Apache Spark on Kubernetes: Understanding Driver, Executors, and Data Flow

1. Overview

Apache Spark is a distributed computing framework designed for large-scale data processing. When running Spark on Kubernetes, understanding how **driver** and **executor pods** interact, and how data flows between them, is crucial.

2. Components of Spark Application

Driver: - Central coordinator of a Spark application. - Maintains DAG (Directed Acyclic Graph) of transformations. - Schedules tasks on executors. - Receives results of actions (if requested).

Executor: - Runs on worker nodes. - Holds partitions of data. - Performs transformations and actions locally. - Sends results or intermediate aggregation to driver if needed.

3. Components of Master and Worker Nodes (Standalone Spark)

Master Node Components: - **Spark Master process**: Manages cluster resources and schedules applications. - **Driver program**: Can run on master node or a client machine, responsible for creating SparkContext and DAG. - **Web UI**: Monitors jobs, stages, and tasks.

Worker Node Components: - **Executor processes**: Run Spark tasks on partitions. - **BlockManager**: Manages storage of RDD partitions in memory/disk. - **Task scheduler**: Receives task instructions from driver. - **Shuffle Service**: Handles data exchange between executors during shuffle operations.

4. Transformations vs Actions

- **Transformations (lazy operations)**: `filter`, `map`, `flatMap`.
- Only update the **logical plan** on the driver.
- No data movement occurs until an action is triggered.
- **Actions (trigger computation)**: `collect()`, `count()`, `take(n)`, `write()`.
- Cause executors to process partitions.
- May send results to driver or storage.

5. Lineage and Intermediate Data

```
df1 = df0.filter(...)
df3 = df1.filter(...)
```

- `df1` and `df3` are **logical plans on the driver**, not materialized data.

• Data is physically processed **on executors**.
• Without caching, transformations are recomputed each time an action is triggered.
• Caching stores intermediate results **in executor memory** to avoid recomputation:

```
df1.cache()
```

6. Data Partitioning

• DataFrames/RDDs are split into **partitions**.
• Each executor works on its **local partition**.
• Transformations like `filter()` operate **within partitions**.
• Shuffles or repartitions move data **between executors** as needed.

7. Data Combination and Aggregation

Without `collect()`: - Executors may perform **local aggregation**. - Partial results may be combined among executors (shuffle) before producing final output. - Driver only receives **final aggregated result**, not all raw data.

With `collect()`: - Each executor sends its partition data to the **driver** over the network. - Driver combines results into a **single list of rows**. - Only use `collect()` for small datasets to avoid exhausting driver memory.

Example: Max Value

```
Partitions: [3,8,1], [7,2,9], [4,6,5]
Executor local max: 8, 9, 6
Driver receives: [8, 9, 6] → global max = 9
```

- Driver combines **partial maxima** to compute global maximum. - For complex aggregations (e.g., `groupBy`), shuffle may happen and final aggregation may occur **on executors or driver** depending on result size.

8. Spark on Kubernetes

Pods and Containers: - Driver runs in **driver pod** (usually one container). - Executors run in **executor pods** (one container each). - Pods may reside on **different worker nodes** for isolation and scalability. - Communication between driver and executors happens via **network (RPC)**.

Optional Sidecars: - Driver or executor pods may have sidecar containers for logging, monitoring, or metrics.

Pod Placement: - Scheduler places pods based on **resource availability, node selectors, and affinity rules**. - Driver and executors can end up on same or different nodes.

9. Summary of Key Points

1. Driver orchestrates tasks, maintains DAG, and receives minimal results.
2. Executors process partitions, apply transformations, and perform local aggregations.
3. `collect()` explicitly instructs driver to receive data from executors.
4. Intermediate DataFrames like `df.filter()` reside in **executors**, not driver.
5. Global aggregation (max, sum, count) uses **partial aggregation on executors**; driver combines small results.
6. Spark on Kubernetes: driver and executors run in pods; placement can vary.
7. Caching reduces recomputation and improves performance.
8. Master node manages resources and scheduling; worker nodes run executors and manage data storage.