Almacenamiento de Archivos Descargados en un Entorno Spark con Yarn y Múltiples Clústeres

1. Introducción

Apache Spark se ha consolidado como un sistema de procesamiento distribuido fundamental para abordar cargas de trabajo de big data, proporcionando capacidades de computación rápida y generalizada.[1] En este ecosistema, Hadoop YARN (Yet Another Resource Negotiator) desempeña un papel crucial como plataforma de gestión de recursos para clústeres Hadoop, encargándose de la asignación de recursos a diversas aplicaciones, incluyendo Spark.[5]

En este contexto, surge la interrogante planteada por el usuario respecto a la posibilidad de almacenar un archivo descargado dentro de una aplicación Spark, que se ejecuta en un entorno Yarn con múltiples clústeres, directamente en el sistema de archivos local de los nodos de trabajo. Esta cuestión se contrapone a la práctica común de utilizar sistemas de archivos distribuidos como HDFS (Hadoop Distributed File System) para la persistencia de datos en tales entornos.

La práctica habitual en arquitecturas de big data que emplean Spark y Yarn es la utilización de sistemas de archivos distribuidos como HDFS debido a su capacidad para manejar grandes volúmenes de datos de manera escalable y tolerante a fallos. Sin embargo, la idea de guardar archivos localmente plantea interrogantes sobre la viabilidad técnica y las implicaciones prácticas en un entorno distribuido y multi-clúster.

Este informe tiene como objetivo principal responder de manera exhaustiva y autorizada a la pregunta del usuario, explorando la factibilidad técnica y las consecuencias de optar por el almacenamiento local en contraposición a HDFS en un entorno Spark on Yarn con múltiples clústeres. Se busca determinar si el almacenamiento local es una opción viable y recomendable, o si HDFS sigue siendo la alternativa preferida y, posiblemente, la única práctica. Para lograr este objetivo, el informe se estructurará abordando las capacidades de interacción de Spark con diversos sistemas de archivos, la gestión de recursos de Yarn en configuraciones multi-clúster, las diferencias fundamentales entre el almacenamiento local y distribuido en este contexto, los aspectos de configuración relevantes tanto en Spark como en Yarn, las implicaciones en el rendimiento, y las mejores prácticas para la gestión de archivos descargados en un entorno distribuido.

2. Entendiendo las Interacciones de Spark con el Sistema de Archivos

Spark está diseñado para interactuar con una amplia gama de sistemas de archivos a través de la interfaz Hadoop FileSystem.[1] Esta flexibilidad permite a Spark operar con sistemas de archivos locales, HDFS, almacenamiento en la nube como Amazon S3 y Azure Data Lake Storage, así como con otros sistemas compatibles con Hadoop.[1] Un ejemplo de la configuración para acceder a sistemas de archivos específicos se puede observar en el parámetro spark.hadoop.fs.cos.object-storage.endpoint.[14]

Para acceder al sistema de archivos local, Spark utiliza el esquema de URI file://.[15] Este esquema permite a Spark leer y, potencialmente, escribir en los discos locales de los nodos donde se ejecutan las tareas de Spark. Como se sugiere en el fragmento [16], la sintaxis "file:///ruta local" se puede emplear para guardar archivos en el sistema local.

Spark SQL proporciona funciones genéricas de carga y guardado que son compatibles con diversos formatos de archivo y sistemas de almacenamiento.[19] Por ejemplo, la función dataframe.write().csv("ruta") permite escribir un DataFrame en formato CSV, donde la "ruta" puede ser una ruta de archivo local (utilizando file://) o una ruta en HDFS. El fragmento [15] menciona explícitamente la posibilidad de guardar archivos tanto en una ruta local (especificando 'file://') como en HDFS.

El comportamiento predeterminado de Spark al escribir archivos a menudo implica la creación de un directorio que contiene múltiples archivos de partes (uno por cada partición) y un archivo _SUCCESS.[19] Para forzar la escritura en un único archivo, se pueden utilizar las funciones repartition(1) o coalesce(1), tal como se indica en [22] y.[23] Sin embargo, es importante notar que existen limitaciones en cuanto al control del nombre exacto del archivo de salida al utilizar estos métodos.

La arquitectura de Spark está intrínsecamente diseñada para el procesamiento distribuido, lo que se refleja en su capacidad para interactuar con sistemas de archivos locales. No obstante, su comportamiento predeterminado y sus optimizaciones están orientados hacia el almacenamiento distribuido. La razón subyacente radica en que Spark distribuye las operaciones a través de múltiples ejecutores, cada uno de los cuales podría estar funcionando en nodos diferentes. Por lo tanto, escribir en una ruta de archivo local desde estos ejecutores significaría que los archivos se crearían en máquinas distintas, lo que podría no ser el resultado deseado para un único archivo descargado. La salida predeterminada de múltiples archivos de partes es un reflejo de este modelo de procesamiento distribuido.

3. Gestión de Recursos de YARN y Acceso al Sistema de Archivos en Entornos Multi-Clúster

La arquitectura de YARN se compone de varios componentes clave, incluyendo el ResourceManager, el NodeManager y el ApplicationMaster.[5] El ResourceManager es responsable de la asignación de los recursos del clúster a las aplicaciones. El NodeManager, por su parte, gestiona la ejecución de contenedores en los nodos individuales. El ApplicationMaster se encarga de la gestión del ciclo de vida de una aplicación específica dentro del clúster.

Para la gestión de múltiples sub-clústeres, YARN ofrece el concepto de federación.[26] La federación permite tratar varios clústeres YARN más pequeños como si fueran un único clúster de gran tamaño. En esta arquitectura, el Router y el AMRMProxy desempeñan roles importantes en la gestión de las solicitudes de las aplicaciones y la asignación de recursos a través de los diferentes sub-clústeres.

YARN también gestiona el acceso al sistema de archivos para las aplicaciones que se ejecutan dentro de contenedores.[27] Los contenedores tienen acceso al sistema de archivos local del nodo en el que se están ejecutando. Los parámetros de configuración como yarn.nodemanager.local-dirs definen los directorios locales a los que los contenedores pueden acceder.[14] Además, YARN se encarga de la distribución de los archivos relacionados con la aplicación, como los archivos JAR, a través del clúster.

En un entorno multi-clúster Yarn gestionado por federación, cada ejecutor de Spark se ejecuta dentro de un contenedor en un nodo específico dentro de uno de los sub-clústeres. Estos contenedores tienen acceso al sistema de archivos local de ese nodo, lo que sugiere que la escritura local es técnicamente posible desde la perspectiva de la contenedorización de YARN. La función fundamental de YARN es la gestión de recursos a nivel de nodo. Los contenedores, la unidad de ejecución, operan dentro de los límites de los recursos de un único nodo, incluyendo su sistema de archivos. La federación abstrae la complejidad de múltiples clústeres, pero en última instancia se basa en NodeManagers individuales para gestionar los recursos locales.

4. Factibilidad Técnica de Guardar en Sistemas de Archivos Locales en Spark on YARN

Desde una perspectiva técnica, Spark sí puede configurarse para escribir directamente en los discos locales de los nodos de trabajo cuando se ejecuta en YARN en una configuración multi-clúster. El uso del esquema de URI file:// en las operaciones de escritura de Spark debería permitir la escritura en el sistema de archivos local del nodo donde se está ejecutando el ejecutor que realiza la escritura.[15] Sin embargo, como se muestra en el fragmento [18], un intento de utilizar saveAsTextFile(s"file://ruta/al/archivo") en modo yarn-cluster puede provocar una IOException.

Existen varios desafíos significativos asociados con este enfoque en un entorno distribuido. En primer lugar, la **localidad de los datos** es un factor crítico. Un archivo guardado localmente por un ejecutor no será directamente accesible por otros ejecutores que se ejecuten en nodos diferentes o en clústeres distintos, a menos que se implementen mecanismos explícitos de transferencia de datos. En segundo lugar, si el objetivo es guardar un único archivo descargado, la escritura desde múltiples ejecutores a sus respectivos sistemas de archivos locales resultaría en un archivo fragmentado a través del clúster. En tercer lugar, el nodo del **driver de Spark**, especialmente en el modo de despliegue cluster, se ejecuta dentro del clúster y podría no tener acceso directo a los sistemas de archivos locales de todos los nodos de trabajo.[18] Finalmente, la **complejidad de un entorno multi-clúster** gestionado por federación añade una capa adicional de dificultad. En este escenario, predecir qué nodo o nodos de trabajo específicos ejecutarán la tarea de guardar el archivo se vuelve aún más complejo, lo que dificulta la gestión o la localización del archivo guardado.

Si bien es técnicamente posible que los ejecutores individuales escriban en sus sistemas de archivos locales, este enfoque presenta desafíos significativos relacionados con la consistencia, la accesibilidad y la gestión de los datos en un entorno distribuido y multi-clúster. La fortaleza de Spark reside en el procesamiento y la gestión de datos distribuidos. Guardar un único archivo lógico a través de múltiples sistemas de archivos locales independientes socava este paradigma. La falta de una visión centralizada o de un mecanismo para coordinar estas escrituras locales lo hace poco fiable para la mayoría de los casos de uso que implican un único archivo descargado.

5. Configuración de Spark y YARN para el Acceso de Escritura al Sistema de Archivos Local

Existen parámetros de configuración en Spark que podrían influir en la escritura de archivos locales. Uno de ellos es spark.local.dir, que define el directorio para archivos temporales, shuffles y datos intermedios en cada nodo ejecutor.[14] Es importante destacar que este parámetro se utiliza principalmente para el uso interno de Spark y no para guardar la salida final de los datos. Otro parámetro relevante es spark.yarn.app.container.log.dir, que especifica la ubicación de los registros de la aplicación dentro del contenedor YARN.[27] El fragmento [14] muestra cómo se puede sobrescribir el valor de spark.local.dir al enviar aplicaciones Spark.

En cuanto a YARN, si bien gestiona los directorios locales a los que los contenedores pueden acceder a través de yarn.nodemanager.local-dirs, no existe una configuración

específica para habilitar o deshabilitar ampliamente el acceso de escritura a todas las rutas del sistema de archivos local. Los permisos en el sistema de archivos local de cada nodo son los que principalmente rigen esta capacidad. El fragmento [33] analiza el container-executor en YARN, que ejecuta los contenedores como el usuario que envió la aplicación y establece permisos restringidos en los archivos y directorios locales utilizados por los contenedores. Esto sugiere un enfoque de seguridad que podría no permitir fácilmente la escritura en rutas locales arbitrarias sin los permisos adecuados.

Si una aplicación Spark intenta escribir en una ruta local utilizando file:///, el directorio debe existir en el nodo de trabajo donde se está ejecutando el ejecutor, y el usuario que ejecuta el ejecutor debe tener permisos de escritura.[18] La excepción IOException: Mkdirs failed to create file mencionada en [18] pone de manifiesto este posible problema.

Configurar Spark para escribir en directorios locales específicos es posible, pero requiere una gestión cuidadosa de la existencia y los permisos de los directorios en todos los nodos de trabajo involucrados. El modelo de seguridad de YARN, si bien proporciona aislamiento, añade otra capa de complejidad para garantizar que la aplicación Spark tenga los derechos necesarios. La naturaleza distribuida de Spark on Yarn implica que las configuraciones que afectan el acceso al sistema de archivos deben ser consistentes en todos los nodos de trabajo. Simplemente establecer una ruta local en el código de Spark no garantiza su existencia o capacidad de escritura en cada nodo que pueda ejecutar la tarea.

6. Implicaciones y Limitaciones de Rendimiento del Almacenamiento Local

Al comparar el rendimiento de la escritura en discos locales versus HDFS en un trabajo Spark distribuido, se observan varios factores relevantes.[34] Para conjuntos de datos pequeños o cálculos intermedios, la escritura en el disco local podría ofrecer una latencia menor debido a la evitación de la sobrecarga de red asociada con HDFS. Sin embargo, este beneficio se limita al alcance del disco local de un único ejecutor. HDFS, por otro lado, está diseñado para un alto rendimiento y escritura paralela, especialmente para grandes conjuntos de datos distribuidos en múltiples nodos. El fragmento [34] presenta un escenario que podría parecer contradictorio, donde la lectura desde el sistema de archivos local fue más rápida que la lectura desde HDFS, posiblemente debido a la localidad de los datos dentro de un clúster pequeño. No obstante, este resultado podría no ser aplicable a las operaciones de escritura o a configuraciones multi-clúster más grandes y complejas.

Varios factores influyen en el rendimiento, como la E/S de red, la E/S de disco y la serialización de datos.[36] La escritura en HDFS implica inherentemente E/S de red, ya que los bloques de datos se distribuyen y replican a través del clúster. El rendimiento de la E/S del

disco local depende del tipo y la velocidad del almacenamiento local en cada nodo de trabajo. La sobrecarga de serialización puede afectar el rendimiento general tanto de las escrituras locales como de las escrituras en HDFS, dependiendo del formato de archivo utilizado.

En términos de tolerancia a fallos, redundancia de datos y escalabilidad en un entorno multi-clúster, el almacenamiento local presenta limitaciones significativas.[3] El almacenamiento local en un nodo de trabajo típicamente carece de la tolerancia a fallos y la redundancia incorporadas en HDFS, donde los datos se replican en múltiples nodos. Si un nodo de trabajo con un archivo guardado localmente falla, el archivo podría perderse o volverse inaccesible. La escalabilidad también es una limitación importante, ya que la gestión de datos dispersos en los discos locales de numerosos nodos de trabajo en múltiples clústeres se vuelve operativamente compleja y no proporciona una solución de almacenamiento unificada y escalable como HDFS.

Si bien las escrituras en el disco local podrían parecer más rápidas en ciertos escenarios limitados, la falta de tolerancia a fallos, redundancia y escalabilidad inherentes al almacenamiento local lo convierten en una mala elección para la gestión de datos importantes en un entorno Spark distribuido y multi-clúster. HDFS, a pesar de la posible sobrecarga de red, ofrece la robustez necesaria para tales escenarios. La razón fundamental es que los principales beneficios de utilizar un sistema distribuido como Spark on Yarn son la escalabilidad y la resiliencia. Confiar en los sistemas de archivos locales niega estas ventajas para el almacenamiento del archivo descargado. HDFS está específicamente diseñado para manejar datos de manera distribuida y tolerante a fallos, lo que lo convierte en la opción más apropiada para los datos gestionados dentro del ecosistema Spark/Yarn.

La siguiente tabla resume las diferencias clave entre el uso del sistema de archivos local y HDFS en un entorno Spark on Yarn:

Característica	**Sistema de Archivos Local (en Nodos de Trabajo)**	**HDFS (Hadoop Distributed File System)**
Escalabilidad	Limitada a la capacidad de	Altamente escalable, abarca

	nodos individuales.	miles de nodos.
Tolerancia a Fallos	Típicamente ninguna; pérdida de datos si falla un nodo.	Alta; los datos se replican en múltiples nodos.
Redundancia de Datos	Ninguna; los datos existen solo en el nodo donde se guardaron.	Alta; se mantienen múltiples copias de los datos.
Accesibilidad	Solo accesible directamente por procesos en el mismo nodo.	Accesible por cualquier nodo en el clúster Spark/Yarn.
Rendimiento	Potencialmente menor latencia para el acceso local por el ejecutor que guarda.	Mayor latencia para el acceso individual debido a la sobrecarga de red, pero alto rendimiento para lecturas/escrituras distribuidas.
Gestión	Requiere la gestión del almacenamiento en cada nodo individual.	Gestión centralizada del almacenamiento en todo el clúster.
Casos de Uso (Spark)	Limitados; principalmente para datos temporales, locales al nodo o desarrollo.	Almacenamiento primario para datos de entrada y salida en aplicaciones Spark distribuidas.

7. El Rol del Driver de Spark y los Modos de Despliegue

El modo en que el driver de Spark interactúa con los ejecutores varía según el modo de despliegue utilizado (cliente vs. clúster).[18] En el modo cliente, el driver se ejecuta en la máquina cliente que envía la aplicación Spark y se comunica directamente con los ejecutores que se ejecutan en el clúster YARN. En contraste, en el modo clúster, el driver se ejecuta dentro de un proceso ApplicationMaster dentro del propio clúster YARN, y el cliente que envía la aplicación puede desconectarse después de la iniciación.

La ubicación del driver tiene un impacto significativo en su capacidad para interactuar

directamente con los sistemas de archivos de los nodos de trabajo. En el modo clúster, el driver típicamente no se encuentra en el mismo nodo que la mayoría de los ejecutores y no tiene acceso directo a sus sistemas de archivos locales. Intentar escribir en una ruta file:// desde el driver en este modo probablemente resultaría en que el archivo se guarde en el nodo donde se está ejecutando el ApplicationMaster (y por lo tanto, el driver), y no en los nodos de trabajo que realizan el procesamiento de datos. En el modo cliente, el driver podría tener un acceso de red más directo a los nodos de trabajo, pero aún así no tendría inherentemente permisos de escritura o una visión compartida de sus sistemas de archivos locales.

Si se necesitara la escritura directa desde el driver (aunque generalmente se desaconseja para datos distribuidos), se podrían considerar soluciones alternativas. Una posibilidad sería utilizar collect() para llevar los datos al driver (si son lo suficientemente pequeños) y luego utilizar operaciones estándar de E/S de archivos para guardarlos en el sistema de archivos local del driver.[16] Sin embargo, esto anula el propósito del procesamiento distribuido y puede generar problemas de escalabilidad. El fragmento [47] menciona que cambiar el modo de despliegue de clúster a cliente permitió realizar escrituras de archivos correctamente en la máquina donde se ejecutó el driver.

El modo de despliegue influye considerablemente en dónde se ejecuta el driver de Spark y, por lo tanto, en su capacidad para interactuar con los sistemas de archivos locales de los nodos de trabajo. En el modo clúster, que es más común para las implementaciones en producción, la escritura directa en los discos locales de los nodos de trabajo desde el driver no es un enfoque directo ni recomendado. La arquitectura de Spark on Yarn en modo clúster está diseñada para que el driver coordine tareas distribuidas, no para que actúe como un punto central para la E/S de datos al almacenamiento local de los nodos de trabajo. Esto crearía un cuello de botella y socavaría la naturaleza distribuida del sistema.

La siguiente tabla resume la ubicación del driver en diferentes modos de despliegue y su interacción con los sistemas de archivos locales de los nodos de trabajo:

Modo de Despliegue	**Ubicación del Driver**	**Interacción con Sistemas de Archivos Locales de Nodos**

		de Trabajo
Cliente	En la máquina donde se envía la aplicación.	Interacción directa limitada; requeriría acceso de red y permisos adecuados. La escritura en file:// típicamente guarda en el sistema de archivos local de la máquina cliente.
Clúster	Dentro de un ApplicationMaster en el clúster YARN.	Sin acceso directo a los sistemas de archivos locales de los nodos de trabajo en la configuración típica. La escritura en file:// guarda en el nodo donde se ejecuta el ApplicationMaster.

8. Casos de Uso y Ejemplos de Almacenamiento Local (y por qué podrían ser limitados)

Existen casos de uso potenciales, aunque limitados, donde podría considerarse el almacenamiento en sistemas de archivos locales. Esto podría incluir el almacenamiento de archivos intermedios pequeños y temporales que solo necesita un ejecutor específico para un cálculo local posterior. Sin embargo, incluso para esto, típicamente se utiliza spark.local.dir. En escenarios de desarrollo o pruebas donde Spark se ejecuta en modo local (no en un clúster con Yarn), la escritura en el sistema de archivos local es común.[16] El fragmento [16] menciona que el código Scala proporcionado funciona bien cuando se ejecuta en modo local. También podrían existir casos extremos donde un algoritmo no Hadoop que se ejecuta en el mismo nodo necesita acceder directamente a la salida de una tarea Spark desde el disco local.[49] Sin embargo, esto introduce un acoplamiento estrecho y reduce la portabilidad de la aplicación Spark.

Se pueden encontrar ejemplos de intentos de guardar localmente en una configuración de clúster Yarn, que a menudo resaltan los posibles resultados. El ejemplo en [18] utilizando df.coalesce(1).rdd.saveAsTextFile(s"file://ruta/al/archivo") en modo yarn-cluster resultó en una IOException, lo que indica problemas de permisos o de directorio. El fragmento [18] muestra un intento similar y el rastreo de la pila resultante, enfatizando aún más los desafíos.

Estos casos de uso son limitados y generalmente no se recomiendan para entornos de producción, especialmente en configuraciones multi-clúster. La falta de compartición de datos, tolerancia a fallos y gestión centralizada hacen que el almacenamiento local sea inadecuado para la mayoría de las tareas de procesamiento de datos distribuidos.

9. Mejores Prácticas y Alternativas: Aprovechando HDFS para la Gestión de Archivos Distribuidos

Es fundamental enfatizar las ventajas de utilizar HDFS para la gestión de archivos en un entorno Spark distribuido.[3] HDFS ofrece escalabilidad para almacenar petabytes de datos en miles de nodos, tolerancia a fallos a través de la replicación de datos, localidad de los datos que permite a Spark procesar datos en los nodos donde residen minimizando el tráfico de red, e integración con el ecosistema Hadoop y otras herramientas de procesamiento de datos.

Las mejores prácticas para escribir y gestionar archivos descargados en HDFS incluyen el uso de las funciones de escritura incorporadas de Spark con rutas HDFS (por ejemplo, dataframe.write().csv("hdfs://namenode:puerto/ruta")). También se recomienda considerar formatos de archivo como Parquet y ORC para un almacenamiento y recuperación eficientes de datos estructurados.[13] Estos formatos ofrecen evolución de esquemas, compresión y almacenamiento columnar, que son beneficiosos para el análisis de big data. Es importante gestionar los permisos de archivo y el control de acceso a través de los mecanismos de HDFS y organizar los archivos en directorios lógicos dentro de HDFS para una mejor gestión.

Si la descarga ocurre en un único nodo (por ejemplo, el driver en modo cliente), el archivo debe moverse a HDFS antes de ser procesado por la aplicación Spark distribuida. Esto se puede hacer utilizando el comando hdfs dfs -put de Hadoop o la API del sistema de archivos de Spark. Para las descargas que ocurren de forma distribuida (por ejemplo, cada ejecutor descarga una parte de los datos), los datos deben escribirse directamente en HDFS como parte del trabajo de Spark.

La práctica recomendada para la gestión de archivos descargados en un entorno Spark on Yarn multi-clúster es aprovechar HDFS. Sus principios de diseño de escalabilidad, tolerancia a fallos y localidad de los datos se alinean perfectamente con la naturaleza distribuida de Spark y Yarn, asegurando un procesamiento de datos eficiente y fiable. HDFS proporciona una capa de almacenamiento centralizada y gestionada accesible a todos los nodos del clúster Spark/Yarn (o clústeres federados). Esto elimina los desafíos asociados con la gestión de datos a través de los sistemas de archivos locales de los nodos de trabajo

individuales y garantiza la consistencia y la resiliencia de los datos.

10. Conclusión y Recomendaciones

En resumen, si bien técnicamente es posible que los ejecutores individuales escriban en sus sistemas de archivos locales en un entorno Spark on Yarn con múltiples clústeres, este enfoque presenta desafíos significativos en términos de gestión de datos, accesibilidad, tolerancia a fallos y escalabilidad. Además, existen limitaciones relacionadas con el acceso del driver de Spark a los sistemas de archivos locales de los nodos de trabajo, especialmente en el modo clúster.

Por lo tanto, la respuesta clara a la pregunta del usuario es que **no se recomienda** guardar archivos descargados en el sistema de archivos local de los nodos de trabajo en un entorno Spark on Yarn multi-clúster, y no es la forma prevista de manejar los datos dentro de esta arquitectura.

El enfoque preferido es utilizar HDFS para la gestión de datos distribuidos en tales entornos debido a sus ventajas inherentes en escalabilidad, tolerancia a fallos y localidad de los datos.

Las recomendaciones para la gestión de archivos descargados son:

- Descargar los archivos directamente en HDFS dentro de la aplicación Spark o moverlos a HDFS si se descargan fuera del contexto de Spark.
- Utilizar formatos de archivo eficientes como Parquet u ORC para almacenar datos estructurados en HDFS.
- Gestionar la organización y los permisos de los archivos dentro de HDFS.

En conclusión, adherirse al paradigma de gestión de datos distribuidos que ofrece HDFS es crucial para construir aplicaciones de big data robustas y escalables con Spark on Yarn a través de múltiples clústeres.

Obras citadas

1. What is Spark? - Introduction to Apache Spark and Analytics - AWS, fecha de acceso: marzo 31, 2025, https://aws.amazon.com/what-is/apache-spark/
2. Apache Spark FAQ, fecha de acceso: marzo 31, 2025, https://spark.apache.org/faq.html
3. Hadoop vs Spark - Difference Between Apache Frameworks - AWS, fecha de acceso: marzo 31, 2025, https://aws.amazon.com/compare/the-difference-between-hadoop-vs-spark/
4. Apache Spark vs. Hadoop: Key Differences and Use Cases | OpenLogic by Perforce,

fecha de acceso: marzo 31, 2025, https://www.openlogic.com/blog/spark-vs-hadoop
5. YARN(Yet Another Resource Negotiator) Architecture | by AS | Medium, fecha de acceso: marzo 31, 2025, https://medium.com/@jwbtmf/yarn-yet-another-resource-manager-architecture-21d117364b26
6. Understanding YARN architecture and features - Cloudera Documentation, fecha de acceso: marzo 31, 2025, http://docs.cloudera.com.s3-website-us-east-1.amazonaws.com/HDPDocuments/HDP3/HDP-3.0.1/data-operating-system/content/apache_yarn.html
7. Hadoop YARN Architecture | GeeksforGeeks, fecha de acceso: marzo 31, 2025, https://www.geeksforgeeks.org/hadoop-yarn-architecture/
8. Hadoop cluster resource management – YARN | HPC, Big data & information security, fecha de acceso: marzo 31, 2025, https://hrouhani.org/hadoop-cluster-resource-management-yarn/
9. Understanding basics of HDFS and YARN - Cloudera Community - 248860, fecha de acceso: marzo 31, 2025, https://community.cloudera.com/t5/Community-Articles/Understanding-basics-of-HDFS-and-YARN/ta-p/248860
10. Yarn Cluster Setup - Datagaps Help Documentation, fecha de acceso: marzo 31, 2025, https://help.datagaps.com/articles/v2024-4-1-0/yarn-cluster
11. spark.apache.org, fecha de acceso: marzo 31, 2025, https://spark.apache.org/docs/latest/rdd-programming-guide.html#:~:text=Spark%20can%20create%20distributed%20datasets,using%20SparkContext%20's%20textFile%20method.
12. Apache Spark - Wikipedia, fecha de acceso: marzo 31, 2025, https://en.wikipedia.org/wiki/Apache_Spark
13. Explain Types of Data file formats in Big Data through Apache spark | by MultiTech | Medium, fecha de acceso: marzo 31, 2025, https://informationit27.medium.com/explain-types-of-data-file-formats-in-big-data-through-apache-spark-669f812c75e4
14. spark.local.dir configuration parameter - IBM, fecha de acceso: marzo 31, 2025, https://www.ibm.com/docs/en/watsonx/w-and-w/1.1.x?topic=jobs-sparklocaldir-configuration-parameter
15. Save DataFrame as CSV File in Spark - Kontext, fecha de acceso: marzo 31, 2025, https://kontext.tech/article/357/save-dataframe-as-csv-file-in-spark
16. How to save Spark RDD to local filesystem - scala - Stack Overflow, fecha de acceso: marzo 31, 2025, https://stackoverflow.com/questions/40226326/how-to-save-spark-rdd-to-local-filesystem
17. Loading and Saving Your Data | Spark Tutorial - Intellipaat, fecha de acceso: marzo 31, 2025, https://intellipaat.com/blog/tutorial/spark-tutorial/loading-and-saving-your-data/
18. Writing files to local system with Spark in Cluster mode - Stack Overflow, fecha de acceso: marzo 31, 2025, https://stackoverflow.com/questions/40786093/writing-files-to-local-system-with-spark-in-cluster-mode
19. CSV Files - Spark 3.5.4 Documentation, fecha de acceso: marzo 31, 2025, https://spark.apache.org/docs/3.5.4/sql-data-sources-csv.html
20. Generic Load/Save Functions - Spark 3.5.4 Documentation, fecha de acceso: marzo 31,

2025, https://spark.apache.org/docs/3.5.4/sql-data-sources-load-save-functions.html
21. Data Sources - Spark 3.5.3 Documentation - Apache Spark, fecha de acceso: marzo 31, 2025, https://spark.apache.org/docs/3.5.3/sql-data-sources.html
22. How to Save PySpark Dataframe to a Single Output File | Engineering for Data Science, fecha de acceso: marzo 31, 2025, https://engineeringfordatascience.com/posts/how_to_save_pyspark_dataframe_to_single_output_file/
23. Writing out single files with Spark (CSV or Parquet) - MungingData, fecha de acceso: marzo 31, 2025, https://www.mungingdata.com/apache-spark/output-one-file-csv-parquet/
24. Spark Creating Directories Instead of Files When Saving Parquet - Reddit, fecha de acceso: marzo 31, 2025, https://www.reddit.com/r/dataengineering/comments/1ayehhc/spark_creating_directories_instead_of_files_when/
25. Access Apache Hadoop YARN application logs - Azure HDInsight - Learn Microsoft, fecha de acceso: marzo 31, 2025, https://learn.microsoft.com/en-us/azure/hdinsight/hdinsight-hadoop-access-yarn-app-logs-linux
26. YARN Federation - Apache Hadoop 3.4.1 – Hadoop, fecha de acceso: marzo 31, 2025, https://hadoop.apache.org/docs/stable/hadoop-yarn/hadoop-yarn-site/Federation.html
27. Running Spark on YARN - Spark 3.5.5 Documentation, fecha de acceso: marzo 31, 2025, https://spark.apache.org/docs/latest/running-on-yarn.html
28. docs/running-on-yarn.md · ccc6dc0f4b62837c73fca0e3c8b9c14be798b062 · Embedded Systems Public / spark · GitLab, fecha de acceso: marzo 31, 2025, https://git.uni-due.de/embedded-systems-public/spark/-/blob/ccc6dc0f4b62837c73fca0e3c8b9c14be798b062/docs/running-on-yarn.md
29. Can you read/write directly to hard disk from a spark job? - Stack Overflow, fecha de acceso: marzo 31, 2025, https://stackoverflow.com/questions/46793208/can-you-read-write-directly-to-hard-disk-from-a-spark-job
30. Apache Spark Cluster Deploy Modes(Part-03) | by Kiran vutukuri - Medium, fecha de acceso: marzo 31, 2025, https://medium.com/@kiranvutukuri/apache-spark-cluster-deploy-modes-part-03-ffce83c8d96a
31. Cluster Mode Overview - Spark 3.5.5 Documentation, fecha de acceso: marzo 31, 2025, https://spark.apache.org/docs/latest/cluster-overview.html
32. Creating the Apache Spark working directories - IBM, fecha de acceso: marzo 31, 2025, https://www.ibm.com/docs/en/izoda/1.1.0?topic=structure-creating-apache-spark-working-directories
33. Configuring Apache Hadoop YARN Security - Cloudera Documentation, fecha de acceso: marzo 31, 2025, https://docs.cloudera.com/runtime/7.3.1/yarn-security/yarn-security.pdf
34. Spark local vs hdfs permormance - Stack Overflow, fecha de acceso: marzo 31, 2025, https://stackoverflow.com/questions/34763437/spark-local-vs-hdfs-permormance
35. Understanding Apache Spark Memory Management on yarn: A Deep Dive with Real Cluster Metrics | by Kumar | Medium, fecha de acceso: marzo 31, 2025,

https://medium.com/@InsightfulEnginner/understanding-apache-spark-memory-management-on-yarn-a-deep-dive-with-real-cluster-metrics-a6ce2783a502
36. Apache Spark Performance Benchmarks show Kubernetes has caught up with YARN, fecha de acceso: marzo 31, 2025, https://spot.io/blog/apache-spark-performance-benchmarks-show-kubernetes-has-caught-up-with-yarn/
37. Hadoop vs. Spark: 5 Key Differences and Using Them Together - Granulate, fecha de acceso: marzo 31, 2025, https://granulate.io/blog/hadoop-vs-spark-5-key-differences-and-using-them-together/
38. Apache Spark vs Hadoop - A detailed technical comparison - Macrometa, fecha de acceso: marzo 31, 2025, https://www.macrometa.com/event-stream-processing/apache-spark-vs-hadoop
39. Hadoop vs Spark: Key Differences in Big Data Analytics - Veritis, fecha de acceso: marzo 31, 2025, https://www.veritis.com/blog/hadoop-vs-spark-all-you-need-to-know-about-big-data-analytics/
40. Hadoop vs Spark: A Comparative Study - Cloudlytics, fecha de acceso: marzo 31, 2025, https://cloudlytics.com/hadoop-vs-spark-a-comparative-study/
41. Tuning - Spark 3.5.5 Documentation - Apache Spark, fecha de acceso: marzo 31, 2025, https://spark.apache.org/docs/latest/tuning.html
42. How to Optimize Your Apache Spark Jobs: Top 10 Approaches and Best Practices for Performance Tuning | by Manoj Kumar Das | Medium, fecha de acceso: marzo 31, 2025, https://medium.com/@manoj.kdas37/how-to-optimize-your-apache-spark-jobs-top-10-approaches-and-best-practices-for-performance-tuning-4630ae864f52
43. Explore best practices for Spark performance optimization - IBM Developer, fecha de acceso: marzo 31, 2025, https://developer.ibm.com/blogs/spark-performance-optimization-guidelines/
44. Hadoop vs. Spark: What's the Difference? - IBM, fecha de acceso: marzo 31, 2025, https://www.ibm.com/think/insights/hadoop-vs-spark
45. What is the advantage of using spark with HDFS as file storage system and YARN as resource manager? - Stack Overflow, fecha de acceso: marzo 31, 2025, https://stackoverflow.com/questions/54380931/what-is-the-advantage-of-using-spark-with-hdfs-as-file-storage-system-and-yarn-a
46. Difference between using hdfs:// and yarn in spark - Stack Overflow, fecha de acceso: marzo 31, 2025, https://stackoverflow.com/questions/36258480/difference-between-using-hdfs-and-yarn-in-spark
47. Spark (Scala) Writing (and reading) to local file system from driver - Stack Overflow, fecha de acceso: marzo 31, 2025, https://stackoverflow.com/questions/38122039/spark-scala-writing-and-reading-to-local-file-system-from-driver
48. Running Spark on Top of a Hadoop YARN Cluster – utho Docs, fecha de acceso: marzo 31, 2025, https://utho.com/docs/database/hadoop/install-configure-run-spark-on-hadoop-yarn-cluster/
49. Hadoop Yarn write to local file system - java - Stack Overflow, fecha de acceso: marzo 31, 2025, https://stackoverflow.com/questions/45785811/hadoop-yarn-write-to-local-file-system

50. File Formats and Data Sources in PySpark | by Dhanashri Saner | Medium, fecha de acceso: marzo 31, 2025, https://medium.com/@dhanashrisaner.30/file-formats-and-data-sources-in-pyspark-14de6a029b74