Apache Spark পূর্ণাঙ্গ বাংলা টিউটোরিয়াল (Conceptual + Practical + Example Explanation)

1. পরিচিতি ও Apache Spark কী?

সহজ ভাষায় (Analogy)

তুমি ধরা যাক বিশাল একটা লাইব্রেরির মালিক, যেখানে হাজার হাজার বই আছে

তুমি তোমার বন্ধুদের নিয়ে একটা টিম গঠন করো, যারা বইগুলো ভাগাভাগি করে একসাথে পড়বে

যদি কারো বই হারিয়ে যায়, অন্য কেউ আবার সেই বই দিয়ে সাহায্য করবে

তাই তোমাদের দল দ্রুত কাজ করতে পারে, আর বই পড়ার কাজ অনেক সহজ হয়

Formal Description

Apache Spark হলো একটি ওপেন-সোর্স, দ্রুত, ইন-মেমোরি ভিত্তিক ডিস্ট্রিবিউটেড ডেটা প্রসেসিং ইঞ্জিন

এটি বড় ডেটাসেটকে অনেকগুলো worker node-এ ভাগ করে, parallel ও fault-tolerant ভাবে প্রসেস করতে সক্ষম

Spark বিভিন্ন ধরনের কাজ করতে পারে — SQL query, machine learning, stream processing, graph processing ইত্যাদি

Spark এর speed অনেকটাই MapReduce এর তুলনায় দ্রুত, কারণ এটি in-memory computation করে

কোড উদাহরণ

python

CopyEdit

```
    from pyspark.sql import SparkSession

spark = SparkSession.builder \
   .appName("MySparkApp") \
```

.getOrCreate()

Explanation:

SparkSession হলো Spark এর নতুন entry point যেখান থেকে তুমি Spark এর সব ফিচার ব্যবহার করতে পারবে

.builder.appName() দিয়ে application এর নাম দাও, .getOrCreate() দিয়ে Spark context তৈরি করো বা পুরানোটি নিয়ে আসো

2. RDD (Resilient Distributed Dataset)

সহজ ভাষায় (Analogy)

তোমার কাছে অনেক বইয়ের প্যাকেট আছে যেগুলো তোমার বন্ধুদের মধ্যে ভাগ করে দেওয়া হয়েছে

যদি কেউ তার প্যাকেট হারিয়ে ফেলে, অন্য কেউ আবার সেই বই দিয়ে সাহায্য করতে পারে

এই বইয়ের প্যাকেটগুলোই Spark এর RDD

Formal Description

RDD হলো Spark এর lowest-level distributed data abstraction যা নিম্নলিখিত গুণাবলী রাখে:

- Immutable: একবার তৈরি হলে আর পরিবর্তন করা যায় না পরিবর্তনের মানে নতুন RDD তৈরি
- Distributed: ডেটা ক্লাস্টারের বিভিন্ন worker node-এ ভাগ করা থাকে
- Fault Tolerant: lineage graph থাকার কারণে কোনো ডেটা হারালে পুনরুদ্ধার সম্ভব
- Lazy Evaluation: transformations তখনই execute হয় না, যতক্ষণ না কোনো action কল করা হয়
- Partitioned: ডেটা partition এ ভাগ করে parallel execution হয়

কোড + Explanation

python

CopyEdit

```
# ধরো SparkContext sc আগে থেকে তৈরি আছে


# ১  একটা লিস্ট থেকে RDD তৈরি করলাম
```

```
    rdd = sc.parallelize([10, 20, 30, 40])

    # ২  map transformation: প্রতিটি মানকে ২ দিয়ে গুণ করলাম
    rdd2 = rdd.map(lambda x: x * 2)

    # ৩  filter transformation: ৫০ এর চেয়ে ছোট মান রাখলাম
    rdd3 = rdd2.filter(lambda x: x < 50)

    # ৪  action: RDD থেকে সব মান collect করে আনা
result = rdd3.collect()

print(result)   # Output: [20, 40]
```

Code Explanation:

- sc.parallelize() দিয়ে তোমরা ডেটা Spark এর RDD তে রূপান্তর করো এটি distributed হয়
- map() হলো transformation, যা lazy থাকে, অর্থাৎ এখনো execute হয় না
- filter() আরেকটা transformation যা শর্ত অনুযায়ী ডেটা ফিল্টার করে
- collect() হলো action, যা lazy evaluation ট্রিগার করে এবং সব worker node থেকে ডেটা driver এ নিয়ে আসে
- Output হলো [20, 40] কারণ প্রথমে সব সংখ্যা দ্বিগুণ হলো [20, 40, 60, 80], তারপর filter এ ৫০ এর নিচে যেগুলো আছে সেগুলো বাছাই হলো

বাস্তব উদাহরণ

ধরা যাক, তোমার কাছে রেস্টুরেন্টের গ্রাহকদের রিভিউ স্কোরের তালিকা আছে

তুমি এমন গ্রাহকদের খুঁজছো যাদের রিভিউ স্কোর ৪ এর নিচে, যাতে তাদের সাথে যোগাযোগ করা যায়

python

CopyEdit

```
    reviews = sc.parallelize([5, 4, 3, 5, 2, 1, 4])
    bad_reviews = reviews.filter(lambda x: x < 4).collect()
print(bad_reviews)  # Output: [3, 2, 1]
```

Explanation:

এই কোড দিয়ে তুমি দ্রুত বড় ডেটার মধ্য থেকে প্রয়োজনীয় subset বের করতে পারো, Spark এর distributed শক্তি কাজে লাগিয়ে

3. Spark Architecture

সহজ ভাষায় (Analogy)

তুমি বড় একটা প্রজেক্ট পরিচালনা করছো যেখানে—

- তুমি (Driver) পুরো প্রজেক্টের ম্যানেজার,
- Supervisor (Cluster Manager) কাজের জন্য লোক ও resource দিচ্ছে,
- কর্মীরা (Executors) তোমার নির্দেশ অনুযায়ী কাজ করছে

Formal Description

Spark architecture প্রধান ৪টি অংশ নিয়ে গঠিত—

Component	Description
Driver Program	User এর কোড রান করে, Application Lifecycle manage করে
Cluster Manager	ক্লাস্টারের resource allocate করে
Executors	Worker nodes-এ Task execute করে
DAG Scheduler	Task গুলো logical stage এ ভাগ করে scheduling করে

Spark এর execution flow:

Application Job(s) Stage(s) Task(s)

Architecture Diagram ব্যাখ্যা:

- Driver হলো তোমার main program যা SparkContext বা SparkSession তৈরি করে এবং tasks distribute করে
- Cluster Manager (যেমন YARN, Mesos, Standalone) Executors কে resource দেয়
- Executors হলো worker nodes, যারা বাস্তবে ডেটা প্রসেসিং করে
- Driver DAG scheduler কে DAG হিসেবে তোমার কাজের স্টেপগুলো দেয়, যা ছোট ছোট Stage ও Task এ ভাগ হয়

4. DAG (Directed Acyclic Graph)

সহজ ভাষায় (Analogy)

তুমি যখন কাজ করো, তুমি ধাপে ধাপে পরিকল্পনা করো, যেখানে প্রথম কাজ শেষ না হলে পরবর্তী কাজ শুরু হয় না

এই কাজের ধারাবাহিকতা ও dependency নিয়ে তৈরি হয় একটা গ্রাফ যার কোনো cycle থাকে না

Formal Description

Spark lazy evaluation এ সকল transformation গুলো DAG আকারে জমা হয়

DAG হলো একটি directed acyclic graph যা Spark কে ডেটা প্রসেসিং এর স্টেপগুলো জানায়

Spark DAG scheduler এই গ্রাফকে ছোট ছোট stage এ ভাগ করে এবং প্রতিটি stage এর কাজ executor এ task হিসেবে পাঠায়

কোড + Debug Example

python

CopyEdit

```
rdd = sc.textFile("file.txt")
rdd2 = rdd.map(lambda x: x.upper())
rdd3 = rdd2.filter(lambda x: "SPARK" in x)
print(rdd3.toDebugString())
```

Explanation:

.toDebugString() দিয়ে তুমি দেখতে পারবে Spark কীভাবে তোমার কোডের স্টেপগুলো DAG হিসেবে সাজিয়েছে

প্রথমে textFile থেকে ডেটা নেবে, map দিয়ে uppercase করবে, filter দিয়ে “SPARK” থাকা লাইন বেছে নিবে

5. Spark Application vs Job vs Stage vs Task

সহজ ভাষায়

- Application: তোমার Spark প্রোগ্রাম
- Job: যখন তুমি কোনো Action কল করো তখন Spark একটা Job শুরু করে

- Stage: Job কে ছোট ছোট অংশে ভাগ করা হয়, যেখানে shuffle হয়
- Task: প্রতিটি partition এর জন্য একক কাজ

Formal Description

Unit	Description
Application	Spark Session বা Context এ রান করা পুরো প্রোগ্রাম
Job	Action কল করার ফলে Spark শুরু করে
Stage	Shuffle boundary অনুযায়ী Job কে ভাগ করা হয়
Task	প্রতিটি partition এর জন্য আলাদা কাজ execute হয়

6. SparkContext vs SparkSession

সহজ ভাষায়

আগে শুধু SparkContext ছিলো, যা মূলত RDD-centric, আর নতুন SparkSession এসেছে যেটা সব ধরনের Spark API (SQL, Hive, ML) এর জন্য unified interface

Formal Description

Feature	SparkContext	SparkSession
Introduced in	Spark 1.x	Spark 2.x
API Focus	RDD-based	DataFrame, Dataset, SQL-based
Usage	Low-level API	Unified high-level API

Usage Example

python

CopyEdit

```
from pyspark.sql import SparkSession

spark = SparkSession.builder.appName("MyApp").getOrCreate()
```

```
sc = spark.sparkContext

rdd = sc.parallelize([1, 2, 3])
df = spark.createDataFrame([(1, "A"), (2, "B")], ["id", "name"])
```

7. Spark SQL

সহজ ভাষায়

SQL দিয়ে structured ডেটার ওপর সহজে query চালানোর পদ্ধতি

Formal Description

Spark SQL হলো Spark এর একটি module যা relational queries ও DataFrame API ব্যবহার করে structured ডেটার বিশ্লেষণ করে

Spark SQL Catalyst optimizer ব্যবহার করে query দ্রুত execute করে

কোড + উদাহরণ

python
CopyEdit

```
df = spark.read.csv("people.csv", header=True, inferSchema=True)
df.createOrReplaceTempView("people")

result = spark.sql("SELECT name FROM people WHERE age > 25")
result.show()
```

Explanation:

- CSV থেকে DataFrame তৈরি করা হলো
- DataFrame কে temporary SQL view এ রূপান্তরিত করা হলো
- SQL query চালিয়ে filtered নামগুলো আনা হলো

8. Hive এবং Spark সম্পর্ক

সহজ ভাষায়

Hive হলো একটা data warehouse যা HDFS এ structured ডেটা রাখে

Spark Hive Support দিয়ে Hive এর টেবিল থেকে ডেটা পড়ে SQL query চালাতে পারে

Formal Description

Spark Hive Integration দিয়ে Spark Hive Metastore থেকে টেবিল স্কিমা নেয় এবং HiveQL query execute করে

Hive এর তুলনায় Spark in-memory কাজ করায় অনেক দ্রুত

উদাহরণ

python

CopyEdit

```
spark = SparkSession.builder \
    .appName("HiveApp") \
    .enableHiveSupport() \
    .getOrCreate()

     spark.sql("SELECT * FROM hive_table").show()
```

9. Use Case Matrix

প্রয়োজনে	ব্যবহার করো
Low-level ETL বা custom logic	RDD
Structured Data ও SQL Query	DataFrame / Spark SQL
Hive Warehouse Integration	Spark with Hive Support
Performance ও Scalability	Spark SQL / SparkSession

10. Spark Data Sources & Formats

সহজ ভাষায়

Spark বিভিন্ন ফাইল ফরম্যাট পড়তে পারে যেমন CSV, JSON, Parquet ইত্যাদি

Formal Description

Spark এ DataFrame reader API দিয়ে বিভিন্ন ফরম্যাটের ডেটা ইনপুট

নেয়:

- CSV: comma-separated
- JSON: key-value structure
- Parquet/ORC: columnar storage, দ্রুত query এর জন্য উপযোগী

Example

python

CopyEdit

```
df_csv = spark.read.csv("data.csv", header=True, inferSchema=True)
df_json = spark.read.json("data.json")
df_parquet = spark.read.parquet("data.parquet")
```

11. Spark Partitioning & Shuffling

সহজ ভাষায় (Analogy)

তুমি একটা বড় দলের মধ্যে কাজ ভাগ করছো, কিন্তু কাজের মাঝে মাঝে এমন অংশ থাকে যেখানে সদস্যরা একে অপরের থেকে কিছু তথ্য আদান-প্রদান করবে

ধরো তোমরা ১০ জন, এবং দুই গ্রুপে ভাগ হয়েছো — গ্রুপ A ও গ্রুপ B

তবে কাজের কিছু ধাপে তোমাকে গ্রুপ A থেকে গ্রুপ B-তে ডেটা শেয়ার করতে হবে, তখন তোমাদেরকে একটি বড় মিটিং করতে হবে — এটিই Spark এ shuffle

Formal Description

Partitioning হলো ডেটাকে ছোট ছোট ভাগে ভাগ করার প্রক্রিয়া, যা parallelism বাড়ায় এবং computational efficiency বাড়ায়

Spark এর RDD বা DataFrame partitioned হয় যাতে একসাথে অনেক কাজ parallel করা যায়

Shuffle হলো partition গুলোর মধ্যে ডেটা movement, যা network, disk I/O ও CPU খরচ বাড়ায়, কারণ ডেটা এক জায়গা থেকে অন্য জায়গায় যায় এটি expensive operation

Practical Impact

- ভালো partitioning মানে workload balanced থাকে, যা স্পার্ক কর্মক্ষমতা বাড়ায়
- বেশি shuffle হলে performance degrade হয়, তাই shuffle কমানোর চেষ্টা করো

- ReduceByKey, Join ইত্যাদি operation এ shuffle হয়

কোড উদাহরণ

python

CopyEdit

```
    rdd = sc.parallelize(range(10), 2)  # 2 partition তৈরি করলাম


    # এখানে map এ key-value tuple তৈরি করলাম
    rdd2 = rdd.map(lambda x: (x % 2, x))


    # partitionBy দিয়ে shuffle হয়েছে (repartitioned by key)
rdd3 = rdd2.partitionBy(2)

print(rdd3.glom().collect())  # partition-wise ডেটা দেখাবে
```

12. Spark Caching & Persistence

সহজ ভাষায় (Analogy)

তুমি যখন কোন বইয়ের কোনো পৃষ্ঠা বারবার পড়ো, তখন তুমি সেই পৃষ্ঠা বই থেকে বার বার খুঁজে না নিয়ে নিজের সামনে রেখে পড়ো, যাতে দ্রুত কাজ হয়

Spark এ ডেটা বা intermediate results বারবার ব্যবহার করার জন্য cache বা persist করা হয়

Formal Description

- cache() হলো ডেটাকে default memory তে রাখা, যাতে পরে তা দ্রুত পাওয়া যায়
- persist() দিয়ে তোমরা ডেটা memory, disk, বা combination এ রাখতে পারো
- এটি expensive computation avoid করে, response time কমায়

Practical Use

- Iterative algorithms (যেমন machine learning) এ cache করা দরকার
- Spark UI তে cache storage দেখে বুঝবে কোন RDD বা DataFrame cached আছে

কোড উদাহরণ

python

CopyEdit

```
    rdd = sc.parallelize([1, 2, 3, 4, 5])
```

```
    # cache করলে প্রথম action এ cache হবে
rdd_cached = rdd.cache()

print(rdd_cached.count())    # প্রথম action - cache হবে
print(rdd_cached.collect())  # cache থেকে দ্রুত ডেটা আসবে
```

13. Spark UI & Debugging

সহজ ভাষায় (Analogy)

তুমি একটা বড় প্রজেক্টের ম্যানেজার, এবং কাজ কেমন চলছে তা জানার জন্য তুমি প্রতিদিন রিপোর্ট দেখো

Spark UI হলো সেই রিপোর্টিং টুল যা তোমাকে Spark cluster এর কাজের অবস্থা, রিসোর্স ব্যবহার, ও পারফরমেন্স দেখায়

Formal Description

- Spark UI তে job, stage, task এর detail logs, timing, shuffle, executor info পাওয়া যায়
- এটি তোমাকে bottleneck বুঝতে ও optimization করতে সাহায্য করে
- Web UI default port 4040 এ চলে, local machine থেকে ব্রাউজ করে access করা যায়

Practical Tips

- Long-running jobs এ Spark UI তে সময় সময় জটিলতা দেখে performance tuning করো
- Stage বা task failure হলে UI তে error trace পাবে

14. Spark Streaming (Introduction)

সহজ ভাষায় (Analogy)

তুমি একটা নদীর পানি ধরে বিশ্লেষণ করতে চাও, কিন্তু নদীর পানি কখনো থামে না

Spark Streaming হলো সেই পাইপলাইন যা এই চলমান (live) পানি, অর্থাৎ ডেটা স্ট্রিম থেকে টুকরো টুকরো করে (micro-batches) নিয়ে কাজ করে

Formal Description

- Spark Streaming real-time data প্রসেসিংয়ের জন্য, ডেটাকে ছোট ছোট micro-batches এ ভাগ করে process করে
- Structured Streaming হলো Spark Streaming এর উন্নত ভার্সন যা declarative API দেয়
- Supports Windowed computations, Stateful processing ইত্যাদি

Use Cases

- Real-time log processing
- Sensor data analytics
- Fraud detection

কোড উদাহরণ (Structured Streaming)

python

CopyEdit

```
df = spark.readStream.format("socket") \
.option("host", "localhost") \
.option("port", 9999).load()

words = df.selectExpr("explode(split(value, ' ')) as word")

word_counts = words.groupBy("word").count()

query = word_counts.writeStream.outputMode("complete").format("console").start()

query.awaitTermination()
```

15. Spark MLlib (Machine Learning)

সহজ ভাষায় (Analogy)

তুমি একটা অটোমেটেড মেশিন বানাতে চাও যা তোমার ডেটা দেখে সিদ্ধান্ত নিতে পারে, ভবিষ্যৎ আন্দাজ করতে পারে

Spark MLlib হলো সেই লাইব্রেরি যা বড় ডেটাতে scalable machine learning মডেল তৈরি করতে সাহায্য করে

Formal Description

- MLlib এ classification, regression, clustering, collaborative filtering ইত্যাদি

algorithms আছে

- Pipeline API দিয়ে complex ML workflows তৈরি করা যায়
- Spark এর distributed পরিবেশে parallel training হয়

Use Cases

- Spam detection
- Customer segmentation
- Recommendation systems

কোড উদাহরণ (Simple Logistic Regression)

python

CopyEdit

```
from pyspark.ml.classification import LogisticRegression
from pyspark.ml.feature import VectorAssembler

data = spark.createDataFrame([
    (0, 1.0, 3.0),
    (1, 2.0, 5.0),
    (0, 1.3, 2.3),
    (1, 3.3, 4.4)
], ["label", "feature1", "feature2"])

assembler = VectorAssembler(inputCols=["feature1", "feature2"], outputCol="features")
data_prepared = assembler.transform(data)

lr = LogisticRegression(featuresCol="features", labelCol="label")
model = lr.fit(data_prepared)

predictions = model.transform(data_prepared)
predictions.show()
```

16. Error Handling & Best Practices

সহজ ভাষায়

কোনো কাজ করলে ভুল হতেই পারে, তাই সাবধানে কাজ করো, ভুল বুঝে ঠিক করো

Lazy evaluation বুঝে Action ও Transformation ঠিকমতো ব্যবহার করো

Formal Tips

- যথাযথ partitioning করো, যাতে workload balanced হয়
- Shuffle কমানোর চেষ্টা করো
- Cache করো, যখন প্রয়োজন
- Spark UI ব্যবহার করে performance monitor করো
- Exception handling দিয়ে Spark job fail হলে retry ব্যবস্থা রাখো

17. Spark Integration with Other Tools

সহজ ভাষায়

Spark একা চলবে না, অন্য অনেক টুলের সাথে মিলে কাজ করবে — যেমন Hadoop ডেটা, Kafka স্ট্রিম, Hive টেবিল, JDBC ডেটাবেস ইত্যাদি

Formal Description

- Hadoop HDFS থেকে ডেটা পড়া যায়
- Kafka থেকে স্ট্রিম ডেটা নিয়ে কাজ করা যায়
- Hive Metastore ও টেবিল থেকে ডেটা নিয়ে Spark SQL চালানো যায়
- JDBC দিয়ে relational DB তে ডেটা লিখতে বা পড়তে পারো

18. Broadcast Variables & Accumulators

সহজ ভাষায় (Analogy)

Broadcast Variables:

তুমি ধরা যাক, তোমার একটি বড় ম্যাপ বা তথ্যের তালিকা আছে যেটা তোমার পুরো টিমকে দিতে হবে তবে তুমি চাইছো একবার ম্যাপটা সবাইকে পাঠানো হোক, যেন প্রত্যেকজনের কাছে আলাদা করে অনেক বার না পাঠাতে হয় Spark এ broadcast variables ঠিক এমনই কাজ করে — একবার ডেটা পাঠানো হয় সব worker node-এ, আর সবাই সেটা ভাগাভাগি করে ব্যবহার করে

Accumulators:

তুমি যদি অনেক লোকের কাজ থেকে মোট ফলাফল বা সংখ্যা সংগ্রহ করতে চাও, যেমন কয়জন কাজ সঠিকভাবে করেছে বা কতবার error এসেছে, তখন accumulator কাজ দেয় ধরো, তোমার টিমের প্রত্যেক সদস্য তাদের কাজের progress জানাচ্ছে, আর তুমি সেই সংখ্যা গুলো জমা

রাখছো Spark এ accumulator হলো এমন একটা ভেরিয়েবল যা worker node গুলো থেকে তথ্য যোগ করতে পারে, কিন্তু driver শুধু পড়তে পারে

Formal Description

- Broadcast Variable: Spark এর একটি read-only variable যা driver থেকে executors এ efficient ভাবে distribute করা হয় এটি executor গুলোর মাঝে shared data হিসেবে কাজ করে, বিশেষ করে যখন বড় বা static data (যেমন lookup table) প্রয়োজন হয় এতে network overhead কমে
- Accumulator: একটি write-only variable যা distributed task গুলো থেকে aggregation বা summation করতে ব্যবহার হয় worker nodes value বাড়ায়, কিন্তু driver থেকে শুধুমাত্র পড়া যায় সাধারণত counters বা sums এর জন্য ব্যবহৃত

কোড উদাহরণ ও ব্যাখ্যা

python

CopyEdit

```
# Broadcast Variable Example
broadcastVar = sc.broadcast([2, 4, 6, 8])

rdd = sc.parallelize([1, 2, 3, 4, 5, 6, 7, 8, 9])

# শুধুমাত্র broadcastVar এর মধ্যে থাকা সংখ্যা গুলো ফিল্টার করব
filtered = rdd.filter(lambda x: x in broadcastVar.value).collect()

print(filtered)  # Output: [2, 4, 6, 8]
```

ব্যাখ্যা:

broadcastVar.value executor গুলোতে copy হয় তাই প্রত্যেক worker node আলাদা করে পুরো list না নিয়ে, shared broadcast data ব্যবহার করে ফিল্টারিং করে এতে network cost কমে

python

CopyEdit

```
# Accumulator Example
accum = sc.accumulator(0)
```

```
def count_negatives(x):
    global accum
    if x < 0:
        accum += 1


    rdd = sc.parallelize([10, -1, 20, -3, 5, -2])

rdd.foreach(count_negatives)

print("Negative numbers count:", accum.value)  # Output: 3
```

ব্যাখ্যা:

প্রত্যেক worker node যখন count_negatives ফাংশন চালায়, তখন accum variable এর মান বাড়ে শেষে driver সেই মান পড়ে নেয় তবে worker node থেকে সরাসরি পড়া যায় না, শুধুমাত্র যোগ করা যায়

19. DataFrame vs Dataset (Spark SQL)

সহজ ভাষায় (Analogy)

DataFrame হলো একটি এক ধরনের এক্সেল শীটের মতো যেখানে row ও column থাকে, কিন্তু প্রতিটা column এর ডেটার ধরন স্পষ্ট হয় না Dataset হলো DataFrame এর উন্নত সংস্করণ যা programmatically প্রতিটি row এর ডেটার ধরন নির্দিষ্ট করে রাখে (type safe)

Python এ Dataset API নেই, তবে Scala/Java তে Dataset strong typed API যা কম্পাইল টাইমে ডেটার ধরন চেক করে, ফলে error কম হয়

Formal Description

বৈশিষ্ট্য	DataFrame	Dataset
Introduced In	Spark 1.3	Spark 1.6
Type Safety	Untyped (Generic Row objects)	Strongly typed (Compile-time safety)
API Language	Python, Scala, Java	Scala, Java
Optimizati	Catalyst Optimizer	Catalyst Optimizer

on

Use Case	SQL querying, general data analysis	Complex typed transformations

Example (Scala/Java তে Dataset)

scala

CopyEdit

```
case class Person(name: String, age: Int)

    val ds = spark.read.json("people.json").as[Person]

ds.filter(_.age > 30).show()
```

ব্যাখ্যা:

এখানে Person class দিয়ে typed Dataset তৈরি করা হলো, যেখানে name ও age এর ধরন compile time এ নির্দিষ্ট ফলে ভুল কম হয়

20. Partition Pruning & Predicate Pushdown

সহজ ভাষায় (Analogy)

ধরো তুমি একটা বড় বইয়ের বইমেলায় গেছো, কিন্তু তুমি শুধু নির্দিষ্ট অংশের বই দেখতে চাও তোমার সময় বাঁচাতে বইমেলার লোকেরা সেই নির্দিষ্ট অংশের বুকগুলো আলাদা করে সাজিয়ে দেয় তোমার জন্য

Spark এ partition pruning ও predicate pushdown ঠিক এরকম কাজ করে — query এর সময় ডেটার ছোট অংশ সিলেক্ট করে ডেটা লোড করে, পুরো ডেটা না পড়ে

Formal Description

- Partition Pruning: Spark যখন partitioned টেবিল থেকে query চালায়, তখন শুধু প্রয়োজনীয় partition গুলোতে ডেটা পড়ে যেমন, partition column যদি year=2024 হয়, আর query তে year=2024 filter থাকে, Spark শুধু ঐ partition scan করবে
- Predicate Pushdown: Query এর filter বা condition গুলো data source (যেমন Parquet, ORC, JDBC) এর কাছে পাঠানো হয়, যাতে কম ডেটা read হয় এবং network

overhead কমে

Example

python

CopyEdit

```
df = spark.read.parquet("sales_data")  # Assume partitioned by year


# partition pruning হবে কারণ filter applied on partition column
filtered_df = df.filter(df.year == 2024)

filtered_df.show()
```

ব্যাখ্যা:

filter(df.year == 2024) ব্যবহার করলে Spark শুধু year=2024 partition থেকে ডেটা নিয়ে কাজ করবে, বাকি partitions skip করবে এটাই partition pruning

21. Window Functions in Spark SQL

সহজ ভাষায় (Analogy)

তুমি একটা ফুটবল টিমের খেলোয়াড়দের স্কোরের তালিকা নিয়ে কাজ করছো, এবং চাচ্ছো যে দলের মধ্যে প্রত্যেক খেলোয়াড়ের rank বের করো কিন্তু পুরো টিমের বদলে তুমি position অনুযায়ী আলাদা আলাদা rank চাও এই কাজটা Spark এর window functions দিয়ে হয়

Formal Description

Window function হলো Spark SQL এর analytic function যা ডেটা এর একটি নির্দিষ্ট window বা গ্রুপ এর মধ্যে calculations করে যেমন rank, row_number, moving average ইত্যাদি এটি partitionBy ও orderBy ব্যবহার করে নির্দিষ্ট window নির্ধারণ করে

Example

python

CopyEdit

```
from pyspark.sql.window import Window
import pyspark.sql.functions as F
```

```
data = [("TeamA", "Player1", 20),
        ("TeamA", "Player2", 15),
        ("TeamB", "Player3", 30),
        ("TeamB", "Player4", 25)]

    df = spark.createDataFrame(data, ["team", "player", "score"])

windowSpec = Window.partitionBy("team").orderBy(F.desc("score"))

df.withColumn("rank", F.rank().over(windowSpec)).show()
```

ব্যাখ্যা:

এখানে প্রতিটি টিমের মধ্যে খেলোয়াড়দের স্কোর অনুযায়ী rank দেওয়া হয়েছে partitionBy("team") দ্বারা আলাদা আলাদা টিমের জন্য rank শুরু হয়েছে

22. Checkpointing in Spark

সহজ ভাষায় (Analogy)

তুমি অনেক বড় একটা গেম খেলছো মাঝে মাঝে সেই গেমের অবস্থা save করে রাখো, যাতে হঠাৎ বিদ্যুৎ গেলে তুমি আবার শেষ সেভ থেকে শুরু করতে পারো Spark এ checkpointing ঠিক এরকম — intermediate computation save করে রেখে, failure থেকে সহজে recovery সম্ভব হয়

Formal Description

Checkpoint হলো Spark এর একটি feature যা RDD বা DataFrame এর lineage (dependency chain) থেকে মুক্তি দিয়ে intermediate state persistent storage (HDFS বা local disk) এ সংরক্ষণ করে এটি long lineage থাকার কারণে failure recovery ও memory management সহজ করে

Usage Example

python

CopyEdit

```
    sc.setCheckpointDir("/tmp/spark-checkpoints")
```

```
    rdd = sc.parallelize([1, 2, 3, 4, 5])

rdd_checkpointed = rdd.checkpoint()

rdd_checkpointed.count()
```

ব্যাখ্যা:

setCheckpointDir() দিয়ে checkpoint ডিরেক্টরি সেট করা হয় checkpoint() কল করলে Spark lineage বাদ দিয়ে সেই রিসাল্ট সেভ করে

23. Custom Partitioners

সহজ ভাষায় (Analogy)

তুমি তোমার কাজগুলো নিজস্ব নিয়মে ভাগ করতে চাও, যেমন তুমি গ্রুপে ভাগ করার সময় নিজের পছন্দ অনুযায়ী সদস্যদের ভাগ করো Spark এ custom partitioner দিয়ে ডেটাকে নিজের মতো ভাগ করার সুযোগ পাওয়া যায়

Formal Description

Partitioner Spark কে বলে কিভাবে key-value ডেটা partition হবে Spark ডিফল্ট hash partitioner ব্যবহার করে, কিন্তু প্রয়োজনে তুমি নিজের custom partitioner তৈরি করে দিতে পারো যা নির্দিষ্ট logic অনুসারে partition করে

Example (Scala)

```
scala
CopyEdit
class CustomPartitioner(partitions: Int) extends Partitioner {
  def numPartitions: Int = partitions

  def getPartition(key: Any): Int = {
    key match {
      case s: String => s.charAt(0).toInt % partitions
      case _ => 0
    }
  }
}
```

```
    val rdd = sc.parallelize(Seq(("apple", 1), ("banana", 2), ("carrot", 3)))

val partitionedRDD = rdd.partitionBy(new CustomPartitioner(3))
```

24. Spark GraphX (Graph Processing)

সহজ ভাষায় (Analogy)

তুমি যদি social network বা যেকোনো network এর মধ্যে সম্পর্ক বিশ্লেষণ করতে চাও, Spark GraphX সেই কাজটা করে দেয় — যেখানে nodes এবং edges নিয়ে বিশ্লেষণ হয়

Formal Description

GraphX Spark এর একটি API যা large-scale graph processing ও analytics এর জন্য এটি Pregel API ব্যবহার করে graph algorithms যেমন PageRank, Connected Components, Triangle Counting ইত্যাদি চালায়

Example

scala

CopyEdit

```
    import org.apache.spark.graphx._

    val vertices = sc.parallelize(Seq((1L, "Alice"), (2L, "Bob"), (3L, "Charlie")))
    val edges = sc.parallelize(Seq(Edge(1L, 2L, "friend"), Edge(2L, 3L, "follow")))

val graph = Graph(vertices, edges)

val ranks = graph.pageRank(0.1).vertices

ranks.collect.foreach(println)
```

25. Spark Structured Streaming vs DStream

সহজ ভাষায় (Analogy)

DStream হলো Spark এর পুরনো লাইভ ডেটা প্রসেসিং API যা micro-batch এ কাজ করে Structured Streaming হলো নতুন API যা DataFrame/Dataset API এর ওপর ভিত্তি করে, যা সহজে scalable ও fault-tolerant

Formal Description

- DStream: Spark Streaming এর পুরানো API, RDD ভিত্তিক
- Structured Streaming: Spark 2.x থেকে চালু, DataFrame/Dataset API ভিত্তিক Declarative, সহজে ব্যবহারযোগ্য, বিভিন্ন sink/source সাপোর্ট করে

কোড উদাহরণ (Structured Streaming)

python

CopyEdit

```
df = spark.readStream.format("socket").option("host", "localhost").option("port", 9999).load()

word_counts = df.selectExpr("explode(split(value, ' ')) as word").groupBy("word").count()

query = word_counts.writeStream.outputMode("complete").format("console").start()

query.awaitTermination()
```

26. Spark Performance Tuning Tips

সহজ ভাষায়

Spark কে দ্রুত ও ভালো কাজ করানোর জন্য কিছু নিয়ম মেনে চলা দরকার — যেমন workload balance রাখা, অপ্রয়োজনীয় shuffle এড়িয়ে চলা, ডেটা cache করা, ইত্যাদি

Formal Tips

- Partition Size: প্রতিটি partition এর সাইজ 64MB থেকে 128MB রাখো
- Shuffle কমাও: কম shuffle মানে network ও disk I/O কম হবে
- Caching: Iterative algorithm এ cache করো
- Broadcast Variables: ছোট lookup tables broadcast করো

- Avoid Wide Transformations: বেশি shuffle হয় এমন operations এড়িয়ে চলো
- Use Tungsten & Catalyst: Spark এর optimizer গুলো enable রাখো

27. Common Spark Errors & Fixes

সহজ ভাষায় ও সমাধান

Error	কারন	সমাধান
Memory Overflow	Executor RAM কম	executor memory বাড়াও
Serialization Error	Object serializable নয়	Kryo serializer ব্যবহার করো
Skewed Data	Uneven data distribution	Salting technique প্রয়োগ করো
Task Failures	Resource shortage বা bug	Retry settings adjust করো, bug fix করো

28. Spark Configuration Parameters Overview

Parameter	Description	Default
spark.executor.memory	Executor node RAM size	1g
spark.driver.memory	Driver node RAM size	1g
spark.sql.shuffle.partitions	Shuffle operation এর partition সংখ্যা	200
spark.default.parallelism	Default task parallelism সংখ্যা	cluster CPU core

29. Integration with Cloud Platforms

সহজ ভাষায়

Spark সহজে cloud platform এর service গুলো যেমন AWS EMR, Azure Databricks, Google Dataproc এর সাথে কাজ করে এসব প্ল্যাটফর্ম Spark cluster চালাতে সুবিধা দেয়

Formal Description

- Cloud storage থেকে যেমন S3, Azure Data Lake Storage থেকে Spark ডেটা পড়তে পারে
- Cloud based cluster management, auto scaling সুবিধা
- Managed Spark services দিয়ে DevOps কম হয়, সহজে Spark চালানো যায়