Ministry of Higher Education and Scientific Research
University of Mila
Institute of Mathematics and Computer Science
Department of Computer Science
Master 2 I2A – Big Data
2025/2026

Practical Works 8 - The DzStore Data Pipeline (Spark + MongoDB)

Context: You are the Lead Data Engineer for **DzStore**, Algeria's fastest-growing electronics e-commerce platform. The CEO wants to launch a "Black Friday" event. Your infrastructure must handle high-speed transactions and provide real-time analytics on trending products.
Objective: Link your Storage layer (MongoDB) with your Processing layer (Spark) to build the backend for these features.
Prerequisites:

- Google Colab (Recommended).
- MongoDB Atlas Account (The "DzStore" Production DB).

Part 1: The "Black Friday" Stress Test (CAP Theorem)

Scenario: During Black Friday, thousands of users click "Buy Now" simultaneously.

- **The Checkout Microservice** needs speed (Latency is king).
- **The Billing Microservice** needs safety (Durability is king).

We need to measure the trade-off to decide which configuration to use for which service.

1.1 Setup (Python)

Connect to your Atlas cluster.

```
!pip install pymongo

from pymongo import MongoClient, WriteConcern
import time
import statistics

# Replace with your Atlas Connection String
URI = "mongodb+srv://student:pass@cluster0.mongodb.net/?retryWrites=true&w=majority"

# 1. Checkout Service Config (Optimized for Speed)
# We accept risk of data loss if the primary crashes, but we want speed.
client_checkout = MongoClient(URI)
db = client_checkout['dzstore_db']
coll_checkout = db.get_collection('orders_fast', write_concern=WriteConcern(w=1))

# 2. Billing Service Config (Optimized for Safety)
# We CANNOT lose payment data. We wait for acknowledgment from the majority of nodes.
client_billing = MongoClient(URI)
coll_billing = db.get_collection('orders_secure', write_concern=WriteConcern(w='majority'))
```

1.2 The Benchmark

Simulate 100 simultaneous orders and measure how long the user has to wait.

```
def run_stress_test(collection, service_name):
    latencies = []
    print(f"--- Stress Testing: {service_name} ---")

    # Clean start
    collection.drop()
```

```
    # Simulate 100 orders
    for i in range(100):
        order_payload = {
            "order_id": f"BF_{i}",
            "user": "Amine",
            "items": ["Gaming Laptop", "Mouse"],
            "total": 150000,
            "timestamp": time.time()
        }

        start = time.time()
        collection.insert_one(order_payload)
        end = time.time()

        latencies.append((end - start) * 1000) # ms

    avg_lat = statistics.mean(latencies)
    print(f"Average User Wait Time: {avg_lat:.2f} ms")
    print("-" * 30)

run_stress_test(coll_checkout, "Checkout Service (w=1)")
run_stress_test(coll_billing, "Billing Service (w=majority)")
```

Architectural Decision:

- Based on your results, explain why using `w='majority'` for the live "Add to Cart" button might crash the user experience during Black Friday.

Part 2: Spark & MongoDB Integration (The Bridge)

Now we need to connect our Analytics Engine (Spark) to our Operational Database (MongoDB).

2.1 Configuration

Run this in Colab to install Spark and the MongoDB Connector.

```
# 1. Install Dependencies
!apt-get install openjdk-8-jdk-headless -qq > /dev/null
!wget -q [https://dlcdn.apache.org/spark/spark-3.5.0/spark-3.5.0-bin-
hadoop3.tgz](https://dlcdn.apache.org/spark/spark-3.5.0/spark-3.5.0-bin-hadoop3.tgz)
!tar xf spark-3.5.0-bin-hadoop3.tgz
!pip install -q findspark

# 2. Environment Variables
import os
os.environ["JAVA_HOME"] = "/usr/lib/jvm/java-8-openjdk-amd64"
os.environ["SPARK_HOME"] = "/content/spark-3.5.0-bin-hadoop3"

# 3. Initialize Spark Session with DzStore Config
import findspark
findspark.init()

from pyspark.sql import SparkSession

spark = SparkSession.builder \
    .master("local[*]") \
    .appName("DzStore_Analytics") \
    .config("spark.mongodb.read.connection.uri", URI) \
    .config("spark.mongodb.write.connection.uri", URI) \
    .config("spark.jars.packages", "org.mongodb.spark:mongo-spark-connector_2.12:10.2.1") \
    .getOrCreate()
```

Part 3: The Marketing Campaign (Optimization)

Scenario: The Marketing team wants to send a promo code to **VIP Users** (Total Spend > 100,000 DA) living in **Oran**. We have 10,000 users in MongoDB. If we load all of them into Spark to filter them, we waste bandwidth. We must ensure Spark "pushes" the filter to MongoDB.

3.1 Data Seeding

Generate dummy user data in MongoDB.

```
# Generate 2000 users
users_data = []
cities = ["Algiers", "Oran", "Constantine", "Setif"]
import random

for i in range(2000):
    users_data.append({
        "user_id": i,
        "city": random.choice(cities),
        "total_spend": random.randint(1000, 200000) # Some are VIPs, some are not
    })

db.users.drop()
db.users.insert_many(users_data)
print("DzStore User Database Populated.")
```

3.2 The Smart Query

Use Spark to find the target audience.

```
# 1. Connect to the Collection
df_users = spark.read.format("mongodb") \
    .option("database", "dzstore_db") \
    .option("collection", "users") \
    .load()

# 2. Define the VIP Criteria
# "Find users in Oran who spent > 100,000"
vip_users = df_users.filter(
    (df_users["city"] == "Oran") &
    (df_users["total_spend"] > 100000)
)

# 3. Check the Plan
print("--- Execution Plan ---")
vip_users.explain()

# 4. View Results
vip_users.show(5)
```

Task: Check the `explain()` output. Look for `PushedFilters`.

- Does it show EqualTo(city, Oran) and GreaterThan(total_spend, 100000)?
- If yes, you successfully optimized the query!

Part 4: "Trending Products" Pipeline (ETL Project)

Scenario: The "Hot Right Now" widget on the homepage is broken. We need to calculate which products are getting the most views *right now*.

- **Source:** The web servers generate raw CSV logs (`web_traffic.csv`).
- **Logic:** Use Spark to count views per product.
- **Destination:** Save the counts to MongoDB (`trending_products` collection) so the frontend can display them.

4.1 The Raw Logs (Source)

Create a simulated log file.

```
log_data = """timestamp,user_id,product_name,action
2025-12-06 10:00,101,iPhone 15,view
2025-12-06 10:01,102,Samsung S24,view
2025-12-06 10:02,101,iPhone 15,add_to_cart
2025-12-06 10:03,103,PlayStation 5,view
2025-12-06 10:04,104,iPhone 15,view
2025-12-06 10:05,102,Samsung S24,buy
2025-12-06 10:06,105,PlayStation 5,view
2025-12-06 10:07,106,iPhone 15,view"""
```

```
with open("web_traffic.csv", "w") as f:
    f.write(log_data)
```

4.2 The Processing Logic (Transform)

Use PySpark to calculate popularity.

```
from pyspark.sql.functions import desc

# 1. Load CSV
df_logs = spark.read.csv("web_traffic.csv", header=True, inferSchema=True)

# 2. Filter for 'view' actions only
df_views = df_logs.filter(df_logs["action"] == "view")

# 3. Aggregation: Count views per product
df_trending = df_views.groupBy("product_name").count()

# 4. Sort by popularity
df_trending = df_trending.orderBy(desc("count"))

print("--- Trending Products (Calculated in Spark) ---")
df_trending.show()
```

4.3 Serving the Data (Load)

Write the result back to MongoDB so the website can read it.

```
# Write to the 'trending_products' collection
df_trending.write.format("mongodb") \
    .option("database", "dzstore_db") \
    .option("collection", "trending_products") \
    .mode("overwrite") \
    .save()

print("Data successfully pushed to Production DB!")
```

Verification: Go to your MongoDB Atlas dashboard. You should see a new collection `trending_products` with documents like: `{ "product_name": "iPhone 15", "count": 3 }`