Introduction to Big Data with Spark and Hadoop

Module 6 Glossary: Monitoring and Tuning

Welcome! This alphabetized glossary contains many of the terms in this course. This comprehensive glossary also includes additional industry-recognized terms not used in course videos. These terms are essential for you to recognize when working in the industry, participating in user groups, and in other professional certificate programs.

Estimated reading time: 5 minutes

Term	Definition
Apache Spark user interface	Provides valuable insights, organized on multiple tabs, about the running application. The **Jobs** tab displays all jobs in the application, including job status. The **Stages** tab reports the state of tasks within a stage. The **Storage** tab shows the size of any RDDs or DataFrames that persisted to memory or disk. The **Environment** tab includes any environment variables and system properties for Spark or the JVM. The **Executor** tab displays a summary that shows memory and disk usage for any executors in use for the application. Additional tabs display based on the type of application in use.
Application dependency issues	Spark applications can have many dependencies, including application files such as Python script files, Java JAR files, and even required data files. Applications depend on the libraries used and their dependencies. Dependencies must be made available on all cluster nodes, either by preinstallation, including the dependencies in the Spark-submit script bundled with the application, or as additional arguments.
Application resource issues	CPU cores and memory can become an issue if a task is in the scheduling queue and the available workers do not have enough resources to run the tasks. As a worker finishes a task, the CPU and memory are freed, allowing the scheduling of another task. However, if the application asks for more resources that can ever become available, the tasks might never be run and eventually time out. Similarly, suppose that the executors are running long tasks that never finish. In that case, their resources never become available, which also causes future tasks to never run, resulting in a timeout error. Users can readily access these errors when they view the UI or event logs.
Application-id	A unique ID that Spark assigns to each application. These log files appear for each executor and driver process that the application runs.
Data validation	The practice of verifying the integrity, quality, and correctness of data used within Spark applications or data processing workflows. This validation process includes checking data for issues such as missing values, outliers, or data format errors. Data validation is crucial for ensuring that the data being processed in Spark applications is reliable and suitable for analysis or further processing. Various techniques and libraries, such as Apache Spark's DataFrame API or external tools, can be employed to perform data validation tasks in Spark environments.
Directed acyclic graph (DAG)	Conceptual representation of a series of activities. A graph depicts the order of activities. It is visually presented as a set of circles, each representing an activity, some connected by lines, representing the flow from one activity to another.
Driver memory	Refers to the memory allocation designated for the driver program of a Spark application. The driver program serves as the central coordinator of tasks, managing the distribution and execution of Spark jobs across cluster nodes. It holds the application's control flow, metadata, and the results of Spark transformations and actions. The driver memory's capacity is a critical factor that impacts the feasibility and performance of Spark applications. It should be configured carefully to ensure efficient job execution without memory-related issues.
Environment tab	Entails several lists to describe the environment of the running application. These lists include the Spark configuration properties, resource profiles, properties for Hadoop, and the current system properties.
Executor memory	Used for processing. If caching is enabled, additional memory is used. Excessive caching results in out-of-memory errors.
Executors tab	A component of certain distributed computing frameworks and tools used to manage and monitor the execution of tasks within a cluster. It typically presents a summary table at the top that displays relevant metrics for active or terminated executors. These metrics may include task-related statistics, data input and output, disk utilization, and memory usage. Below the summary table, the tab lists all the individual executors that have participated in the application or job, which may include the primary driver. This list often provides links to access the standard output (stdout) and standard error (stderr) log messages associated with each executor process. The Executors tab serves as a valuable resource for administrators and operators to gain insights into the performance and behavior of cluster executors during task execution.
Job details	Provides information about the different stages of a specific job. The timeline displays each stage, where the user can quickly see the job's timing and duration. Below the timeline, completed stages are displayed. In the parentheses beside the heading, users will see a quick view that displays the number of completed stages. Then, view the list of stages within the job and job metrics, including when the job was submitted, input or output sizes, the number of attempted tasks, the number of succeeded tasks, and how much data was read or written because of a shuffle.
Jobs tab	Commonly found in Spark user interfaces and monitoring tools, it offers an event timeline that provides key insights into the execution flow of Spark applications. This timeline includes crucial timestamps such as the initiation times of driver and executor processes, along with the creation timestamps of individual jobs within the application. The Jobs tab serves as a valuable resource for monitoring the chronological sequence of events during Spark job execution.
Multiple related jobs	Spark application can consist of many parallel and often related jobs, including multiple jobs resulting from multiple data sources, multiple DataFrames, and the actions applied to the DataFrames.
Parallelization	Parallel regions of program code executed by multiple threads, possibly running on multiple processors. Environment variables determine the number of threads created and calls to library functions.
Parquet	A columnar format that is supported by multiple data processing systems. Spark SQL allows reading and writing data from Parquet files, and Spark SQL preserves the data schema.
Serialization	Required to coordinate access to resources that are used by more than one program. An example of why resource serialization is needed occurs when one program is reading from a data set and another program needs to write to the data set.
Spark data persistence	Also known as caching data in Spark. Ability to store intermediate calculations for reuse. This is achieved by setting persistence in either memory or both memory and disk. Once intermediate data is computed to generate a fresh DataFrame and cached in memory, subsequent operations on the DataFrame can utilize the cached data instead of reloading it from the source and redoing previous computations. This feature is crucial for accelerating machine learning tasks that involve multiple iterations on the same data set during model training.

Term	Definition
Spark History server	Web UI where the status of running and completed Spark jobs on a provisioned instance of Analytics Engine powered by Apache Spark is displayed. If users want to analyze how different stages of the Spark job are performed, they can view the details in the Spark history server UI.
Spark memory management	Spark memory stores the intermediate state while executing tasks such as joining or storing broadcast variables. All the cached and persisted data will be stored in this segment, specifically in the storage memory.
Spark RDD persistence	Optimization technique that saves the result of RDD evaluation in cache memory. Using this technique, the intermediate result can be saved for future use. It reduces the computation overhead.
Spark standalone	Here, the resource allocation is typically based on the number of available CPU cores. By default, a Spark application can occupy all the cores within the cluster, which might lead to resource contention if multiple applications run simultaneously. In the context of the standalone cluster manager, a ZooKeeper quorum is employed to facilitate master recovery through standby master nodes. This redundancy ensures high availability and fault tolerance in the cluster management layer. Additionally, manual recovery of the master node can be achieved using file system operations in the event of master failure, allowing system administrators to intervene and restore cluster stability when necessary.
SparkContext	When a Spark application is being run, as the driver program creates a SparkContext, Spark starts a web server that serves as the application user interface. Users can connect to the UI web server by entering the hostname of the driver followed by port 4040 in a browser once that application is running. The web server runs for the duration of the Spark application, so once the SparkContext stops, the server shuts down, and the application UI is no longer accessible.
Stages tab	Displays a list of all stages in the application, grouped by the current state of either completed, active, or pending. This example displays three completed stages. Click the Stage ID Description hyperlinks to view task details for that stage.
Storage tab	Displays details about RDDs that have been cached or persisted to memory and written to disk.
Unified memory	Unified regions in Spark shared by executor memory and storage memory. If executor memory is not used, storage can acquire all the available memory and vice versa. If the total storage memory usage falls under a certain threshold, executor memory can discard storage memory. Due to complexities in implementation, storage cannot evict executor memory.
User code	Made up of the driver program, which runs in the driver process, and the functions and variables serialized that the executor runs in parallel. The driver and executor processes run the application user code of an application passed to the Spark-submit script. The user code in the driver creates the SparkContext and creates jobs based on operations for the DataFrames. These DataFrame operations become serialized closures sent throughout the cluster and run on executor processes as tasks. The serialized closures contain the necessary functions, classes, and variables to run each task.
Workflows	Include jobs created by SparkContext in the driver program. Jobs in progress run as tasks in the executors, and completed jobs transfer results back to the driver or write to disk.

Author(s)

- Niha Ayaz Sultan