**CLUSTER DE SERVIDORES**

Un cluster es un sistema basado en la unión de varios servidores que trabajarán de forma paralela como si de uno solo se tratara, es decir, tenemos un servidor formado por otros servidores. Podemos tener un cluster de 2, 10, 20 ó 100 servidores. Cuantos más servidores tengamos, más potencia de *hardware* estará disponible.

**Conceptos**

Un cluster es un conjunto de equipos independientes que realizan alguna tarea común en la que se comportan como un solo equipo.

* Ofrece uno o varios de los siguientes servicios:
  + Alto rendimiento
  + Alta disponibilidad
  + Balanceo de carga
  + Escalabilidad
* Surgieron como alternativa al crecimiento vertical de los ordenadores: Al aumentar los requisitos computacionales, en vez de sustituir el equipo por uno nuevo más potente, se añade una máquina más y se crea un cluster entre ambas.
* Con la generalización del uso de tecnologías de virtualización y cloud computing, los clústeres se están convirtiendo en componentes comunes en todo tipo de instalaciones.
* Los equipos dentro de un cluster reciben el nombre de nodos.

**Tipos de máquinas**

* **Máquinas físicas.** La solución tradicional. Salvo para situaciones específicas, está en progresivo desuso por ser la opción más costosa.
* **Máquinas virtuales** mucho más versátil que el uso de máquinas físicas ya que permite ajustar las características de los equipos a las necesidades reales. Se puede crear un cluster de máquinas virtuales en un solo equipo físico.
* **Instancias de cloud** Añade dos nuevos conceptos que se adaptan muy bien a la tecnología de clústeres: automatización y elasticidad Obviamente pueden utilizarse distintas combinaciones de las tres opciones anteriores.

**Tipos de Clusters: Cluster de Alto Rendimiento**

* High Performance Computing Cluster o HPC cluster.
* Se suelen utilizar equipos físicos.
* Imprescindibles para cálculos complejos que requieran gran capacidad computacional.
* En la mayor parte de los casos, los grandes ordenadores han sido sustituidos por clústeres de equipos ”convencionales” (arquitectura x86).
* Se utilizan principalmente en centros de investigación, ingeniería y otras actividades que requieran cálculos complejos.

Visita recomendada: [TOP500](http://www.top500.org)

**Tipos de Clusters: Cluster de Alta Disponibilidad**

* High availability (HA)
* Implementa tolerancia a fallos (failover) para garantizar el servicio en caso de que ocurra algún fallo.
* Utiliza nodos redundantes que sustituyen o complementan a los existentes.
* Se suelen implementar con balanceo de carga para repartir las peticiones entre varios nodos.
* Ampliamente utilizado en servicios de Internet .

**Balanceo de carga**

* Es una propiedad que permite repartir el trabajo entre varios nodos del cluster.
* Se utiliza algún algoritmo para el reparto de tarea entre los nodos del cluster: aleatorio, round robin, carga de los nodos, tiempo de respuesta, . . .
* Aunque no es habitual, se puede implementar un cluster de balanceo de carga que no incluya alta disponibilidad.

**Escalabilidad. Conceptos**

* Es una propiedad que permite modificar el tamaño del cluster en función de las necesidades.
* Es difícil y lento implementar un cluster escalable con equipos físicos.
* Con instancias de cloud se puede ir aún más allá, se habla algunas veces de elasticidad en lugar de escalabilidad.
* Tipos de escalado:
  + Escalado vertical: varían las características de los nodos (CPU, RAM, espacio de almacenamiento)
  + Escalado horizontal: varía el número de nodos del cluster.

**Escalabilidad. Elasticidad**

Objetivo Conseguir un cluster escalable de forma automática y dinámica que se adapte en cada momento a las necesidades

* Con el término escalabilidad se asume que el clúster siempre se crece, mientras que con la elasticidad se remarca la adaptación a cada momento (creciendo o decreciendo)
* Amazon EC2: Amazon **Elastic** Compute Cloud
* IaaS: Escalabilidad al alcance de cualquiera porque se paga por uso. Cuesta lo mismo utilizar 10 nodos durante una hora que un nodo durante 10 horas.
* Se utiliza principalmente escalado horizontal

**Orquestación o automatización**

* Configuration Management Software (CMS), también se utilizan los términos orchestration y automatization
* No se puede pensar en elasticidad sin un sistema totalmente automático de configuraci´on de los nodos
* Un sistema de orquestación o automatización permite crear máquinas virtuales, instancias de cloud, instalar paquetes, configurar servicios o utilizar plantillas de forma totalmente automática
* Los administradores de sistemas se centran cada vez m´as en crear recetas y plantillas para sus despliegues
* Puppet, chef, salt, ansible, . . .

| **Mascotas (Escalado vertical)** | **Ganado (Escalado Horizontal)** |
| --- | --- |
| * Tienen nombre * Son únicas * Se las cuida y mantiene | * Número en vez de nombre * Indistinguibles entre sí * Cuando no sirven, se sustituyen |
| Máquinas físicas o virtuales | Clusters e instancias de cloud |

“Future application architectures should use Cattle but Pets with strong configuration management are viable and still needed”. Tim Bell

**Almacenamiento. Conceptos**

* Los sistemas de ficheros tradicionales solo pueden montarse en un equipo.
* Es muy común compartir ficheros en red en el esquema conocido como NAS (Network-Attached Storage), donde una aplicación permite el acceso a ficheros a distintos clientes.
* NAS utiliza los conocidos protocolos NFS, CIFS, FTP, HTTP, . . .
* En muchos casos, los clusters requieren algún sistema de almacenamiento compartido con más propiedades (acceso a nivel de bloque, concurrencia, tolerancia a fallos o mayor escalabilidad, por ejemplo).

**Almacenamiento. SAN (Storage Area Network)**

* Una red de almacenamiento (SAN) es una red dedicada que ofrece almacenamiento a nivel de bloque.
* Una SAN está compuesta por nodos de almacenamiento, redes de alta velocidad (Gigabit Ethernet, 10G Ethernet, Myrinet o Infiniband) y el resto de nodos
* Un dispositivo de bloques puede ser un disco, una partición, un volumen lógico, . . .
* Los protocolos iSCSI, Fibre Channel (FC) o ATA over Ethernet (AoE) utilizados en SAN, permiten que los nodos manejen un dispositivo de bloques remoto como un dispositivo de bloques local
* Para que varios nodos puedan montar el mismo dispositivo de bloques, deben utilizar un sistema de ficheros para SAN, como Oracle Cluster File System (OCFS) o Red Hat Global FileSystem (GFS).

**Almacenamiento. Sistemas de ficheros distribuidos**

* No necesitan utilizar los protocolos iSCSI, FC o AoE porque utilizan su propio protocolo para comunicar cliente y servidor.
* El objetivo de un sistema de ficheros distribuidos es proporcionar almacenamiento remoto a sus clientes de forma transparente, ya que estos lo verán como si se tratase de alm. local.
* Los sistemas de ficheros distribuidos deben poseer tolerancia a fallos, gran escalabilidad, utilización en sistemas heterogéneos y control de concurrencia entre otras propiedades
* Algunos de los sistemas de ficheros distribuidos son:
  + Lustre
  + Ceph
  + Google FileSystem (GFS)
  + GlusterFS
  + Windows DFS

**Almacenamiento. DRBD**

* Distributed Replicated Block Device.
* Una opción específicamente desarrollada para clústeres de alta disponibilidad.
* Se replica un determinado dispositivo de bloques por una red dedicada.
* Se denomina por analogía *Network based RAID1.*
* Puede utilizarse en modo síncrono o asíncrono.
* Incluido en el kernel linux desde la versión 2.6.33

**Resumen**

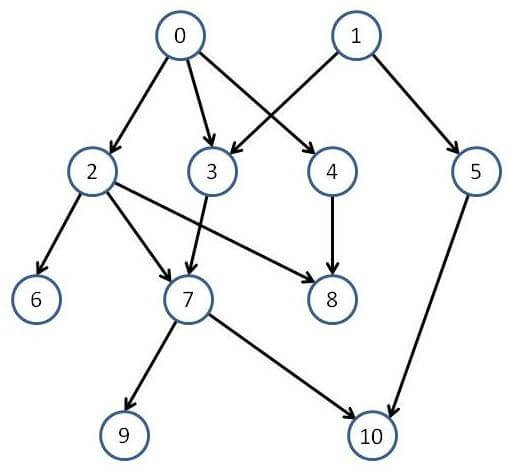
* Los clústeres de alta disponibilidad se están convirtiendo en elementos cada vez más comunes en todo tipo de infraestructuras TIC.
* La tendencia en el manejo de los nodos del cluster (y en general del resto de los servidores) es mediante aplicaciones de orquestación.
* Es muy importante que el cluster sea escalable (o incluso elástico) para adaptarse a diferentes situaciones.
* Un cluster requiere de soluciones específicas de almacenamiento.

**LINUX - Apache Spark** ([Apache Spark: qué es y cómo funciona](https://geekytheory.com/apache-spark-que-es-y-como-funciona/))

**C**ombina un sistema de computación distribuida a través de clusters de ordenadores con una manera sencilla y elegante de escribir programas. Fue creado en la Universidad de Berkeley en California y es considerado el primer software de código abierto que hace la programación distribuida realmente accesible a los científicos de datos.

Es sencillo entender Spark si lo comparamos con su predecesor, [MapReduce](http://geekytheory.com/fundamentos-de-apache-hadoop-y-mapreduce/), el cual revolucionó la manera de trabajar con grandes conjuntos de datos ofreciendo un modelo relativamente simple para escribir programas que se podían ejecutar paralelamente en cientos y miles de máquinas al mismo tiempo. Gracias a su arquitectura, MapReduce logra prácticamente una relación lineal de escalabilidad, ya que si los datos crecen es posible añadir más máquinas y tardar lo mismo.

Spark mantiene la escalabilidad lineal y la tolerancia a fallos de MapReduce, pero amplía sus bondades gracias a varias funcionalidades: **DAG y RDD.**



**DAG (Grafo Acíclico Dirigido)** es un grafo dirigido que no tiene ciclos, es decir, para cada nodo del grafo no hay un camino directo que comience y finalice en dicho nodo. Un vértice se conecta a otro, pero nunca a sí mismo.

Spark soporta el flujo de datos acíclico. Cada tarea de Spark crea un DAG de etapas de trabajo para que se ejecuten en un determinado cluster. En comparación con **MapReduce**, el cual crea un DAG con dos estados predefinidos (Map y Reduce), los grafos DAG creados por Spark pueden tener cualquier número de etapas. Spark con DAG es más rápido que MapReduce por el hecho de que no tiene que escribir en disco los resultados obtenidos en las etapas intermedias del grafo. MapReduce, sin embargo, debe escribir en disco los resultados entre las etapas Map y Reduce.

Gracias a una completa **API (interfaz de programación de aplicaciones)**, es posible programar complejos hilos de ejecución paralelos en unas pocas líneas de código.

**RDD (Resilient Distributed Dataset)**

Apache Spark mejora con respecto a los demás sistemas en cuanto a la computación en memoria. RDD permite a los programadores realizar operaciones sobre grandes cantidades de datos en clusters de una manera rápida y tolerante a fallos. Surge debido a que las herramientas existentes tienen problemas que hacen que se manejen los datos ineficientemente a la hora de ejecutar algoritmos iterativos y procesos de minería de datos. En ambos casos, mantener los datos en memoria puede mejorar el rendimiento considerablemente.

Una vez que los datos han sido leídos como objetos RDD en Spark, pueden realizarse diversas operaciones mediante sus APIs. Los dos tipos de operaciones que se pueden realizar son:

* **Transformaciones:** tras aplicar una transformación, obtenemos un nuevo y modificado RDD basado en el original.
* **Acciones:** una acción consiste simplemente en aplicar una operación sobre un RDD y obtener un valor como resultado, que dependerá del tipo de operación.

Dado que las tareas de Spark pueden necesitar realizar diversas acciones o transformaciones sobre un conjunto de datos en particular, es altamente recomendable y beneficioso en cuanto a eficiencia el almacenar RDDs en memoria para un rápido acceso a los mismos. Mediante la función caché() se almacenan los datos en memoria para que no sea necesario acceder a ellos en disco.

El almacenamiento de los datos en memoria caché hace que los algoritmos de machine learning ejecutados que realizan varias iteraciones sobre el conjunto de datos de entrenamiento sea más eficiente. Además, se pueden almacenar versiones transformadas de dichos datos.

**Modelo de programación**

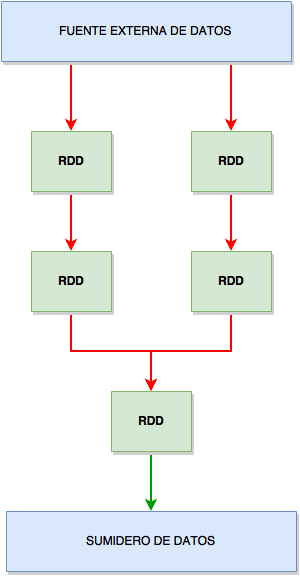
Un programa típico se organiza de la siguiente manera:

1. A partir de una variable de entorno llamada context se crea un objeto RDD leyendo datos de fichero, bases de datos o cualquier otra fuente de información.
2. Una vez creado el RDD inicial se realizan transformaciones para crear más objetos RDD a partir del primero. Dichas transformaciones se expresan en términos de programación funcional y no eliminan el RDD original, sino que crean uno nuevo.
3. Tras realizar las acciones y transformaciones necesarias sobre los datos, los objetos RDD deben converger para crear el RDD final. Este RDD puede ser almacenado.

Un pequeño ejemplo de **código en Python que cuenta el número de palabras que contiene un archivo** sería el siguiente:

my\_RDD = spark.textFile("hdfs://...") words = my\_RDD.flatMap(lambda line : line.split(" ")) .map(lambda word : (word, 1)) .reduceByKey(lambda a, b : a + b) words.saveAsTextFile("hdfs://...")

Cuando el programa comienza su ejecución crea un grafo similar al de la figura siguiente en el que los nodos son objetos RDD y las uniones entre ellos son operaciones de transformación. El grafo de la ejecución es un DAG y, cada grafo es una unidad atómica de ejecución. En la figura siguiente, las líneas rojas representan transformación y las verdes operación.

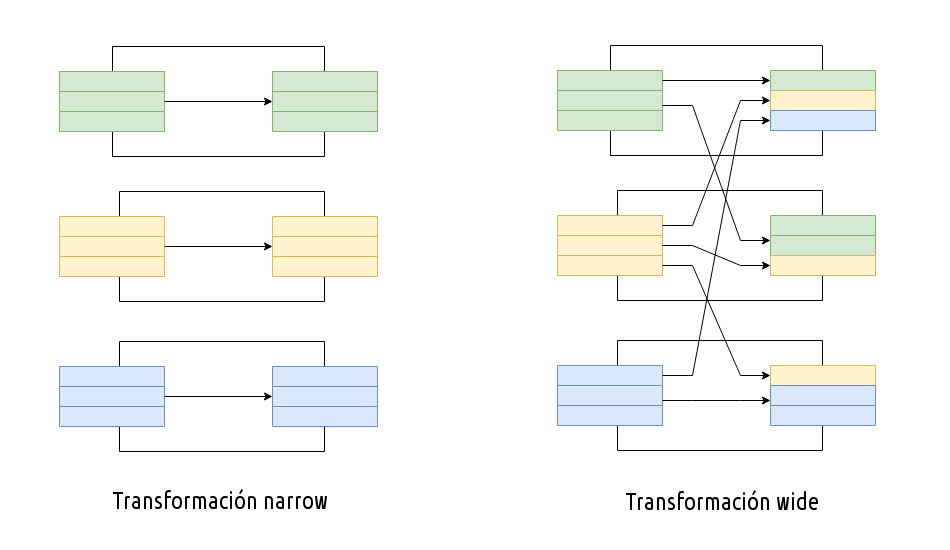


**Tipos de transformaciones**

Es muy posible que los datos con los que se necesite tratar estén en diferentes objetos RDD, por lo que Spark define dos tipos de operaciones de transformación:

* **Narrow transformation:** se utiliza cuando los datos que se necesitan tratar están en la misma partición del RDD y no es necesario realizar una mezcla de dichos datos para obtenerlos todos. Algunos ejemplos son las funciones filter(), sample(), map() o flatMap().
* **Wide transformation:** se utiliza cuando la lógica de la aplicación necesita datos que se encuentran en diferentes particiones de un RDD y es necesario mezclar dichas particiones para agrupar los datos necesarios en un RDD determinado. Ejemplos: groupByKey() o reduceByKey().

Una representación gráfica de ambos tipos de transformaciones es la que se puede apreciar en la figura siguiente:

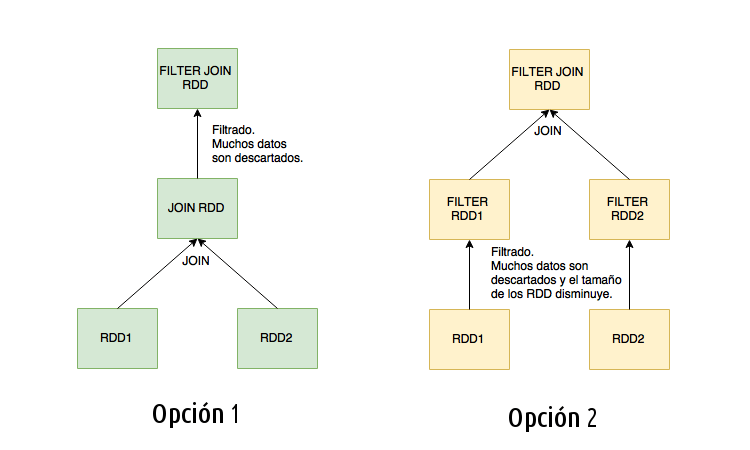


En algunos casos es posible realizar un reordenamiento de datos para reducir la cantidad de datos que deben ser mezclados. A continuación se muestra un ejemplo de un JOIN entre dos objetos RDD seguido de una operación de filtrado.

Por ejemplo, dados dos objetos RDD (RDD1 y RDD2), con variables ’a’ y ’b’, se va a realizar una operación de JOIN entre ambos conjuntos de datos para los casos en los que ’a’ sea mayor que 5 y ’b’ sea menor que 10:

SELECT a, b FROM RDD1 JOIN RDD2 WHERE a>5 AND b<10

Esta operación puede realizarse de dos maneras, tal y como se aprecia en la imagen de abajo. La primera opción consiste en una implementación muy simple en la que primero se realiza el JOIN entre los objetos RDD y luego se filtran los datos. Sin embargo, en la segunda opción, primero se realiza el filtrado por separado en ambos RDD y luego se hace el JOIN.



La segunda opción es más eficiente debido a que el filtrado y posterior unión de los datos se hace por separado. Podría decirse la mezcla o barajado de datos es la operación que más coste tiene, por lo que Apache Spark proporciona un mecanismo que genera un plan de ejecución a partir de un DAG que minimiza la cantidad de datos que son mezclados. El plan de ejecución es el siguiente:

1. Primero se analiza el DAG para determinar el orden de las transformaciones.
2. Con el fin de minimizar el mezclado de datos, primero se realizan las transformaciones narrow en cada RDD.
3. Finalmente se realiza la transformación wide a partir de los RDD sobre los que se han realizado las transformaciones narrow.

**Conclusiones**

Apache Spark es una herramienta útil y eficiente para tareas de procesamiento masivo de datos (Big Data y Machine Learning). Está en constante desarrollo y se actualiza frecuentemente. Además, su documentación es muy completa y la comunidad cada vez se hace más grande.

***Práctica Cluster 1.1 (***[***Cómo crear un clúster de servidores con Apache Spark***](https://geekytheory.com/como-crear-un-cluster-de-servidores-con-apache-spark/)***)***