partecipazione media in alta, media o bassa.

Il **Mapper** prende in input il file generato nella fase di Join, quindi quello contenente le coppie `<tag,questions>`, esegue il mapping del valore in input nella corrispondente classe java `Questions` e scrive un file con coppie nella forma: `<<tag>,<<openQuestions><questionCount><totalAnswers>>>`. Ovviamente nella fase di mapping, dato che si gestisce una domanda alla volta, in `openQuestions` ci sarà o 0 o 1 a seconda se la domanda è stata chiusa o meno. Per lo stesso motivo, nel `questionCount` ci sarà un 1 (rappresentante la domanda corrente) e in `totalAnswers` il numero di risposte ottenute dalla domanda in esame.
Il **Combiner** esegue una pre-aggregazione dei dati sommando `openQuestions`, `questionCount` e `totalAnswers`.
Il **Reducer** esegue la somma finale di tutti i valori relativi allo stesso tag; e con i risultati ottenuti calcola il tasso di apertura (`openingRate`) e la partecipazione media (`averageParticipation`) degli utenti.
A questo punto, il **Reducer** può caricare le `Properties`, relative alla partecipazione media minima e massima calcolate nel Job `MinMax`, e infine discretizzare la partecipazione media del tag nelle fasce *bassa*, *media* e *alta*

3.2.2 Implementazione in SparkSQL

Il comando per eseguire il Job è il seguente:
`./run.sh --spark --job 2 [options] <output_path>`.
È possibile specificare l'opzione `--no-save` per mantenere l'output del job su HDFS senza spostarlo nella cartella locale da cui è stato eseguiro il comando.

- Link diretto alla cronologia dell'applicazione su YARN: `http://isi-vclust0.csr.unibo.it:18089/history/application_1552648127930_3237/jobs/`.
- File di Input:
  - `/user/mvandi/exam/data/questions.csv` e
  - `/user/mvandi/exam/data/question_tags.csv`.
- File di Output: Il percorso del file di Output è fornito dall'utente che esegue lo script.

In questo Job SparkSQL vengono creati innanzitutto due `DataFrame` relativi ai file *.csv* di input. Durante il *load* dei file, viene controllato se esistono già le relative tabelle Parquet:

- se esistono, i `DataFrame` vengono creati a partire da queste al fine di ottimizzare i tempi di caricamento; infatti l'accesso alle tabelle Parquet è più veloce rispetto ai file memorizzati in HDFS.
- se non esistono, i `DataFrame` vengono creati a partire dai file *.csv* e subito dopo create le rispettive tabelle Parquet.

Una volta creati i due `DataFrame`: `questionsDF` e `questionsTagDF` è possibile ottenere i risultati richiesti, eseguendo operazioni *sql-like*.
Da `questionsDF` si filtrano le domande che hanno una `creationDate` compresa tra *2012-01-01T00:00:00Z* e *2014-12-31T23:59:59Z* e quelle che non sono state cancellate; quindi con `deletionDate` uguale a `null`. Dopo di che si esegue il `join` tra i due `DataFrame`.
Di seguito elencheremo le operazioni effettuate sul `DataFrame` ottenuto tramite join:

1. `withColumn`; si aggiunge una colonna `openQuestions`; se `closedDate` è `null` avrà valore 0, altrimeni 1.

2. `groupBy`; si raggruppano i record per `tag` aggregando i valori di `openQuestions` tramite somma; allo stesso modo quelli in `answerCount`; e creando una nuova colonna `questionCount` rappresentante il numero di record (= numero di domande per tag).

3. `where`; si filtrano i tag che compaiono in più di una domanda (`questionCount` maggiore di 1).

4. `withColumn`; si aggiunge una colonna per calcolare la partecipazione media; `averageParticipation` è determinata dal rapporto tra `totalAnswers` e `questionCount`.

5. `cache`; si fa il cache del `DataFrame` in memoria al fine di memorizzare le operazioni effettuate.

A questo punto è possibile individuare la partecipazione media minima e massima, utilizzando gli operatori di minimo e di massimo forniti dalle librerie di SparkSQL. In particolare, eseguiamo un `crossJoin` del `DataFrame` precedentemente creato con un secondo `DataFrame` che ha come unico record i valori di `minParticipation` e `maxParticipation`.
Con queste informazioni, è ora possibile eseguire la discretizzazione della partecipazione media e calcolare il tasso di apertura per tutti i tag.
Il risultato, infine, viene scritto su Parquet.

3.2.3 Conclusioni

L'utilizzo di Apache Spark SQL ha permesso di migliorare notevolmente i tempi di esecuzione rispetto allo stesso job implementato in Apache Hadoop MapReduce, oltre a migliorare la qualità del codice grazie al suo stile funzionale.
L'utilizzo di un linguaggio di interrogazione SQL-like ha reso immediata la scrittura del job in questione; a differenza dell'implementazione in Apache Hadoop MapReduce, la quale ha richiesto uno sforzo maggiore e non si è dimostrata triviale.