Scalable and Cloud Programming

A.A. 2024-2025

Co-purchase Analysis

Federica Grisendi, ID 0001145353

20 febbraio 2025

1 Lavoro svolto

L'obiettivo di questo progetto è stato implementare un'analisi di co-acquisto dei prodotti basata su Apache Spark e Scala, sfruttando la potenza del calcolo distribuito per migliorare le prestazioni rispetto a un'implementazione sequenziale.

Il progetto realizzato elabora il dataset fornito contenente coppie di codici identificativi interi *(ordine, prodotto)* e determina quante volte due prodotti sono stati acquistati nel medesimo ordine. Per eseguire tale analisi, il programma esegue i seguenti passaggi:

1. Carica il dataset e ne memorizza i dati sotto forma di RDD (*Resilient Distributed Datasets*).
2. Determina dinamicamente il numero di partizioni in base al numero di nodi disponibili nel cluster e il numero di core per nodo, al fine di ottimizzare la parallelizzazione.
3. Raggruppa i prodotti acquistati nello stesso ordine con l'obiettivo di generare le coppie di prodotti co-acquistati.
4. Conta la frequenza di ogni coppia di prodotti e salva i risultati in un unico documento di output.

L'implementazione è stata testata su cluster composti rispettivamente da uno, due, tre e quattro nodi *worker* per valutare la scalabilità e il miglioramento delle prestazioni con l'aumento delle risorse di calcolo.

2 Approccio utilizzato

L'approccio adottato sfrutta RDD e operazioni distribuite per garantire scalabilità e parallelizzazione.

2.1 Determinazione dinamica del numero di partizioni

Il numero di partizioni viene calcolato come il massimo tra

$$nCores \cdot nNodes \cdot 2 \quad \textbf{e} \quad sc.defaultParallelism \cdot 2$$

dove:

- $nNodes$ è il numero di nodi nel cluster.
- $nCores$ è il numero di core per nodo.
- $sc.defaultParallelism$ rappresenta il livello di parallelismo predefinito di Spark.

Questo garantisce una migliore allocazione delle risorse del cluster.

2.2 Elaborazione delle coppie di prodotti

Dopo aver caricato il dataset:

1. I dati vengono partizionati con *HashPartitioner* per distribuire il carico di lavoro in modo bilanciato sui nodi worker del cluster.
2. I prodotti vengono raggruppati per ordine e trasformati in coppie di prodotti co-acquistati.
3. Il conteggio delle coppie avviene utilizzando il metodo *reduceByKey*, un'operazione distribuita ottimizzata per il calcolo su cluster.

3 Analisi di scalabilità e prestazioni

L'implementazione è stata testata su un cluster Spark con 1, 2, 3 e 4 nodi worker, misurando il tempo di esecuzione e il miglioramento delle prestazioni in termini di *speed-up* e *Strong Scaling Efficiency*.
Dove:

- *Speed-up*: misura il miglioramento delle prestazioni, in termini di tempo, dell'esecuzione con n nodi rispetto al tempo di esecuzione su un singolo nodo:

$$S(n) = \frac{T(1)}{T(n)}$$

- *Strong Scaling Efficiency*: valuta l'efficienza della scalabilità aumentando il numero di nodi:

$$SSE(n) = \frac{S(n)}{n} = \frac{T(1)}{n \cdot T(n)}$$

3.1 Risultati ottenuti

3.1.1 Cluster "*single node*"

La versione sequenziale del programma genera i seguenti risultati:

Numero di partizioni	8 (corrispondente a $nCores \cdot nNodes \cdot 2 = 4 * 1 * 2$)
Tempo di esecuzione	$T(1) = 702.81$ secondi

3.1.2 Cluster a due nodi

La versione parallela del programma eseguita su un cluster composto da due nodi genera i seguenti risultati:

Numero di partizioni	16 (corrispondente a $nCores \cdot nNodes \cdot 2 = 4 * 2 * 2$)
Tempo di esecuzione	$T(2) = 407.15$ secondi
Speed-up	$S(2) = \frac{T(1)}{T(2)} = \frac{702.81}{407.15} = 1.72$
Strong Scaling Efficiency	$SSE(2) = \frac{S(2)}{n} = \frac{1.72}{2} = 0.86$

3.1.3 Cluster a tre nodi

La versione parallela del programma eseguita su un cluster composto da tre nodi genera i seguenti risultati:

Numero di partizioni	24 (corrispondente a $nCores \cdot nNodes \cdot 2 = 4 * 3 * 2$)
Tempo di esecuzione	$T(3) = 383.03$ secondi
Speed-up	$S(3) = \frac{T(1)}{T(3)} = \frac{702.81}{383.03} = 1.83$
Strong Scaling Efficiency	$SSE(3) = \frac{S(3)}{n} = \frac{1.83}{3} = 0.61$

3.1.4 Cluster a quattro nodi

La versione parallela del programma eseguita su un cluster composto da quattro nodi genera i seguenti risultati:

Numero di partizioni	32 (corrispondente a $nCores \cdot nNodes \cdot 2 = 4 * 4 * 2$)
Tempo di esecuzione	$T(4) = 247.08$ secondi
Speed-up	$S(4) = \frac{T(1)}{T(4)} = \frac{702.81}{247.08} = 2.84$
Strong Scaling Efficiency	$SSE(4) = \frac{S(4)}{n} = \frac{2.84}{4} = 0.71$

3.2 Conclusioni

La seguente tabella riassume i parametri d'interesse relativi all'esecuzione del programma sui diversi cluster:

Nodi	Tempo di esecuzione	Speed-up	Strong Scaling Efficiency
1	702.81	-	-
2	407.15	1.73	0.86
3	383.03	1.84	0.61
4	247.08	2.86	0.71

Tabella 1: Confronto prestazionale.

L'implementazione mostra un buon livello di scalabilità, con un significativo miglioramento delle prestazioni aumentando il numero di nodi. Tuttavia, come previsto, l'efficienza della scalabilità diminuisce con l'aumentare del numero di nodi a causa della maggiore latenza di comunicazione e gestione delle partizioni.