Big Data and Reasoning

Domande

Google e Hadoop:

- Come ha influenzato Google lo sviluppo di Hadoop?:

```
Google ha influenzato lo sviluppo di Hadoop in quanto ha fornito il modello di programmazione MapReduce e il file system distribuito GFS.
```

- Quali sono alcune caratteristiche chiave dell'architettura di Google?

```
Alcune caratteristiche chiave dell'architettura di Google sono:
```

- Distribuzione dei dati
- Distribuzione dei calcoli
- Tolleranza ai guasti
- Scalabilità

Tipi di Database Alternativi:

- Cosa ha portato alla creazione di database grafici?

```
I database grafici sono stati creati per gestire le relazioni tra le entità in modo più efficiente rispetto ai database relazionali. Un esempio di
```

- Qual è la differenza tra un database relazionale e un triplestore?

```
Un triplestore è un database che memorizza i dati in triple RDF (entità:attributo:valore). Usa SparkQL per interrogare. E' un database non relazion
```

Berkeley Analytics Data Stack:

- In che modo Spark migliora l'elaborazione distribuita rispetto a MapReduce?

```
Spark migliora l'elaborazione distribuita rispetto a MapReduce in quanto è più veloce e più facile da usare. Inoltre, non ha bisogno di salvare i f
```

- Qual è il ruolo di Tachyon in questo stack?

```
Tachyon è un file system distribuito che fornisce un'interfaccia di memoria condivisa per i cluster. E' stato inventato per lavorare con Spark e pu
```

MapReduce - Introduzione:

- Qual è l'obiettivo principale di MapReduce?

```
L'obiettivo principale di MapReduce è quello di fornire un modello di programmazione per elaborare grandi quantità di dati in parallelo su un clust
```

- In che modo MapReduce gestisce i guasti delle macchine?

```
MapReduce affronta i guasti delle macchine riavviando i task falliti e garantendo che l'output sia coerente e senza perdite,  anche in caso di guas
```

Dettagli sull'Implementazione di MapReduce:

- Spiega il ruolo di un combiner in MapReduce.

```
Il combiner è un'ottimizzazione che può essere utilizzata per ridurre il traffico di rete e la quantità di dati scritti su disco. In particolare è
```

- Come si accede ai parametri di configurazione in Hadoop?

```
I parametri di configurazione in Hadoop sono accessibili tramite la classe Configuration.
```

Hadoop con Altri Linguaggi:

- Come funziona Hadoop Streaming con linguaggi diversi da Java?

Hadoop Streaming consente di utilizzare linguaggi diversi da Java in quanto utilizza lo standard input e output per comunicare con i programmi MapR

- Qual è il vantaggio di utilizzare l'API di streaming?

L'API di streaming consente di utilizzare linguaggi diversi da Java per scrivere programmi MapReduce. In particolare, i programmi MapReduce possono

Chaining MapReduce Jobs:

- Come vengono concatenati i lavori MapReduce?

I lavori MapReduce vengono concatenati utilizzando la classe ChainMapper e ChainReducer, che consentono di concatenare più mapper e reducer in un s

- Qual è il ruolo della classe ChainMapper?

La classe ChainMapper consente di concatenare più mapper in un singolo job.

Joining Data in MapReduce:

- Spiega la differenza tra reduce-side join e map-side join.

In un reduce-side join, non ci sono limiti sui dati che vengono mandati in output, e questo rende il carico di lavoro molto elevato, poiché per far

- Come funziona l'utilizzo di Distributed Cache per i join in MapReduce?

Distributed Cache in MapReduce consente di distribuire file ai nodi del cluster. Viene utilizzato per i join, consentendo ai mappers di accedere a

Monoidi e Combiner in MapReduce:

- Cos'è un monoid e perché è importante in MapReduce?

`Un monoid è un insieme con un'operazione binaria che è associativa e ha un elemento neutro. Inoltre, bisogna rispettare l'associatività e la chiusura. Ad esempio, la somma è un monoide:

```
- Elemento neutro: 0
- Associatività: (a+b)+c = a+(b+c)
- Chiusura: a+b è sempre un numero $\in$ R`
```

- In che modo i monoidi semplificano l'implementazione di combiner?

I monoidi semplificano l'implementazione di combiner perché consentono di eseguire operazioni di riduzione locali in modo efficiente durante la fas

- In che situazioni sarebbe utile utilizzare Distributed Cache?

Distributed Cache è utile quando è necessario fornire file o risorse ausiliarie a tutti i nodi del cluster durante l'esecuzione di un lavoro MapRed

Column-Oriented Databases:

- Qual è la differenza tra un database orientato alle colonne e uno orientato alle righe?

Un database orientato alle colonne organizza i dati per colonne anziché per righe, offrendo vantaggi nelle query analitiche OLAP. In un database or

- Perché l'architettura C-STORE è considerata ibrida?

In breve, l'architettura C-STORE cerca di combinare il meglio dei due mondi, offrendo un'architettura ibrida che può essere adattata per affrontare

- Come interagiscono tra loro Spark, Mesos e Tachyon?

Spark è un framework di elaborazione distribuita che può essere eseguito su Mesos, un sistema operativo per cluster che consente di eseguire applic

Graph Databases:

- Quando è preferibile utilizzare un database grafico rispetto a un database relazionale?

I database grafici sono preferibili quando le relazioni complesse tra dati devono essere rappresentate come grafi, come nelle reti sociali. Ad esem

Secondary Sort in MapReduce:

- Cos'è il secondary sort e quando è utile in MapReduce?

```
Il secondary sort è un'ottimizzazione che consente di ordinare i valori in output per chiave. In particolare, i valori vengono ordinati in ordine c
```

- Spiega il ruolo del comparatore di raggruppamento nel secondary sort.

```
Il comparatore di raggruppamento nel secondary sort consente di raggruppare i valori in output per chiave.
```

MapReduce Input e Output Formats:

- Qual è il ruolo degli InputFormat e OutputFormat in MapReduce?

```
Gli InputFormat e OutputFormat in MapReduce consentono di definire il formato di input e output dei dati. In particolare, gli InputFormat consenton
```

- Come implementare un formato di input personalizzato in Hadoop?

```
Per implementare un formato di input personalizzato in Hadoop, è necessario implementare l'interfaccia InputFormat e l'interfaccia RecordReader.
```

MapReduce Custom Partitioner:

- Perché potresti aver bisogno di un partizionatore personalizzato in MapReduce?

```
Potresti aver bisogno di un partizionatore personalizzato in MapReduce per controllare la distribuzione dei dati tra i reducer.
```

Apache HBase:

- Qual è il ruolo di Apache HBase nell'ecosistema Hadoop?

```
HBase è un database distribuito non relazionale (NoSQL) che fornisce accesso in tempo reale e la memorizzazione di grandi quantità di dati su clust
```

- In che modo HBase differisce da HDFS?

```
HBase è un database NoSQL distribuito che consente l'accesso in tempo reale ai dati, mentre HDFS è un sistema di file distribuito progettato per l'
```

OLAP e OLTP in Hadoop:

- Spiega le differenze tra OLAP e OLTP.

```
Una sistema OLAP (Online Analytical Processing) è un sistema di elaborazione che consente di analizzare grandi quantità di dati in modo efficiente.
```

- Perché le colonne sono organizzate in modo diverso in un sistema OLAP rispetto a un sistema OLTP?

```
Nel contesto OLAP, i dati vengono organizzati per consentire un rapido accesso e analisi dei dati, solitamente organizzati per colonne per ottimizz
```

MapReduce Join Operations:

- Descrivi le diverse tecniche per implementare operazioni di join in MapReduce.

```
Reduce-side join: Copia l'output di tutti i mappers, effettua uno shuffle e uno sorting, quindi esegue il join durante la fase di riduzione. Map-si
```

- Quando preferiresti una join di tipo reduce-side rispetto a una join di tipo map-side?

```
Conviene quando hai tipi di dati non strutturati ed e' piu' semplice da implementare. Ma in generale, mai.
```

Fault Tolerance in MapReduce:

- Come affronta MapReduce i problemi di failover e fault tolerance?

```
Riassegna il task ad un altro nodo, quando un worker fallisce. Se fallisce un worker mapper allora va rieseguto tutto, se fallisce il reducer, non
```

- Descrivi cosa succede in caso di fallimento di un worker o del master in MapReduce.

```
Se fallisce il master, tutto il processo viene annullato perche' non puo' coordinare il flow.
```

Berkeley Data Analytics Stack:

- Quali componenti costituiscono la Berkeley Data Analytics Stack?

`Sono:

- Spark: Framework per eseguire operazioni di mapreduce in memoria. Sfrutta I RDD Cioe dataset condivisi sulla rete
- Mesos: Cioe' l'alternativa a YARN ma in memoria
- Tachyon: Un file system distribuito in memoria, cioe' l'alternativa al HDFSs
  `
- In cosa si differenzia da Hadoop e qual è il suo vantaggio?

```
Si differenzia dal fatto che il tutto avviene in memoria in modo piu' efficiente, fornendo anche API che facilitano la scrittura di programmi che u
```

MapReduce Map-Side Optimization:

- Spiega l'ottimizzazione lato mappa in MapReduce.

```
Le ottimizzazioni map-side sono utili perche' permettono di eseguire operazioni di riduzione volumetrica sui dati per diminuire la quantita' di dat
```

Monoids in MapReduce:

- Cosa sono i monoidi e perché sono rilevanti in MapReduce?

`I monoidi sono un insieme con un'operazione binaria che è associativa e ha un elemento neutro. Inoltre, bisogna rispettare l'associatività e la chiusura. Ad esempio, la somma è un monoide:

```
- Elemento neutro: 0
- Associatività: (a+b)+c = a+(b+c)
- Chiusura: a+b è sempre un numero $\in$ R`
```

- Come puoi sfruttare i monoidi per ottimizzare le prestazioni in MapReduce?

```
Sfruttando i monoidi, è possibile implementare combiner che eseguono operazioni di riduzione locali, riducendo così il volume di dati da trasmetter
```

MapReduce Bloom Filters:

- Come vengono utilizzati i filtri di Bloom in MapReduce?
-

```
I filtri di Bloom vengono utilizzati per approssimare operazioni di semi-join, filtrando in modo efficiente i dati non necessari prima della fase d
```

- In quali situazioni i filtri di Bloom possono migliorare le prestazioni?

```
I filtri di Bloom possono migliorare le prestazioni in situazioni in cui una parte significativa dei dati può essere esclusa in modo efficiente pri
```

- Come funziona Amazon Dynamo
  Amazon's Dynamo è un'alternativa ai database relazionali che sfrutta la coerenza eventual per garantire disponibilità e tolleranza alle partizioni.
  Dynamo supporta l'accesso basato sulla chiave primaria, e i valori recuperati tramite una ricerca per chiave sono oggetti binari non strutturati (senza una struttura specifica). Le caratteristiche principali di Dynamo sono:
  Consistent hashing: Viene utilizzato il valore hash della chiave primaria per determinare i nodi nel cluster responsabili di quella chiave. Ciò consente di aggiungere o rimuovere nodi dal cluster con un minimo impatto sulla riconfigurazione complessiva.
  Consistenza regolabile: L'applicazione può specificare compromessi tra coerenza, prestazioni di lettura e prestazioni di scrittura, permettendo di adattare il sistema alle esigenze specifiche.
  Versionamento dei dati: Poiché le operazioni di scrittura non vengono mai bloccate, è possibile che ci siano versioni multiple di un oggetto nel sistema. Queste versioni multiple devono essere gestite dall'applicazione o dagli utenti.
  In sostanza, Dynamo offre un modo flessibile di gestire dati distribuiti, consentendo agli sviluppatori di bilanciare la coerenza dei dati con le prestazioni di lettura e scrittura in base alle esigenze dell'applicazione.
- Come funziona BigTable?
  BigTable è un sistema di database non relazionale basato su colonne che utilizza il Google File System per lo storage. Si tratta di una mappa ordinata multidimensionale e distribuita in modo sparso. La chiave della mappa è composta da una chiave di riga, una chiave di colonna e un timestamp per gestire diverse versioni delle stesse informazioni. Ogni valore nella mappa è un array di byte non interpretato, consentendo la memorizzazione di dati

senza uno schema predeterminato. Le righe sono identificate da chiavi arbitrarie e le operazioni di lettura o scrittura su una singola chiave di riga sono atomiche. Le chiavi di colonna sono organizzate in famiglie di colonne, ciascuna delle quali ha un significato comune. I componenti principali includono il cliente, il server principale (che gestisce la suddivisione dinamica delle tavolette) e il server delle tavolette (che gestisce le richieste di lettura/scrittura). Un servizio di lock distribuito, chiamato Chubby, assicura la coerenza e l'accesso diretto ai server delle tavolette.

- Implementa un codice che allena un modello di regressione lineare con SparkML.

```java
import org.apache.spark.ml.regression.LinearRegression;
import org.apache.spark.ml.regression.LinearRegressionModel;
import org.apache.spark.ml.regression.LinearRegressionTrainingSummary;

//Imports

// Load training data.
Dataset<Row> training = spark.read().format("libsvm")
  .load("data/mllib/sample_linear_regression_data.txt");

//Facciamo regressoine, prendiamo ilmodello

LinearRegression lr = new LinearRegression()


//Bisogna aggiungere la colonna della label, quindi usiamo uno StringIndexer

StringIndexer indexer = new StringIndexer()
  .setInputCol("label")
  .setOutputCol("indexedLabel")
  .fit(training);

// Bisogna aggiungere la colonna delle features, quindi usiamo uno VectorIndexer

VectorIndexer featureIndexer = new VectorIndexer()
  .setInputCol("features")
  .setOutputCol("indexedFeatures")
  .setMaxCategories(4) // features con > 4 valori distinti vengono considerate continue
  .fit(training);

// Dividiamo i dati in training e test set (70% training e 30% test)

Dataset<Row>[] splits = training.randomSplit(new double[]{0.7, 0.3});

Dataset<Row> trainingData = splits[0];
Dataset<Row> testData = splits[1];

// Trainiamo il modello
LinearRegressionModel lrModel = lr.fit(trainingData);

// Facciamo le predizioni
Dataset<Row> predictions = lrModel.transform(testData);

// Selezioniamo le colonne da visualizzare
predictions.select("prediction", "label", "features").show(5);

// Selezioniamo le metriche di valutazione
RegressionEvaluator evaluator = new RegressionEvaluator()
  .setLabelCol("indexedLabel")
  .setPredictionCol("prediction")
  .setMetricName("rmse");

// Calcoliamo l'errore quadratico medio
double rmse = evaluator.evaluate(predictions);
System.out.println("Root Mean Squared Error (RMSE) on test data = " + rmse);

LinearRegressionTrainingSummary trainingSummary = lrModel.summary();
System.out.println("numIterations: " + trainingSummary.totalIterations());

// Print the coefficients and intercept for linear regression
LinearRegressionModel lrModel = (LinearRegressionModel)(model.stages()[1]);

System.out.println("Coefficients: "
  + lrModel.coefficients() + " Intercept: " + lrModel.intercept());
```

```
// Summarize the model over the training set and print out some metrics
LinearRegressionTrainingSummary trainingSummary = lrModel.summary();
```

- Implementa un codice che usa Spark per calcolare la media di un RDD.

```
// Classe ColumnType
public class ColumnType {
    public String name;
    public String type;

    public ColumnType(String name, String type) {
        this.name = name;
        this.type = type;
    }
}

// Spark session
SparkSession session = SparkSession.builder().master(Main.master).appName("Understanding").getOrCreate();
        session.sparkContext().setLogLevel("error");

// Carica un dataset
Dataset<Row> df = session.read().format("csv").option("header", "true").option("inferSchema", "true").load("data/iris.csv");

// Facciamo un array di colonne a mano
ArrayList<ColumnType> types = new ArrayList<>();
types.add(new ColumnType("sepal_length", "double"));
types.add(new ColumnType("sepal_width", "double"));
types.add(new ColumnType("petal_length", "double"));
types.add(new ColumnType("petal_width", "double"));

// Puliamo i daati
for (ColumnType type : types) {
    df = df.withColumn(type.name, df.col(type.name).cast(type.type));
}

// Calcoliamo la media
df.agg(avg("sepal_length")).show();

// Creazioend i un job che calcola la media che chiama il metodo call

Dataset<Row> newDf = df.map(new MapFunction<Row, Row>() {
    @Override
    public Row call(Row row) throws Exception {
        return RowFactory.create(row.getDouble(0), row.getDouble(1), row.getDouble(2), row.getDouble(3));
    }
  }, Encoders.bean(Row.class)).toDF();

// Calcoliamo la media
newDf.agg(avg("sepal_length")).show();

// Salviaom il dataset in csv
newDf.write().format("csv").save("data/iris_cleaned.csv");
```

- Implementa un codice che sfrutta MapReduce per eseguire un join tra due dataset.

```
// Immaginiamo di essere in un nuovo file

// connectionFactory
public static Configuration getConfiguration() {
        Configuration conf = new Configuration();
        conf.set("hbase.zookeeper.quorum", "master,slave1,slave2");
        return conf;

    }

// Import job for artists
public static void importJob(Configuration conf) {
        try {
            Job job = Job.getInstance(conf, "Import from HDFS to HBase");
            job.setJarByClass(Main.class);

            job.setMapperClass(ImportMapper.class);

            job.setMapOutputKeyClass(ImmutableBytesWritable.class);
            job.setMapOutputValueClass(Put.class);

            TableMapReduceUtil.initTableReducerJob("songs", ImportReducer.class, job);

            FileInputFormat.addInputPath(job, new Path("/user/hadoop/intermediate2/songs.csv"));

            try {
                System.out.println("Initializing the final job");
                job.waitForCompletion(true);
                System.out.println("Job finsihed");
            } catch (ClassNotFoundException e) {
                e.printStackTrace();
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        } catch (IOException e) {
            System.err.println("Something whent wrong during the Import Job");
        }
    }

// Import job for songs
public static void importJob2(Configuration conf) {
        try {
            Job job = Job.getInstance(conf, "Import from HDFS to HBase");
            job.setJarByClass(Main.class);

            job.setMapperClass(ImportMapper2.class);

            job.setMapOutputKeyClass(ImmutableBytesWritable.class);
            job.setMapOutputValueClass(Put.class);

            TableMapReduceUtil.initTableReducerJob("artists", ImportReducer2.class, job);

            FileInputFormat.addInputPath(job, new Path("/user/hadoop/intermediate2/artists.csv"));

            try {
                System.out.println("Initializing the final job");
                job.waitForCompletion(true);
                System.out.println("Job finsihed");
            } catch (ClassNotFoundException e) {
                e.printStackTrace();
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        } catch (IOException e) {
            System.err.println("Something whent wrong during the Import Job");
```

```
        }
    }

// Mapper for artists
public static class ImportMapper2 extends Mapper<LongWritable, Text, ImmutableBytesWritable, Put> {
        private static final byte[] CF_SONG = "song".getBytes();
        private static final byte[] Q_SONG = "song".getBytes();

        public void map(LongWritable key, Text value, Context context) throws IOException, InterruptedException {
            String[] values = value.toString().split(",");
            String rowKey = values[0];
s
            Put put = new Put(Bytes.toBytes(rowKey));
            put.addColumn(CF_SONG, Q_SONG, Bytes.toBytes(values[1]));

            context.write(new ImmutableBytesWritable(Bytes.toBytes(rowKey)), put);
        }
    }

// Mapper for songs
public static class ImportMapper extends Mapper<LongWritable, Text, ImmutableBytesWritable, Put> {
        private static final byte[] CF_SONG = "song".getBytes();
        private static final byte[] Q_SONG = "song".getBytes();

        public void map(LongWritable key, Text value, Context context) throws IOException, InterruptedException {
            String[] values = value.toString().split(",");
            String rowKey = values[0];

            Put put = new Put(Bytes.toBytes(rowKey));
            put.addColumn(CF_SONG, Q_SONG, Bytes.toBytes(values[1]));

            context.write(new ImmutableBytesWritable(Bytes.toBytes(rowKey)), put);
        }
    }

// Reducer for artists
public static class ImportReducer2 extends TableReducer<ImmutableBytesWritable, Put, ImmutableBytesWritable> {
        public void reduce(ImmutableBytesWritable key, Iterable<Put> values, Context context)
                throws IOException, InterruptedException {
            for (Put put : values) {
                context.write(key, put);
            }
        }
    }

// Reducer for songs
public static class ImportReducer extends TableReducer<ImmutableBytesWritable, Put, ImmutableBytesWritable> {
        public void reduce(ImmutableBytesWritable key, Iterable<Put> values, Context context)
                throws IOException, InterruptedException {
            for (Put put : values) {
                context.write(key, put);
            }
        }
    }

// Main
public static void main(String[] args) throws IOException {
        Configuration conf = getConfiguration();
        Connection connection = ConnectionFactory.createConnection(conf);

        Admin admin = connection.getAdmin();

        TableDescriptorBuilder tableDescriptorBuilder = TableDescriptorBuilder.newBuilder(TableName.valueOf("songs"));

        ColumnFamilyDescriptorBuilder columnFamilyDescriptorBuilder = ColumnFamilyDescriptorBuilder
```

```
                .newBuilder(Bytes.toBytes("song"));

        ColumnFamilyDescriptor columnFamilyDescriptor = columnFamilyDescriptorBuilder.build();

        tableDescriptorBuilder.setColumnFamily(columnFamilyDescriptor);

        TableDescriptor tableDescriptor = tableDescriptorBuilder.build();

        admin.createTable(tableDescriptor);

        TableDescriptorBuilder tableDescriptorBuilder2 = TableDescriptorBuilder.newBuilder(TableName.valueOf("artists"));

        ColumnFamilyDescriptorBuilder columnFamilyDescriptorBuilder2 = ColumnFamilyDescriptorBuilder
                .newBuilder(Bytes.toBytes("song"));

        ColumnFamilyDescriptor columnFamilyDescriptor2 = columnFamilyDescriptorBuilder2.build();

        tableDescriptorBuilder2.setColumnFamily(columnFamilyDescriptor2);

        TableDescriptor tableDescriptor2 = tableDescriptorBuilder2.build();

        admin.createTable(tableDescriptor2);

        // Importiamo i dati
        importJob(conf);
        importJob2(conf);

        // Creiamo un nuovo job per fare il join
        Job job = Job.getInstance(conf, "Join");
        job.setJarByClass(Main.class);
        job.setMapperClass(JoinMapper.class);
        job.setReducerClass(JoinReducer.class);
        job.setOutputKeyClass(Text.class);
        job.setOutputValueClass(Text.class);

        // Leggiamo da hbase
        TableMapReduceUtil.initTableMapperJob("songs", new Scan(), JoinMapper.class, Text.class, Text.class, job);
        TableMapReduceUtil.initTableReducerJob("join", JoinReducer.class, job);

        try {
            System.out.println("Initializing the final job");
            job.waitForCompletion(true);
            System.out.println("Job finsihed");
        } catch (ClassNotFoundException e) {
            e.printStackTrace();
        } catch (InterruptedException e) {
            e.printStackTrace();
        }

        // Chiudiamo la connessione
        connection.close();
    }

// Mapper for join
public static class JoinMapper extends TableMapper<Text, Text> {
        public void map(ImmutableBytesWritable row, Result value, Context context)
                throws IOException, InterruptedException {
            String song = Bytes.toString(value.getValue(Bytes.toBytes("song"), Bytes.toBytes("song")));
            String artist = Bytes.toString(value.getValue(Bytes.toBytes("song"), Bytes.toBytes("artist")));

            context.write(new Text(song), new Text(artist));
        }
    }

// Reducer for join
```

```
public static class JoinReducer extends TableReducer<Text, Text, ImmutableBytesWritable> {
        public void reduce(Text key, Iterable<Text> values, Context context)
                throws IOException, InterruptedException {
            for (Text value : values) {
                Put put = new Put(Bytes.toBytes(key.toString()));
                put.addColumn(Bytes.toBytes("song"), Bytes.toBytes("artist"), Bytes.toBytes(value.toString()));
                context.write(new ImmutableBytesWritable(Bytes.toBytes(key.toString())), put);
            }
        }
    }
```