

**Report on Big Data project:  
Analisi delle Aperture nel gioco degli  
scacchi**

Giovanni Poggi - Mat. 938735

June 17, 2021

# Contents

<b>1</b>	<b>Introduzione</b>	<b>3</b>
1.1	Descrizione del Dataset . . . . .	3
1.2	Descrizione del file . . . . .	3
<b>2</b>	<b>Data Preparation</b>	<b>4</b>
<b>3</b>	<b>Job</b>	<b>5</b>
3.1	Implementazione MapReduce . . . . .	5
3.1.1	Job 1 - Calcolo totale partite vinte, perse e pareggiate in base all'apertura . . . . .	5
3.1.2	Job 2 - Calcolo delle percentuali di vittoria ed ordinamento	7
3.1.3	Note . . . . .	8
3.1.4	Job 3 - Calcolo della percentuale di utilizzo dell'apertura	8
3.1.5	Note . . . . .	8
3.2	Implementazione Spark . . . . .	8
3.2.1	Spark . . . . .	9
3.2.2	Note . . . . .	9

# 1 Introduzione

## 1.1 Descrizione del Dataset

Nella realizzazione di questo Progetto si è effettuata l'analisi del dataset *Chess Games - 6.2 Million chess games played on LiChess* scaricabile da qui: [https://www.kaggle.com/arevel/chess-games?select=chess\\_games.csv](https://www.kaggle.com/arevel/chess-games?select=chess_games.csv).

Il dataset contiene 6,25 milioni di partite di scacchi giocate su *lichess.org* nel Luglio 2016. È composto da una sola tabella, in formato csv ed occupa complessivamente 4.1 GB di memoria.

## 1.2 Descrizione del file

La tabella contiene un totale di 15 colonne che descrivono tutti gli elementi caratteristici di una partita di Scacchi.

Qui di seguito la spiegazione delle colonne presenti:

1. **Evento**: Tipo di gioco;
2. **Bianco**: ID o Nome Utente del giocatore che utilizza le pedine bianche;
3. **Nero**: ID o Nome Utente del giocatore che utilizza le pedine nere;
4. **Risultato**: Il risultato della partita (1-0 significa che ha vinto il Bianco e 0-1 significa che ha vinto il Nero, altri risultati indicano il pareggio o particolari partite speciali);
5. **Data UTC**: Data di inizio partita;
6. **Ora UTC**: Ora di inizio partita;
7. **WhiteElo**: Valutazione ELO del giocatore Bianco (Sistema di punteggi dei giocatori di scacchi, questo sistema permette la creazione di classifiche a punti autocorrettive nel tempo);
8. **BlackElo**: Valutazione ELO del giocatore Nero;
9. **WhiteRatingDiff**: Differenza di punti in classifica del bianco dopo la partita;
10. **BlackRatingDiff**: Differenza di punti in classifica del nero dopo la partita;
11. **ECO**: Apertura in codifica ECO (sistema di classificazione delle aperture);
12. **Apertura**: Nome dell'apertura;
13. **TimeControl**: Tempo del gioco in secondi;
14. **Terminazione**: Motivo della fine del gioco;
15. **AN**: Movimenti in formato Movetext;

In particolare, in questo progetto ci si focalizzerà principalmente sulle colonne riportanti le Aperture ed il Risultato delle varie Partite effettuate. Qui di seguito un paio di immagini di come si presentano i dati nella tabella:


▲ Event Game type	▲ White White's ID	▲ Black Black's ID	▲ Result Game Result (1-0 White wins) (0-1 Black wins)	📅 UTCDate UTC Date
Blitz 37% Classical 24% Other (2405818) 38%	118945 unique values	115946 unique values	1-0 50% 0-1 46% Other (240218) 4%	
Classical	eisaaaa	HAMID449	1-0	2016.06.30
Blitz	go4jas	Sergei1973	0-1	2016.06.30

Figure 1: First Part of Table CSV

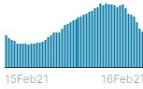
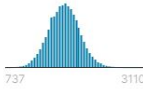
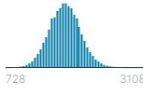


📅 UTCtime UTC time	# WhiteElo White's ELO	# BlackElo Black's ELO	# WhiteRatingDiff White's rating points difference after the game	# BlackRatingDiff Black's rating points difference after the game
				
22:00:01	1981	1896	11.0	-11.0
22:00:01	1641	1627	-11.0	12.0

Figure 2: Second Part of Table CSV

## 2 Data Preparation

Il file utilizzato per questo progetto è stato memorizzato in questo percorso: `hdfs:/user/gpoggi/chessanalysis`.

Il dataset è consistente, non presenta problemi relativi a dati mancanti, non sembrano esserci dati duplicati o di formato errato. Di conseguenza si è deciso di non effettuare alcuna operazione di pulizia preliminare dei dati.

▲ ECO	▲ Opening	▲ TimeControl	▲ Termination	▲ AN
Opening in ECO encoding	Opening name	Time of the game in seconds.	Reason of the game's end	Movements in Movetext format.
A00	Van't Kruijs Opening	300+0	Normal	6188849 unique values
A40	Scandinavian Defen...	180+0	Time forfeit	
Other (5542427)	Other (6010845)	Other (4232429)	Other (14759)	
D18	Slav Defense	300+5	Time forfeit	1. d4 d5 2. c4 c6 3. e3 a6 4. Nf3 e5 5. cxd5 e4 6. Ne5 cxd5 7. Qa4+ Bd7 8. Nxd7 Nxd7 9. Nc3 Nf6 10. ...
C28	King's Pawn Opening: 2.b3	300+0	Normal	1. e4 e5 2. b3 Nf6 3. Bb2 Nc6 4. Nf3 d6 5. d3 g6 6. Nbd2 Bg7 7. g3 Be6 8. Bg2 Qd7 9. 0-0 0-0 10. c3 ...

Figure 3: Third Part of Table CSV

### 3 Job

I due job concordati sono stati i seguenti:

1. *Date le aperture prese dal dataset, si effettui un calcolo del numero totale di partite in cui sono state utilizzate, quante volte la partita è stata vinta, quante persa e quante pareggiata;*
2. *Dati i risultati precedentemente ottenuti, si effettui il calcolo della percentuale di vittorie di tali Aperture e si disponga un ordinamento discendente di tali sulla base della percentuale;*

L'implementazione dei due Job è stata effettuata con MapReduce e Spark.

#### 3.1 Implementazione MapReduce

Il Job è stato implementato come lista ordinata di più Job MapReduce eseguiti in successione. Per una più corretta comprensione dell'output si è pensato di realizzare delle cartelle intermedie create alla fine di ogni Job e memorizzate all'interno dei percorsi `hdfs:/user/gpoggi/stageout` e `hdfs:/user/gpoggi/outputchess`. La cartella di visualizzazione dell'output finale a fine computazione è salvata all'indirizzo `hdfs:/user/gpoggi/finaloutput`. Per la realizzazione dell'elaborato si è pensato di dividere la consegna in tre Job di MapReduce, descritti qui di seguito.

##### 3.1.1 Job 1 - Calcolo totale partite vinte, perse e pareggiate in base all'apertura

Questo primo Job è composto dalle seguenti fasi:

- **Map:** In questa prima operazione si andranno a leggere tutti i dati, riga per riga, dal file di input `chess_games.csv` e si manterranno solamente i

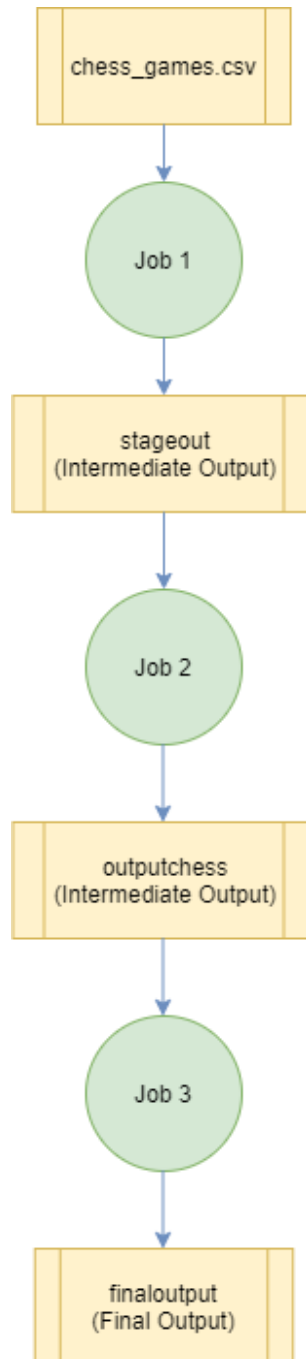


Figure 4: Job Schema

dati utili ai fini del progetto. L'output di questa Map consiste in coppie chiave-valore del tipo Opening-RisultatoPartita (es. Slav Defense, 1-0);

- **Reduce:** L'operazione di Reduce consiste nel prendere le coppie di valori fornite precedentemente dal Mapper e per ogni Apertura calcolare il numero di partite totali, il numero di partite vinte, il numero delle partite perse ed il numero di partite pareggiate in base allo score assegnato (1-0, 0-1, etc..). In output viene fornito un file di appoggio che possa permettere al secondo Job di eseguire il proprio compito;

Il numero di Mapper è di 33.

Il numero di Reducer scelto è di 5.

Dopo svariati tentativi si è notato che con il numero di default di Reducer (20) il tempo di esecuzione è di 1 minuto e 24 secondi circa, di conseguenza si è optato per 5 Reducer in quanto il tempo di esecuzione si è ridotto a 58 secondi.

Si tiene comunque a precisare che questi risultati sono indicativi, in quanto il Cluster poteva essere sovraccarico nel momento dei test. Si è comunque tentata diverse volte l'esecuzione del Job e si è scelto il risultato migliore.

Link YARN: [http://isi-vclust0.csr.unibo.it:19888/jobhistory/job/job\\_1622038346572\\_0816](http://isi-vclust0.csr.unibo.it:19888/jobhistory/job/job_1622038346572_0816)

### 3.1.2 Job 2 - Calcolo delle percentuali di vittoria ed ordinamento

Questo secondo Job è composto dalle seguenti fasi di elaborazione:

- **Map:** In questa operazione di Map si andranno a leggere i valori salvati dal Job precedentemente eseguito e si prenderanno i valori relativi alle colonne delle Vittorie e delle Partite totali andando a calcolare la percentuale di vittorie sul totale delle partite giocate. L'output di questa Map consiste in coppie chiave-valore del tipo Percentuale-StringaPrecedentePresente in modo da effettuare un ordinamento in base alla percentuale grazie al *set-SortComparatorClass*;
- **Reduce:** L'operazione di Reduce consiste nel ritrasformare i dati precedentemente ottenuti nel formato leggibile e standard su cui si sono eseguiti tutti i precedenti calcoli (Andando quindi a reimpostare come Chiave l'Apertura e come valori tutti i risultati ottenuti precedentemente) ed infine si salveranno i risultati ottenuti nella cartella di *outputchess/*;

Il numero di Mapper è di 5.

Il numero di Reducer scelto è di 1.

Si è notato che con il numero di default di Reducer (5) il tempo di esecuzione è di 27 secondi, di conseguenza si è optato per la scelta di un solo Reducer così da ottenere un tempo di esecuzione di 20 secondi.

Si tiene comunque a precisare che questi risultati sono indicativi, in quanto il Cluster poteva essere sovraccarico nel momento dei test. Si è comunque tentata diverse volte l'esecuzione del Job e si è scelto il risultato migliore.

Link YARN: [http://isi-vclust0.csr.unibo.it:19888/jobhistory/job/job\\_1622038346572\\_0817](http://isi-vclust0.csr.unibo.it:19888/jobhistory/job/job_1622038346572_0817)

### 3.1.3 Note

Si precisa che le percentuali del Job 2 possono essere molto variabili o apparire semplicemente troppo alte o basse perchè spesso sono aperture utilizzate una sola volta (con vittoria o sconfitta) e di conseguenza la percentuale di successo o fallimento di tale potrebbe risultare del 100%.

### 3.1.4 Job 3 - Calcolo della percentuale di utilizzo dell'apertura

Questo terzo Job è composto dalle seguenti fasi di elaborazione:

- **Map:** In questa operazione di Map si andranno semplicemente a leggere i valori salvati dal Job precedentemente eseguito e si effettuerà un controllo sulla correttezza del numero di valori in input;
- **Reduce:** L'operazione di Reduce consiste nel calcolare il numero totale di partite giocate e successivamente effettuare il calcolo della percentuale dell'utilizzo di ogni apertura sul totale. Infine verranno ritrasformati i dati precedentemente ottenuti nel formato leggibile e standard su cui si sono eseguiti i precedenti calcoli (Andando quindi a reimpostare come Chiave l'Apertura e come valori tutti i risultati ottenuti precedentemente) e si salveranno i risultati ottenuti nella cartella di *finaloutput/*;

Il numero di Mapper è 1.

Il numero di Reducer è 1.

In quest'ultimo Job non c'erano particolari differenze di esecuzione dato che il numero di Default di Reducer è anche quello più efficiente, con un tempo di esecuzione è di 27 secondi.

Si tiene comunque a precisare che questi risultati sono indicativi, in quanto il Cluster poteva essere sovraccarico nel momento dei test. Si è comunque tentata diverse volte l'esecuzione del Job e si è scelto il risultato migliore.

Link YARN: [http://isi-vclust0.csr.unibo.it:8088/cluster/app/application\\_1622038346572\\_0818](http://isi-vclust0.csr.unibo.it:8088/cluster/app/application_1622038346572_0818)

### 3.1.5 Note

Si precisa che le percentuali del Job 3 possono risultare molto spesso troppo basse o nulle perchè una determinata apertura viene effettuata molto meno rispetto al totale delle partite giocate. Su grandi numeri, questa percentuale risulterà di gran lunga inferiore al 0,001%.

## 3.2 Implementazione Spark

Nella implementazione degli stessi Job precedenti utilizzando Spark, si è utilizzato come output intermedio il percorso `hdfs:/user/gpoggi/sparkstageout/` e come output finale il percorso `hdfs:/user/gpoggi/sparkoutputchess/` per salvare il file csv finale ottenuto.



### 3.2.1 Spark

Il file di input `chess_games.csv` è stato caricato in un RDD denominato *chessRdd*. Le operazioni eseguite su questo dataset sono state:

- Filtrazione dei dati opportuni dal file di input delle colonne di interesse;
- Calcolo del numero totale di partite giocate, partite vinte, partite perse e partite pareggiate e salvataggio su file;
- Tramite il file intermedio, calcolo delle percentuali di vittoria e di utilizzo dell'apertura;
- Ordinamento per percentuale di vittoria;

Per una più corretta comprensione dei calcoli e dei risultati si è pensato di separare le due operazioni di calcolo delle percentuali e salvare i dati intermedi (`chess_games_output.csv`) e finali (`chess_games_output_final.csv`) in directory separate.

Si è inoltre provata l'esecuzione dell'implementazione in Spark più volte, per determinare il miglior numero di *Executor* e *Core per Executor*. Lanciando l'esecuzione con tutte le possibili combinazioni di valori tra i due parametri, si è notato che la migliore configurazione per eseguire il programma è: *spark2-submit -num-executors 2 -executor-cores 2 -class ChessAnalysisSparkJobs BDE-spark-poggi.jar*. Tramite l'utilizzo di 2 Executors e 2 Core per Executor si riesce ad ottenere un tempo di 42 Secondi circa (Media di tutti i tempi ottenuti). Con tutte le altre configurazioni il tempo impiegato risultava maggiore di tale soglia (Maggiore di 1 Minuto).

Si tiene comunque a precisare che questi risultati sono indicativi, in quanto il Cluster poteva essere sovraccarico nel momento dei test. Si è comunque tentata diverse volte l'esecuzione del Job e si è scelto il risultato migliore.

Nella realizzazione del progetto in Spark non è stato effettuato caching in quanto gli RDD vengono utilizzati una sola volta.

Per una corretta comprensione del lavoro svolto, di seguito il link relativo ai Job in Spark:

Link Spark 1: [http://isi-vclust0.csr.unibo.it:8088/proxy/application\\_1622038346572\\_0827/](http://isi-vclust0.csr.unibo.it:8088/proxy/application_1622038346572_0827/)

Link Spark 2: [http://isi-vclust0.csr.unibo.it:18089/history/application\\_1622038346572\\_0828](http://isi-vclust0.csr.unibo.it:18089/history/application_1622038346572_0828)

### 3.2.2 Note

Si precisa che le percentuali dei risultati possono essere molto variabili o apparire semplicemente troppo alte o basse perchè spesso sono aperture utilizzate una sola volta (con vittoria o sconfitta) e di conseguenza la percentuale di successo o fallimento di tale potrebbe risultare del 100%.

▼ DAG Visualization

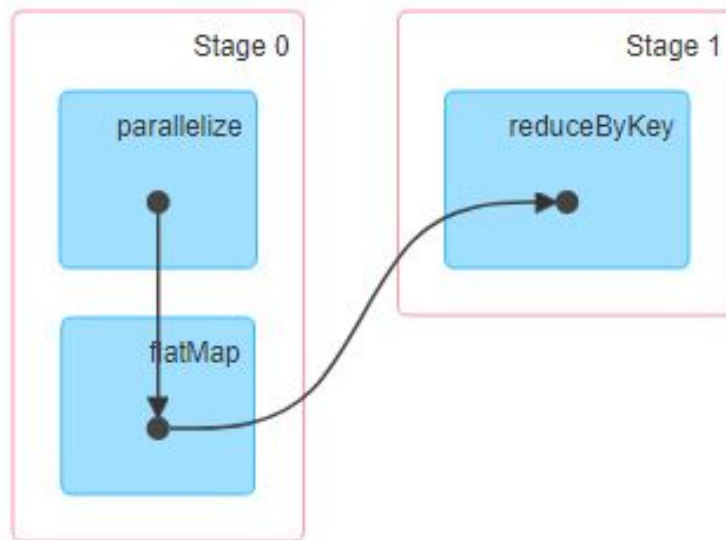


Figure 5: DAG Schema Spark 1

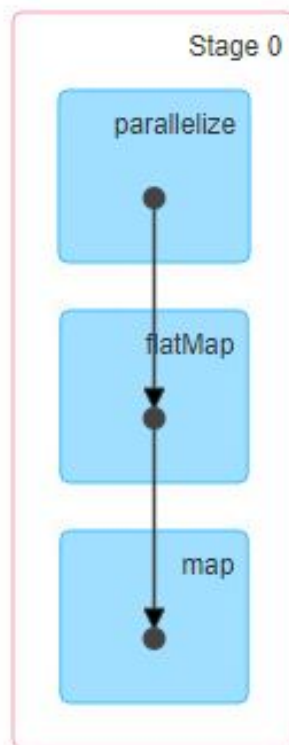


Figure 6: DAG Schema Spark 2