APACHE SPARK II

Apache Spark es un motor de procesamiento rápido en memoria con API de desarrollo elegantes y expresivas para permitir a los trabajadores de datos ejecutar de manera eficiente el aprendizaje automático de streaming o las cargas de trabajo SQL que requieren un acceso interactivo rápido a los conjuntos de datos. Apache Spark consta de núcleo de Spark y un conjunto de bibliotecas.

El núcleo es el motor de ejecución distribuido y las API de Java, Scala y Python ofrecen una plataforma para el desarrollo de aplicaciones distribuidas.

Las bibliotecas adicionales creadas sobre el núcleo permiten las cargas de trabajo para streaming, SQL, procesamiento de gráficos y aprendizaje automático. SparkML, por ejemplo, está diseñado para la ciencia de datos y su abstracción facilita la ciencia de datos.

Para planificar y llevar a cabo los cálculos distribuidos, Spark utiliza el concepto de trabajo, que se ejecuta en los nodos de trabajo mediante etapas y tareas. Spark consta de un controlador, que orquesta la ejecución en un clúster de nodos de trabajo.

El controlador también es responsable de realizar el seguimiento de todos los nodos de trabajo, así como del trabajo que realiza actualmente cada uno de los nodos de trabajo.

Echemos un vistazo a los diversos componentes un poco más. Los componentes clave son el controlador y los ejecutores que son todos los procesos JVM (procesos Java):

1. Controlador: El controlador programa las aplicaciones, programa principal. Si utiliza el shell de Spark, se convierte en el programa Driver y el controlador inicia los ejecutores en el clúster y también controla las ejecuciones de tareas.

2. Ejecutor: A continuación, los ejecutores que se ejecutan en el programa de trabajo del clúster. Dentro del ejecutor, se ejecutan las tareas o cálculos individuales. Podría haber uno o más ejecutores en cada nodo de trabajo y, de forma similar, podría haber varias tareas dentro de cada ejecutor. Cuando Driver se conecta al administrador de clústeres, el administrador de clústeres asigna recursos para ejecutar ejecutores.

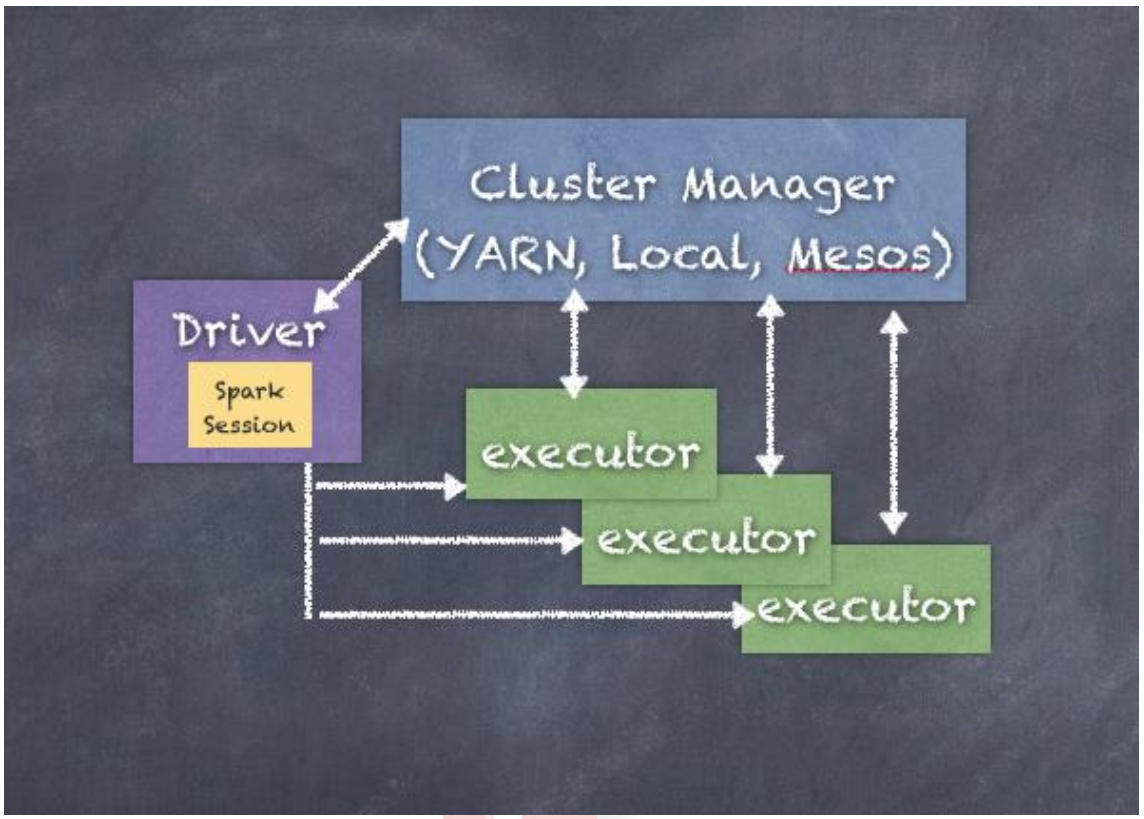
Nota

El administrador de clústeres podría ser un administrador de clúster independiente, YARN o Mesos.

El Administrador de clústeres es responsable de la asignación y asignación de recursos entre los nodos de proceso que forman el clúster.

Normalmente, esto se realiza al tener un proceso de administrador que conoce y administra un clúster de recursos y asigna los recursos a un proceso de solicitud como Spark. Veremos los tres gestores de clústeres diferentes: independiente, YARN y Mesos más abajo en las siguientes secciones.

A continuación, se muestra cómo funciona Spark a un alto nivel:



El punto de entrada principal a un programa Spark se llama. Él está dentro del componente Driver y representa la conexión con el clúster junto con el código para ejecutar el programador y la distribución y orquestación de tareas. `SparkContext SparkContext`

Nota

En Spark 2.x, se ha introducido una nueva variable llamada., , y ahora son variables miembro de la. `SparkSession SparkContext SQLContext HiveContext SparkSession`

Al iniciar el programa Driver, los comandos se emiten al clúster mediante el archivo, y, a continuación, los ejecutores ejecutarán las instrucciones. Una vez completada la ejecución, el programa Driver completa el trabajo. En este punto, puede emitir más comandos y ejecutar más trabajos. `SparkContext`

Nota

La capacidad de mantener y reutilizar la es una ventaja clave de la arquitectura de Apache Spark, a diferencia del marco de Hadoop donde cada trabajo o consulta de Hive o Pig Script inicia el procesamiento completo desde cero para

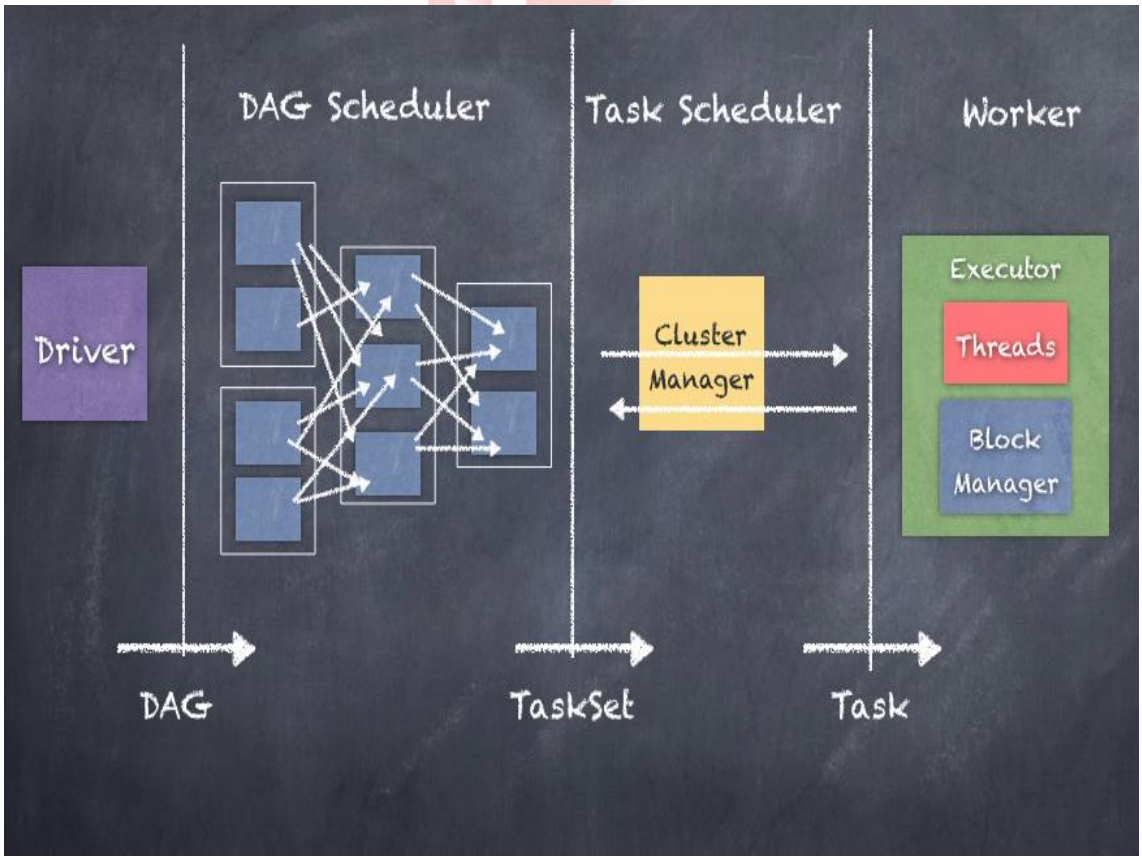
cada tarea que queremos ejecutar, también utilizando un disco caro en lugar de memoria. `SparkContext MapReduce`

Se puede utilizar para crear RDD, acumuladores y variables de difusión en el clúster. Solo uno puede estar activo por proceso JVM/Java. Debe el activo antes de crear uno nuevo. `SparkContext SparkContext stop() SparkContext`

El controlador analiza el código y serializa el código de nivel de bytes en los ejecutores que se van a ejecutar. Cuando realizamos cualquier cálculo, los cálculos se realizarán realmente en el nivel local por cada nodo, utilizando el procesamiento en memoria.

El proceso de analizar el código y planificar la ejecución es el aspecto clave implementado por el proceso de controlador.

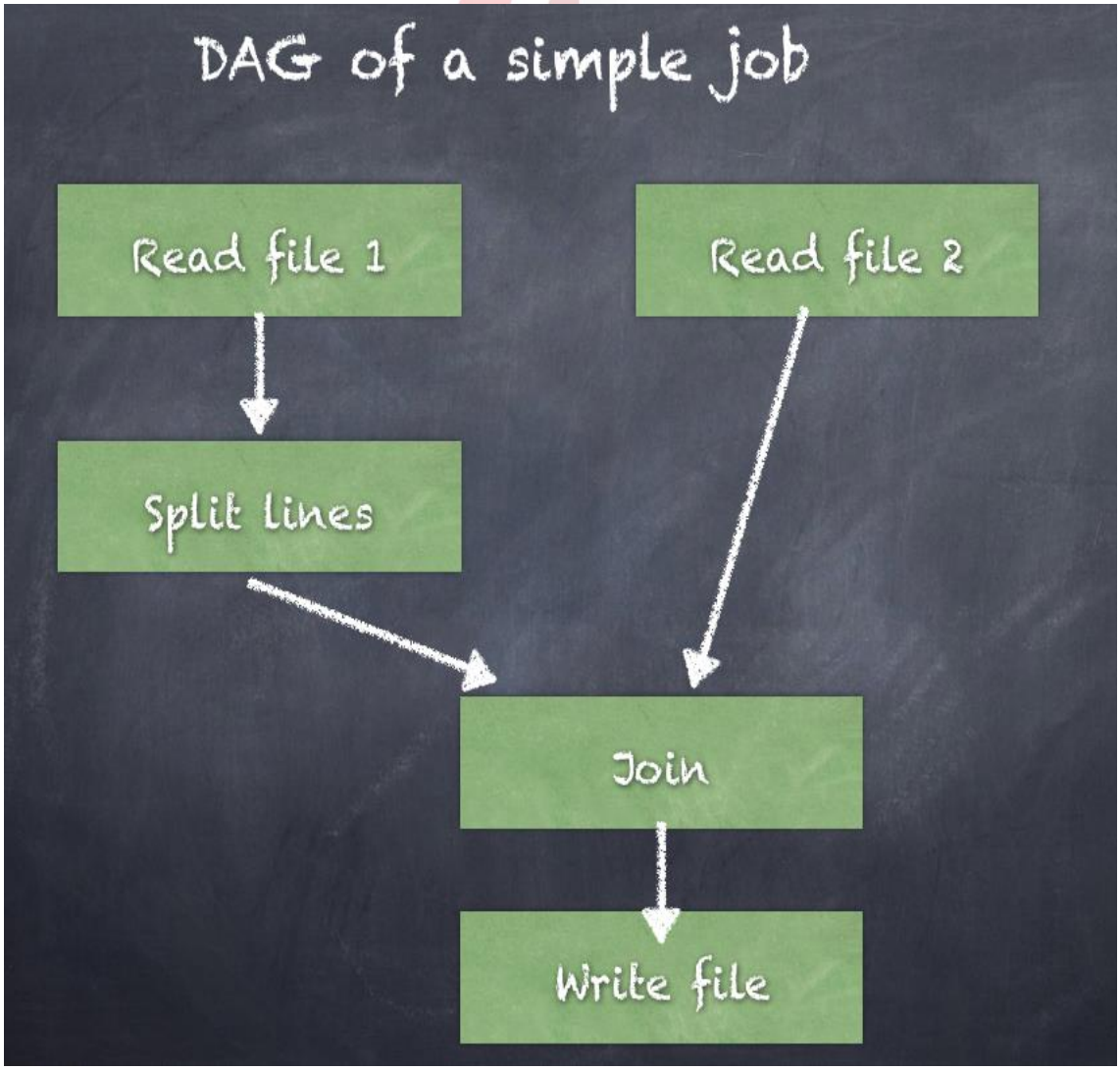
A continuación, se muestra cómo Spark Driver coordina los cálculos en todo el clúster:



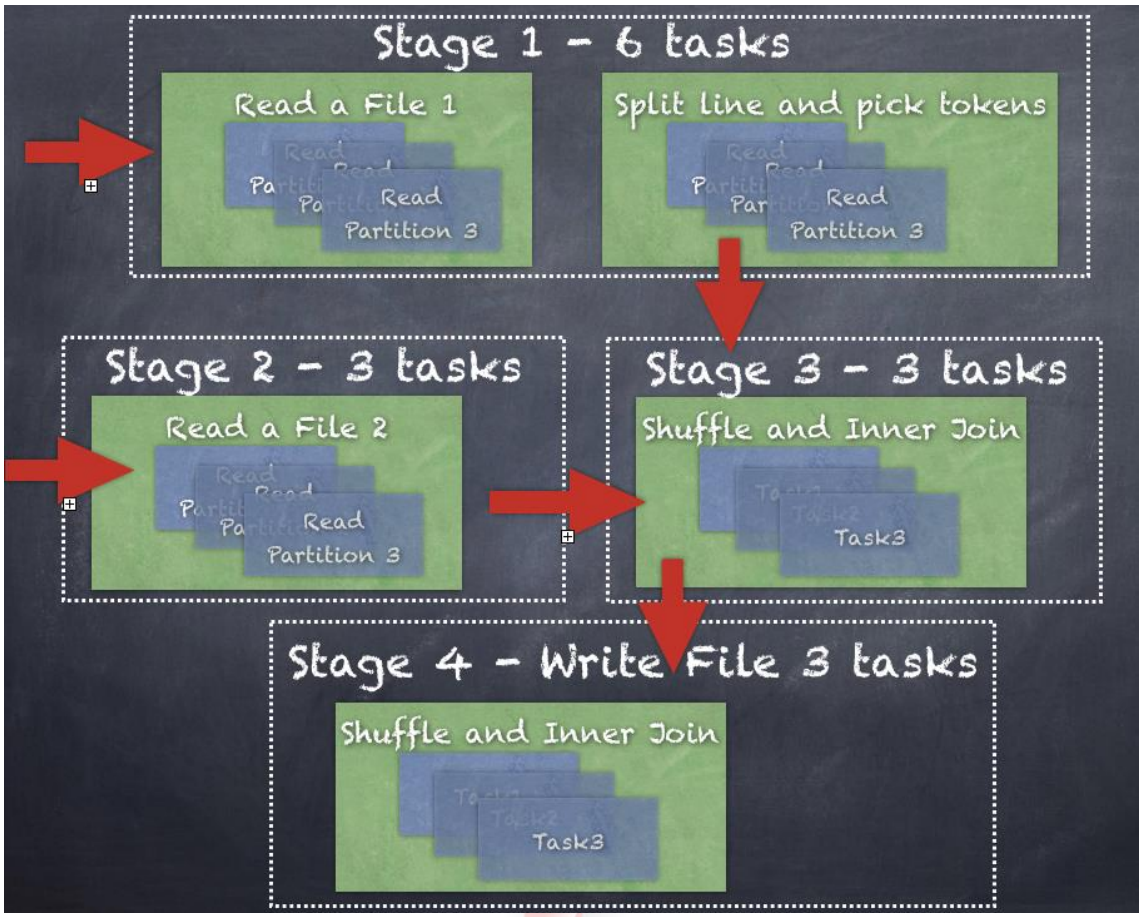
El gráfico acíclico dirigido (DAG) es la salsa secreta del marco. El proceso de controlador crea un DAG de tareas para un fragmento de código que intenta ejecutar mediante el marco de procesamiento distribuido.

A continuación, el programador de tareas ejecuta realmente el DAG en fases y tareas comunicándose con el Administrador de clústeres para que los recursos ejecuten los ejecutores. Un DAG representa un trabajo y un trabajo se divide en subconjuntos, también denominados etapas, y cada etapa se ejecuta como tareas mediante un núcleo por tarea.

En las dos ilustraciones siguientes se muestra una ilustración de un trabajo sencillo y cómo se divide el DAG en etapas y tareas; el primero muestra el trabajo en sí, y el segundo diagrama muestra las etapas en el trabajo y las tareas:



El siguiente diagrama ahora desglosa el trabajo/DAG en etapas y tareas:



El número de etapas y en qué consisten las etapas viene determinado por el tipo de operaciones. Normalmente, cualquier transformación entra en la misma etapa que la anterior, pero cada operación como reducir o barajar siempre crea una nueva etapa de ejecución. Las tareas forman parte de una etapa y están directamente relacionadas con los núcleos que ejecutan las operaciones en los ejecutores.

Nota

Si utiliza YARN o Mesos como administrador de clústeres, puede utilizar el programador YARN dinámico para aumentar el número de ejecutores cuando es necesario realizar más trabajo, así como para eliminar los ejecutores inactivos.

El controlador, por lo tanto, administra la tolerancia a errores de todo el proceso de ejecución. Una vez completado el trabajo por el controlador, la salida se puede escribir en un archivo, base de datos o simplemente en la consola.

Nota

Recuerde que el código en el propio programa Driver tiene que ser completamente serializable, incluidas todas las variables y objetos. La excepción a menudo vista no es una excepción serializable, que es el resultado de incluir variables globales desde fuera del bloque.

El proceso de controlador se encarga de todo el proceso mientras se supervisan y administran los recursos utilizados, como ejecutores, etapas y tareas, asegurándose de que todo funciona según lo planeado y recuperándose de errores como errores de tareas en nodos de ejecutor o nodos de ejecutor completos en su conjunto.