ARQUITECTURA DE HADOOP

Hadoop Versión 1

La idea básica es mover la computación cerca de los datos, con el objetivo de acceder a ellos eficientemente y trabajar de forma óptima.

La estructura básica es Máster/Slave, existe un único JobTracker (que asigna tareas) en el cluster y un TaskTracker (que ejecuta tareas) en cada nodo, Figura 1 (el Job Tracker puede estar en el nodo que contiene el NameNode o no).

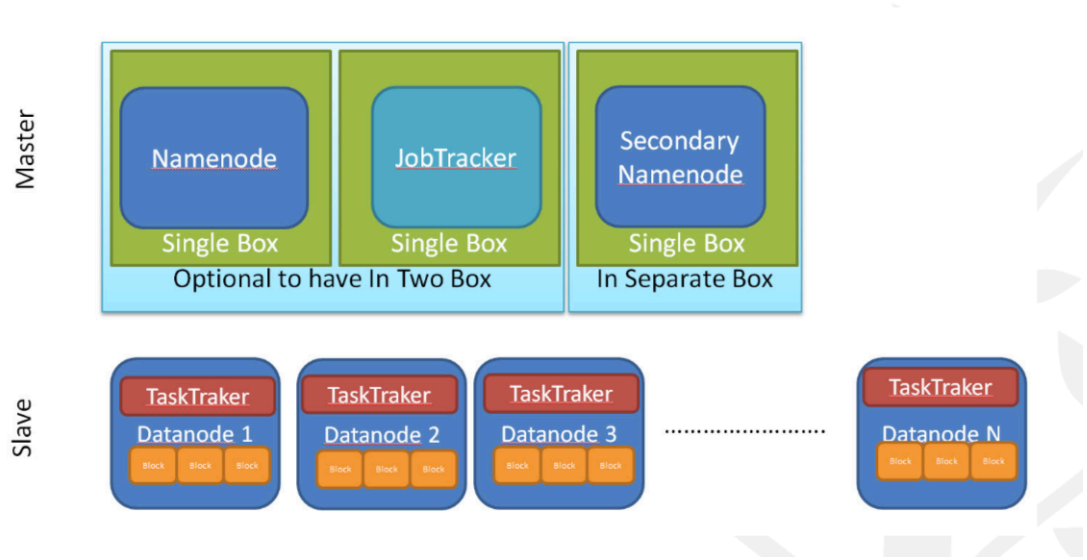

Figura 1. Arquitectura de la versión 1.0 de Hadoop

El flujo de trabajo es el siguiente, Figura 2: El JobTracker recibe una solicitud de trabajo por parte del cliente y se encarga de repartir y monitorizar el trabajo MapReduce en los TaskTrakers, por lo tanto, reparte el trabajo de forma balanceada por todos los nodos del cluster. Los TaskTrakers de los nodos son los encargados de ejecutar el trabajo y comunicarse con el JobTracker. Para ejecutar el trabajo debe ponerse en contacto con el Datanode para saber que bloques de datos debe procesar (los más cercanos posibles al nodo) para completar el trabajo.

Es fundamental monitorizar el trabajo de los TaskTrackers por parte del JobTracker por si existen fallos y otro nodo debe ejecutar lo que ha fallado o no se ha podido ejecutar en otro.

La asignación de recursos y la secuenciación de los trabajos se produce de forma interna dentro del framework.

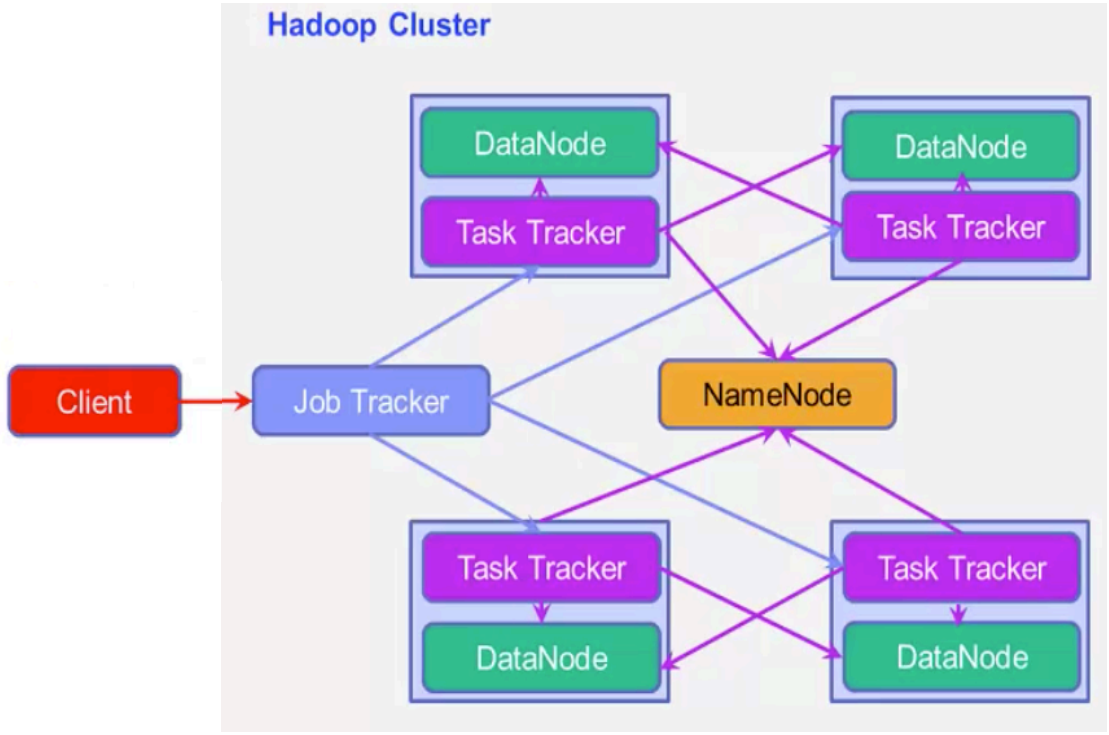

Figura 2. Flujo de trabajo en la arquitectura de la versión 1.0 de Hadoop

Hadoop Versión 2: YARN (Yet Another Resource Negociator) Version 1

Tal como está diseñada la arquitectura de la versión 1, únicamente se pueden ejecutar trabajos MapReduce y de uno en uno, pero Hadoop se puede usar para algo más que únicamente trabajos MapReduce (se podría usar para cualquier otro cómputo que necesite una distribución de datos en un cluster). Por tanto, la idea básica es sacar la asignación de recursos y la secuenciación de los trabajos fuera del propio framework, de manera que se pueda usar por varios trabajos a la vez e, incluso, para que pueda se usado por varios frameworks a la vez (Hadoop y Spark, por ejemplo).

Por otro lado, se debe producir una secuenciación de los trabajos más eficiente, ya que el hecho de balancear la carga de trabajo entre todos los nodos del cluster puede no ser lo más óptimo dependiendo del trabajo a realizar, es posible que no sea necesario llevar el trabajo a todos los nodos del cluster.

El ResourceManager sustituye al JobTracker pero se sitúa fuera del framework y el NodeManager sustituye a los TaskTrackers en cada nodo, figura 3, pero sus funciones son diferenes. El NodeManager es el responsable para cada nodo del contenedor o contenedores que tiene asignados, por lo que debe monitorizar el uso de sus recursos (CPU, memoria, red, etc.) y reportarlo al ResourceManager.

Al igual que en la versión de Hadoop 1 el nodo Master contiene el NameNode, que será consultado por el ResourceManager y los nodos Slave contendrán los DataNode.

Cuando se lanza una aplicación por parte del cliente, se contacta con el ResourceManager, paso 1 de la figura 3, que la asigna a uno de los nodos para que se ejecute dentro de un contenedor (que tiene asignados una serie de recursos como memoria, CPU, etc. dependiendo de cómo se haya configurado YARN), pasos 2a y 2b de la figura 3, si el trabajo necesita más recursos para poderse completar (varios mappers y reducers, varios mappers y un solo reducer o algo completamente diferente que no tiene nada que ver con un proceso MapReduce habitual), el nodo se pone en contacto de nuevo con el ResourceManager y se negocian los recursos necesarios para poder ejecutar el trabajo, paso 3 de la figura 3. Con YARN el ResourceManager sabe de la capacidad de trabajo de cada nodo mediante comunicación con el NodeManager que está corriendo dentro de cada nodo.

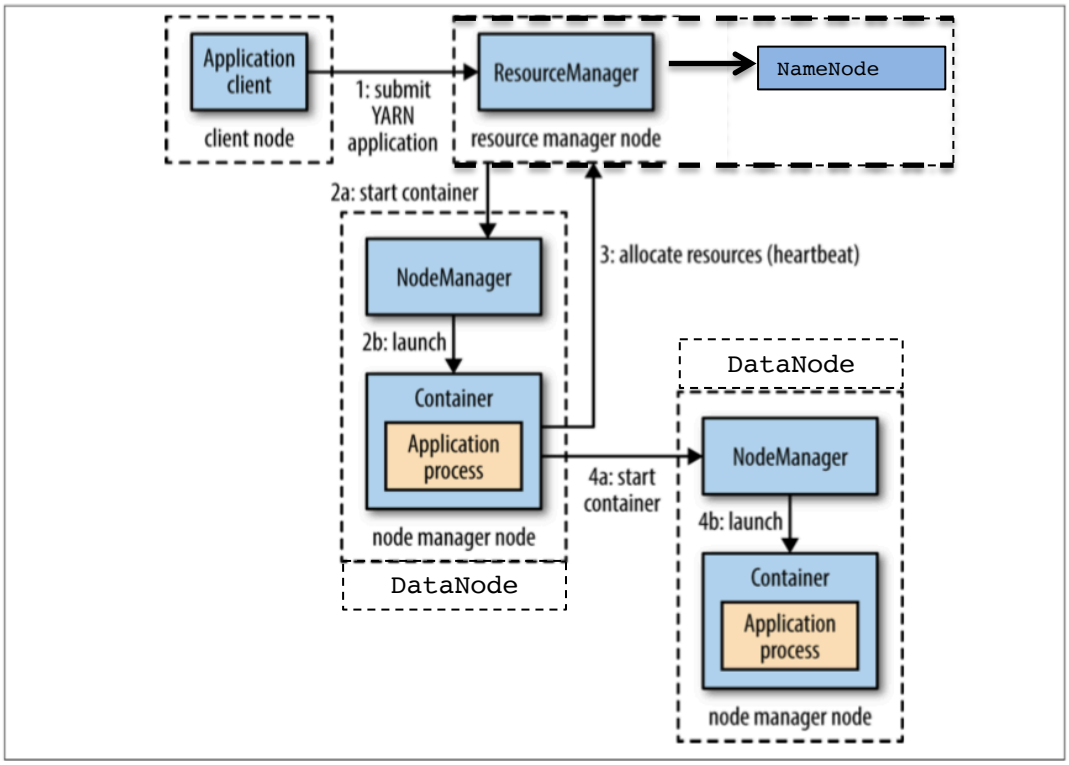

Figura 3. Arquitectura de la versión 2 (YARN) de Hadoop

Si son necesarios más recursos se asignan los contenedores necesarios para completar el trabajo de forma distribuida, paso 4 de la figura 3.

Para que quede más claro, la figura 4 muestra un ejemplo de flujo de trabajo de dos peticiones que se realizan a la vez al ResourceManager. Estas peticiones (Application Master –App. Mstr.- en la figura 4) se lanzan en dos nodos diferentes.

Como se puede ver cada trabajo puede necesitar una capacidad de trabajo diferente, es decir, la ejecución de un número de contenedores diferentes, y estos contenedores se ejecutarán en los nodos más óptimos para ello (con mayor capacidad, con mayor bloque de datos necesarios para el trabajo,…), El ResourceManager es la pieza de la arquitectura que controla qué nodos tienen capacidad para nuevos trabajos, esta información se la proporciona cada uno de los NodeManager de cada nodo.

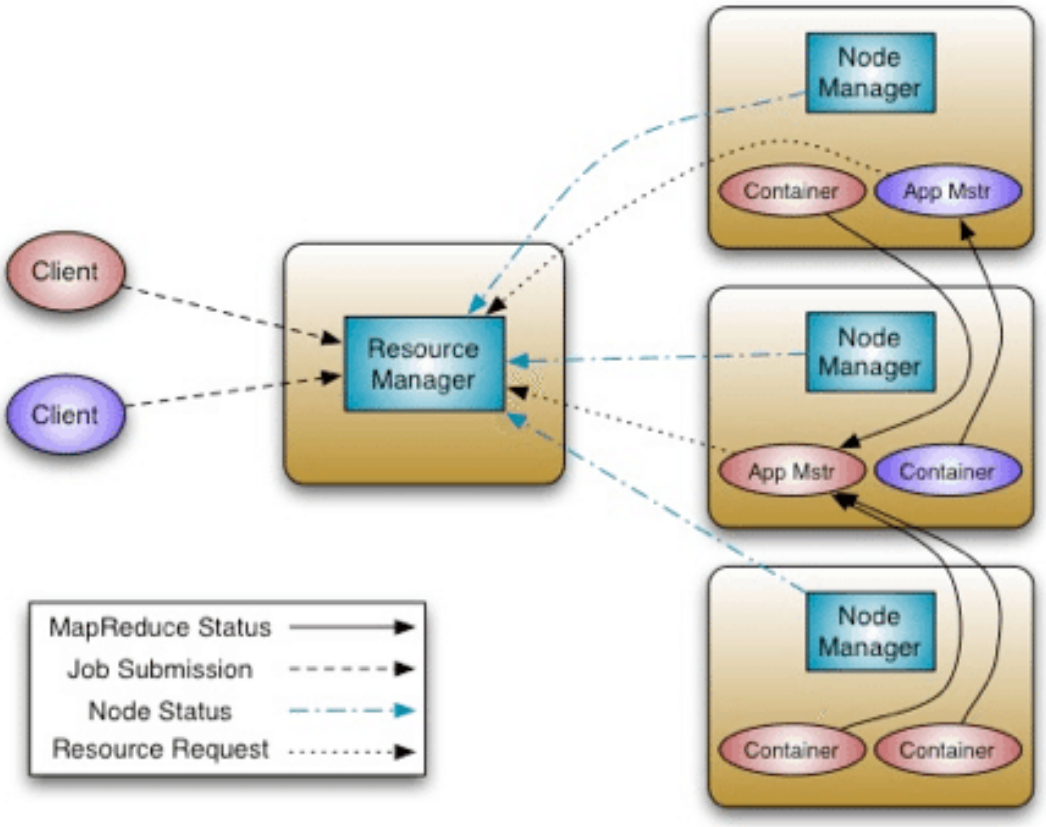

Figura 4. Ejemplo de flujo de trabajo para dos tareas en la arquitectura YARN de Hadoop

Hadoop Versión 3: YARN Versión 2

Las principales diferencias con la versión 2 de Hadoop son:

-Para HDFS:

La principal diferencia entre Hadoop V.2 y Hadoop V.3 es que la solución de Hadoop V.2 para tolerancia a fallos es proporcionada por la técnica de replicación donde cada bloque de información se copia para crear 2 réplicas. Esto significa que, en lugar de almacenar una pieza de información, Hadoop V.2 almacena tres veces más. Esto plantea el problema de espacio en disco.

En Hadoop 3, la tolerancia a fallos la proporciona la codificación de borrado (erasure coding). Este método permite recuperar un bloque de información utilizando un bloque de réplica y el llamado "bloque de paridad". Hadoop 3 crea un bloque de paridad de cada dos bloques de datos (utiliza un método parecido a la compresión de archivos). Esto requiere solo 1,5 veces más espacio en disco. El nivel de tolerancia a fallos en Hadoop 3 sigue siendo el mismo, pero se requiere menos espacio en disco para sus operaciones.

-Para la ejecución de trabajos MapReduce:

Hay varios cambios significativos que mejoran la usabilidad y la escalabilidad:

• YARN 2 soporta los flujos - grupos lógicos de la aplicación YARN.

• La separación entre los procesos de recopilación (escritura de datos) y los procesos de servicio (lectura de datos) mejora la escalabilidad.