Spark Batch Processing

Progetto di Sistemi e Architetture per Big Data

Matteo Federico
0321569

Elisa Verza
0311317

***Abstract*—Questo documento si pone lo scopo di spiegare il metodo adoperato e lo stack architetturale utilizzato per rispondere a 3 query assegnate nel primo progetto del corso di "Sistemi e Architetture per Big Data" della facoltà di Ingegneria Informatica magistrale dell'università di Roma Tor Vergata.**

I. Introduzione

Il progetto svolto consiste nello svolgimento di tre query su dati, elaborati tramite processamento batch, relativi ad azioni scambiate sui mercati di Parigi, Francoforte\Xetra e Amsterdam nel periodo 08/11/2021- 14/11/2021.
Il lavoro è stato diviso in quattro fasi:

- Progettazione dell'architettura
- Preprocessing dei dati
- Analisi e svolgimento delle query
- Analisi delle performance.

I framework utilizzati sono: Apache Spark per il processamento batch, Nifi per data ingestion e preprocessing, HDFS per storage su file system distribuito e MongoDB per il salvataggio dei risultati su DB noSQL. L'intera architettura inoltre è stata sviluppata in modo distribuito su container Docker in un cluster locale utilizzando Docker Compose come orchestratore.

II. Architettura

Uno schema dell'architettura di alto livello è mostrato in 1. Il sistema è containerizzato, ed ogni container espone le porte necessarie per poter accedere all'UI del relativo servizio offerto.
Il csv contenente i dati da processare viene scaricato direttamente da nifi tramite una chiamata get http. I dati preprocessati da nifi vengono poi salvati su HDFS, il quale ha più nodi, ognuno dei quali corrisponde ad un container, in cui uno è il master e tutti gli altri i worker. Anche Spark ha un container master e più container worker, tramite il master è possibile avviare l'esecuzione delle query, che come prima operazione farà il retrieve del csv contenente i dati da HDFS. I csv di output di Spark con i risultati delle query vengono uniti e il risultato aggregato viene salvato in un csv caricato sull'HDFS, Nifi procederà quindi a fare l'inject dei dati caricati per poi salvarli su MongoDB.

III. Dataset

Per lo svolgimento delle query è stata asseganata una porzione di un dataset originariamente messo a disposizione per l'evento "Call for Grand Challenge Solution" di debs del 2022. Il dataset contiene un insieme di dati finanziari dove ogni riga rappresenta uno scambio di strumenti finanziari, suddivisi in 39 campi. Quelli di maggior interesse, utili per rispondere alle query sono i seguenti:

- ID: una stringa formata da due parti separate da un punto, la prima parte rappresenta la sigla dello strumento finanziario, la seconda rappresenta il suo mercato di appartenenza.
- SecType: un campo che può assumere solo due valori: "E" (equities) e "I" (indices), e permette di differenziare le azioni dagli indici.
- Last: é il campo contenente il prezzo dell'ultima offerta per lo strumento finanziario.
- Trading Time e Trading Date: questi due campi sono presenti solo per le azioni, quindi per le entry con SecType posto ad "E", e rappresentano rispettivamente l'orario in cui occorre l'evento (in formato "HH:MM:ss.mmm") e la data (in formato "dd-mm-yyyy").

Il dataset inoltre si compone di un header composto da 11 righe iniziali di commento che hanno come primo carattere "#" ed hanno il compito di spiegare le finalità del dataset, una riga contenente l'effettivo header del file csv, e un'ulteriore riga di commento che va a dare una spiegazione di ogni colonna del dataset.

IV. NIFI

NIFI é stato utilizzato per l'ingestion dei dati, in particolare, é stato utilizzato per prendere il dataset e caricarlo all'interno del sistema e in seguito, dopo l'elaborazione delle query da parte di Spark, ha avuto il compito di prendere i file da HDFS e salvare i risultati su MongoDB.

A. Ingestion e preprocessing

In una prima fase viene preso il dataset tramite url. Successivamente viene effettuato il preprocessing sul dataset applicando una serie di filtraggi e trasformazioni iniziali comune a tutte le query per fare in modo che Saprk, al momento effettivo del processamento, potesse lavorare su dati più puliti possibile. In particolare è stato:

- Eliminato l'header
- Ridotto il numero di colonne, al fine di lasciare solo le informazioni necessarie (ovvero le 5 colonne descritte in precedenza)
- Eliminate le righe che avevano uno dei campi selezionati al punto prima vuoti o privi di informazione.

Per il salvataggio del dataset su HDFS è stata creata un'apposita cartella chiamata "/cartellaNIFI".
E' riportata la configurazione di processori NIFI usati per implementare il processo descritto in figura 2.

B. Salvataggio risultati

A seguito del salavataggio da parte di Spark dei risultati sul HDFS nell'apposita "/cartellaResult" subentra nuovamente Nifi perfarne ingestion e convertire i file in JSON per caricarli su un database NOSQL, MongoDB. In particolare gli elementi di ogni query sono stati caricati su un Database all'interno di MongoDB chiamato Result il quale contiene una collection per ogni query. Lo schema dei processori NIFI utilizzati è rappresentatna nella figura3

V. Implementazione Query

A. Query 1

L'obiettivo della prima query è quello di trovare per ogni ora, il valore minimo, medio e massimo del prezzo di vendita per tutte le azioni (aventi quindi SecType="E") scambiate sul mercato di Parigi("FR").
A tale scopo è stato creato un RDD a partire dalla lettura del dataset da HDFS. E' stata poi effettuata una trasformazione map che ha permesso di ottenere un RDD dove ogni elemento é composto da sette componenti ovvero il nome dell'azione, il mercato, il SecType, il valore, la data, l'ora e una strigna composta da data/azione/ora. Su questo nuovo RDD si è proceduto al filtraggio (tramite la trasformazione filter) per tenere gli elementi che avessero campo SecType ad "E" e il campo mercato uguale a "FR". E' stata poi effettuata una map per trasformare ogni elemento in una coppia ("data/ora/azione","valore") e quest'ultimo RDD é stato memorizato in ram con il metodo cache().
In seguito per il calcolo del minimo e del massimo è stata utilizzata una reduceByKey usando come funzione aggregatrice rispettivamente min e max, mentre per la media una combineByKey che ci ha permesso di calcolare per ogni chiave la somma dei valori ed un contatore con il numero di occorrenze della chiave in modo tale da poter applicare una mapValues in cui si esegue il quoziente tra i due valori mantenendo il contatore delle occorrenze per avere il numero di eventi usati per l'esecuzione della query.
Si hanno così tre RDD risultanti, uno per il minimo, uno per il massimo e uno per la media con il numero di elementi che sono stati uniti tramite una fullOuterJoin. Infine viene salvato il risultato ottenuti su HDFS.
E' possibile vedere il DAG della query uno in figura 7

B. Query 2

La seconda query chiede di trovare per ogni giorno, per ogni azione la media e la deviazione standard della variazione del prezzo di vendità su finestre temporali di un ora, infine farne una classifica e trovare per ogni giorno una lista delle cinque azioni che hanno ottenuto la migliore e la peggiore variazione media.
Come operazioni preliminari viene letto il dataset dal HDFS in un RDD. Si procede quindi ad effettuare un'operazione di raggruppamento, con una groupBy, che ha permesso di creare un RDD di tipo chiave valore, utilizzando come chiave una stringa cosi formata: data/ora/ID e il valore dell'azione come valore. Sono seguite due trasformazioni effettuate tramite map. La prima per ogni giorno, per ogni ora, per ogni azione trova l'ultimo valore inserito prima del passaggio all'ora successiva. Quindi ogni entry del RDD ha chiave, composta da data/ora/ID, e il valore più recente all'interno dell'ora. La seconda ha permesso di trasformare ogni elemento in una coppia che ha come primo valore una stringa formata da "data/azione" e come secondo una lista contenente ora e valore dell'azione.
Successivamente con un raggruppamento, tramite groupByKey, si è aggregato sulla chiave, ovvero la stringa "data/azione", che quindi ha ora come valore una lista di valori e di ora successivamente ordinata in modo ascendente. E' stata effettuata un'altra tasformazione di map in modo tale che ogni entry del RDD contenesse la stringa "data/azione", una lista contenente le variazioni orarie per l'azione in questione e il conteggio degli eventi considerati. Viene poi effettuato un filtraggio per eliminare gli elementi che non hanno variazione di valore e quindi presentano la lista vuota.
Infine una trasformazione stima la media e la deviazione standard per ogni elemento dell'RDD. L'RDD risultante è stato salvato in memoria con l'operazione di cache(), in modo tale da potervi ricorrere sia per trovare gli elementi migliori che gli elementi peggiori.
Per soddisfare la prima richiesta le entry sono state prima raggruppate per data e ordinate in modo decrescente secondo la media ecosì presi i primi 5 elementi. Per la seconda richiesta è stato seguito lo stesso metodo, ma ordinando in modo decrescente secondo la media.
Infine i risultati così ottenuti sono stati salvati come file csv all'interno di HDFS.
E' possibile vedere il DAG della query due in figura **??**

C. Query 3

La terza query richiedeva di calcolare il 25-esimo, 50-esimo e 75-esimo percentile per mercato (Francia, Amsterdam, Francoforte) sulla la variazione di prezzo giornaliera delle singole azioni.
A seguito della lettura del dataset da HDFS viene creato un RDD con i seguenti campi: data.azione, ora e valore azione. Si procede con il raggruppamento sulla chiave data.azione in modo tale che si abbia come valore una lista di coppie ora, valore azione. In questo modo è possibile filtrare per eliminare le entry che presentano un solo valore per giornata e

di cui quindi non è possibile calcolare la variazione di prezzo. Con una trasformazione tramite map si ordinano le coppie ora, valore azione sul campo ora e si procede a calcolare la variazione giornaliera del prezzo facendo la differenza tra il primo e l'ultimo elemento della lista ordinata. In questo modo ogni entry sarà composta da una nuova chiave formata nel seguente modo: data_mercato, dove il mercato di appartenenza dell'azione viene ricavato dall'ID da cui viene scartato il nome dell'azione, la variazione di prezzo e dal numero di eventi considerati per effettuare il calcolo che quindi sarà un numero fisso posto a due. A seguito di un nuovo raggruppamento sulla chiave si passa ad un'ultima trasformazione che prende in input le entry costituite dalla chiave e da una lista contenete le coppie, variazione di prezzo, conteggio eventi e ordina la lista sulla variazione di prezzo e se ne prendono quelle della lista poste nella posizione 25%, 50%, 75% della lunghezza totale. Inoltre viene fatta la somma dei contatori per vedere il numero totale degli eventi considerati. L'output della trsformazione viene salvato come file csv all'interno di HDFS.
E' possibile vedere il DAG della query tre in figura **??**

VI. Grafana

Come strumento di visualizzazione dei risultati delle query è stato utilizzato il framework grafana, anche quest'ultimo containerizzato ed esposto sulla porta 3000.
Grafana offre una serie di connettori per riceve i dati e poter visuallizzare tramite delle dashbord contenenti infografiche.
Tra i vari tipi di connettori di grafana messi a disposizione, si è deciso di utilizzare quello per i file csv il quale tramite il protocollo http esegue un operazione di get sull'URL passato nella configurazione. In particolare sono stati dichiarati tre connettori, uno per ogni query, in modo tale che venisse richiesto un file contenente i risultati per connettore.
E' stato anche instanziato anche un web server chiamato "Simple" contenerizzato con un volume attaccato alla cartella dei risultati e che espone verso la rete interna tra i container i file dei risultati.
Per ogni Query è stata quindi creata una dashboard contenente varie infografiche che permettono di visualizzare in modo più semplice e diretto i risultati ottenuti:

- La prima query è stata rappresentata con:
  - due grafici di tipo Time series che vanno ha vedere il numero di azioni che occorrono nel tempo e il valore medio nel tempo.
  - un grafico a torta che rappresenta le 10 azioni francesi che hanno avuto il maggior numero di transazioni
  - e un grafico Gauge per rappresentare il valore medio dei valori massimi, dei valori minimi e dei valori medi di tutte le transazioni francesi in tutti i giorni.
- La seconda query è stata rappresentata con una dashborad composta dai seguenti grafici:
  - un grafico di tipo Time Series che permette di visualizzare la classifica per ogni giorno e mette in evidenza la presenza ripetuta di alcune azioni più volte all'interno delle varie classifiche
  - due grafici di tipo "Bar Gauge" che mostrano le azioni che hanno riportato la migliore variazione e la peggiore variazione per ogni giorno.
- Per l'ultima query i risultati ottenuti evidenziano la mancanza di transazioni sul mercato di Amsterdam e di Francoforte. Pertanto i grafici considerati coprono solo il mercato di Parigi e rappresentano l'andamento dei percentili nel tempo e il loro valore nei vari giorni sottolineando che per ogni giorno la mediana rimane comunque sullo zero.

VII. Analisi dei risultati e performance

Per l'analisi delle performance si è lavorato su due fronti differenti:

- La prima analisi voleva andare a trovare il minimo numero di nodi Spark che andassero a minimizzare i tempi di risposta del sistema per le tre query, in particolare, sono state definite quattro configurazioni ognuna con un numero differente di worker Spark. Per ogni configurazione e query sono state effettuate dieci simulazioni, prendendo per ognuna i tempi di esecuzione tramite la libreria Time messa a disposizione da python.
  Il risultato di queste analisi ha messo in luce che con l'aumentare del numero di nodi worker andasse anche ad aumentare il tempo di elaborazione di tutte le query. La spiegazione di questo risultato può essere trovata nel fatto che il dataset di partenza aveva una dimensione di circa 450MB, ma già dopo l'applicazione del preprocessamento la dimensione del dataset é diminuità a 25MB (una diminuzione di un fattore 18 della dimensione originaria) per tanto il dataset é abbastanza piccolo e quindi con l'aumentare del numero di nodi il tempo effettivo di miglioramento dei nodi é minore del tempo di overhead che hanno i nodi per sincronizzarsi tra loro. Nella figura seguente vi é riportato l'andamento dei tempi di elaborazione per le varie Query 4.
- La seconda analisi ha lo scopo di confrontare i tempi di esecuzione delle query implementate utilizzando gli RDD rispetto a quelle realizzate eseguendo solo query sql. Quindi si confortano le api di più basso livello (RDD) con quelle con un livello di astrazione più alto (SQL). Infatti si nota come i tempi siano nettamente migliori nel caso degli RDD come atteso. La prima query è quella che presenta minor differenza nei tempi, con un incremento percentuale circa del 54.5%. Per quanto riguarda la seconda query l'incremento percentuale sale notevolmente arrivando al 317.4%. Infine per la terza query sono state proposte tre implementaizioni, oltre alle due già discusse una terza rappresenta un ibrido in quanto dopo aver calcolato la variazione giornaliera di prezzo per azione, come fatto per gli RDD, si effettua la coversione in dataframe e vengono calcolati i percentili tramite delle query sql. La soluzione ibrida e quella con gli RDD risultano essere equivalenti, mentre la query sql pura subisce un incremento percentuale del 291.3%.

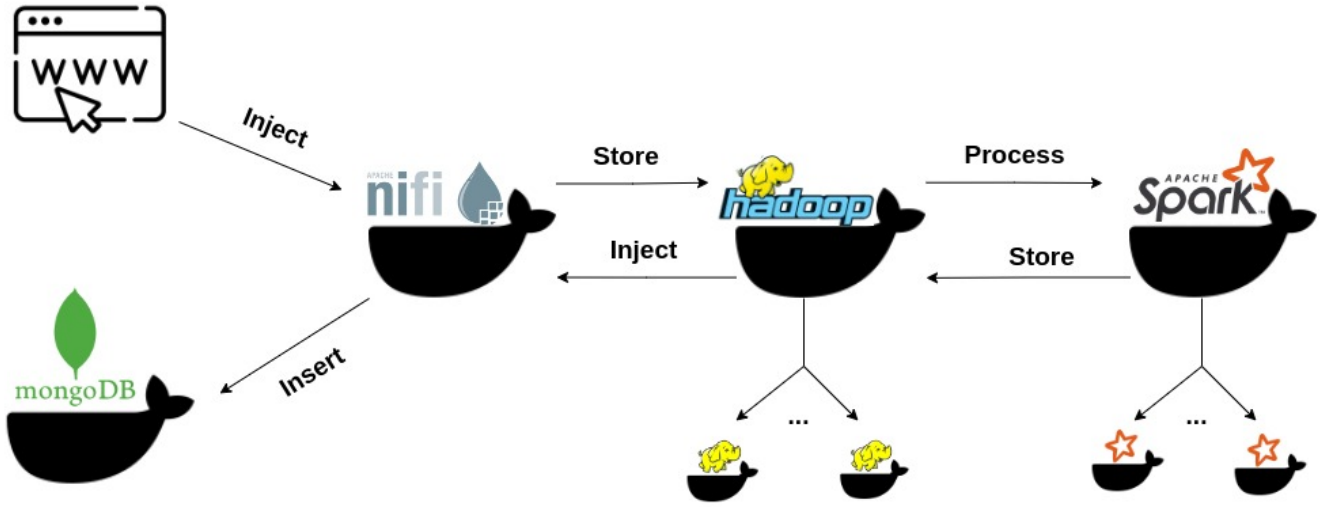

Fig. 1. Architettura

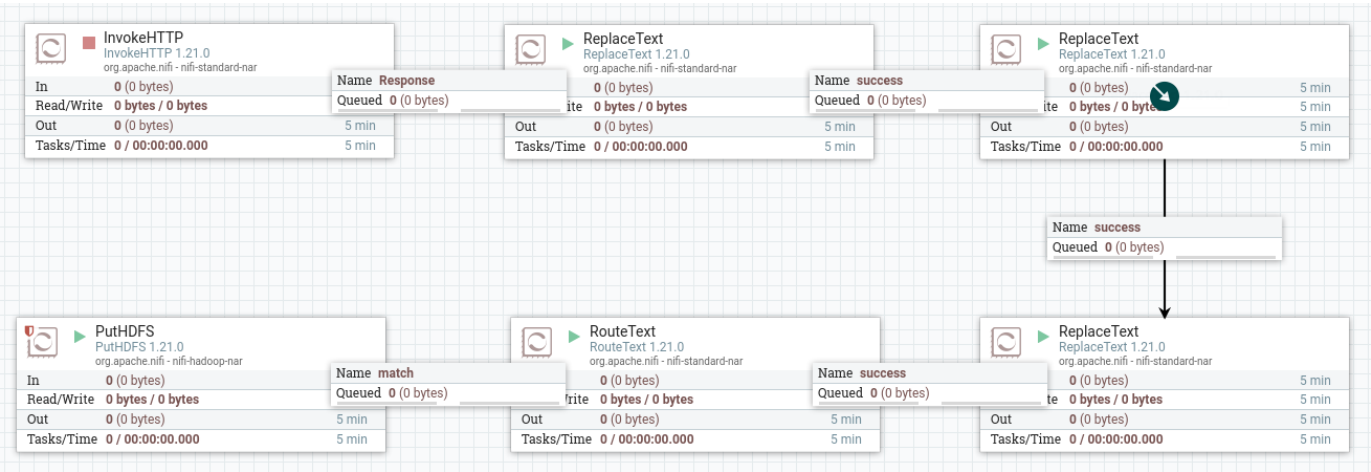

Fig. 2. Schema NIFI Fase 1

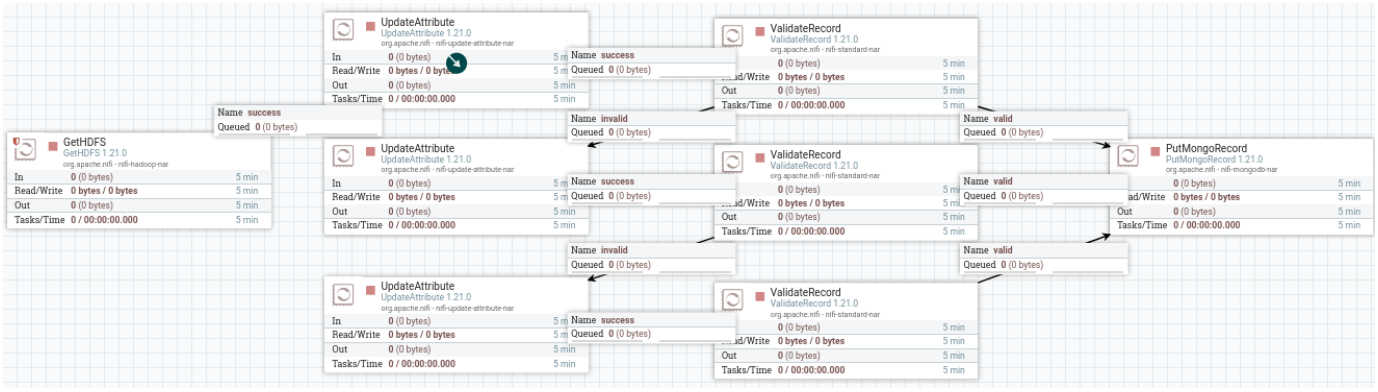

Fig. 3. Schema NIFI Fase 2

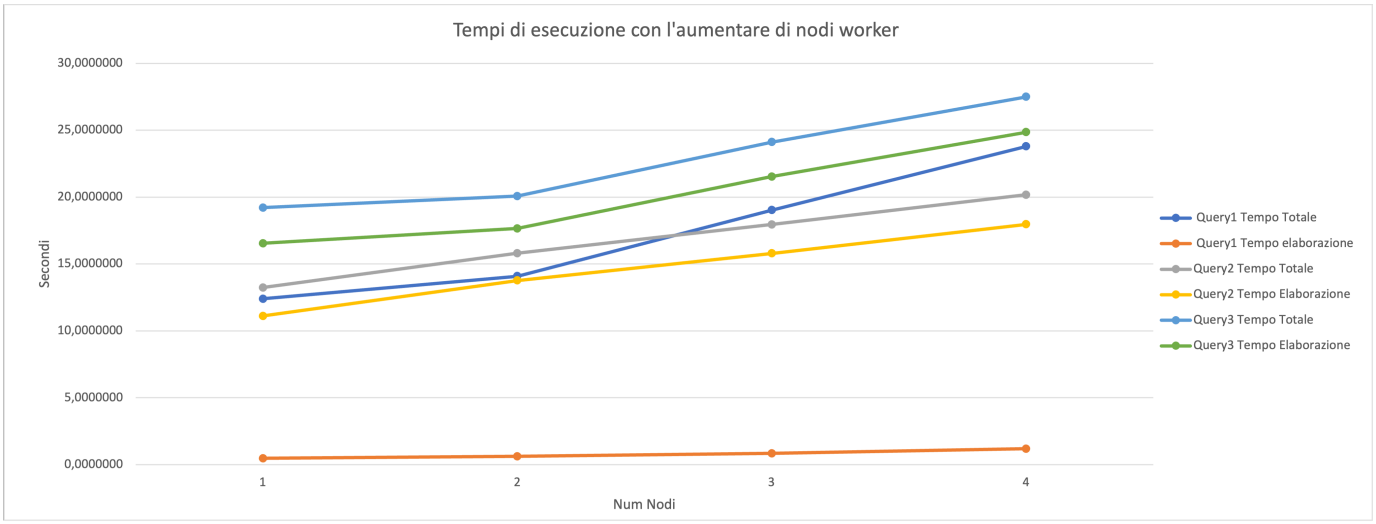

Fig. 4. GraficoPerformance1

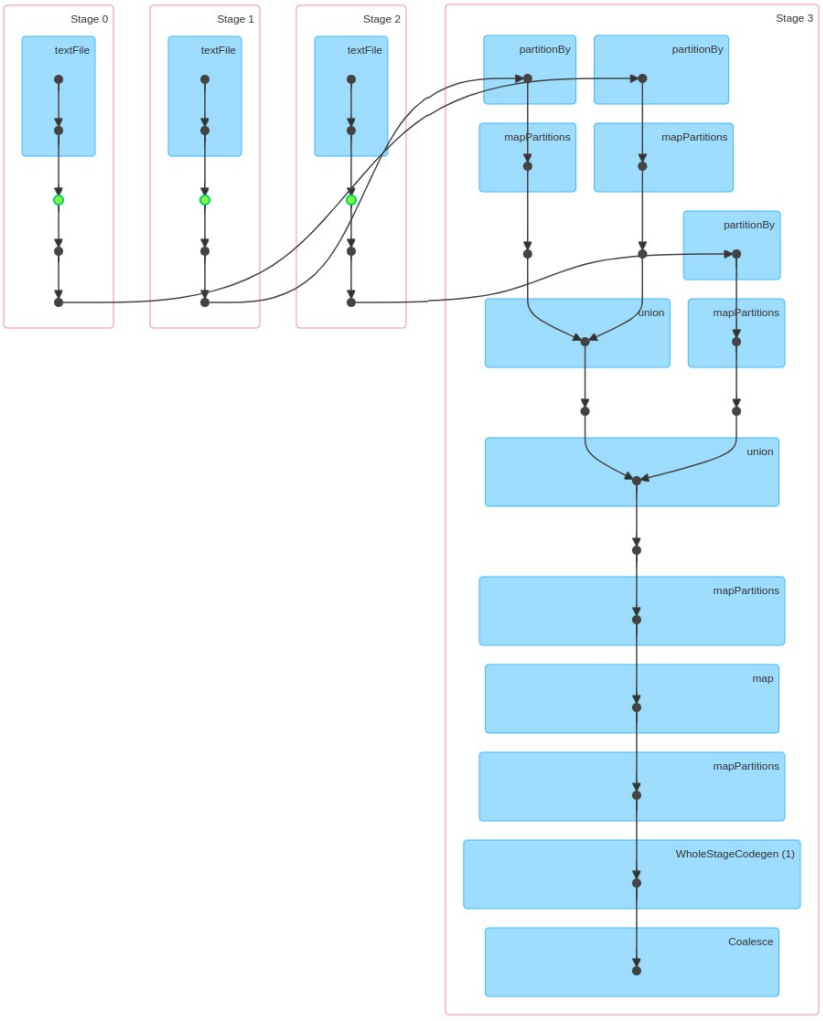

Fig. 5. DAG query 1

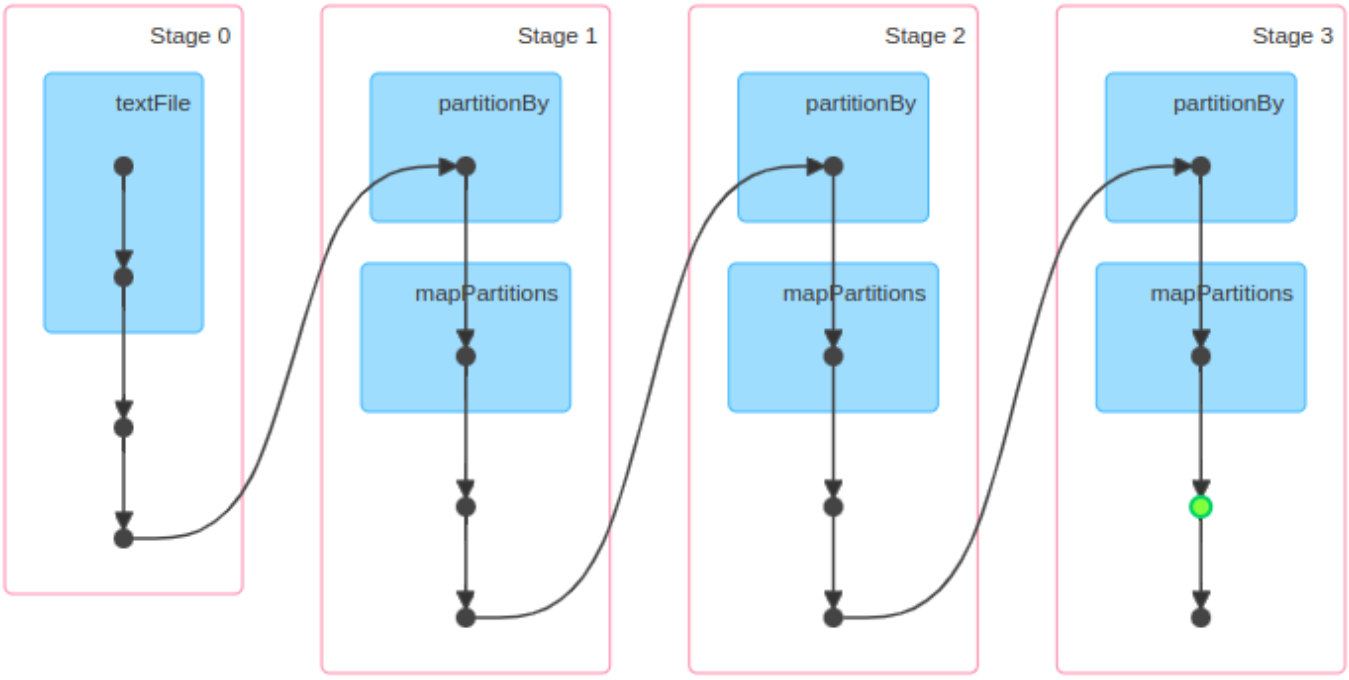

Fig. 6. DAG query 2

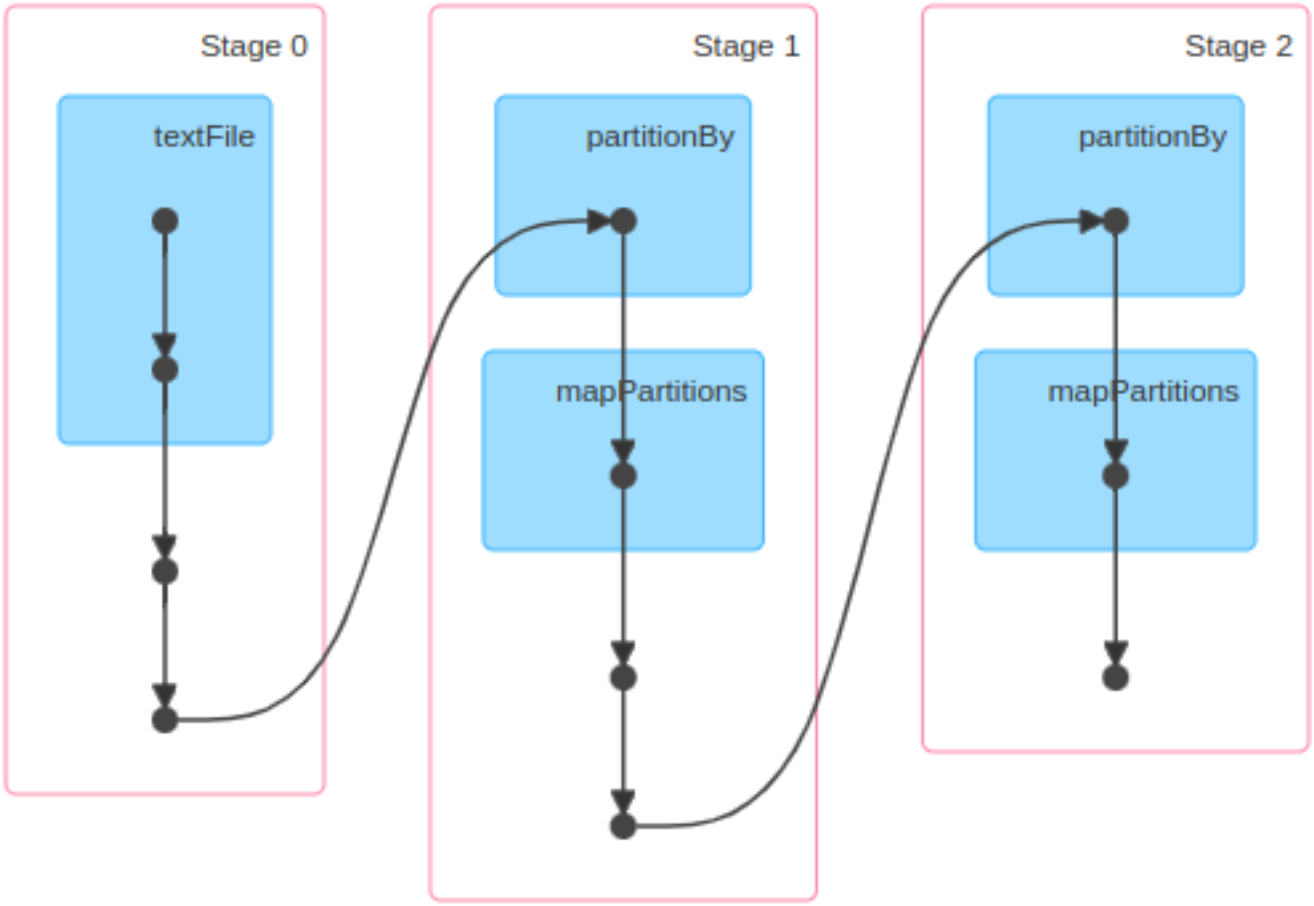

Fig. 7. DAG query 3