Informe Técnico

Abstract: Apache Hadoop es un framework basado en JAVA que soporta aplicaciones distribuidas. Permite a las aplicaciones trabajar con miles de nodos y petabytes de datos. Hadoop se inspiró en los documentos Google para MapReduce y Google File System (GFS). Hadoop es un proyecto de alto nivel Apache y con una gran comunidad base

Document Id.:	**CESGA-2013-002**
Date:	**24/01/13**
Responsible:	Diego Nieto
Status:	**FINAL**

CESGA

Hadoop y *MapReduce*.

Diego Nieto

Index

1 Introducción

Apache Hadoop es un *framework* basado en JAVA que soporta aplicaciones distribuidas. Permite a las aplicaciones trabajar con miles de nodos y petabytes de datos. *Hadoop* se inspiró en los documentos Google para MapReduce y Google File System (GFS). Hadoop es un proyecto de alto nivel Apache y con una gran comunidad base. Yahoo! ha sido el mayor contribuidor al proyecto.

Hadoop: Procesamiento de enormes cantidades de datos (TB y PB) en grandes clusters de comodity hardware. Esta formado por 2 sistemas:

- Almacenamiento: HDFS
- Procesamiento: MapReduce

Y aporta una serie de ventajas:

- Bajo coste
- Facilidad de uso
- Tolerancia a fallos

2 Arquitectura

Cuatro procesos (*dæmons*) principales:

- En el *master*: *namenode* y *jobtracker*

- En los workers: *datanode* y *tasktracker*

- *namenode* y *datanodes*: sistema HDFS

- *jobtracker* y *tasktrackers*: trabajos *MapReduce*

Ficheros de configuración en *$HADOOP_HOME/conf:*

- *core-site.xml*: configuración principal, valores por defecto en *hadoop.apache.org/common/docs/current/core-default.html*

- *hdfs-site.xml*: configuración del HDFS, valores por defecto en *hadoop.apache.org/common/docs/current/hdfs-default.html*

- *map-red-site.xml*: configuración del MapReduce, valores por defecto en *hadoop.apache.org/common/docs/current/mapred-default.html*

2.1 HDFS

Hadoop puede acceder a diferentes tipos de filesystems (local, HDFS, KFS, S3,. . .)

Ventajas HDFS: *Hadoop Distributed File System*:

- Diseñado para almacenar ficheros muy grandes en commodity hardware
- Elevado ancho de banda
- Fiabilidad mediante replicación

- Tolerancia a fallos

Inconvenientes

- Elevada latencia
- Poco eficiente con muchos ficheros pequeños
- Modificaciones siempre al final de los ficheros
- No permite múltiples *writers*

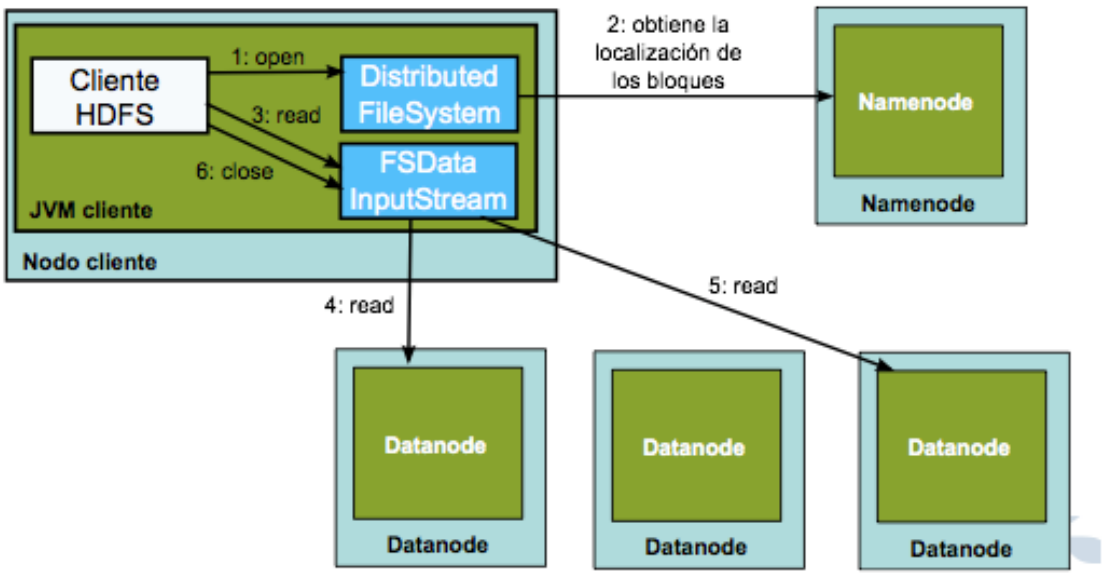
Los principales procesos vinculados al HDFS son:

- *Namenode*: Mantiene la información (metadatos) de los ficheros que residen en el HDFS
- *Datanode*: Mantienen los datos y se encargan de la replicación de los mismos
- *Secondary Namenode*: Mantienen *checkpoints* del *Namenode*

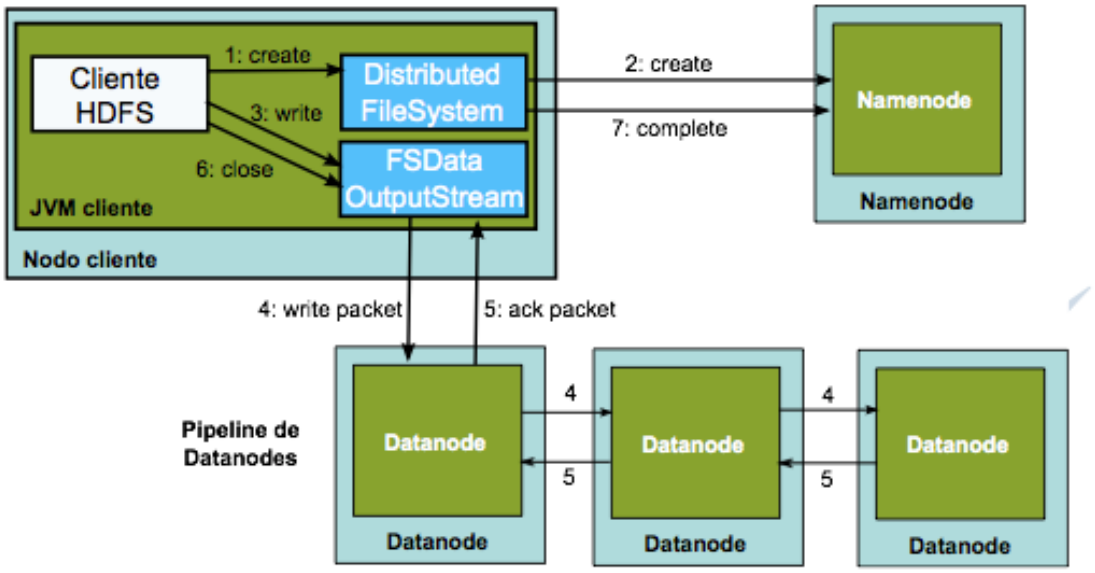
La interfaz principal para acceder al HDFS es mediate CLI:

 # hadoop dfs –help

Lectura de datos HDFS



Escritura de datos HDFS



(imágenes cortesía de Tomás Fernández Pena del CITIUS-USC)

2.2 MapReduce

Modelo de programación funcional en paralelo diseñado para escalabilidad y tolerancia a fallos en grandes sistemas de *commodity hardware*:

- Basado en la combinación de operaciones Map y Reduce
- Diseñado originalmente por *Google*
- Usado en múltiples operaciones
- Manejo de varios petabytes diarios
- Popularizado por la implementación open source *Apache Hadoop*
- Usado por *Facebook, Last.fm, Rackspace, yahoo, Amazon Web Services…*

Ejemplos de algunas aplicaciones de *MapReduce*:

- En *Google*:
  - Construcción de índices para el buscador (*pagerank*)
  - *Clustering* de artículos en *Google News*
  - Búsqueda de rutas en *Google Maps*
  - Traducción estadística

- En Facebook:
  - Minería de datos
  - Optimización de ads
  - Detección de spam
  - Gestión de logs

- En I+D+i:
  - Análisis astronómico
  - bioinformática
  - física de partículas
  - simulación climática
  - procesamiento del lenguaje natural

Organizaciones que usan *Hadoop*:

- *A9.com*
- *AOL*
- *Booz Allen Hamilton*

- *EHarmony*
- *eBay*
- *Facebook*
- *Fox Interactive Media*
- *Freebase*
- *IBM*
- *ImageShack*
- *ISI*
- *Joost*
- *Last.fm*
- *LinkedIn*
- *Meebo*
- *Metaweb*
- *Mitula15*
- *The New York Times*
- *Ning*
- *Powerset (ahora parte de Microsoft)*
- *Rackspace*
- *StumbleUpon16*
- *Tuenti*
- *Twitter*
- *Veoh*
- *Zoosk*
- *1&1*

3 Objetivos de las pruebas

Los objetivos principales de los set de pruebas son:

- Dimensionar rendimiento y escalabilidad del HDFS de *Hadoop*.

- Probar algoritmos en *MapReduce* para dimensionar la escalabilidad y su capacidad de paralelización.

- Comprobar la capacidad de integración de *Hadoop* y *MapReduce* con otras soluciones y herramientas open para proporcionar soluciones de computación y almacenamiento alternativas:

    - Que aprovechen mejor los recursos existentes.
    - Que aporten una reducción de costes y consumo sin disminución de potencia
    - Que sean escalables

4 Descripción de hardware

- El *bechmark* se divide en 2 sets de pruebas. Un primer set para el *cluster SVGD*, centrado en dimensionar la escalabilidad y paralelización de *MapReduce* y un segundo set, para el *cluster SVGD2* , orientado a dimensionar el rendimiento del sistema de ficheros distribuido de *Hadoop* (HDFS).

- En el cluster SVGD se han utilizado 16 nodos con 2 procesadores *AMD Opteron Processor 6174 2.2 GHz*, por nodo (24 *cores* por nodo en total) y con conectividad *Gigabit Ethernet*.

- Para el cluster SVGD2, se han utilizado 8 nodos con 2 procesadores *Intel Sandy Bridge E5-2670 2.6 GHz* por nodo (16 *cores* por nodo en total) y con conectividad *Infiniband* de 40Gbps.

4.1 Descripción y ejecución del benchmark SVGD

Este set de pruebas se centra en analizar la escalabilidad y capacidad de paralelización de *MapReduce*.

La instalación base ha constado de 4/8/16 nodos en el SVGD:

- 1x *Master* (*namenode, secondary namenode, jobtracker, datanode y tasktracker*)
- 3/7/15 *slaves/workers* (*datanode* y *tasktracker*)

Como *dataset* se ha utilizado *freebase* (enciclopedia libre), una base de datos de 5GB, y como ejemplo de código *MapReduce* se ha utilizado el clásico *WordCount* (conteo de palabras y repeticiones de las mismas).

Los tiempos de carga no son muy significativos ya que la carga no se realiza en paralelo, pero si el procesamiento de datos. También se puede observar que utilizar ficheros comprimidos, no aporta mucho a la hora de procesar los datos:

# nodos	***freebase (min:seg)***	***freebase gz***	***wordcount (min:seg)***	***wordcount gz (min:seg)***
16	7:59	2:43	3:07	3:52
8	7:54	2.42	4:33	5:30
4	7:59	2.43	7:15	7:48

Todas las máquinas han consumido entre un 10-15% de CPU. La memoria está configurada para que no se supere el consumo de 2GB de RAM (1GB para el *datanode* y 1 GB para el *tasktracker*). En principio se puede tunear el número de *threads* para un proceso map o reduce, pudiendose optimizar al el uso de la memoria RAM. Por ejemplo si disponemos de una aplicación *MapReduce*, en donde el proceso map consume más memoria que el reduce, se podría tunear el sistema para que asigne más memoria al proceso map.

La ejecución de los trabajos de procesamiento se ha realizado via interfaz CLI

 # hadoop jar wordcount.jar WordCount /user/dnieto/freebase /user/dnieto/freebase-out

La monitorización de la ejecución de los trabajo se ha realizado a través del interfaz web del *jobtracker*.

4.2 Descripción y ejecución del benchmark SVGD2

Este *set* de pruebas se centra en analizar el rendimiento y la escalabilidad del sistema de ficheros distribuido HDFS de Hadoop

La instalación base ha constado de 9 nodos en el SVGD2:

- 1x *Master* (*namenode, secondary namenode y jobtracker*)
- 8x *Slaves* (*datanodes y tasktrackers*)

Como prueba se van a utilizar los datos generados por el sistema de *consumo/accounting* de las aplicaciones en los sistemas *HPC* del CESGA. Estos datos se encuentran en un SGBD relacional, cuyo esquema está en 3FN, configurado para transacciones (*INSERTS* y *UPDATES*). Este sistema está montado sobre una *VM* con 3GB de RAM y 2 Vproc.

Para obtener un análisis de los datos de consumo es necesario ejecutar una *query* que obtiene dichos datos por ejecutable, aplicación y máquina. Las tablas del esquema consultado albergan del orden de 1000K *rows* y no siguen una estrategia de particionado. El cuello de botella se detecta en la tabla en donde se almacenan los datos del consumo de todos los ejecutables y aplicaciones de los sistemas *HPC* del CESGA. Dicha tabla, que tiene unas 20 columnas cuyos valores se han calculado mediante operaciones aritméticas básicas, alberga unos 20M de registros (5GB),. Dicha *query* tarda más de 60 minutos en ejecutarse, con un consumo de memoria y procesador de más del 90%, quedando la *VM* casi inaccesible durante la operación de consulta.

Una primera observación al respecto es que dicho sistema transaccional no está pensado para ejecutar consultas que consulten masivamente datos de distintas tablas y esquemas sobre los que aplican operaciones aritméticas y de ordenamiento.

Para ello sería necesario crear un sistema *OLAP*, optimizado para consultas y no para transacciones. El problema de crear este sistema es que habría que utilizar *hardware* adicional y mantenerlo, además del sistema transaccional, con todos los costes que esto conllevaría (mantenimiento, personal, formación, etc)

Aquí es donde entra en juego ***Hadoop*** y su *interfaz/API SQL* llamada ***Hive***. *Hadoop* nos permite cargar los datos de consumo *as-provided* sin necesidad de aplicar cambios u operaciones *ETL* sobre los datos fuente y *Hive* nos permite consultar dichos datos, de la misma manera que lo haríamos en el SGBD relacional original, utilizando *SQL* y todo de manera transparente para el usuario.

Internamente, *Hadoop* almacena y distribuye entre los distintos nodos los datos (particionado) y *hive* se encarga de dividir la consulta en varias partes que son asignadas a trabajos MapReduce, cuyos procesos map y reduce son paralelizables entre los distintos nodos del cluster. De esta manera estamos aprovechando recursos de computación existentes para realizar tareas que en un principio no podrían ser asignadas a este tipo de recursos.

Los tiempos de carga no son muy significativos ya que la carga no se realiza en paralelo. El tiempo total de carga de 5,3GB ha sido de 5 minutos y 10 segundos.

A continuación se detalla la ***query[1] ad-hoc*** a ejecutar:

hive> INSERT OVERWRITE TABLE cons_grouped

SELECT consumo.fechafin, maquinas.nombre, instituciones.nombre, consumo.idusuario, aplicaciones.nombre, ejecutables.nombre, consumo.estado, COUNT(consumo.idconsumo), SUM(consumo.systcpu), SUM(consumo.usercpu), SUM(consumo.elapcpu), MAX(consumo.memory), SUM(consumo.io), SUM(consumo.rw)

FROM maquinas

JOIN aplicejec ON (maquinas.idmaquina = aplicejec.idmaquina)

JOIN aplicaciones ON (aplicaciones.idaplic = aplicejec.idaplic)

JOIN ejecutables ON (ejecutables.idejec = aplicejec.idejec)

JOIN consumo ON (consumo.idaplicejec = aplicejec.idaplicejec)

JOIN instituciones ON (instituciones.idinstit = consumo.idinstit)

GROUP BY consumo.fechafin, maquinas.nombre, instituciones.nombre, consumo.idusuario, aplicaciones.nombre, ejecutables.nombre, consumo.estado;

Al analizar la complejidad de la *query ad-hoc*, *hive* divide la misma en distintos trabajos *MapReduce* que envía al *jobtracker*, que se encarga de repartirlos entre los distintos nodos. En este caso cada operación JOIN es asignada a un trabajo, siendo el cómputo final de trabajos 6.

Todas las máquinas han consumido entre un 30-40% de CPU.

La memoria está configurada para que no se supere el consumo de 4GB de RAM (2GB para el datanode y 2GB para el tasktracker).

Como en el *benchmark* anterior, se ha configurado el número de procesos y de threads vinculados a operaciones map o reduce, para optimizar al máximo el uso de la memoria y de los procesadores.

Como cada nodo dispone de 16 cores (2 procesadores 8-*cores*), el número de tareas máximo para maps o reduces es de 16 por nodo (8 maps / 8 reduces)

La ejecución de los trabajos de procesamiento se ha realizado via interfaz CLI de hive.

La monitorización de la ejecución de los trabajos se ha realizado a través del interfaz web del *jobtracker*, donde están todos trabajos ejecutados:

En la sección de "*Retired Jobs*" se puede consultar los detalles *(memoria consumida, tiempo de cpu, operaciones MapReduce, tiempo de ejecución etc*) de cada uno de los 6 trabajos *MapReduce* que componen la *query*. Aclarar que cada trabajo o *job* es paralelizado a lo largo del *cluster* y su ejecución es secuencial, es decir, se ejecuta el primer trabajo y se paraleliza. En cuanto acabe se ejecuta el segundo trabajo y se paraleliza, así hasta completar el número de trabajos que componen la query.

El throughtput total del HDFS y el rendimiento se puede analizar en la siguiente tabla:

JobID	HDFS_BYTES_READ	HDFS_BYTES_WRITTEN
job001	322.532,00	588.042,00
job002	597.224,00	658.165,00
job003	861.963,00	782.900,00
job004	3.424.465.408,00	1.978.859.253,00
job005	2.523.293.683,00	38.935.929,00
job006	39.635.882,00	34.001.967,00
Total bytes	5.989.176.692,00	2.053.826.256,00
Texec	426,63	426,63
Bytes to Gbits	44,62	15,30
Gbits/s	0,104594051	0,035867702
Bytes to Mbytes	5711,698136	1958,672486
MB/s	13,38794303	4,59103318
Throughput concurrente (MB/s)	**856,8283541**	**293,8261235**

Podemos observar que el *throughput concurrente* es la medida de rendimiento a utilizar, que según las referencias de los bechmarks[1] TestDFSIO y Terasort, se obtiene multiplicando el ancho de banda medio por tarea por el número de tareas *map* disponibles en el *cluster*. En nuestro caso 64 (8 nodos * 8 tareas map/nodo) maps disponibles en el cluster.

El resultado final nos da una idea de la capacidad de *Hadoop* y *MapReduce* para aprovechar los recursos de computación y almacenamiento distribuidos existentes en el CESGA.

[1] http://www.michael-noll.com/blog/2011/04/09/benchmarking-and-stress-testing-an-Hadoop-cluster-with-terasort-testdfsio-nnbench-mrbench/

#nodos/cores	*query*[1]
1 / 2	3.392 segundos (61 min. 32 sec.)
8/16	426,332 segundos (7 min.)

5 Conclusiones

Como conclusión podemos afirmar que las posibilidades que *Hadoop*, *MapReduce* y *hive* pueden ofrecer como servicio de computación distribuido son muy interesantes.

La posibilidades que Hadoop puede ofrecer a una organización son muchas:

- Aportar visibilidad y competitividad en servicios del paradigma Bigdata
- Gestionar un servicio con muy pocos despliegues en instalaciones nacionales, constituyendo un imán para futuros proyectos y colaboraciones entre instituciones y empresas
- Mejor aprovechamiento de los recursos realizando un despliegue de *Hadoop incremental/ondemand*.

Dependiendo de las necesidades de cada organización se podría empezar con un servicio básico y poco hardware dedicado, permitiendo que la comunidad de usuarios interesada probase sus algoritmos MapReduce. Los usuarios podrían disponer de un conjunto de comandos *CLI* para gestionar y ejecutar sus aplicaciones *MapReduce*.

Como conclusión final, este benchmark también ha analizado la capacidad de *Hadoop y hive* como sistema de almacenamiento distribuido y como un *Datawarehouse OLAP* para realizar un análisis básico. El ahorro de costes en hardware y software, así como el aprovechamiento y optimización de recursos de computación hace posible que este despliegue se pueda utilizar para otras tareas similares de almacenamiento, análisis, sumarización y minería de datos (pe. La web de la organización o empresa, aplicaciones internas de gestión … etc).