Evaluación de la herramienta de código libre Apache Hadoop

Universidad Carlos III de Madrid
Escuela Politécnica Superior

Proyecto Fin de Carrera

INGENIERÍA TÉCNICA DE TELECOMUNICACIÓN: TELEMÁTICA

Autora:

MªCarmen Palacios Díaz-Zorita

Tutor:

Norberto Fernández García

Leganés, Noviembre de 2011

Resumen

La capacidad de los discos duros ha aumentado en gran medida a lo largo de los años. Las aplicaciones modernas manejan grandes volúmenes de datos y en ocasiones, los usuarios se encuentran con ficheros de varios gigabytes e incluso mayores. Los sistemas de ficheros tradicionales alcanzan pronto sus límites con esta clase de datos y rendimiento. Además, al análisis de estos grandes volúmenes de información resulta complejo, sino irrealizable, en una única computadora o servidor.

Como una de las posibles soluciones a este problema se encuentra MapReduce, un modelo de programación diseñado por Google. En concreto en este trabajo se utilizan una implementación código libre de MapReduce llamada Hadoop.

Este proyecto trata de hacer un estudio de esta herramienta, Hadoop, en la que se analiza el rendimiento de una aplicación MapReduce escrita en Java. El estudio se realizará sobre varios *clusters* de máquinas, aprovechando la capacidad de procesamiento de estos y la fiabilidad y tolerancia a fallos que facilita el modelo MapReduce.

Abstract

The capacity of hard drives has increased greatly over the years. Modern applications handle large volumes of data, at times, users are faced with multi-gigabyte files and even higher. The traditional file systems soon reach their limits with this kind of data and performance. In addition, the analysis of these large volumes of information is complex and sometimes impossible, on a single computer or server.

MapReduce could be considered as a possible solutions to this problema. It is a programming model designed by Google. Specifically, an open source implementation of MapReduce, called Hadoop, is used in this paper.

The aim of this project is to study this tool, Hadoop, analising the performance of a Java based MapReduce application. This study will be carried out by using several computer *clusters* taking advantage of their processing capability, reliability and failover, which make the MapReduce model easier.

ÍNDICE DE CONTENIDOS

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

1. INTRODUCCIÓN

1.1. Motivación del Proyecto

Hoy por hoy, vivimos en la era de los datos. La explosión de datos está sucediendo a todo nivel en todos los dispositivos electrónicos, aplicaciones, individuos y organizaciones. De acuerdo al "Estudio del Universo digital" de IDC [1], el año pasado excedimos 1.2 Zettabytes con un pronóstico de crecimiento de 44x en la presente década. El recurso humano asociado solo crecerá 1.4x, lo que representa una enorme oportunidad para la industria.

Estos grandes volúmenes de datos pueden venir de varias partes [2]:

- La bolsa de Nueva York genera un terabyte de nuevos datos comerciales por día.
- Facebook contiene aproximadamente 10.000 millones de fotos, las cuales requieren un Petabyte de almacenamiento.
- Ancestry.com, el portal de información genealógica, almacena cerca de 2.5 Petabytes.
- El gran colisionador de partículas producirá 15 Petabytes de datos por año [3].
- Google procesa 20 Petabytes de datos diariamente.

En la actualidad, la tarea de administrar una base de datos mayor a 3 TB requiere una buena solución. Los lugares donde se almacenan los datos pueden soportar hasta 80 TB en sistemas con memoria compartida (SMP) y el salto a los Petabytes generalmente requiere procesamiento paralelo. En Estados Unidos, el 60% de bases de datos en producción de empresas supera ya 1 TB de información y de acuerdo a Forrester el 13% supera los 15 TB [4]. Los grandes sitios de Internet son sin duda los que tienen la mayor oportunidad en el denominado *clickstream analysis.*

El procesamiento de grandes volúmenes de datos involucra:

a) Almacenamiento de los datos
b) Lectura y escritura de los datos
c) Procesamiento de los datos.

En cuanto al almacenamiento de los datos, una solución es comprar varios discos duros de alta capacidad y conectarlos mediante un sistema RAID (del inglés *Redundant Array of Independent Disks*), lo cual dependiendo de la aplicación puede llegar a ser muy costoso o más aún no obtener el espacio suficiente. De igual forma, la lectura y escritura de información parece simple, sin embargo, el tamaño de los archivos o datos es relevante; copiar unos cuantos "Megas" puede ser muy rápido (no importando si es de manera local o usando la red), pero copiar "Teras" puede tener tiempos prohibitivos.

Finalmente, está el procesamiento de los datos, en donde para reducir tiempos de respuesta, normalmente se requiere del uso de más de un procesador.

En los últimos tiempos, para resolver y facilitar estos aspectos, han aparecido herramientas de código libre que permiten manejar grandes volúmenes de datos en sistemas distribuidos.

Google, por su parte, también ha creado herramientas de este tipo como *Google MapReduce* [5] y *Google File System* (GFS) [6].

GFS es un sistema de archivos distribuido y escalable para aplicaciones distribuidas que acceden a los datos de forma intensiva. Esta herramienta provee de tolerancia a fallos mientras se ejecuta sobre máquinas de bajo coste, y es capaz de generar un alto rendimiento a un gran número de peticiones [6].

Este sistema de archivos ha cumplido exitosamente las necesidades de almacenamiento. Y es ampliamente usado en Google como la plataforma de almacenamiento para la generación y procesamiento de datos, usados por varios servicios así como también en trabajos de investigación y desarrollo que requiere grandes conjuntos de datos.

El *cluster* más grande hasta la fecha provee cientos de terabytes de almacenamiento a través de miles de discos en un número equivalente de máquinas, y es accedido simultáneamente por cientos de cliente.

Por otro lado, MapReduce es un modelo de programación que permite el desarrollo de aplicaciones con procesamiento en grandes conjuntos de datos [7]. En MapReduce los usuarios especifican una función *map* que procesa un par clave/valor y genera un conjunto intermedio de pares clave/valor. Posteriormente la función *reduce* mezcla todos los valores intermedios asociados con la clave intermedia. Los programas escritos en este estilo funcional son automáticamente paralelizados y ejecutados sobre grandes *cluster*s de máquinas de propósito general.

En concreto, en el presente proyecto de fin de carrera centramos nuestra atención en Hadoop, la evaluación de la implementación de código libre de estas dos tecnologías, desarrollada dentro del proyecto Apache Hadoop.

1.2. Objetivos

El principal objetivo de este proyecto es el estudio y análisis de la herramienta Apache Hadoop.

Este estudio se extenderá a la programación de diferentes aplicaciones con procesamiento de grandes volúmenes de datos. Las ejecuciones de las pruebas se realizarán con un volumen de datos variable, así como con escenarios con diferente número de máquinas, para observar el comportamiento de la herramienta en parámetros tales como el tiempo de procesamiento o el consumo de espacio en disco.

El programa de pruebas consistirá en distintas aplicaciones que recibirán un fichero de entrada y procesarán los datos devolviendo un fichero de salida con diferente información. Como entrada se utilizará un fichero que contiene información sobre la estructura de enlaces de Wikipedia.

Una vez realizadas todas las ejecuciones se recogerán los datos obtenidos y se estudiarán. Se compararán los tiempos y la carga del sistema de cada escenario y se plasmarán en una gráfica, para posteriormente obtener conclusiones.

1.3. Estructura de la memoria

Esta memoria se encuentra dividida en distintos capítulos en los que se explican todos los temas relacionados con este proyecto. A continuación se muestra una breve descripción de cada uno de los capítulos que conforman la memoria.

- Capítulo 1: Introducción

Se trata del capítulo en el que se detallan cuáles son las motivaciones iniciales y los objetivos que se marcaron para la realización del proyecto.

- Capítulo 2: Estado del Arte

En este capítulo se explica brevemente en qué consisten las tecnologías utilizadas durante el desarrollo de este proyecto. Tanto la tecnología MapReduce y Google File System, como su equivalente de código libre Hadoop y HDFS

- Capítulo 3: Diseño e Implementación

En este capítulo se describirán los aspectos de instalación y configuración de Hadoop. Se describirá el entorno de pruebas, especificando las máquinas y direcciones IP utilizadas, las funciones de maestro/esclavo y la conectividad. Se explicarán las distintas pruebas realizadas y cómo se han resuelto con la herramienta Hadoop. También se hablará de los problemas acaecidos y cómo se han resuelto.

- Capítulo 4: Validación

En este capítulo se mostrarán las pruebas que se han hecho para asegurarnos de que el código funciona correctamente, así como los resultados obtenidos de la evaluación empírica sobre el funcionamiento de la herramienta Hadoop.

- Capítulo 5: Conclusiones y líneas futuras

En este capítulo se mostrará el grado en el que se han alcanzado los objetivos, lecciones aprendidas, dificultades al realizar el proyecto y posibles ideas para seguir trabajando en el tema.

- Anexo A: Presupuesto

En este anexo se muestra la planificación llevada a cabo, los materiales necesarios y el coste detallado del proyecto.

- Anexo B: Manual de instalación y ejecución

En este anexo se describen los pasos que tiene que seguir un usuario para conseguir instalar la herramienta Hadoop en un entorno Linux. También se explicarán los pasos necesarios para ejecutar la aplicación desarrollada, para que cualquier usuario pueda utilizar y entender en todo momento cómo se debe utilizar. La explicación se acompañará de ilustraciones aclarativas del proceso.

- Anexo C: Desarrollo con Eclipse

El desarrollo del código Java fue escrito con la ayuda del compilador para Java Eclipse. En este anexo se describe de forma detallada tanto el proceso de instalación de la herramienta Eclipse, como del pluging de Hadoop que permite integrar los servicios de la plataforma Hadoop con esta potente herramienta de desarrollo.

2. ESTADO DEL ARTE

En este capítulo se pretende explicar de manera general en qué consisten las tecnologías utilizadas durante el desarrollo del proyecto. Se explicarán tanto la tecnología MapReduce y Google File System, como su equivalente de código libre Hadoop y HDFS.

2.1. MapReduce

2.1.1. Introducción

MapReduce es el nombre dado a la combinación de dos procesos separados necesarios para la extracción de valores de un gran número de orígenes de datos distintos. La parte *map* funciona como extractor y asigna valores a determinadas claves para un único documento. La parte *reduce* realiza la función de acumulación y combina las claves de múltiples documentos para crear un valor reducido (combinado) único para cada clave a partir de los múltiples valores generados.

Lo interesante de esta forma de trabajo es que es fácil de implementar en un *cluster*. Muchas de las tareas del mundo real se pueden expresar con este modelo [5].

MapReduce es un framework introducido por Google en 2004 para dar soporte a la computación paralela sobre grandes colecciones de datos en grupos de ordenadores. Se han escrito implementaciones de MapReduce en C++, Java, Python y otros lenguajes. El procesamiento computacional puede

realizarse sobre los datos almacenados, ya sea en un sistema de archivos (no estructurado) o dentro de una base de datos (estructurado).

MapReduce permite procesar los datos que están en el *cluster* de forma paralela aprovechando el sistema de archivos distribuido. Cada servidor del *cluster* trata de procesar la parte de los datos que posee localmente. Existen dos fases principales en el proceso: *Map* y *Reduce*.

- **Map:** El método Map recibe como entrada un par (clave, valor) y su salida es uno o varios pares (clave-i, valor-i). Para cada (clave1, valor1) devuelve una lista de (clave2, valor2):

$$(\text{clave1}, \text{valor1}) \rightarrow [(\text{clave2}, \text{valor2})]$$

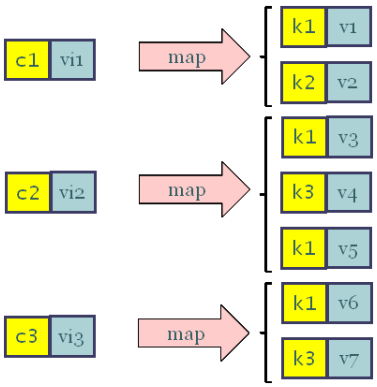


Figura 2.1: Función *Map*

El sistema se encarga de mezclar y ordenar resultados intermedios en función de las claves.

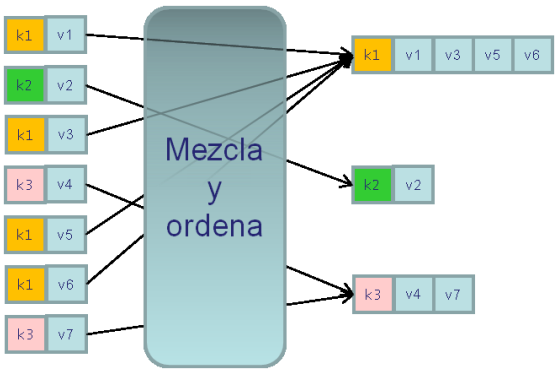


Figura 2.2: Mezcla y ordenación

- **Reduce**: El método *Reduce* recibe como entrada un par (clave, lista de valores) y la salida es un par (clave, valor).

  Para cada clave2, toma la lista de valores asociada y los combina en uno solo:

$$(\text{clave2}, [\text{valor2}]) \rightarrow (\text{clave2}, \text{valor2})$$

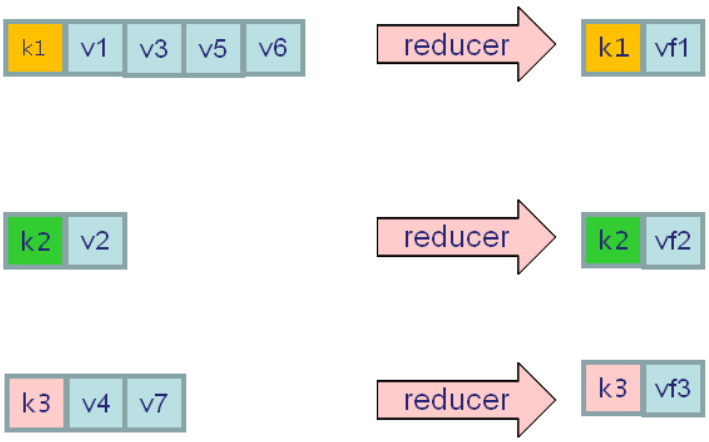


Figura 2.3: Función *Reduce*

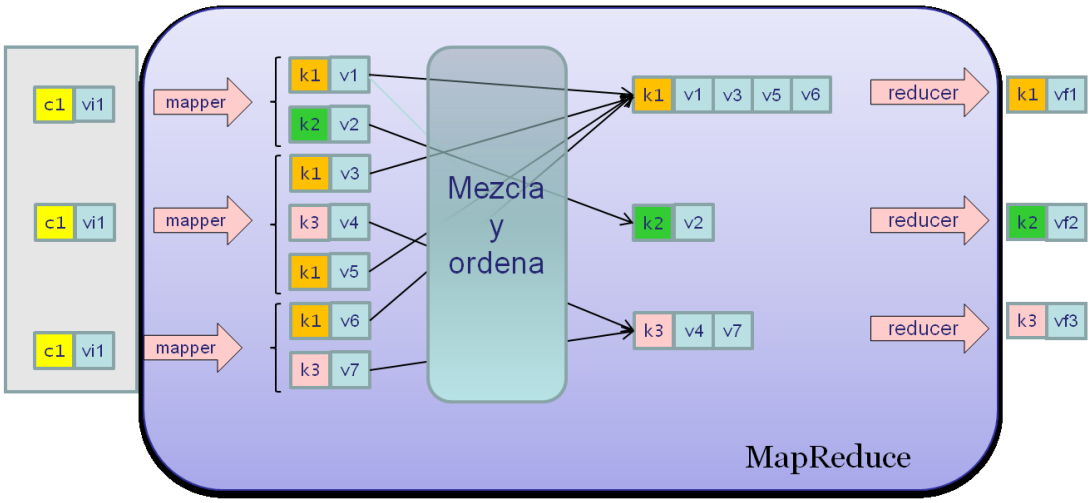


Figura 2.4: Esquema general MapReduce

Una ventaja de MapReduce es que permite el procesamiento distribuido de las operaciones *map* y *reduce*. Siempre que cada operación de asignación sea independiente de las demás, todas las operaciones *map* se puede realizar en paralelo, aunque en la práctica esta paralelización se ve limitada por la fuente de datos y/o el número de CPU's. Del mismo modo, un conjunto de operaciones *reduce* pueden llevar a cabo la fase de reducción. Todo lo que se requiere es que todas las salidas de la operación *map* que comparten la misma clave se presenten a la misma operación *reduce* y al mismo tiempo. Si bien este proceso a menudo puede parecer ineficiente en comparación con los algoritmos que son más secuenciales, MapReduce se puede aplicar a conjuntos de datos mucho mayores de lo que un servidor, por grande que sea, puede manejar [7]. Una gran granja de servidores puede usar MapReduce para ordenar un petabyte de datos en sólo unas pocas horas. Este método también ofrece posibilidades de recuperarse de un fallo de almacenamiento en los servidores durante la operación: si un *map* o *reduce* falla, el trabajo puede ser reprogramado, asumiendo que los datos de entrada aún están disponible.

2.1.2. Funcionamiento

En los últimos cinco años, se han implementado cientos de algoritmos para procesar grandes cantidades de datos en bruto. La mayoría de estos cálculos son conceptualmente sencillos. Sin embargo, los datos de entrada suelen ser grandes y los cálculos tienen que ser distribuidos a través de cientos o miles de máquinas con el fin de realizarlo en una cantidad de tiempo razonable. La cuestión de cómo paralelizar el cálculo, distribuir los datos y manejar los puntos de fallo, dificultan el algoritmo original introduciendo una gran cantidad de código complejo para tratar estos temas [5].

Como solución a este problema, se ha pensado en una nueva abstracción que permita expresar los cálculos simples que se están tratando de realizar, pero ocultando los detalles complejos de paralelización, tolerancia a fallos,

distribución de los datos y balanceo de carga en una biblioteca. El paradigma MapReduce solucionaría estos problemas [5].

Las funciones *map* y *reduce* de MapReduce se definen sobre datos estructurados en pares clave/valor. La función *map*, escrita por el usuario, recibe un par clave/valor y devuelve un conjunto de pares clave/valor intermedio:

$$\text{Map} \quad (k1, v1) \rightarrow \text{list}\ (k2, v2)$$

Esta función se aplica en paralelo a cada par del conjunto de datos de entrada produciendo una lista de pares (k2, v2) por cada llamada. MapReduce agrupa todos los valores intermedios asociados con la misma clave *k* y se los pasa a la función *reduce*.

La función *reduce* recibe esa clave y su conjunto de valores asociados y los fusiona para formar un conjunto de valores posiblemente más pequeño:

$$\text{Reduce} \quad (k2, \text{list}\ (v2)) \rightarrow \text{list}\ (v3)$$

El resultado de la llamada a la función *Reduce* es una lista (que puede estar vacía o contener un único elemento de valores). Los resultados de las llamadas se recopilan en la lista de resultados buscada.

Para ilustrar en líneas generales el funcionamiento de este modelo de programación, se muestra este ejemplo de MapReduce. Es un proceso para contar las apariciones de cada palabra en un conjunto de documentos. El usuario debería codificar un algoritmo similar al siguiente pseudo-código:

```
map(String name, String document):
  // clave: nombre del documento
  // valor: contenido del documento
  for each word w in document:
    EmitIntermediate(w, 1);

reduce(String word, Iterator partialCounts):
  // word: una palabra
  // partialCounts: una lista parcial de la cuenta agregada
  int result = 0;
  for each v in partialCounts:
    result += ParseInt(v);
  Emit(result);
```

En resumen, cada documento se divide en palabras, y cada palabra se cuenta con valor inicial "1" por la función *map*, utilizando la palabra como la clave del resultado. La función *reduce* suma estos valores y emite el número de ocurrencias totales de cada palabra en cuestión.

2.1.3. Flujo de Datos

Desde la perspectiva del flujo de datos, la ejecución de MapReduce consiste en *M* tareas *map* y *R* tareas *reduce* independientes, cada una de las cuales puede depender de *M* tareas *map*. Generalmente, los resultados intermedios se particionan en *R* trozos para *R* tareas *reduce* [5]. La Figura 2.5: Flujo de datos de un trabajo MapReduce muestra el flujo de datos de alto nivel de un trabajo MapReduce.

La parte estática de MapReduce es una ordenación distribuida a alto nivel, mientras que los elementos dinámicos, que son los que puede definir la aplicación, son:

Proceso que lee la entrada

Divide los datos de entrada en bloques de 16 a 128 MB, siendo asignados cada uno de estos bloques a la función *map* correspondiente. Estos datos serán leídos de un almacenamiento estable (típicamente un sistema de ficheros distribuidos, puesto que normalmente los datos no caben en el sistema de ficheros de un solo nodo) y se generarán pares clave/valor. Un ejemplo común sería la lectura de un directorio entero, y la devolución de un registro por cada línea de cada fichero del directorio.

Función de mapeo (*map*)

Cada función map recibe una serie de pares clave/valor, los procesa individualmente, y devuelve cero o más pares clave-valor de salida. Los tipos de datos de la entrada y la salida de la función *map* pueden ser distintos.

Por ejemplo, si la aplicación realizase un conteo de palabras, la función *map* separaría la línea en palabras y sacaría como resultado la palabra como clave y un "1" como valor.

Función de reducción (*reduce*)

El sistema llama a la aplicación de la función *reduce* una vez por cada clave distinta en el orden establecido. El *reduce* puede iterar a través de los valores asociados con esa clave y obtener como resultado 0 o más valores.

En el ejemplo de la aplicación de conteo de palabras, la función *reduce* toma los valores de entrada, los suma y genera una única salida con la palabra y su suma final.

Función de partición

Las salidas de cada uno de los nodos *map* son asignadas a un nodo *reduce* en concreto a partir del resultado obtenido por la función de partición de la aplicación. Esta función devuelve el índice del *reduce* buscado, dada una clave y un número de nodos *reduce*.

Una función típica es hallar el valor hash de la clave y hacer módulo del número de nodos *reduce*.

Función de comparación

La entrada de cada *reduce* se obtiene de la máquina en la que se ejecutó y ordenó el *map* utilizando la función de comparación de la aplicación.

Esta función de comparación de claves se utiliza para ordenar los resultados finales emitidos por la función *reduce* antes de volver a la lista de claves.

Función de escritura de salida

Escribe la salida de la función *reduce* en un almacenamiento estable, típicamente un sistema de ficheros distribuido.

El principal beneficio de este modelo de programación es la simplicidad. El programador simplemente proporciona una descripción del algoritmo centrada en su funcionalidad.

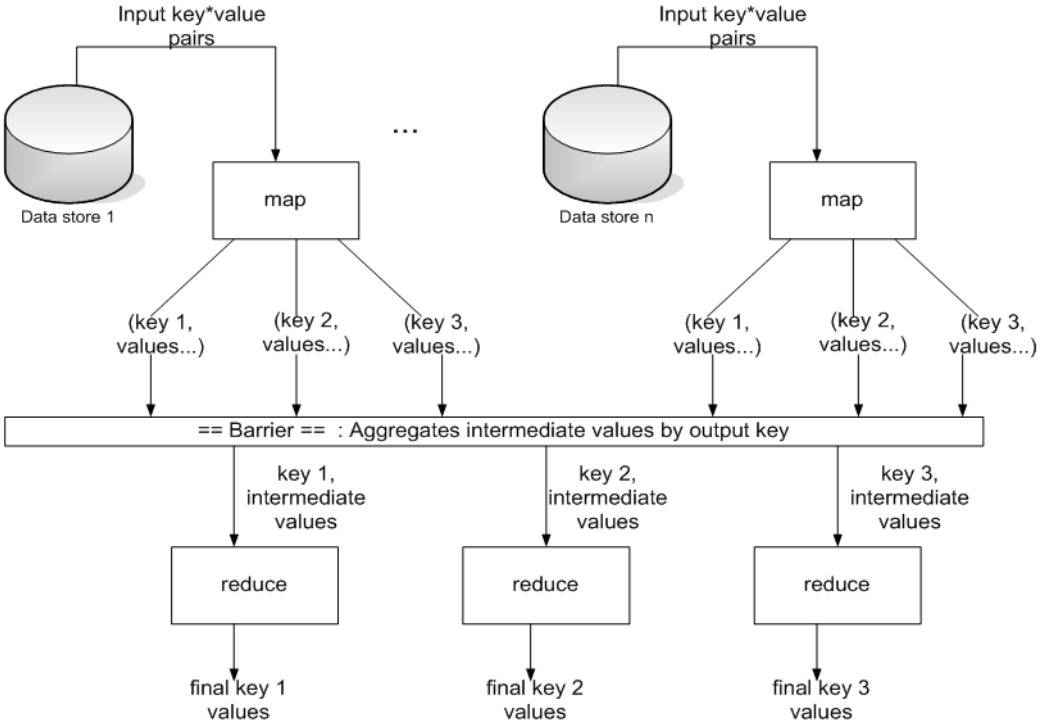


Figura 2.5: Flujo de datos de un trabajo MapReduce [8]

2.1.4. Implementación

Se pueden proponer muchas implementaciones distintas del modelo MapReduce. La elección correcta se basará fundamentalmente en la arquitectura en la que se desee implantar. Así, una implementación será adecuada para una máquina de memoria compartida, otra para un conjunto de máquinas con varios núcleos, e incluso otra para procesadores de tarjetas gráficas (*GPU*s), sistemas Cloud, etc.

El sistema de ejecución de MapReduce planifica tareas *map* y *reduce* sobre los recursos distribuidos, lo que le hace lidiar con muchos problemas, tales como la paralelización, el control de concurrencia, las comunicaciones por red y la tolerancia a fallos. De hecho, lleva a cabo varias optimizaciones para

disminuir el sobrecoste asociado a la planificación, comunicación de red y agrupamiento de resultados intermedios.

Las invocaciones *map* se distribuyen a lo largo de múltiples máquinas particionando automáticamente la información de entrada en un conjunto de *M* subconjuntos. Las invocaciones *reduce* se distribuyen particionando el espacio de claves intermedio en *R* trozos utilizando una función de particionado, así como un número de particiones *R*, que pueden ser personalizadas por el usuario. El sistema de ejecución de MapReduce reparte las tareas *map* y *reduce* entre los recursos distribuidos. Ambos, los pares de entrada *map* y los pares de salida *reduce*, son almacenados en un sistema de ficheros distribuidos la Figura 2.6: Flujo de ejecución de un trabajo MapReduce distribuido muestra el flujo global de ejecución de un trabajo MapReduce en un entorno *cluster* compuesto por un alto número de máquinas usando una aproximación maestro-esclavo [5].

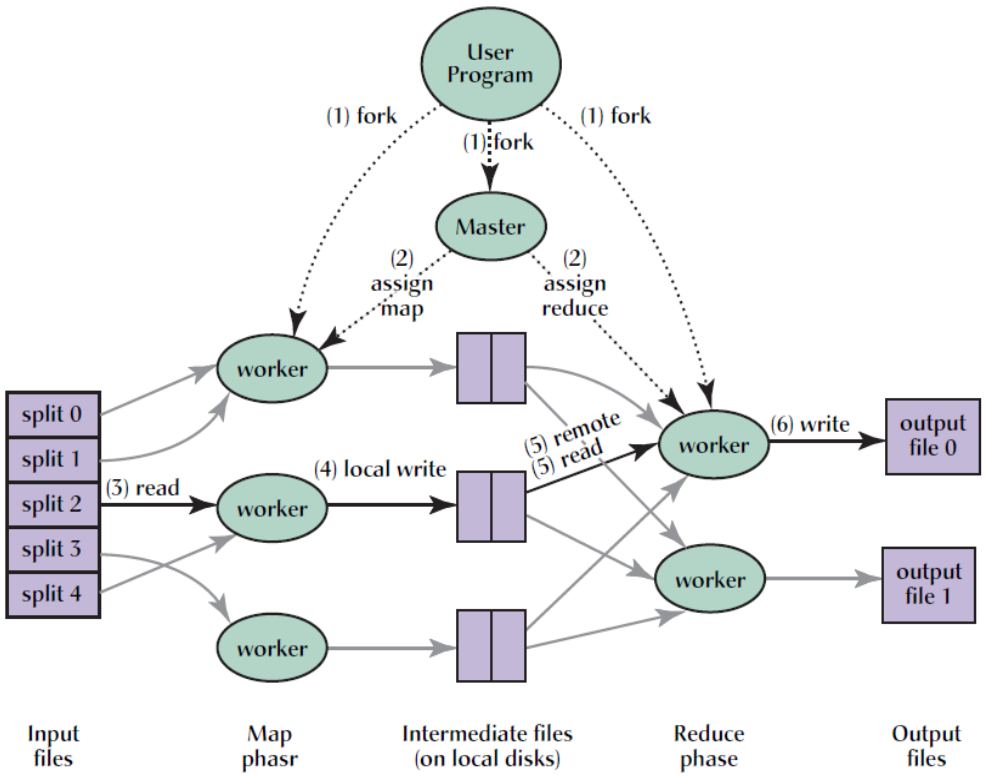


Figura 2.6: Flujo de ejecución de un trabajo MapReduce distribuido [5]

Cuando un usuario invoca una implementación de MapReduce, se ejecutarán por orden las siguientes acciones (los números de la siguiente lista se corresponden con las etiquetas numeradas de la Figura 2.6):

1. La biblioteca de funciones MapReduce que utilice el usuario en el programa primero divide los ficheros de entrada en *M* bloques de típicamente entre 16 y 128 Mb cada uno (pudiendo ser configurados por el usuario). A continuación inicia muchas copias del programa en el *cluster* de máquinas.
2. Una de dichas copias es diferente del resto: la copia maestra. El resto son trabajadores que ejecutan las tareas asignadas por el maestro. Existen *M* tareas *map* y *R* tareas *reduce* para asignar. El maestro elegirá nodos "ociosos" y les asignará a cada uno una tarea *map* o una *reduce.*
3. Cada trabajador al que se le asigna una tarea *map* lee los contenidos del subconjunto correspondiente. Obtiene los pares clave/valor a partir de la información suministrada y se los pasa a la función *map* definida por el usuario. Estos pares clave/valor intermedios generados por la función *map* se almacenan en memoria caché.
4. Periódicamente, estos pares en memoria caché se escriben a disco, particionados en *R* regiones por la función de partición. Las localizaciones de estos pares en disco se transmiten de vuelta al maestro, que es el responsable de redirigirlas a los nodos trabajadores *reduce*.
5. Cuando a un nodo *reduce* le notifica el nodo maestro estas localizaciones, utiliza llamadas remotas para leer el contenido almacenado localmente en los discos de los trabajadores *map*. Cuando ha finalizado de leer toda la información, la ordena por las claves intermedias, de tal forma que todas las ocurrencias asociadas a la misma clave vayan al mismo grupo. Este ordenamiento es necesario porque típicamente muchas claves diferentes se asignan a

la misma tarea *reduce*. Si la cantidad de información intermedia es demasiado grande para almacenarse en memoria, se utiliza un ordenamiento externo.

6. El nodo *reduce* itera sobre la información intermedia ordenada y por cada clave intermedia distinta encontrada, pasa dicha clave y el correspondiente conjunto de valores intermedios a la función *reduce* (también definida por el usuario). La salida de dicha función se añade a un fichero local de esta partición *reduce*.
7. Cuando todas las tareas *map* y *reduce* se hayan completado, el nodo maestro devuelve el control de nuevo al código del usuario.

Tras una ejecución exitosa, la salida del trabajo MapReduce está disponible en los *R* ficheros de salida (uno por tarea *reduce*, con nombres personalizables por el usuario). Por norma general, no hará falta combinar estos *R* ficheros en un único archivo pues se pasan a menudo como entrada a otra llamada MapReduce, o se utilizan desde otra aplicación distribuida que sea capaz de manejar información fraccionada en múltiples ficheros.

2.1.5. Fiabilidad y tolerancia a fallos

MapReduce consigue fiabilidad en sus ejecuciones gracias a su estrategia de reparto de operaciones sobre el conjunto de datos asignados a cada nodo. Las operaciones individuales utilizan operadores atómicos para nombrar ficheros de salida como comprobación para asegurarse de que no existen hilos conflictivos ejecutándose paralelamente. Asimismo, el nodo maestro intenta planificar operaciones *reduce* en el mismo nodo, o en el mismo rack en el que se encuentra el nodo que retiene la información sobre la que se está trabajando. Esta propiedad es deseable ya que conserva el ancho de banda de la red troncal del centro de datos.

Por su parte, la tolerancia a fallos es uno de los aspectos más importantes en un modelo de programación distribuido, ya que se está trabajando con volúmenes muy grandes de datos almacenados en decenas o centenas de

máquinas, con lo que la probabilidad de fallo de una máquina es alta. Se pueden distinguir dos tipos de fallos principalmente:

Fallo de un nodo esclavo

El nodo maestro mantiene varias estructuras de datos. Para cada tarea *map* o *reduce*, almacena el estado ("ocioso", "en progreso" o "completado") y la identidad del nodo esclavo en el que se ejecuta la tarea. El nodo maestro es el túnel a través del que la localización de las regiones de los ficheros intermedios se propaga de los nodos *map* a los nodos *reduce*. Por tanto, para cada tarea *map* completada, el maestro almacena la localización y tamaños de las R regiones de ficheros intermedios generadas por las tareas *map*. Cuando estas tareas se completen, se actualizarán estas localizaciones y se recibirá información acerca del tamaño de datos procesados. Además, esta información se suministrará incrementalmente a los nodos esclavos que ejecuten tareas *reduce* en estado "en progreso". En caso de que el nodo maestro no reciba una respuesta de un esclavo en una cantidad de tiempo determinada, lo marcará como caído y redirigirá a otros nodos el trabajo asignado a ese nodo. Cualquier tarea *map* completada por un nodo esclavo caído volverá a su estado inicial ("ocioso"), y por tanto se volverá seleccionable para planificarse en otro nodo esclavo. De igual forma, cualquier tarea *map* o *reduce* que se ejecute sobre un nodo esclavo caído se resetea a "ocioso" y se vuelve seleccionable.

Las tareas *map* completadas se re-ejecutan en caso de fallo debido a que sus resultados se almacenan en los discos locales de la máquina caída y se vuelven, por tanto, inaccesibles. Las tareas *reduce* completadas no necesitan ser re-ejecutadas ya que sus salidas se almacenan en un sistema de ficheros global. Cuando una tarea *map* se ejecuta primero en el nodo esclavo *A* y luego en el *B* (debido a que *A* ha fallado), todos los esclavos que estén ejecutando tareas *reduce* son notificados de dicha reejecución. Cualquier tarea *reduce* que

no haya leído todavía en ese momento la información del esclavo *A*, lo hará del *B*.

De las anteriores afirmaciones, se puede concluir que MapReduce es tolerante a fallos a gran escala de nodos esclavos.

Fallo del nodo maestro

Es sencillo hacer que el nodo maestro escriba puntos de control periódicos de las estructuras de datos descritas en el anterior epígrafe. Así, si la tarea maestro muere, se puede comenzar una nueva copia desde el último estado guardado. Sin embargo, dado que existe un único nodo maestro, se adopta de forma generalizada la decisión de abortar el trabajo MapReduce si ocurre esta incidencia y reiniciarlo cuando el usuario lo desee, ya que la probabilidad de fallo de un nodo individual se considera pequeña.

2.1.6. Implementaciones

MapReduce es útil en una amplia gama de aplicaciones. Además, el modelo MapReduce se ha adaptado a varios entornos informáticos, como los sistemas con muchos núcleos, entornos de computación voluntaria, entornos *cloud* dinámicos, y entornos móviles.

Existen multitud de implementaciones del modelo MapReduce de las que se pueden destacar [7]:

- El framework MapReduce de Google, implementado en *C++*, con interfaces en Python y Java.
- Hadoop, implementación de código abierto de MapReduce programada en Java que forma parte del proyecto Apache.
- Greenplum, implementación comercial de MapReduce, con soporte para Python, Perl, SQL y otros lenguajes [9].
- Phoenix, implementación de memoria compartida de MapReduce escrita en *C* [10].

- Disco, implementación de código abierto de MapReduce desarrollada por Nokia. Su núcleo está escrito en *Erlang* y los trabajos se suelen escribir en *Python* [11].

Por su carácter de código abierto y su orientación a la ejecución sobre *clusters* de ordenadores, en este proyecto se utilizará Hadoop. Se describe con detalle en la sección 2.3 de este capítulo.

2.1.7. Ejemplos de uso

La principal aplicación de MapReduce consiste en el procesamiento de información en conjuntos de datos homogéneos, siendo útil en un amplio espectro de aplicaciones y arquitecturas, entre las que se pueden destacar [12]:

- Generación de PDFs a gran escala: En 2007, el periódico *New York Times* necesitaba generar PDFs para 11.000.000 de artículos en forma de imágenes escaneadas del documento original. Para conseguirlo se emplearon: *Amazon Simple Storage Service* (S3), un servicio de almacenamiento barato que puede almacenar y recuperar cualquier cantidad de información en cualquier momento; *Amazon Elastic Compute Cloud* (EC2), un entorno de computación virtualizado diseñado para usarse en conjunto con otros servicios Amazon (especialmente S3); y Hadoop. El proceso consistió en que se enviaron 4 TB de artículos escaneados a S3 y que un clúster de máquinas EC2 fue configurado para distribuir la generación de PDFs vía Hadoop. Finalmente, empleando 100 instancias EC2, en 24 horas el periódico fue capaz de convertir 4 TB de artículos escaneados en 1.5 TB de documentos PDF [13].
- Información Geográfica: *Google Maps* ha utilizado MapReduce para solventar ciertos problemas como la localización de carreteras conectadas a una intersección dada, el renderizado de capas de mapas

o la búsqueda del punto de interés más cercano a una dirección o localización dada, etc.

- *PageRank*: El programa por excelencia de posicionamiento web utiliza MapReduce (con ligeras modificaciones) para regenerar completamente el índice de Google de páginas web y valorar cualquier tipo de documento de manera recursiva.
- Construcción de índices inversos: En este escenario, la función *map* parsea cada documento, y emite una secuencia de pares <palabra, id_documento>. La función *reduce* acepta todos los pares de una palabra dada, ordena los correspondientes ids de los documentos y emite una lista de pares <palabra, lista (id_documento)>. El conjunto de todos los pares de salida forma un índice inverso simple.
- Aprendizaje máquina: Se ha utilizado MapReduce para implementar distintos tipos de algoritmos de aprendizaje máquina, demostrando aceleración lineal incrementando el número de procesadores. Un ejemplo de este tipo de aplicaciones es el *framework* de código libre Apache Mahout, que se apoya en Apache Hadoop.

2.2. Google File System

2.2.1. Introducción

El sistema de archivos de Google, GFS (*Google File System*), es un sistema de almacenamiento basado en las necesidades de Google diseñado en octubre de 2003 [6].

Al no ser un sistema de archivos de uso generalista, GFS, ha sido diseñado teniendo en cuenta las siguiente premisas: que un componente falle es la norma no la excepción, los archivos son enormes (archivos de muchos GB son comunes), es muy común que un archivo cambie porque se le añaden datos pero es muy raro que se sobrescriban los datos existentes, el codiseño de las

aplicaciones y de la API del sistema de archivos proporciona un beneficio global.

2.2.2. Diseño

- El sistema está construido para que el fallo de un componente no le afecte.
- El sistema almacena grandes archivos
- La mayoría del trabajo consiste en dos tipos de lecturas: grandes lecturas de datos y pequeñas lecturas aleatorias
- La carga de trabajo también consiste en añadir grandes secuencias de datos a archivos.
- El sistema debe ser diseñado para ofrecer concurrencia a múltiples clientes que quieran el mismo archivo.
- Tener un gran ancho de banda prolongadamente es más importante que una baja latencia.

2.2.3. Arquitectura

La arquitectura de GFS consiste en una gran cantidad de equipos con características parecidas a las de un ordenador personal, pero que sólo son parte de un sistema que los engloba. Ninguno de ellos es esencial, son indistinguibles, como células en un organismo.

Google fue de los primeros en hacer uso de una infraestructura así. Pero el simple hecho de poner un montón de PCs juntos no permite explotar su potencia [14].

Crearon su propia versión modificada de Linux sobre la que diseñaron un sistema de ficheros llamado *Google File System* (GFS). El objetivo de GFS era

poder almacenar información sobre un sistema de ficheros distribuido de forma segura y que soportase una gran carga de trabajo.

GFS es un sistema distribuido que no guarda todos los datos de un fichero en un solo disco duro ni en un solo ordenador, sino que emplea toda una red de ordenadores para hacerlo. GFS divide los datos en trozos, y hace al menos 3 copias de cada trozo. Cada copia irá a un ordenador diferente dentro de la red de Google.

Se emplean dos tipos de servidores, los maestros y los *ChunkServers*. Los primeros almacenan la situación física de los trozos, *chunks*, que componen los ficheros, así como la jerarquía de ficheros y directorios, los metadatos.

Para no saturar a los maestros, los clientes que acceden a ellos cachean, almacenan temporalmente los datos relativos a qué *ChunkServers* contactar para un determinado fichero.

Los *ChunkServers* almacenan los trozos en sí. Los ficheros en GFS son un poco especiales. Normalmente son ficheros muy grandes, más de 100MB, y hay unos cuantos millones de ellos. No se pueden sobrescribir, sólo es posible añadir (*append*) datos al final del fichero. La operación "añadir" es atómica, por lo que no es necesario poseer un sistema de bloqueos muy sofisticado. Si un fichero está realizando una operación "añadir", el resto debe esperar.

Los ficheros serán leídos completamente o por partes. Existen *ChunkServers* primarios y secundarios que almacenan las copias de los *chunks* del primario.

Cuando se escribe en un fichero, el cliente solicita la escritura al *ChunkServer* primario, el primario manda los datos a todas las réplicas y éstas los almacenan a la orden del primario. El primario asigna a los datos a escribir unos números de serie y realiza la escritura. Entonces solicita a los secundarios que ejecuten la escritura con el número de serie. El objetivo es que en el peor de los casos la escritura se realice en el primario y en algunos de los secundarios.

Para no saturar la red, uno de los objetivos de todo este sistema, los datos se envían a los *ChunkServers* de forma lineal, es decir, se envían al primero, que conforme los recibe, los envía al siguiente, como si fuese una cuerda que va pasando por varios aros.

GFS además es capaz de asignar más recursos a aquellos ficheros que más se emplean, de forma que cuanto más importante es un fichero, más copias de seguridad se harán de sus trozos, lo que protege mejor esa información y además, al haber más copias disponibles, acelera el acceso.

GFS no está integrado en el sistema operativo, sino que es accesible a través de librerías, de forma que puede ser empleado desde distintos sistemas operativos.

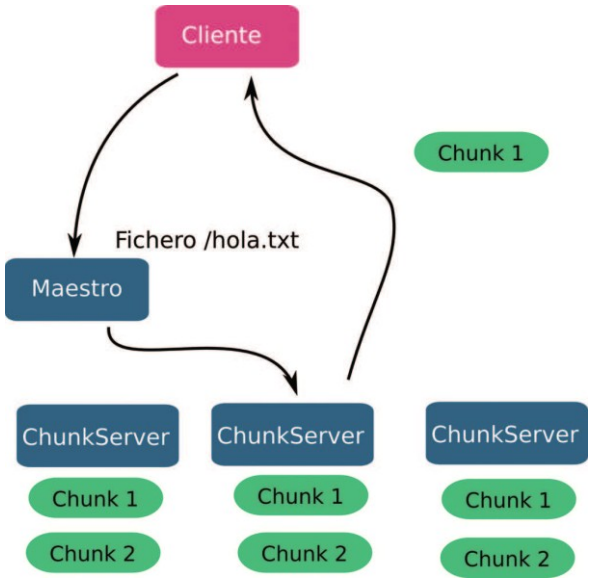


Figura 2.7: Arquitectura de GFS

2.2.4. Máster (Maestro)

Tener un solo maestro simplifica el diseño y permite tener sofisticados métodos de emplazamiento y replicación de trozos usando un conocimiento global.

Se intentan minimizar las lecturas y escrituras para evitar los cuellos de botella. Los clientes nunca escriben datos a través del maestro sino que el cliente le pregunta al maestro que con qué *ChunkServer* puede contactar.

2.2.5. Tamaño de los trozos

El tamaño de los trozos en los que se dividen los ficheros es una de las claves del diseño. GFS usa 64 MB que es mucho más grande que el tamaño del bloque de un sistema de archivos típico. La mayor objeción contra este gran tamaño es la fragmentación interna frente a las muchas ventajas que ofrece un gran tamaño de trozo. Primero, reduce las interacciones del cliente con el maestro ya que las lecturas y escrituras en el mismo trozo sólo requieren una petición inicial al maestro para obtener la localización del trozo. Segundo, al tener un trozo grande es muy probable que se puedan realizar muchas operaciones en un trozo lo que reduce la sobrecarga en la red manteniendo una conexión TCP persistente con el *ChunkServer* durante un periodo de tiempo. Por último, reduce el tamaño de los metadatos almacenados en el maestro lo que permite guardar los metadatos en memoria.

2.2.6. Metadatos

El maestro almacena tres tipos importantes de metadatos: el espacio de nombres de fichero y de trozos, la correspondencia de archivos a trozos y la localización de las réplicas de los trozos. Toda la información está guardada en memoria lo que implica que las operaciones son rápidas. Además, es fácil y eficiente para el maestro realizar comprobaciones periódicas que son usadas para la recolección de basura, la re-replicación en caso de fallos en los *ChunkServer* y la migración de trozos para el balanceo de carga.

El maestro no guarda un registro persistente de qué *ChunkServer* tiene una réplica de un determinado trozo sino que pide esta información a los *ChunkServer* al comenzar. El maestro puede actualizar su información guardada después de esto ya que él controla todos los *ChunkServer* y monitoriza el estado de estos con mensajes regulares de *HeardBeat*.

2.3. Hadoop

2.3.1. Introducción

Como ya se comentó en la sección 1.1, Hadoop es una de las implementaciones de código abierto de MapReduce. Hadoop es un proyecto administrado por *Apache Software Foundation*. Es un *Framework* para el desarrollo de aplicaciones de procesamiento paralelo que utiliza MapReduce. Permite a las aplicaciones trabajar con miles de nodos y petabytes de datos [15].

Muchos colaboradores de Hadoop trabajan en algunas de las compañías más grandes del mundo de la tecnología. Originalmente fue desarrollado y empleado por las grandes empresas dominantes en la Web, como Yahoo y Facebook. Ahora Hadoop es muy utilizado en finanzas, tecnología, telecomunicaciones, medios y entretenimiento, gobierno, instituciones de investigación y otros mercados con gran cantidad de datos. Con Hadoop, las empresas pueden explorar datos complejos mediante el análisis personalizado adaptado a sus datos y necesidades.

Hadoop fue diseñado para analizar de forma rápida y fiable los datos estructurados y los datos complejos. Como resultado, muchas empresas han optado por desplegar Hadoop junto a sus demás sistemas informáticos, lo que les permite combinar los datos antiguos y los nuevos de distintas formas nuevas [16].

Hadoop es muy útil cuando vamos a realizar proyectos que necesiten de escalabilidad. Al disponer los datos de forma distribuida, la búsqueda se puede realizar muy rápidamente ya que Hadoop puede acceder a ella de forma paralela. Y aunque los datos estén distribuidos, no hay que preocuparse de fallos ya que dispone de un sistema de seguridad.

El proyecto Hadoop consta de una serie de subproyectos, que vienen a complementar su funcionalidad, profundizando en aspectos como el tratamiento, flujo e importación de datos, la monitorización de trabajos, etc. La Figura 2.8: Ecosistema de Hadoop con sus principales subproyectos muestra el ecosistema de Hadoop.

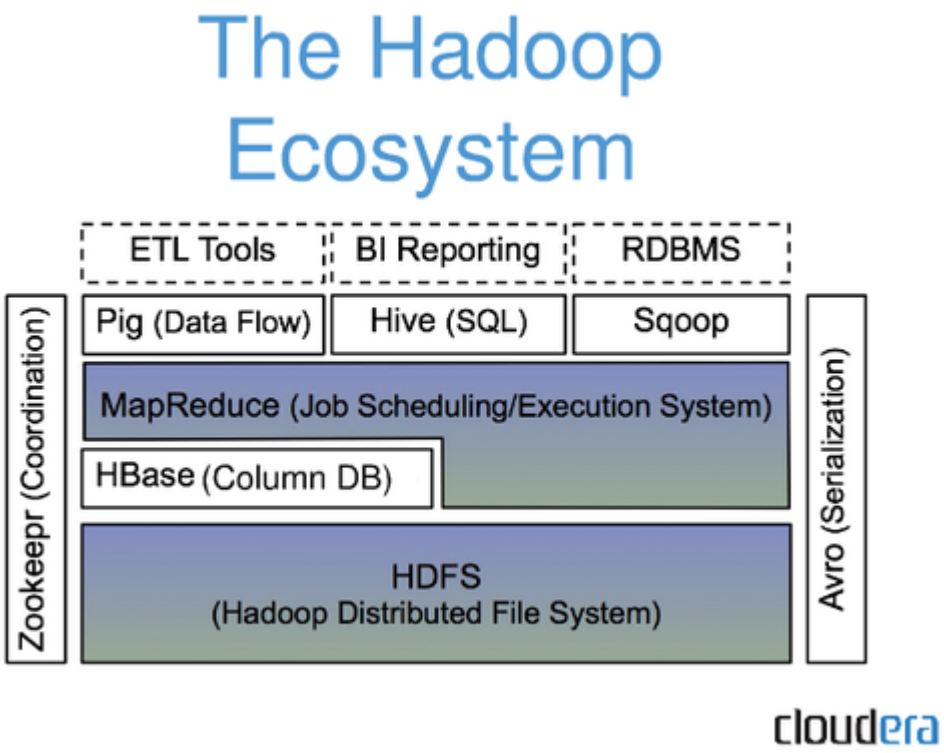


Figura 2.8: Ecosistema de Hadoop con sus principales subproyectos [17]

2.3.2. Arquitectura

La arquitectura de Hadoop se vertebra sobre tres pilares fundamentales:

- **Sistema de ficheros**: Hadoop se apoya para su funcionamiento en un sistema de ficheros distribuido, denominado HDFS.
- **Hadoop MapReduce**: El motor de Hadoop consta de un planificador de trabajos MapReduce, así como de una serie de nodos encargados de llevarlos a cabo.
- **Hadoop Common**: conjunto de utilidades que posibilitan la integración de subproyectos de Hadoop.

Sobre el sistema de ficheros se ubica el motor de MapReduce, que consiste en un planificador de trabajos denominado *JobTracker*, a través del que las aplicaciones cliente envían trabajos MapReduce. Este planificador envía el flujo de trabajo entrante a los nodos *TaskTracker* disponibles en el *cluster*, que se ocuparán de ejecutar las funciones *map* y *reduce* en cada nodo. El planificador trata de mantener esos trabajos tan cerca de la máquina que ha emitido esa información como sea posible. Si el trabajo no puede ser ubicado en el nodo actual en el que la información reside, se le da prioridad a nodos en el mismo rack. Esto reduce el tráfico de red en la red principal del *cluster*. Si un *TaskTracker* falla o sufre un *timeout*, esa parte del trabajo se replanifica, sin embargo, si falla el *JobTracker*, toda la información pendiente de procesarse se pierde.

El *JobTracker* registra los trabajos pendientes de ejecución que residan en el sistema de ficheros. Cuando arranca un *JobTracker* busca esa información, de tal forma que pueda volver a empezar el trabajo a partir del punto en el que se quedó. En versiones de Hadoop anteriores a la utilizada en este proyecto (Hadoop 0.20.2), si se reiniciaba un *JobTracker* toda esta información se perdía.

Sin embargo, esta aproximación presenta alguna limitación:

- Si un *TaskTracker* es muy lento, puede retrasar la operación MapReduce completa, especialmente hacia el final del trabajo, donde todos los procesos trabajadores lanzados pueden acabar esperando por una única tarea lenta. Sin embargo, con la opción de *ejecución especulativa* (ejecución de un código cuyos resultados puede que no sean necesarios) activada, una única tarea puede ser ejecutada en múltiples nodos esclavos.

2.3.3. Modos de ejecución

Hadoop se puede ejecutar de tres formas distintas:

- **Modo local / *standalone*:** Por defecto, Hadoop está configurado para ejecutarse en modo no-distribuido como un proceso Java aislado. Esto es útil para depuración.
- **Modo pseudo-distribuido**: Hadoop también puede ejecutarse en un único modo en un modo pseudo-distribuido donde cada demonio Hadoop se ejecuta en un proceso Java diferente.
- **Modo distribuido:** Esta es la forma de aprovechar toda la potencia de Hadoop, ya que se maximiza el paralelismo de procesos y se utilizan todos los recursos disponibles del *cluster* en el que se va a configurar Hadoop.

2.3.4. Ejemplos de uso

Existe un gran número de aplicaciones y empresas que utilizan Hadoop en sus *clusters*. Entre las más relevantes se pueden citar [18]:

- **Cloudera, Inc**: Una de las principales compañías que dan soporte y formación en Hadoop. Tienen su propia distribución de Hadoop y uno de sus trabajadores, TomWhite, ha escrito uno de los libros de referencia de Hadoop [2]**.**

- **Facebook**: Utilizan Hadoop para almacenar copias de *logs* internos y fuentes de datos de grandes dimensiones, así como de fuente de informes/análisis y aprendizaje máquina. También han utilizado el sub-proyecto de Hadoop, *Hive* y han construido una implementación FUSE (*Filesystem in Userspace*) sobre HDFS.
- **Google**: Uno de los pilares clave para el desarrollo de Hadoop. Junto con IBM, lleva a cabo la iniciativa "*University Initiative to Address Internet-Scale Computing Challenges*" que intenta mejorar el conocimiento de los estudiantes sobre computación paralela para adaptarse a modelos de programación y paradigmas como MapReduce.
- **Twitter**: Utilizan Hadoop para almacenar y procesar *tweets* y ficheros de *log*.
- **Yahoo!:** Una de las empresas promotoras de Hadoop, lo utilizan en más de 40.000 máquinas para temas de búsqueda en web y sistemas auxiliares.

2.4. HDFS: Hadoop Distributed File System

HDFS es el sistema de ficheros utilizado por defecto en Hadoop y está inspirado en el GFS de Google. Es un sistema de ficheros distribuido estructurado en bloques. Los ficheros individuales se parten en bloques de un tamaño fijo. Estos bloques se almacenan a lo largo de un *cluster* de una o más máquinas con capacidad de almacenamiento de información. Las máquinas objetivo que manejan cada bloque se eligen de forma aleatoria por un criterio "bloque a bloque". Así, el acceso a un fichero puede necesitar la cooperación de múltiples máquinas, pero se soportan tamaños de fichero mucho más grandes que en un sistema de ficheros de una única máquina, ya que estos ficheros pueden necesitar más espacio del que tenga un único disco duro.

2.4.1. Arquitectura

El sistema de ficheros se construye a partir de un *cluster* de nodos de información (*DataNodes*), cada uno de los cuales envía bloques de información por la red, usando un protocolo de bloques específico del HDFS. También mandan la misma información por HTTP, permitiendo el acceso a todo el contenido desde un navegador web u otro cliente. Los nodos de información pueden comunicarse para rebalancear su carga, copiarse ficheros y mantener el nivel de replicación de información en un nivel alto. Un ejemplo de esta arquitectura se puede observar en la Figura 2.9.

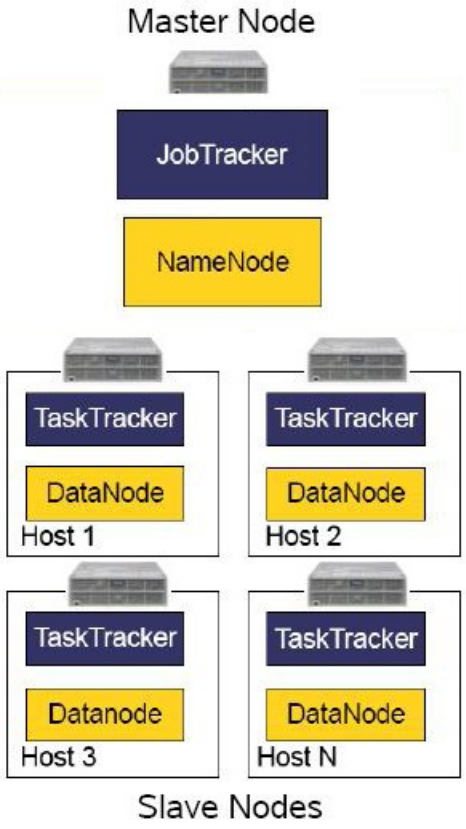


Figura 2.9: Arquitectura del HDFS

El *NameNode* almacena las modificaciones del sistema de ficheros en un log que se añade a un fichero nativo del sistema (*EditLogs)*. Cuando arranca un *NameNode* después de una caída, lee el estado del HDFS de un fichero de imagen (*FSImage)* y luego aplica las modificaciones sobre el mismo a partir

del fichero *EditLogs*. A continuación escribe el nuevo estado en *FSImage* y arranca las operaciones normales con un fichero de modificaciones vacío. Debido a que el *NameNode* fusiona la información de ambos ficheros solo durante el arranque, el fichero de modificaciones puede llegar a crecer mucho a lo largo del tiempo en un *cluster* de bastantes máquinas, y además, el siguiente reinicio del *NameNode* tardará más tiempo.

Un *NameNode* secundario fusiona periódicamente los ficheros *FSImage* y *EditLogs* y mantiene el tamaño del log de modificaciones dentro de un tamaño adecuado.

2.4.2. Características

Con respecto al tema de la disponibilidad, este sistema de ficheros requiere un único servidor, el *NameNode*, que constituye un punto de fallo crítico de una instalación HDFS; si el *NameNode* se cae, el sistema de ficheros entero estará caído. Cuando se levante de nuevo, el *NameNode* debe re-ejecutar todas las operaciones pendientes. El sistema de ficheros HDFS es fiable ya que replica la información entre varios anfitriones. Con el valor de replicación por defecto -3- la información se almacena en tres nodos: dos copias en el mismo *rack*, y otra en uno diferente.

En cuanto a seguridad, comentar que HDFS sigue el modelo POSIX de usuarios y grupos: cada fichero o directorio tiene 3 permisos asociados (lectura, escritura y ejecución), cada uno de los cuales está asociado a tres tipos de usuarios: el propietario del fichero, los usuarios que pertenecen al mismo grupo que el propietario, y el resto de usuarios del sistema. Como el HDFS no facilita el espectro completo del sistema POSIX, algunas combinaciones de bits no tendrán efecto. Por ejemplo, ningún fichero puede ser ejecutado ya que el *bit +x* no se puede asociar a ningún fichero (sólo a directorios). Tampoco se puede escribir en un fichero existente, aunque se puede especificar el *bit +w*.

Entre las tareas adicionales que ejecuta el HDFS se pueden citar:

- Rebalanceo de bloques: Esto se puede conseguir a través de la herramienta de balanceo automática incluida con Hadoop. Esta herramienta balanceará de forma inteligente los bloques a lo largo de los nodos para conseguir una mejor distribución de los mismos dentro de un umbral preestablecido, expresado como un porcentaje (por defecto el 10%). Porcentajes más pequeños hacen que los nodos estén todavía más balanceados, pero pueden necesitar más tiempo para conseguir este estado. El balanceo perfecto (0%) actualmente es muy difícil de conseguir.
- Copia de grandes conjuntos de ficheros: Cuando se migra un gran número de ficheros de una localización a otra (bien desde el HDFS de un *cluster* a otro, bien desde S3 a HDFS o viceversa, etc.), la tarea se debería dividir entre múltiples nodos para permitirles compartir el ancho de banda necesario para el proceso. Hadoop incluye una herramienta *distcp* con este propósito.
- Exclusión de nodos*:* Además de poder añadir nodos al *cluster* de forma dinámica, también se pueden eliminar del mismo mientras está activo. Sin embargo, si no se toman precauciones en esta operación, puede derivar en una pérdida de datos, ya que manejan una única copia de uno o más bloques de ficheros. Los nodos deben retirarse de manera planificada, de forma que permita al HDFS asegurarse de que no haya bloques que sean replicados íntegramente dentro de los *DataNode*s que se vayan a retirar. Para ello, facilita un mecanismo de decomisión de nodos que asegura que este proceso se lleva a cabo de forma segura.
- Verificación del sistema**:** Mediante el comando `bin/Hadoop fsck [ruta] [opciones]` se puede obtener distinta información del sistema de ficheros.

3. DISEÑO E IMPLEMENTACIÓN

En este capítulo se describen los aspectos de instalación y configuración de Hadoop. Se explica el entorno de pruebas, especificando las máquinas y direcciones IP utilizadas, las funciones de maestro/esclavo y la conectividad. Se describirán las distintas pruebas realizadas y cómo se han resuelto con la herramienta Hadoop.

3.1. Instalación y Configuración de Hadoop

3.1.1. Java

Hadoop necesita Java para ejecutarse. Para la realización del proyecto se ha utilizado la versión 1.6.0_22. En el sito oficial de Java se puede descargar la máquina virtual. Una vez descargado el paquete lo instalamos siguiendo los pasos del instalador.

Después de la instalación hemos de comprobar que la máquina es la correcta:

```
mamenpala@semnet:~$ java -version
java version "1.6.0_22"
Java(TM) SE Runtime Environment (build 1.6.0_22-b04)
Java HotSpot(TM) 64-Bit Server VM (build 17.1-b03, mixed mode)
```

3.1.2. Configuración ssh

Hadoop requiere que los procesos puedan comunicarse de forma segura entre sí con los diferentes equipos en la red, para esto es necesaria la utilización de ssh.

Por tanto, debemos asegurarnos de que podemos iniciar sesión ssh sin contraseña en las máquinas del *cluster*. A continuación se procede a realizar la configuración para permitir la comunicación ssh entre las distintas máquinas.

Primero se genera una clave ssh con un password vacío:

```
mamenpala@semnet:~$ ssh-keygen -t rsa -P '' -f ~/.ssh/id_rsa
Generating public/private rsa key pair.
Your identification has been saved in /home/mamenpala/.ssh/id_rsa.
Your public key has been saved in /home/mamenpala/.ssh/id_rsa.pub.
The key fingerprint is:
0e:fa:64:06:9e:85:1c:c7:70:63:8d:87:c2:22:f1:23 mamenpala@semnet
The key's randomart image is:
+--[ RSA 2048]----+
|.. .. ++         |
|... o=o.o        |
|E.o...o.         |
| . o +           |
|    + o S        |
|   . = o         |
|    + + .        |
|     =           |
|      .          |
+-----------------+
```

Una vez generada la clave pública es necesario guardarla en el archivo de claves autorizadas para el usuario actual en cada una de las máquinas que forman parte del *cluster*:

```
mamenpala@semnet:~$ cat ~/.ssh/id_rsa.pub >> ~/.ssh/authorized_keys
```

Es necesario poder acceder desde cada nodo hacia los demás nodos del *cluster*.

```
mamenpala@semnet:~$ ssh-copy-id -i .ssh/id_rsa.pub {ip_slave}
```

Probaremos la conexión a localhost para comprobar que lo hemos configurado todo correctamente. No nos debería pedir contraseña:

```
mamenpala@semnet:~$ ssh localhost
The authenticity of host 'localhost (::1)' can't be established.
RSA key fingerprint is
76:09:2f:5f:63:7d:fa:f2:43:91:a3:cc:aa:2a:61:24.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added 'localhost' (RSA) to the list of known
hosts.
Linux semnet 2.6.32-5-amd64 #1 SMP Sat Oct 30 14:18:21 UTC 2010
x86_64

The programs included with the Debian GNU/Linux system are free
software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
Last login: Sun Sep 18 17:12:57 2011 from 27.red-2-136-
88.dynamicip.rima-tde.net
```

En caso de que falle, podemos hacer lo siguiente para ver dónde está el problema:

- Habilitar el modo debug de ssh: `ssh -vvv localhost` y ver dónde puede estar el problema.
- Comprobar la configuración del servidor SSH en `/etc/ssh/sshd_config`, en particular las opciones `PubkeyAuthentication`, que debería tener como parámetro `yes`, y `AllowUsers` activa. En caso de hacer algún cambio en este fichero, es necesario reiniciar el servidor ssh: `sudo service ssh restart`

3.1.3. Instalación de Hadoop

Para descargar Hadoop, basta con ir a la página de descarga de Apache y descargarse el fichero comprimido en *.tar.gz*.

http://hadoop.apache.org/hdfs/releases.html

La versión de Hadoop utilizada en este proyecto es Hadoop 0.20.2

Después de descargar el fichero hay que descomprimirlo en el directorio deseado, cuya ruta se utilizará en la variable de entorno HADOOP_HOME

3.1.4. Configuración de Hadoop

Vamos a configurar el sistema para un *cluster* de 4 nodos. A continuación se muestran los ficheros de configuración que hay que modificar dependiendo de la estructura que tenga el *cluster* a implementar:

hadoop-env.sh

En este fichero deben aparecer las variables de entorno requeridas en nuestra configuración. Para introducirlas, editamos el fichero que se encuentra en `$HADOOP_HOME/conf/hadoop-env.sh` y añadimos las variables de entorno requeridas

```
# The only required environment variable is JAVA_HOME.  All others are
# optional.  When running a distributed configuration it is best to
# set JAVA_HOME in this file, so that it is correctly defined on
# remote nodes.

# The java implementation to use.  Required.
# export JAVA_HOME=/usr/lib/j2sdk1.5-sun
export JAVA_HOME=/usr/lib/jvm/java-6-sun
export PATH=/usr/lib/jvm/java-6-sun/bin:${PATH}
export HADOOP_HOME=/home/mamenpala/proyecto/hadoop-0.20.2
export PATH=$PATH:$HADOOP_HOME/bin
```

Ficheros conf/*-site.xml

A continuación, se va a configurar el directorio donde Hadoop va a guardar los ficheros de datos, los puertos que escuchará, etc. En este caso, el *cluster* lo compondrá una máquina maestra y tres esclavas. Estos ficheros deben guardarse en todas las máquinas del *cluster.*

El fichero `conf/core-site.xml` determina el nombre del sistema de archivos y el puerto a través del cual recibirá las peticiones del cliente. El fichero `conf/core-site.xml` quedará de la siguiente manera:

```
<?xml version="1.0"?>
<?xml-stylesheet type="text/xsl" href="configuration.xsl"?>

<!-- Put site-specific property overrides in this file. -->

<configuration>
       <property>
              <name>fs.default.name</name>
              <value>hdfs://master:9000</value>
       </property>
</configuration>
```

Para especificarle a Hadoop en donde almacenar los bloques de datos del DFS se puede usar la propiedad `dfs.data.dir` y como valor la ruta del directorio, en el archivo `conf/hdfs-site.xml`. En este caso no se está usando esta propiedad, con lo que se considera la ruta por defecto que es `/tmp/hadoop-${user.name}/dfs/data`. En éste archivo es necesario configurar la replicación de los bloques. El valor predeterminado es 3, pero en éste caso hay cuatro nodos así que se puede poner un valor de `dfs.replication` de 4. También se puede configurar las distintas direcciones y puertos en los que escucharán los distintos nodos.

El fichero `conf/hdfs-site.xml` quedará de la siguiente manera:

```
<?xml version="1.0"?>
<?xml-stylesheet type="text/xsl" href="configuration.xsl"?>

<!-- Put site-specific property overrides in this file. -->

<configuration>
	<property>
		<name>dfs.replication</name>
		<value>4</value>
	</property>

	<property>
		<name>dfs.secondary.http.address</name>
		<value>0.0.0.0:41090</value>
		<description>
			The secondary namenode http server address and
			port.
			If the port is 0 then the server will start on a
			free port.
		</description>
	</property>

	<property>
		<name>dfs.datanode.address</name>
		<value>0.0.0.0:41010</value>
		<description>
			The address where the datanode server will
			listen to.
			If the port is 0 then the server will start on a
			free port.
		</description>
	</property>

	<property>
		<name>dfs.datanode.http.address</name>
		<value>0.0.0.0:41075</value>
		<description>
			The datanode http server address and port.
			If the port is 0 then the server will start on a
			free port.
		</description>
	</property>
```

```
        <property>
                <name>dfs.datanode.ipc.address</name>
                <value>0.0.0.0:41020</value>
                <description>
                        The datanode ipc server address and port.
                        If the port is 0 then the server will start on a
                        free port.
                </description>
        </property>

        <property>
                <name>dfs.http.address</name>
                <value>163.117.141.188:40070</value>
                <description>
                        The address and the base port where the dfs
                        namenode web ui will listen on.
                        If the port is 0 then the server will start on a
                        free port.
                </description>
        </property>

        <property>
                <name>dfs.datanode.https.address</name>
                <value>0.0.0.0:41475</value>
        </property>

        <property>
                <name>dfs.https.address</name>
                <value>0.0.0.0:41470</value>
        </property>

</configuration>
```

Y finalmente en el fichero `conf/mapred-site.xml` se especifica la ubicación del *JobTracker*. En este archivo también se puede configurar donde se almacenarán los archivos temporales usando la propiedad `hadoop.tmp.dir`, aunque en este caso no se está usando esta propiedad, con lo que se considera la ruta por defecto que es `/tmp/hadoop-${user.name}`.

El fichero `conf/mapred-site.xml` quedará de la siguiente manera:

```
<?xml version="1.0"?>
<?xml-stylesheet type="text/xsl" href="configuration.xsl"?>

<!-- Put site-specific property overrides in this file. -->

<configuration>

        <property>
                <name>mapred.job.tracker</name>
                <value>master:9001</value>
        </property>

        <property>
```

```
            <name>mapred.job.tracker.http.address</name>
            <value>163.117.141.188:40030</value>
            <description>
                  The job tracker http server address and port the
                  server will listen on.
                  If the port is 0 then the server will start on a
                  Free port.
            </description>
      </property>

      <property>
            <name>mapred.task.tracker.http.address</name>
            <value>0.0.0.0:41060</value>
            <description>
                  The task tracker http server address and port.
                  If the port is 0 then the server will start on a
                  free port.
            </description>
      </property>
</configuration>
```

Archivos master y slave:

En el nodo maestro debemos editar el archivo `conf/masters`. Este archivo contiene el nombre del host que será el nodo maestro; agregamos la siguiente línea:

```
master
```

En los cuatro nodos debemos indicar cuáles serán los esclavos, para esto editamos el archivo `conf/slaves` agregando las siguientes líneas:

```
master
slave01
slave02
slave03
```

Con las líneas anteriores, estamos indicando al HDFS que va a tener un maestro que será master y cuatro esclavos que serán master, slave01, slave02 y slave03. El nodo principal o maestro actuará a la vez como esclavo. Con esta configuración se aprovechan al máximo los recursos, ya que las tareas como maestro consumen menos recursos que las tareas como esclavos.

3.1.5. Ejecución de Hadoop

Formateo del nodo

Antes de empezar, es necesario formatear el sistema de archivos HDFS del nodo. Para ello, ejecutaremos lo siguiente:

```
mamenpala@semnet:~$ hadoop namenode -format
11/09/18 19:49:17 INFO namenode.NameNode: STARTUP_MSG:
/************************************************************
STARTUP_MSG: Starting NameNode
STARTUP_MSG:   host = semnet/127.0.0.1
STARTUP_MSG:   args = [-format]
STARTUP_MSG:   version = 0.20.2
STARTUP_MSG:   build =
https://svn.apache.org/repos/asf/hadoop/common/branches/branch-0.20 -r
911707; compiled by 'chrisdo' on Fri Feb 19 08:07:34 UTC 2010
************************************************************/
11/09/18 19:49:17 INFO namenode.FSNamesystem: fsOwner=mamenpala,mamenpala
11/09/18 19:49:17 INFO namenode.FSNamesystem: supergroup=supergroup
11/09/18 19:49:17 INFO namenode.FSNamesystem: isPermissionEnabled=true
11/09/18 19:49:17 INFO common.Storage: Image file of size 99 saved in 0
seconds.
11/09/18 19:49:17 INFO common.Storage: Storage directory /tmp/hadoop-
mamenpala/dfs/name has been successfully formatted.
11/09/18 19:49:17 INFO namenode.NameNode: SHUTDOWN_MSG:
/************************************************************
SHUTDOWN_MSG: Shutting down NameNode at semnet/127.0.0.1
************************************************************/
```

Arranque y parada del *cluster*

Para arrancar los demonios de HDFS, hay que ejecutar el comando

```
mamenpala@semnet:~$ start-dfs.sh
```

A continuación iniciamos los demonios de MapReduce, ejecutando en el *JobTracker*:

```
mamenpala@semnet:~$ start-mapred.sh
```

Esto arrancará los "nodos" *NameNode*, *DataNode*, *JobTracker* y el *TaskTracker* en la máquina.

Se pueden ver los procesos ejecutados en cada máquina usando el comando jps.

```
mamenpala@semnet:~$ jps
29610 DataNode
29906 TaskTracker
23746 Jps
29497 NameNode
29713 SecondaryNameNode
29798 JobTracker
```

Los procesos *NameNode*, *SecondaryNameNode* y *JobTracker* son procesos que corren en el maestro, mientras que los procesos *TaskTracker* y *DataNode* deben ejecutarse en los esclavos. En el maestro se presentan los cinco procesos debido a que este nodo funciona como maestro y como esclavo a la vez.

Para detener los demonios del sistema de archivos y de MapReduce, respectivamente, ejecutamos:

```
stop-mapred.sh
stop-dfs.sh
```

3.2. Entorno de pruebas

A continuación se muestra el entorno de pruebas en el que se ha montado el *cluster* e instalado la plataforma Hadoop:

3.2.1. Máquinas

Se han utilizado 4 máquinas que están ubicadas en un laboratorio de la universidad. Estas máquinas se distribuyen de la siguiente forma:

- Una de las máquinas trabaja como maestro en el *cluster*. Esta máquina es la encargada de la administración de los procesos, pero también podrá realizar la labor de esclavo.
- Las otras tres máquinas trabajan como esclavos, sobre los que se ejecutan los procesos MapReduce.

Las direcciones IP de estas máquinas son las siguientes.

- master: 163.117.141.188
- slave01: 163.117.141.189
- slave02: 163.117.141.190
- slave03: 163.117.141.191

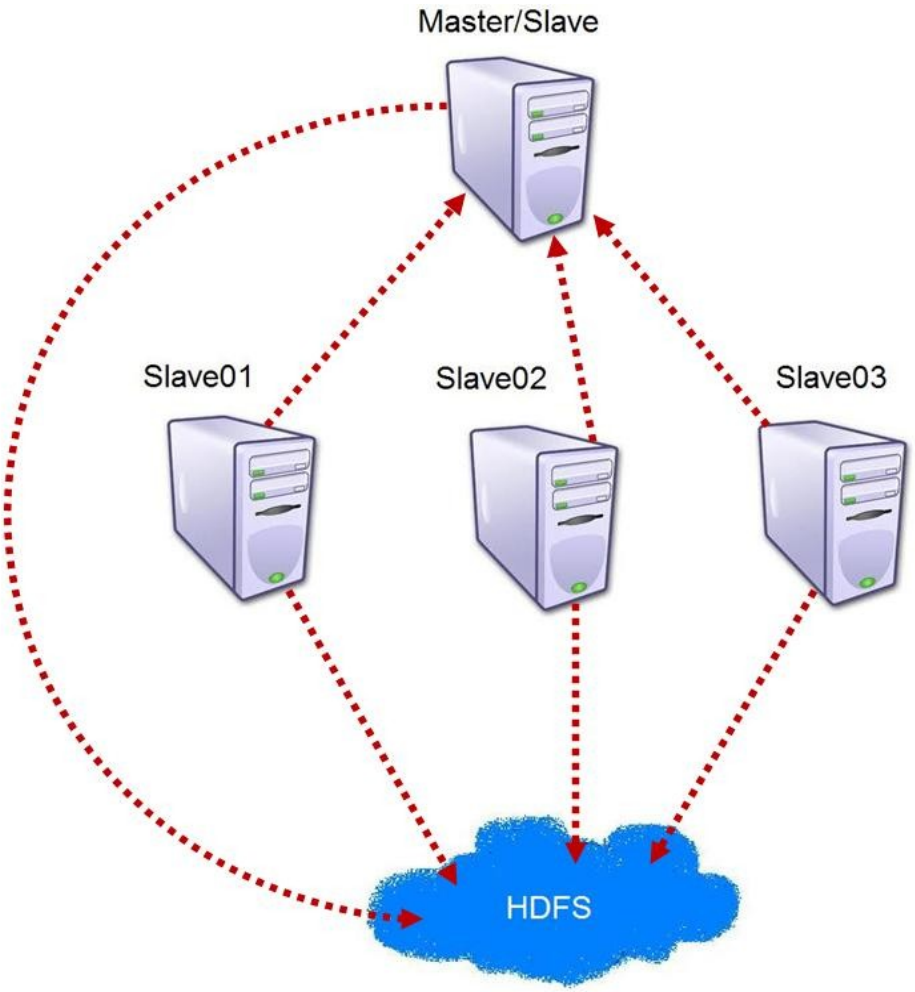


Figura 3.1: Estructura del *Cluster*

3.2.2. Datos

Para comprobar el correcto funcionamiento del *cluster* se han realizado una serie de pruebas que realizan diferentes funciones. Estas pruebas reciben un fichero de entrada que contiene la información sobre la estructura de enlaces de Wikipedia obtenida de un volcado en formato XML de dicha enciclopedia. A partir de este volcado se ha generado una lista de los enlaces de cada página, en la que para cada enlace se tiene el "anchor" (texto en el que se sitúa el enlace) y la URL de Wikipedia a la que se enlaza. El formato del fichero de texto es el siguiente:

Título\tAnchor\tDestino

Donde:

- Título: Es el título de la página de Wikipedia en la que se encuentra el enlace.
- Anchor: Representa el texto del enlace.
- Destino: Es el título de la página destino (a partir de la cual se puede obtener su URL).

Este fichero contenía algunos enlaces que no debían considerarse, como por ejemplo:

*'Tis Pity She's a Whore Théâtre de Paris **:fr:**Théâtre de Paris*

*.hack//Roots . It will be deleted after 2008-02-22.</small> **Image:**Haseo.jpg*

*'Abadilah Report Errors **User_Talk:**Addbot*

*! (album) **Category:**1995 albums **Category:**1995 albums*

Se filtraron los prefijos que representaban idiomas (":en:", ":fr:", etc.) y *namespaces* ("Wikipedia_talk:", "Image:", "User_Talk:", "Category:", etc.), dejando el fichero de entrada únicamente con los enlaces que interesaban.

Una vez que se tiene el fichero de entrada a procesar se realiza sobre éste una serie de pruebas con las que se obtienen unos ficheros de salida que devuelven la información pedida. Estas pruebas se describen en la siguiente sección.

3.3. Pruebas

Para comprobar el correcto funcionamiento del *cluster* se han realizado dos tipos de pruebas que realizan diferentes funciones:

- **Pruebas de comprobación de una correcta instalación y funcionamiento del software:** Estas pruebas son realizadas en la primera fase de desarrollo del proyecto. Para las primeras ejecuciones del software se ha utilizado un fichero de entrada de datos mucho más pequeño que el utilizado en las pruebas finales de evaluación del rendimiento. El objetivo de estas pruebas era conseguir un correcto funcionamiento del programa en sí, por lo que se necesitaba que las ejecuciones fuesen lo más rápidas posibles. El tamaño de este fichero de entrada era de unos 100 MB.
- **Pruebas para la evaluación del rendimiento:** Estas pruebas reciben un fichero de entrada y devuelven un fichero de salida que contiene la información pedida. Las pruebas se realizan con un fichero de entrada de datos de unos 6 GB. Se comienza obteniendo un 10% del fichero y se va aumentando en cada ejecución un 10% cada vez, hasta concluir con el 100% del fichero.

El escenario de pruebas consta de un *cluster* con cuatro máquinas, alojadas en un laboratorio de la universidad. Las pruebas se realizan en cuatro escenarios distintos, dependiendo del número de máquinas que formen el *cluster*.

El primer escenario tendrá un único nodo, *standalone*, es decir, no distribuida y un solo proceso de Java, útil para depuración según la documentación. Es la configuración predeterminada.

El segundo escenario tendrá dos nodos. Una máquina será el esclavo y la otra hará de maestro y esclavo a la vez.

El tercer escenario tendrá tres nodos. Dos máquinas serán los esclavos y la otra hará de maestro y esclavo a la vez.

El cuarto escenario se compondrá por los cuatro nodos, dando lugar al *cluster* completo. Igual que en los escenarios anteriores tres máquinas serán los esclavos y la otra hará de maestro y esclavo a la vez.

El programa de pruebas consistirá en distintas aplicaciones que recibirán el fichero de entrada y procesarán los datos devolviendo un fichero de salida con diferente información.

Una vez realizadas todas las ejecuciones se recogen los datos obtenidos y se estudiarán. Se compararán los tiempos y la carga del sistema de cada escenario y se plasmarán en una gráfica, para posteriormente obtener conclusiones.

Para la realización de esta prueba se han realizado una serie de aplicaciones MapReduce en código Java.

Para cada aplicación es necesario implementar tres clases: la clase *Mapper*, la clase *Reducer* y la propia clase *Job*. En este caso, estas dos últimas clases son iguales para todas las pruebas implementadas, cambiando únicamente el código de la clase *Mapper*.

La clase *Mapper* toma un par de entrada clave/valor y produce un grupo intermedio de pares de claves/valores. La biblioteca de MapReduce agrupa todos los valores intermedios asociados con la misma clave intermedia y los pasa a la función *Reduce*.

La clase *Reducer* recibe la clave intermedia y un grupo de valores para esa clave. Fusiona dichos valores para formar un grupo posiblemente más pequeño de valores. Los valores intermedios les son suministrados a la función *Reducer* a través de un *iterable* (un objeto que le permite a un programador atravesar todos los elementos de un conjunto

independientemente o de su implementación específica). Esto permite manejar las listas de valores que sean demasiado grandes para la memoria.

El código común de las clases *Reducer* es el siguiente:

```
01  import java.io.IOException;
02
03  import org.apache.hadoop.io.IntWritable;
04  import org.apache.hadoop.io.Text;
05  import org.apache.hadoop.mapreduce.Reducer;
06
07  /**
08   * @author M.Carmen Palacios Diaz-Zorita
09   *
10   */
11  public class Prueba1Reducer extends
                Reducer<Text, IntWritable, Text, IntWritable> {
12
13        private IntWritable result = new IntWritable();
14
15        public void reduce(Text key, Iterable<IntWritable> values,
            Context context)  throws IOException, InterruptedException {
16              // initialize the sum for each keyword
17              int sum = 0;
18
19              for (IntWritable val : values) {
20                    // Increment the number of values for this key
21                    sum += val.get();
22              }
23              result.set(sum);
24
25              // create a pair <keyword, number of occurences>
26              context.write(key, result);
27        }
28  }
```

El código común de las clases principales *Job* es el siguiente:

```
01  import org.apache.hadoop.conf.Configuration;
02  import org.apache.hadoop.fs.Path;
03  import org.apache.hadoop.io.IntWritable;
04  import org.apache.hadoop.io.Text;
05  import org.apache.hadoop.mapreduce.Job;
06  import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
07  import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
08
09  /**
10   * @author M.Carmen Palacios Diaz-Zorita
11   *
12   */
13  public class Prueba1 {
14
15        public static void main(String[] args) throws Exception {
16              Configuration conf = new Configuration();
17
18              // create a job with name "prueba1"
19              Job job = new Job(conf, "prueba1");
```

```
20                job.setJarByClass(Prueba1.class);
21                job.setMapperClass(Prueba1Mapper.class);
22                job.setReducerClass(Prueba1Reducer.class);
23
24                // add the Combiner
25                job.setCombinerClass(Prueba1Reducer.class);
26
27                // set output key type
28                job.setOutputKeyClass(Text.class);
                  // set output value type
29                job.setOutputValueClass(IntWritable.class);
30                // set the HDFS path of the input data
                  FileInputFormat.setInputPaths(job, new
31                  Path("prueba1/input"));
32                // set the HDFS path for the output
33                FileOutputFormat.setOutputPath(job, new
34                  Path("prueba1/output"));
35
36                // Wait till job completion
37                System.exit(job.waitForCompletion(true) ? 0 : 1);
38        }
39  }
```

A continuación se muestra un fragmento del fichero de entrada que recibe el programa:

! (album)	*DeSoto Records*	*DeSoto Records*
!Action Pact!	*BBC Radio 1*	*BBC Radio 1*
!Action Pact!	*England*	*England*
!Action Pact!	*John Peel*	*John Peel*
!Hero	*Hero (rock opera)*	*!Hero*
!Hero	*Janitor*	*Janitor*
Evans, Georgia	*population density*	*population density*
Evans, Georgia	*poverty line*	*poverty line*
Evans, Illinois	*217*	*Area code 217*

A continuación se describen las distintas clases *Mapper* para cada una de las pruebas realizadas:

3.3.1. Prueba 1

En esta prueba se quiere obtener el número de enlaces que aparecen en una página.

Para ello la clase *Mapper* toma cada línea del archivo de texto, lo transforma en un array ([title][anchor][link]) con el método *split* de la clase *String* y selecciona la primera posición del array que contiene el título. Se crea el par clave/valor que recibirá el *Reducer,* que contará todos los títulos iguales, obteniendo así el número de enlaces que aparecen en una página.

El código de esta clase *Mapper* es el siguiente:

```
01  import java.io.IOException;
02
03  import org.apache.hadoop.io.IntWritable;
04  import org.apache.hadoop.io.LongWritable;
05  import org.apache.hadoop.io.Text;
06  import org.apache.hadoop.mapreduce.Mapper;
07
08  /**
09   * This program counts the number of anchors that appear on a page of
10   * Wikipedia.
11   *
12   * @author M. Carmen Palacios Diaz-Zorita
13   *
14   */
15  public class Prueba1Mapper extends
                Mapper<LongWritable, Text, Text, IntWritable> {
16
17        // type of output value
18        private final static IntWritable one = new IntWritable(1);
19        // type of output key
20        private Text title = new Text();
21
22        public void map(LongWritable key, Text value, Context context)
                      throws IOException, InterruptedException {
23
24              // Get the array with [title][anchor][link]
25              String[] line = value.toString().split("\t");
26              // set title as each input keyword
27              title.set(line[0]);
28              // create a pair <keyword, 1>
29              context.write(title, one);
30        }
31  }
```

A continuación se muestra un fragmento del resultado obtenido:

! (album) 26
!Action Pact! 14
!Hero 57

3.3.2. Prueba 2

En esta prueba se quiere obtener el número de páginas que apuntan a otra página, es decir, el número de enlaces con destino hacia una página.

Para ello la clase *Mapper* toma cada línea del archivo de texto, lo transforma en un array ([title][anchor][link]) con el método split de la clase String y selecciona la tercera posición del array que contiene el destino. Se crea el par clave/valor que recibirá el *Reducer,* que contará todos los destinos iguales, obteniendo así el número de enlaces con destino hacia una página.

El código de esta clase *Mapper* es el siguiente:

```
01  import java.io.IOException;
02
03  import org.apache.hadoop.io.IntWritable;
04  import org.apache.hadoop.io.LongWritable;
05  import org.apache.hadoop.io.Text;
06  import org.apache.hadoop.mapreduce.Mapper;
07
08  /**
09   * This program counts the number of pages that point to another page.
10   *
11   * @author M.Carmen Palacios Diaz-Zorita
12   *
     */
13  public class Prueba2Mapper extends
14              Mapper<LongWritable, Text, Text, IntWritable> {
15
16        // type of output value
17        private final static IntWritable one = new IntWritable(1);
18        // type of output key
19        private Text link = new Text();

20        public void map(LongWritable key, Text value, Context context)
21                    throws IOException, InterruptedException {
22
23              // Get the array with [title][anchor][link]
24              String[] line = value.toString().split("\t");
25              // set link as each input keyword
26              link.set(line[2]);
27              // create a pair <keyword, 1>
28              context.write(link, one);
29        }
30  }
```

A continuación se muestra un fragmento del resultado obtenido:

! (album) *16*
!Action Pact! *5*
!Hero *5*

3.3.3. Prueba 3

En esta prueba se quiere obtener el número de apariciones de un determinado anchor en una página.

Para ello la clase *Mapper* toma cada línea del archivo de texto, lo transforma en un array ([title][anchor][link]) con el método split de la clase String y selecciona la primera y segunda posición del array que contiene el título y el anchor de una determinada página. Se crea el par clave/valor que recibirá el *Reducer,* que contará todos los títulos + anchor iguales, obteniendo así el número de apariciones de un determinado anchor en una página.

El código de esta clase *Mapper* es el siguiente:

```
01  import java.io.IOException;
02
03  import org.apache.hadoop.io.IntWritable;
04  import org.apache.hadoop.io.LongWritable;
05  import org.apache.hadoop.io.Text;
06  import org.apache.hadoop.mapreduce.Mapper;
07
08  /**
09   * This program counts the number of times that a particular anchor
10   * appears in a Wikipedia page.
11   *
12   * @author M.Carmen Palacios Diaz-Zorita
13   *
14   */
15  public class Prueba3Mapper extends
                Mapper<LongWritable, Text, Text, IntWritable> {
16
17        // type of output value
18        private final static IntWritable one = new IntWritable(1);
19        // type of output key
20        private Text title_anchor = new Text();
21
22        public void map(LongWritable key, Text value, Context context)
                      throws IOException, InterruptedException {
23
24              // Get the array with [title][anchor][link]
25              String[] line = value.toString().split("\t");
26              // set title + anchor as each input keyword
27              title_anchor.set(line[0] + " + " + line[1]);
28              // create a pair <keyword, 1>
29              context.write(title_anchor, one);
30        }
31  }
```

A continuación se muestra un fragmento del resultado obtenido:

! (album) + ! (album)	*16*
!Action Pact! + !Action Pact!	*4*
!Hero + Bethlehem, Pennsylvania	*2*

3.3.4. Prueba 4

En esta prueba se quiere obtener el número de apariciones de un destino en una determinada página.

Para ello la clase *Mapper* toma cada línea del archivo de texto, lo transforma en un array ([title][anchor][link]) con el método split de la clase String y selecciona la primera y tercera posición del array que contiene el título y el destino de una determinada página. Se crea el par clave/valor que recibirá el *Reducer,* que contará todos los títulos + destinos iguales, obteniendo así el número de apariciones de un destino en una determinada página.

El código de esta clase *Mapper* es el siguiente:

```
01  import java.io.IOException;
02
03  import org.apache.hadoop.io.IntWritable;
04  import org.apache.hadoop.io.LongWritable;
05  import org.apache.hadoop.io.Text;
06  import org.apache.hadoop.mapreduce.Mapper;
07
08  /**
09   * This program counts the number of times that the same destination
10   * appears on a Wikipedia page.
11   *
12   * @author M.Carmen Palacios Diaz-Zorita
13   *
14   */
15  public class Prueba4Mapper extends
                Mapper<LongWritable, Text, Text, IntWritable> {
16
17        // type of output value
18        private final static IntWritable one = new IntWritable(1);
19        // type of output key
20        private Text title_link = new Text();
21
22        public void map(LongWritable key, Text value, Context context)
                      throws IOException, InterruptedException {
23
24              // Get the array with [title][anchor][link]
25              String[] line = value.toString().split("\t");
26              // set title + link as each input keyword
27              title_link.set(line[0] + " " + line[2]);
28              // create a pair <keyword, 1>
29              context.write(title_link, one);
30        }
31  }
```

A continuación se muestra un fragmento del resultado obtenido:

! (album) + DeSoto Records 2
!Action Pact! + UK Indie Chart 1
!Hero + Bible 2

3.3.5. Prueba 5

En esta prueba se quiere obtener el número de veces que un anchor aparece apuntando a un desino.

Para ello la clase *Mapper* toma cada línea del archivo de texto, lo transforma en un array ([title][anchor][link]) con el método split de la clase String y selecciona la segunda y tercera posición del array que contiene el anchor y el destino de una determinada página. Se crea el par clave/valor que recibirá el *Reducer,* que contará todos los anchors + destinos iguales, obteniendo así el número de anchor asociado a un determinado destino.

El código de esta clase *Mapper* es el siguiente:

```
01  import java.io.IOException;
02
03  import org.apache.hadoop.io.IntWritable;
04  import org.apache.hadoop.io.LongWritable;
05  import org.apache.hadoop.io.Text;
06  import org.apache.hadoop.mapreduce.Mapper;
07
08  /**
09   * This program counts the number of times that a specific anchor is
10   * associated to a destination.
11   *
12   * @author M.Carmen Palacios Diaz-Zorita
13   *
14   */
15  public class Prueba5Mapper extends
                Mapper<LongWritable, Text, Text, IntWritable> {
16
17        // type of output value
18        private final static IntWritable one = new IntWritable(1);
19        // type of output key
20        private Text anchor_link = new Text();
21
22        public void map(LongWritable key, Text value, Context context)
                      throws IOException, InterruptedException {
23
24              // Get the array with [title][anchor][link]
25              String[] line = value.toString().split("\t");
26              // set anchor + link as each input keyword
27              anchor_link.set(line[1] + " + " + line[2]);
28              // create a pair <keyword, 1>
29              context.write(anchor_link, one);
30        }
31  }
```

A continuación se muestra un fragmento del resultado obtenido:

! (album) + ! (album)	*17*
!Action Pact! + !Action Pact!	*8*
1961 elections + Israeli legislative election, 1961	*6*

4. VALIDACIÓN

Tras el desarrollo del proyecto, éste fue sometido a un banco de pruebas para comprobar su correcto funcionamiento, detectar posibles errores y evaluar cómo evoluciona el tiempo de ejecución y el espacio en disco cuando se varía el tamaño del problema.

A continuación se describirán los resultados de las pruebas realizadas. Estas pruebas se llevaron a cabo con diferentes grupos de archivos de entrada, cada uno de distinto tamaño. Las tareas MapReduce fueron ejecutadas en un *cluster* de nodo único y en *cluster*s de dos, tres y cuatro nodos. Los correspondientes tiempos de ejecución y el espacio en disco utilizado han sido medidos y los resultados plasmados en distintas gráficas para llegar a las conclusiones pertinentes.

Para la configuración, arranque y parada de Hadoop se han creado tres scripts para automatizar estos procesos. De esta manera simplemente hay que lanzar dichos scripts, dependiendo del escenario en el que nos encontremos, ya sea un nodo, dos, tres o cuatro nodos.

4.1. Scripts

El script de configuración, llamado `script_start.sh`, realiza la parte de configuración de Hadoop, copiando los ficheros de configuración correspondientes al escenario deseado en la carpeta de instalación de Hadoop $HADOOP_HOME. También establece las variables de entorno necesarias

para el correcto funcionamiento de la aplicación, formatea el sistema de archivos distribuido, agrega las claves públicas ssh de las máquinas maestro en el archivo .ssh/authorized_keys de las máquinas esclavo, y arranca los demonios de HDFS y de MapReduce.

A continuación se muestra el script `script_start.sh`:

```
01  #!/bin/bash
02  cp -arf /home/mamenpala/proyecto/configNucleos/4Nucleos/conf
    /home/mamenpala/proyecto/hadoop-0.20.2
03
04  echo "#########Creando variables de entorno..."
05  echo "export JAVA_HOME=/usr/lib/jvm/java-6-sun"
06  export JAVA_HOME=/usr/lib/jvm/java-6-sun
07  echo "export PATH=/usr/lib/jvm/java-6-sun/bin:${PATH}"
08  export PATH=/usr/lib/jvm/java-6-sun/bin:${PATH}
09  echo "export HADOOP_HOME=/home/mamenpala/proyecto/hadoop-0.20.2"
10  export HADOOP_HOME=/home/mamenpala/proyecto/hadoop-0.20.2
11  echo "export PATH=$PATH:$HADOOP_HOME/bin"
12  export PATH=$PATH:$HADOOP_HOME/bin
13  hadoop version
14
15  #///Solo en 163.117.141.188
16  hadoop namenode -format
17  #Inicio de los demonios de Mapreduce
18  echo "#########Inicio de los demonios de Mapreduce..."
19  start-mapred.sh
20  #Inicio de los demonios de hdfs
21  echo "#########Inicio de los demonios de hdfs..."
22  start-dfs.sh
23
24
25  cd /home/mamenpala/proyecto/wikiHadoop
26  #Compilacion del codigo java
27  javac -classpath ${HADOOP_HOME}/hadoop-0.20.2-
    core.jar:${HADOOP_HOME}/lib/commons-cli-1.2.jar -d
    /home/mamenpala/proyecto/wikiHadoop/bin
    /home/mamenpala/proyecto/wikiHadoop/src/*.java
28  #Creacion del fichero .jar con los datos compilados
    jar -cvf /home/mamenpala/proyecto/wikiHadoop/prueba3.jar -C
    /home/mamenpala/proyecto/wikiHadoop/bin/ .
29
30
31
32  cd ..
33  cd hadoop-0.20.2
34
35  #Borrado de archivos antiguos por si se hubiesen quedado en el sistema
    de archivos de hadoop
36  echo "#########Borrado de archivos antiguos..."
37  bin/hadoop dfs -rmr /user/mamenpala/src
38  bin/hadoop dfs -rm /user/mamenpala/prueba3.jar
39  bin/hadoop dfs -rmr /user/mamenpala/prueba3/output
40  bin/hadoop dfs -rmr /user/mamenpala/prueba3/input/*
41
42  bin/hadoop dfs -copyFromLocal
    /home/mamenpala/proyecto/wikiHadoop/prueba3.jar .
43  bin/hadoop dfs -copyFromLocal /home/mamenpala/proyecto/wikiHadoop/src .
44  bin/hadoop jar /home/mamenpala/proyecto/wikiHadoop/prueba3.jar
```

El script de arranque, llamado `script_run.sh`, realiza la parte de ejecución de la aplicación, pasando a Hadoop el fichero de entrada a través de un bucle por el cual se va aumentando el tamaño del fichero un 10% en cada ejecución, hasta llegar al 100% del fichero. Cuando termina la ejecución de la aplicación copia los ficheros de salida obtenidos en la carpeta correspondiente del sistema de ficheros local.

A continuación se muestra el script `script_run.sh`:

```
01  #!/bin/bash
02
03  echo "#########Creando variables de entorno..."
04  echo "export JAVA_HOME=/usr/lib/jvm/java-6-sun"
05  export JAVA_HOME=/usr/lib/jvm/java-6-sun
06  echo "export PATH=/usr/lib/jvm/java-6-sun/bin:${PATH}"
07  export PATH=/usr/lib/jvm/java-6-sun/bin:${PATH}
08  echo "export HADOOP_HOME=/home/mamenpala/proyecto/hadoop-0.20.2"
09  export HADOOP_HOME=/home/mamenpala/proyecto/hadoop-0.20.2
10  echo "export PATH=$PATH:$HADOOP_HOME/bin"
11  export PATH=$PATH:$HADOOP_HOME/bin
12  hadoop version
13
14  cd /home/mamenpala/proyecto/
15  salida=$(wc -l wikipedia-links-redirects-sorted.txt-Filtrado.txt)
16  num_lineas=${salida%% "*}
17  x=$((num_lineas/10))
18  #split -l$x wikipedia-links-redirects-sorted.txt-Filtrado.txt
19  ficheros/wikipedia-links-redirects-sorted.txt-Filtrado.
20  echo "Borrando contenido de la carpeta input local..."
21  rm /home/mamenpala/proyecto/pruebas/prueba3/input/wikipedia-links-redirects-
    sorted.txt-Filtrado.partes
22  echo "Borrando contenido de la carpeta output local..."
23  rm /home/mamenpala/proyecto/pruebas/prueba3/output/4Nucleos/*
24  rm /home/mamenpala/proyecto/pruebas/prueba3/output/4Nucleos/logs/*
25  rm /home/mamenpala/proyecto/pruebas/prueba3/output/4Nucleos/logs/logs_tam/*
26  rm /home/mamenpala/proyecto/pruebas/prueba3/output/4Nucleos/logs/logs_time/*
27  touch /home/mamenpala/proyecto/pruebas/prueba3/input/wikipedia-links-
28  redirects-sorted.txt-Filtrado.partes
29  cd hadoop-0.20.2
30  echo "Creando Carpetas en hdfs..."
31  bin/hadoop dfs -rmr /user/*
32  bin/hadoop dfs -mkdir -p prueba3/input
33
34  contador=0
35  partes="aa:ab:ac:ad:ae:af:ag:ah:ai:aj:"
36  IFS=:
37  for i in $partes
38  do
39          contador=$((contador+10))
40          echo "Borrando contenido de la carpera input..."
41          bin/hadoop dfs -rmr /user/mamenpala/prueba3/input/*
42          echo "Concatenando ficheros..."
43          cat /home/mamenpala/proyecto/ficheros/wikipedia-links-redirects-
44           sorted.txt-Filtrado.$i >>
    /home/mamenpala/proyecto/pruebas/prueba3/input/wikipedia-links-redirects-
    sorted.txt-Filtrado.partes
```

```
45          echo "Copiando ficheros de entrada en el sistema de archivos hdfs..."
46          bin/hadoop dfs -copyFromLocal
             /home/mamenpala/proyecto/pruebas/prueba3/input/* prueba3/input
47          secuencia=`seq -s : 1 30`
48          secuencia="$secuencia:"
49          IFS=:
50          for numpruebas in $secuencia
51          do
52                 echo "Borrando carpera output..."
53                 bin/hadoop dfs -rmr /user/mamenpala/prueba3/output
54                 echo "Ejecutando programa..."
55                 bin/hadoop jar /home/mamenpala/proyecto/wikiHadoop/prueba3.jar
                    Prueba3 /user/mamenpala/prueba3/input
                    /user/mamenpala/prueba3/output
56          done
57
58          echo "Copiando fichero de salida al sistema de ficheros local..."
59          bin/hadoop dfs -copyToLocal prueba3/output/part-r-00000
             /home/mamenpala/proyecto/pruebas/prueba3/output/4Nucleos/$contador%.txt
60
61    done
```

El script de parada, llamado `script_stop.sh`, realiza la parte de parada de Hadoop, deteniendo los demonios del sistema de archivos y de MapReduce.

A continuación se muestra el script `script_stop.sh`:

```
01    #!/bin/bash
02    cd /home/mamenpala/proyecto/hadoop-0.20.2
03    echo "#########Borrado de archivos antiguos..."
04    bin/hadoop dfs -rmr /user/mamenpala/src
05    bin/hadoop dfs -rm /user/mamenpala/prueba3.jar
06    bin/hadoop dfs -rmr /user/mamenpala/prueba3/output
07
08    #Parada de los demonios de hdfs
09    echo "#########Parada de los demonios de hdfs..."
10    bin/stop-dfs.sh
11    #Parada de los demonios de Mapreduce
12    echo "#########Parada de los demonios de Mapreduce..."
13    bin/stop-mapred.sh
```

4.2. Escenarios

A continuación se explica de forma detallada los resultados obtenidos en las pruebas realizadas para comprobar el correcto funcionamiento de la herramienta Hadoop, en cada uno de los escenarios propuestos.

Esta comprobación se ha realizado con la prueba número 3, pues es la que más tiempo y recursos necesita. Esto se ha comprobado empíricamente, después de realizar las pruebas y comprobar cuál era la que más tiempo y recursos necesitaba, aunque entre la prueba 3 y la prueba 4 no había mucha diferencia de tiempo. De esta forma se podrá ver si el cambio a un escenario con más o menos máquinas en el *cluster* hace que los tiempos y recursos utilizados varíen de forma notable.

Para cada escenario se presentan los resultados de las medidas de los tiempos obtenidos para cada uno de los procesos Java de Hadoop y los datos obtenidos de la medición del espacio en disco utilizado en cada ejecución. El tamaño del fichero de entrada se ha aumentado un 10% en cada ejecución hasta completar el 100% del fichero en la décima ejecución.

Se han realizado 30 pruebas para cada ejecución. Es decir, para el fichero del 10% se ha repetido la medida 30 veces, para el del 20% otras 30 veces, etc. De este modo se ha podido realizar una media de todos los datos y se han estimado los parámetros con una confianza del 95%.

La medición de los tiempos se ha realizado obteniendo mediante el comando top la diferencia de tiempo desde que están activos cada uno de los procesos Java antes y después de cada ejecución. Estos datos se han tratado con una hoja de cálculo y se han generado las gráficas correspondientes para cada uno de los procesos y escenarios. El formato del tiempo se representa de la siguiente forma: mm:ss,dd siendo minutos:segundos,décimas

La medición del espacio en disco utilizado se realizado obteniendo mediante el comando du el tamaño en el directorio `/tmp/hadoop-mamenpala/dfs/data` justo después de cada ejecución. En este directorio se encuentran únicamente los datos del HDFS. Los *reducers* se conectan a los *mappers* para leer los datos locales y realizar las operaciones con ellos, y la salida de los reducers es la que se escribe como resultado final en el HDFS. Los archivos temporales que se generan como parte de la tarea del *map* resultan difíciles de medir, ya que Hadoop los borra en cuanto los leen los *reducers*.

Igualmente, estos datos se han tratado con una hoja de cálculo y se han generado las gráficas correspondientes para cada escenario. Estas gráficas muestra la evolución de la cantidad de espacio utilizado para cada fichero de entrada.

A continuación se muestran de forma detallada las gráficas obtenidas para cada escenario.

4.2.1. Un Nodo

Esta ejecución ha sido realizada en un solo nodo. Este único nodo realiza las funciones de maestro y esclavo. La máquina utilizada tiene la dirección IP: 163.117.141.188.

Tiempos

DataNode

Tamaño	Media	Desv. Típ	Intervalo de conf (95%)	Intervalo Inferior	Intervalo Superior
10%	00:02,34	00:00,16	00:00,06	00:02,28	00:02,39
20%	00:04,56	00:00,08	00:00,03	00:04,53	00:04,59
30%	00:06,84	00:00,09	00:00,03	00:06,81	00:06,88
40%	00:09,26	00:00,16	00:00,06	00:09,21	00:09,32
50%	00:11,88	00:00,16	00:00,06	00:11,82	00:11,94
60%	00:15,45	00:00,12	00:00,04	00:15,41	00:15,49
70%	00:18,17	00:00,17	00:00,06	00:18,11	00:18,23
80%	00:20,65	00:00,10	00:00,04	00:20,62	00:20,69
90%	00:23,20	00:00,12	00:00,04	00:23,16	00:23,24
100%	00:25,88	00:00,22	00:00,08	00:25,80	00:25,96

Tabla 4-1: Datos del proceso DataNode para el Escenario 1

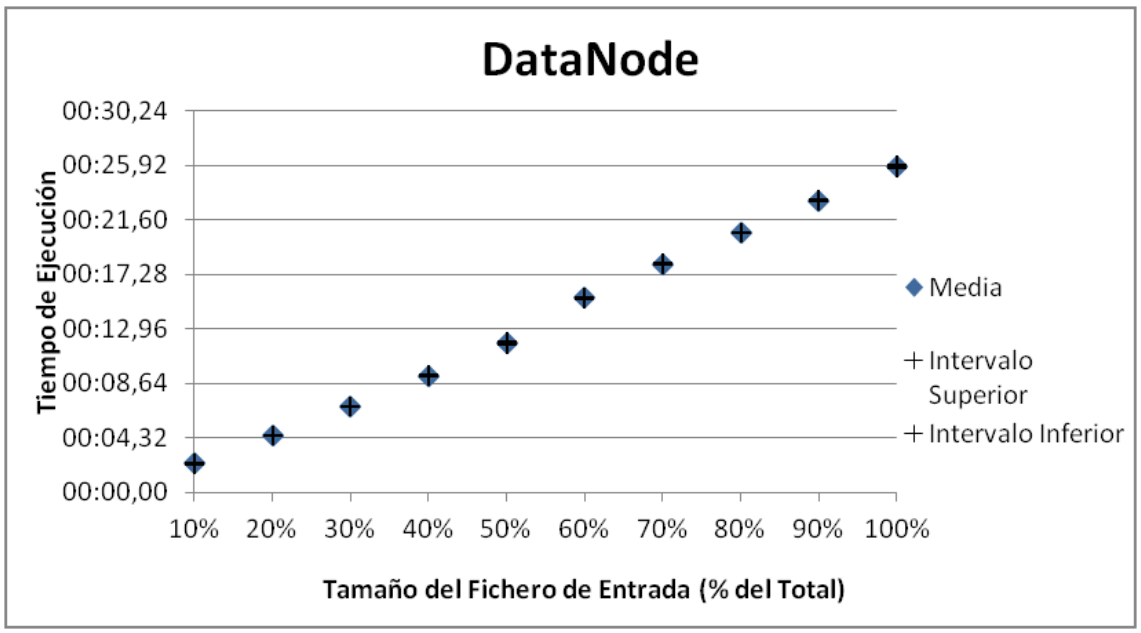


Figura 4.1: Representación de los Datos del proceso DataNode para el Escenario 1

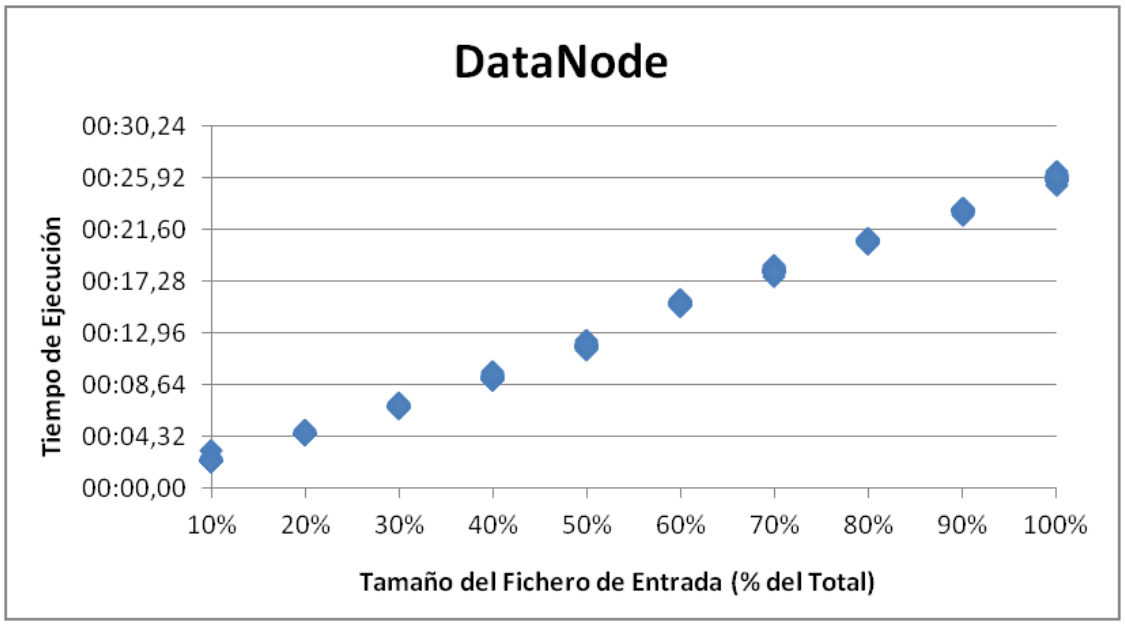


Figura 4.2: Total de los datos del proceso DataNode para el Escenario 1

TaskTracker

Tamaño	Media	Desv. Típ	Intervalo de conf (95%)	Intervalo Inferior	Intervalo Superior
10%	00:01,49	00:00,43	00:00,15	00:01,34	00:01,65
20%	00:02,50	00:00,08	00:00,03	00:02,47	00:02,53
30%	00:03,68	00:00,10	00:00,04	00:03,64	00:03,72
40%	00:04,73	00:00,17	00:00,06	00:04,67	00:04,79
50%	00:06,05	00:00,24	00:00,08	00:05,97	00:06,13
60%	00:07,57	00:00,14	00:00,05	00:07,52	00:07,62
70%	00:08,74	00:00,18	00:00,06	00:08,67	00:08,80
80%	00:10,07	00:00,20	00:00,07	00:10,00	00:10,15
90%	00:11,09	00:00,30	00:00,11	00:10,99	00:11,20
100%	00:12,06	00:00,28	00:00,10	00:11,96	00:12,16

Tabla 4-2: Datos del proceso TaskTracker para el Escenario 1

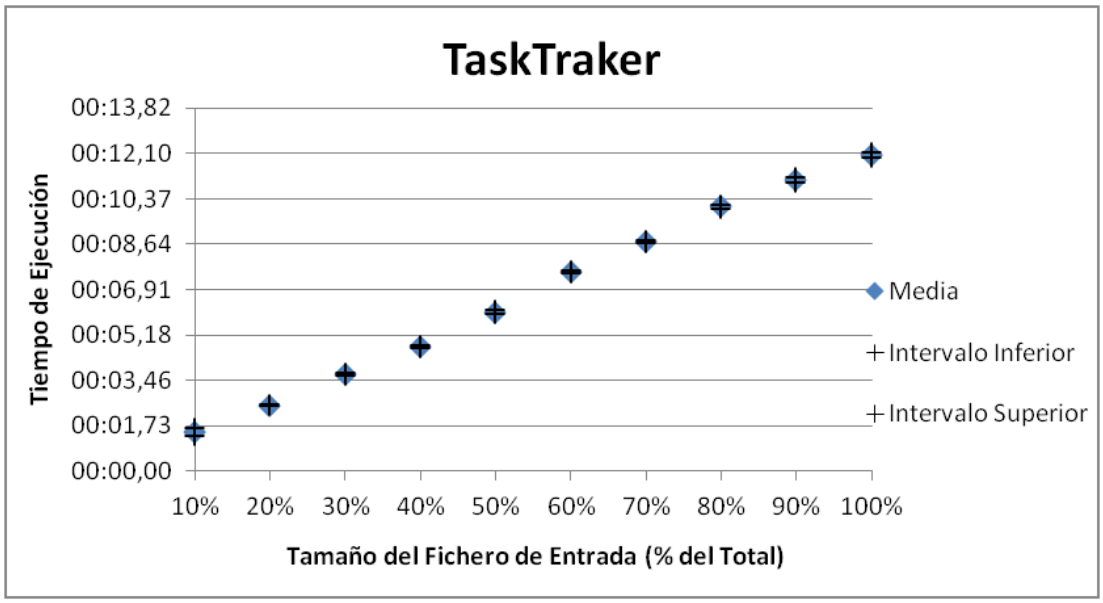


Figura 4.3: Representación de los Datos del proceso TaskTraker para el Escenario 1

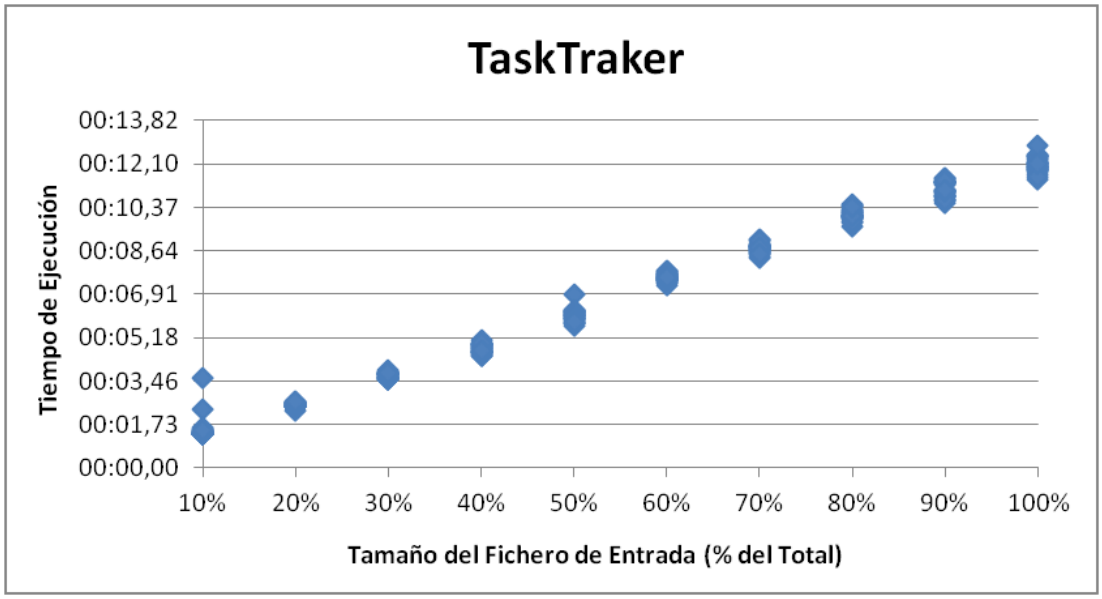


Figura 4.4: Total de los datos del proceso TaskTraker para el Escenario 1

JobTraker

Tamaño	Media	Desv. Típ	Intervalo de conf (95%)	Intervalo Inferior	Intervalo Superior
10%	00:00,31	00:00,20	00:00,07	00:00,24	00:00,38
20%	00:00,35	00:00,06	00:00,02	00:00,33	00:00,37
30%	00:00,39	00:00,04	00:00,01	00:00,38	00:00,41
40%	00:00,48	00:00,04	00:00,01	00:00,46	00:00,49
50%	00:00,58	00:00,18	00:00,06	00:00,52	00:00,64
60%	00:00,65	00:00,03	00:00,01	00:00,64	00:00,66
70%	00:00,74	00:00,04	00:00,01	00:00,72	00:00,75
80%	00:00,86	00:00,08	00:00,03	00:00,83	00:00,88
90%	00:00,92	00:00,05	00:00,02	00:00,91	00:00,94
100%	00:01,03	00:00,12	00:00,04	00:00,99	00:01,08

Tabla 4-3: Datos del proceso JobTracker para el Escenario 1

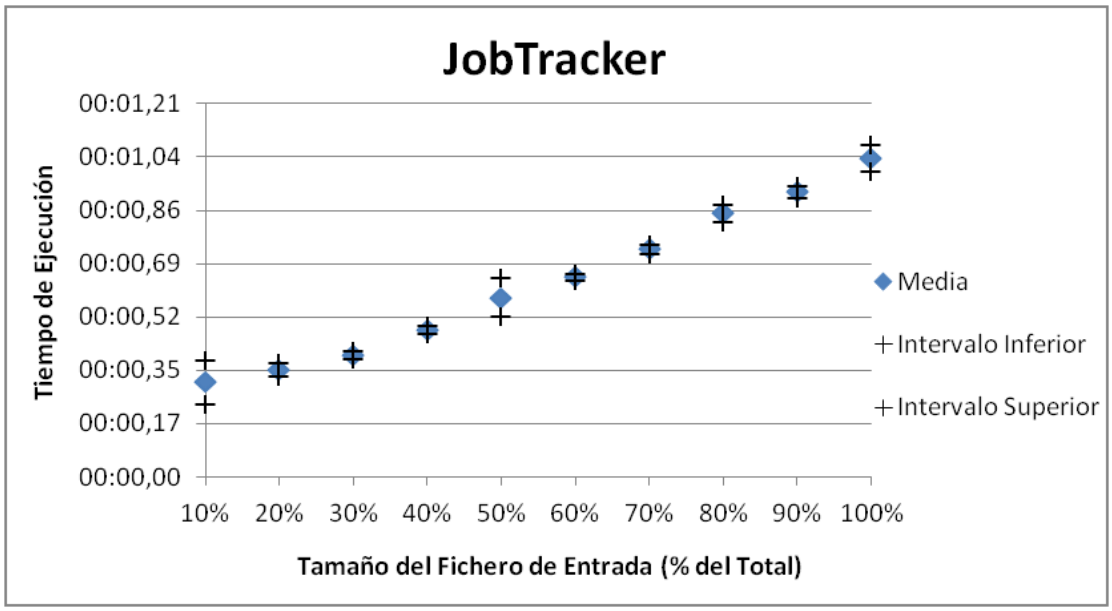


Figura 4.5: Representación de los Datos del proceso JobTraker para el Escenario 1

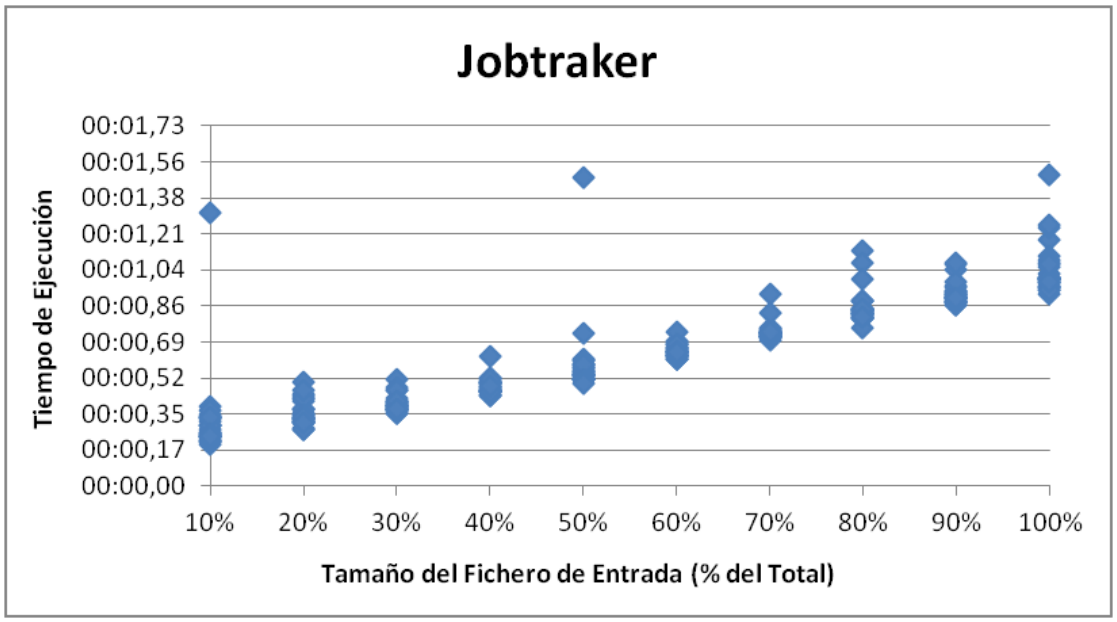


Figura 4.6: Total de los datos del proceso JobTraker para el Escenario 1

NameNode

Tamaño	Media	Desv. Típ	Intervalo de conf (95%)	Intervalo Inferior	Intervalo Superior
10%	00:00,15	00:00,03	00:00,01	00:00,14	00:00,16
20%	00:00,21	00:00,04	00:00,01	00:00,20	00:00,22
30%	00:00,26	00:00,08	00:00,03	00:00,24	00:00,29
40%	00:00,29	00:00,01	00:00,00	00:00,29	00:00,29
50%	00:00,35	00:00,02	00:00,01	00:00,35	00:00,36
60%	00:00,43	00:00,02	00:00,01	00:00,42	00:00,44
70%	00:00,50	00:00,02	00:00,01	00:00,49	00:00,50
80%	00:00,56	00:00,02	00:00,01	00:00,55	00:00,57
90%	00:00,62	00:00,03	00:00,01	00:00,61	00:00,63
100%	00:00,68	00:00,03	00:00,01	00:00,67	00:00,69

Tabla 4-4: Datos del proceso NameNode para el Escenario 1

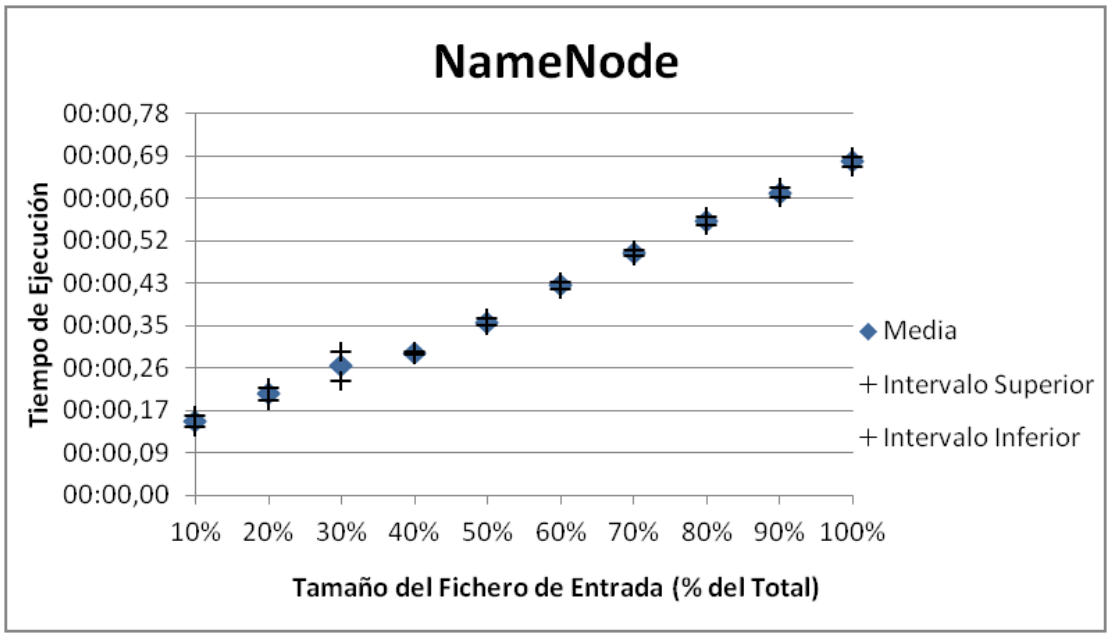


Figura 4.7: Representación de los Datos del proceso NameNode para el Escenario 1

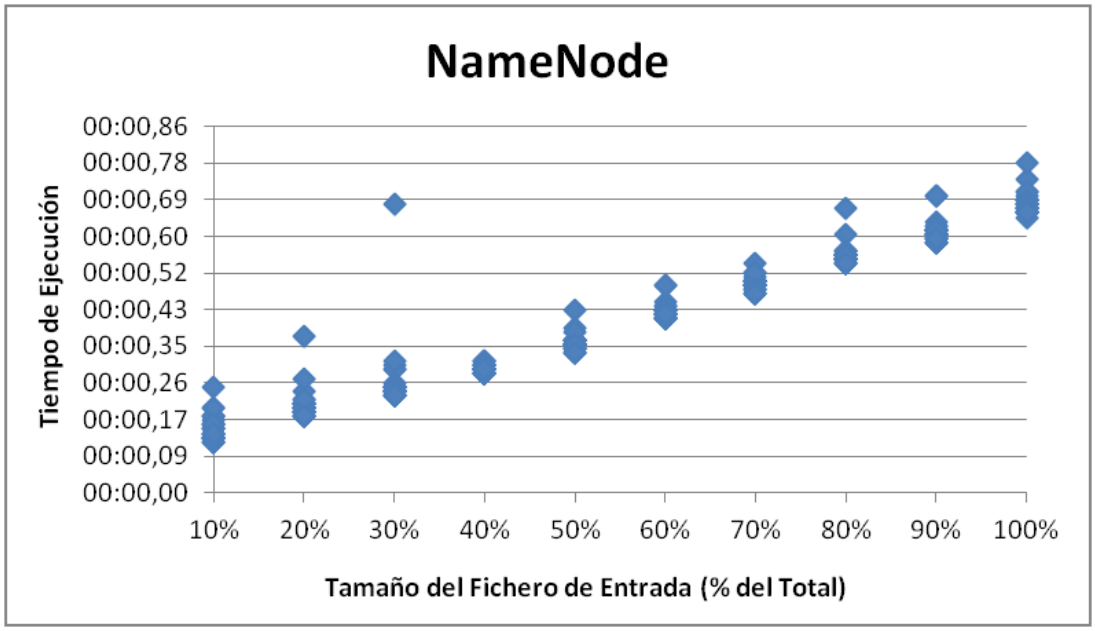


Figura 4.8: Total de los datos del proceso NameNode para el Escenario 1

SecondaryNameNode

Tamaño	Media	Desv. Típ	Intervalo de conf (95%)	Intervalo Inferior	Intervalo Superior
10%	00:00,02	00:00,01	00:00,00	00:00,02	00:00,02
20%	00:00,06	00:00,09	00:00,03	00:00,02	00:00,09
30%	00:00,07	00:00,07	00:00,02	00:00,05	00:00,10
40%	00:00,07	00:00,02	00:00,01	00:00,06	00:00,07
50%	00:00,08	00:00,02	00:00,01	00:00,08	00:00,09
60%	00:00,10	00:00,02	00:00,01	00:00,09	00:00,11
70%	00:00,12	00:00,01	00:00,01	00:00,11	00:00,12
80%	00:00,14	00:00,02	00:00,01	00:00,13	00:00,14
90%	00:00,15	00:00,01	00:00,01	00:00,15	00:00,16
100%	00:00,20	00:00,14	00:00,05	00:00,15	00:00,25

Tabla 4-5: Datos del proceso SecondaryNameNode para el Escenario 1

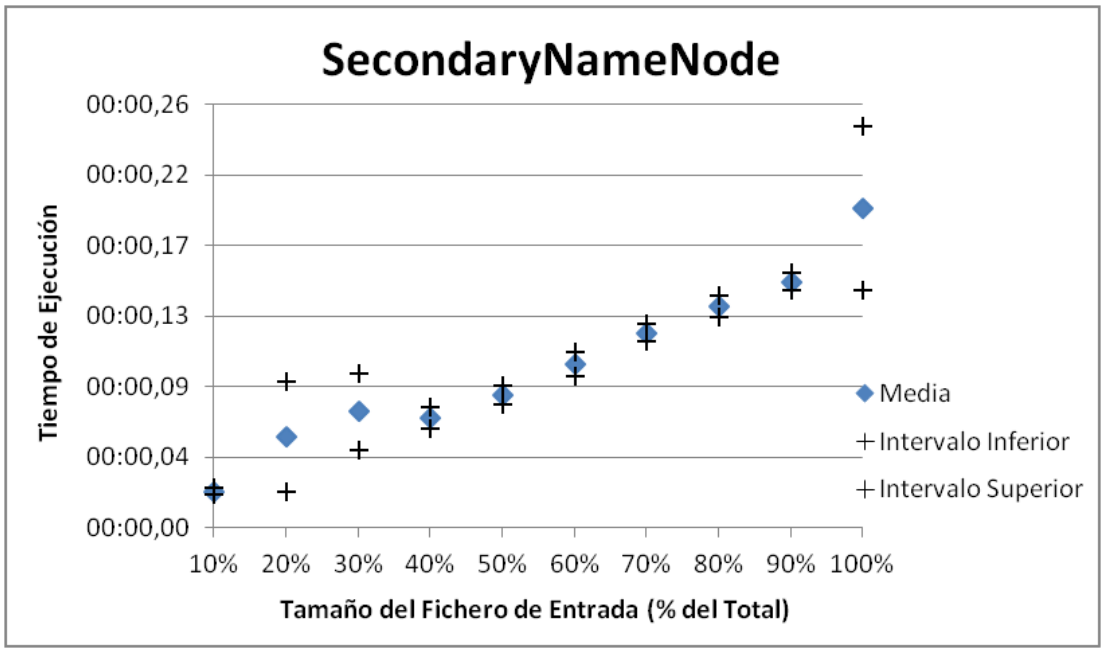


Figura 4.9: Representación de los Datos del proceso SecondaryNameNode para el Escenario 1

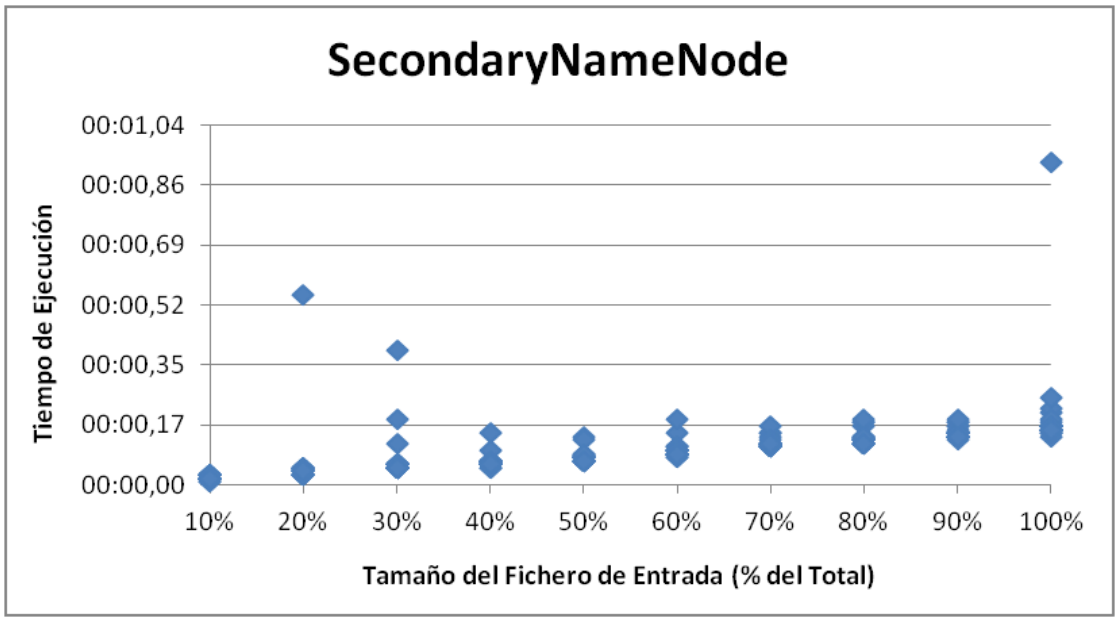


Figura 4.10: Total de los datos del proceso SecondaryNameNode para el Escenario 1

Espacio en disco

Tamaño	Media
10%	929131
20%	1825319
30%	2726698
40%	3632567
50%	4513076
60%	5708244
70%	6636512
80%	7533868
90%	8437583
100%	9341867

Tabla 4-6: Ocupación HDFS. Escenario 1

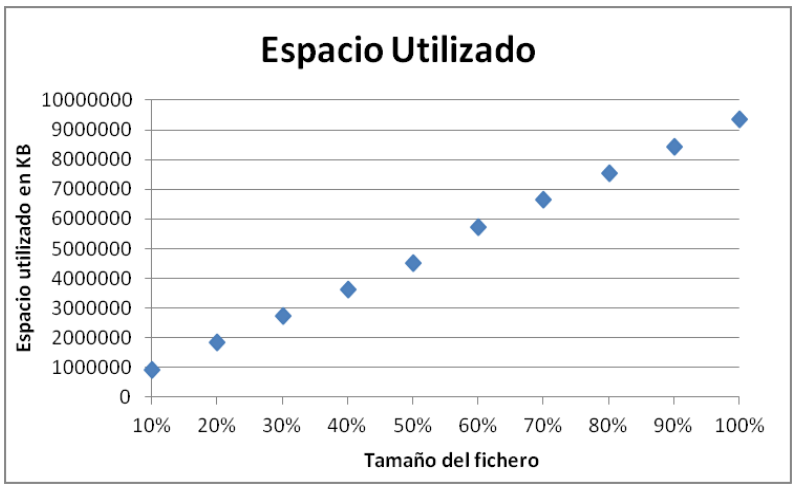


Figura 4.11: Evolución de la ocupación de espacio en disco. Escenario 1

4.2.2. Dos Nodos

Esta ejecución ha sido realizada en un *cluster* con dos nodos. Las máquinas utilizadas tiene las siguientes direcciones IP: master: 163.117.141.188, slave01: 163.117.141.189. La máquina master hace de maestro y esclavo a la vez.

Tiempos

DataNode

Tamaño	Media	Desv. Típ	Intervalo de conf (95%)	Intervalo Inferior	Intervalo Superior
10%	00:03,78	00:01,75	00:00,63	00:03,15	00:04,40
20%	00:09,39	00:04,06	00:01,45	00:07,93	00:10,84
30%	00:09,04	00:02,18	00:00,78	00:08,26	00:09,82
40%	00:16,50	00:07,39	00:02,64	00:13,86	00:19,14
50%	00:24,47	00:09,97	00:03,57	00:20,91	00:28,04
60%	00:34,56	00:11,92	00:04,26	00:30,29	00:38,82
70%	00:32,98	00:14,09	00:05,04	00:27,94	00:38,02
80%	00:42,07	00:16,47	00:05,90	00:36,17	00:47,96
90%	00:51,68	00:17,49	00:06,26	00:45,42	00:57,94
100%	00:44,27	00:18,79	00:06,73	00:37,55	00:51,00

Tabla 4-7: Datos del proceso DataNode para el Escenario 2

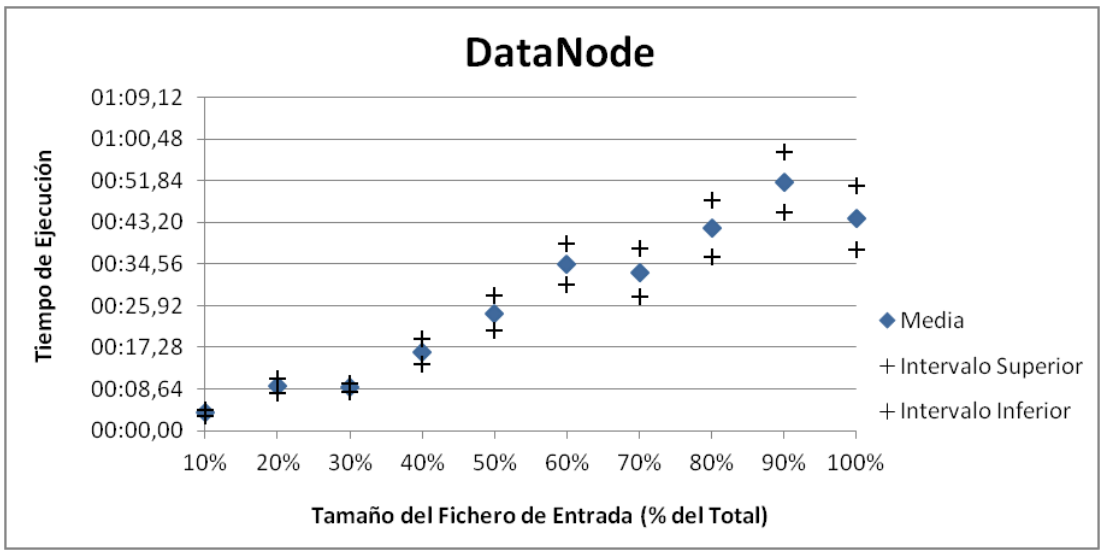


Figura 4.12: Representación de los Datos del proceso DataNode para el Escenario 2

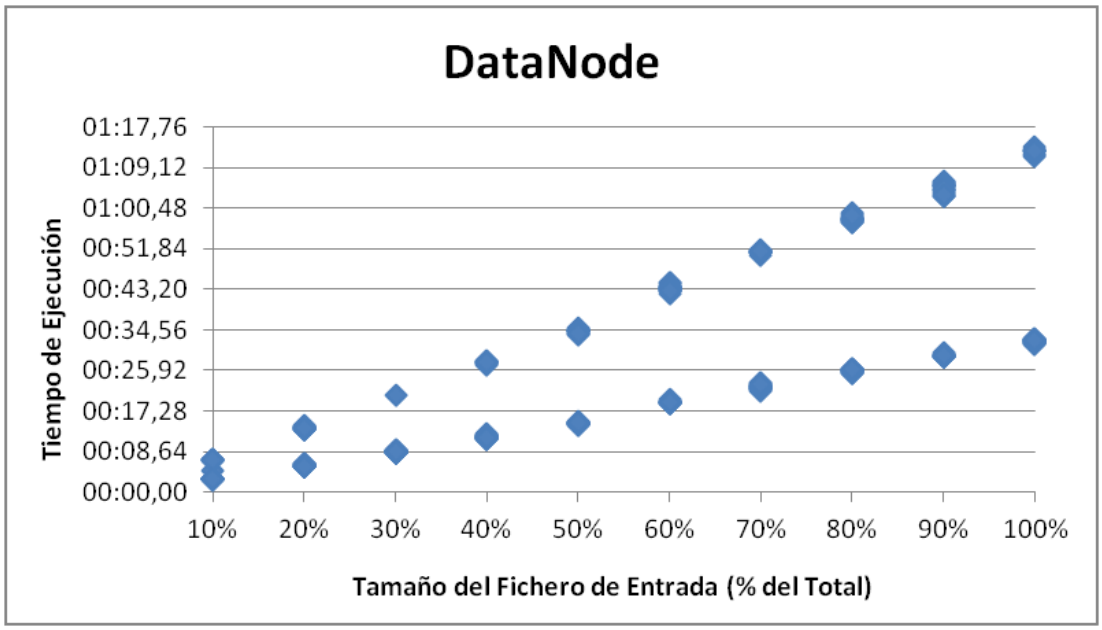


Figura 4.13: Total de los datos del proceso DataNode para el Escenario 2

TaskTracker

Tamaño	Media	Desv. Típ	Intervalo de conf (95%)	Intervalo Inferior	Intervalo Superior
10%	00:01,09	00:00,36	00:00,13	00:00,96	00:01,22
20%	00:01,55	00:00,22	00:00,08	00:01,47	00:01,63
30%	00:02,04	00:00,07	00:00,02	00:02,01	00:02,06
40%	00:02,78	00:00,29	00:00,10	00:02,67	00:02,88
50%	00:03,42	00:00,30	00:00,11	00:03,31	00:03,52
60%	00:04,38	00:00,49	00:00,18	00:04,20	00:04,56
70%	00:04,79	00:00,44	00:00,16	00:04,64	00:04,95
80%	00:05,58	00:00,58	00:00,21	00:05,37	00:05,78
90%	00:06,27	00:00,47	00:00,17	00:06,10	00:06,43
100%	00:06,62	00:00,46	00:00,16	00:06,46	00:06,79

Tabla 4-8: Datos del proceso TaskTracker para el Escenario 2

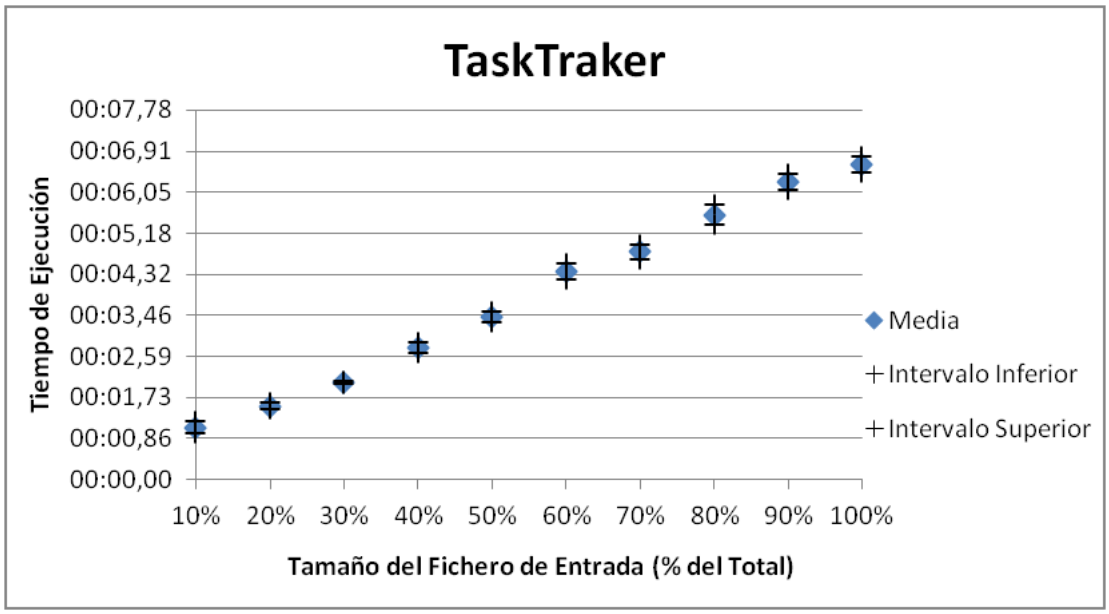


Figura 4.14: Representación de los Datos del proceso TaskTracker para el Escenario 2

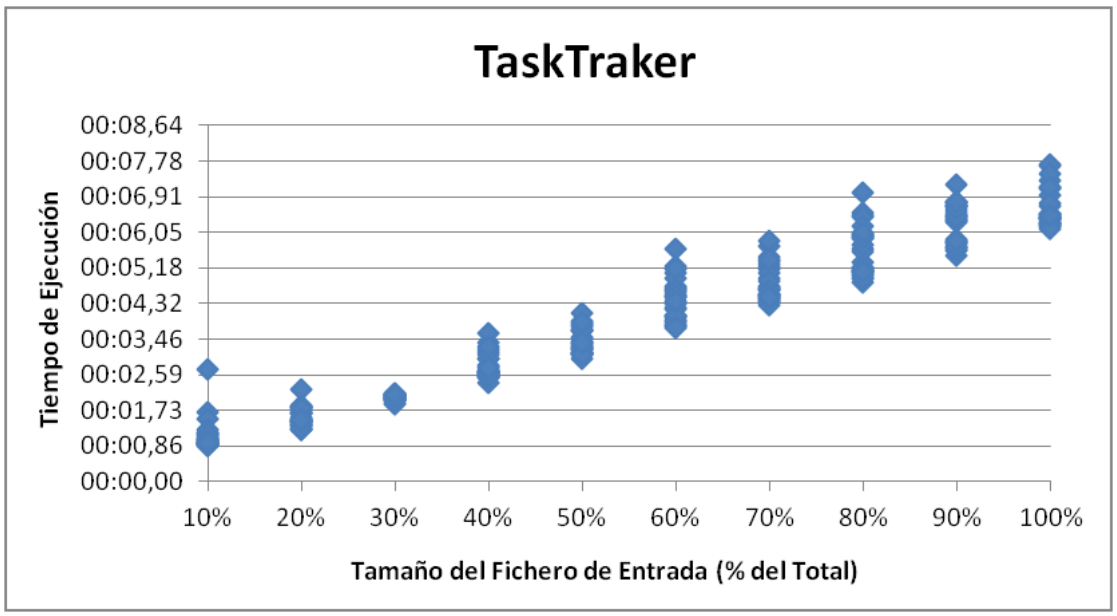


Figura 4.15: Total de los datos del proceso TaskTracker para el Escenario 2

JobTraker

Tamaño	Media	Desv. Típ	Intervalo de conf (95%)	Intervalo Inferior	Intervalo Superior
10%	00:00,35	00:00,35	00:00,13	00:00,22	00:00,47
20%	00:00,32	00:00,06	00:00,02	00:00,29	00:00,34
30%	00:00,33	00:00,04	00:00,01	00:00,32	00:00,35
40%	00:00,42	00:00,05	00:00,02	00:00,40	00:00,44
50%	00:00,48	00:00,03	00:00,01	00:00,47	00:00,49
60%	00:00,58	00:00,03	00:00,01	00:00,57	00:00,60
70%	00:00,65	00:00,03	00:00,01	00:00,63	00:00,66
80%	00:00,74	00:00,05	00:00,02	00:00,72	00:00,76
90%	00:00,82	00:00,10	00:00,03	00:00,78	00:00,85
100%	00:00,87	00:00,08	00:00,03	00:00,84	00:00,89

Tabla 4-9: Datos del proceso JobTracker para el Escenario 2

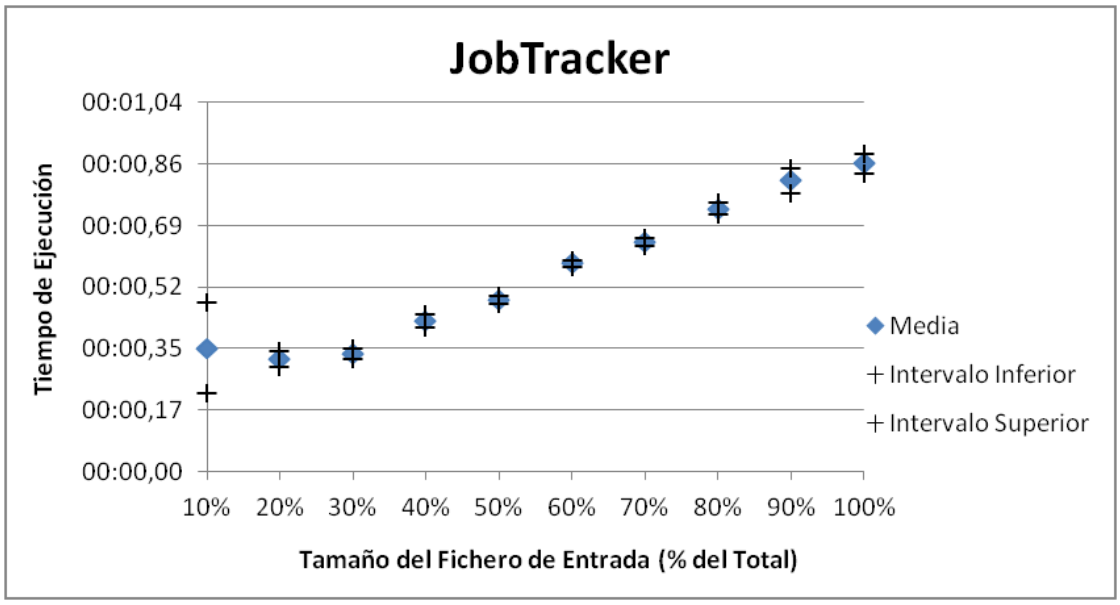


Figura 4.16: Representación de los Datos del proceso JobTracker para el Escenario 2

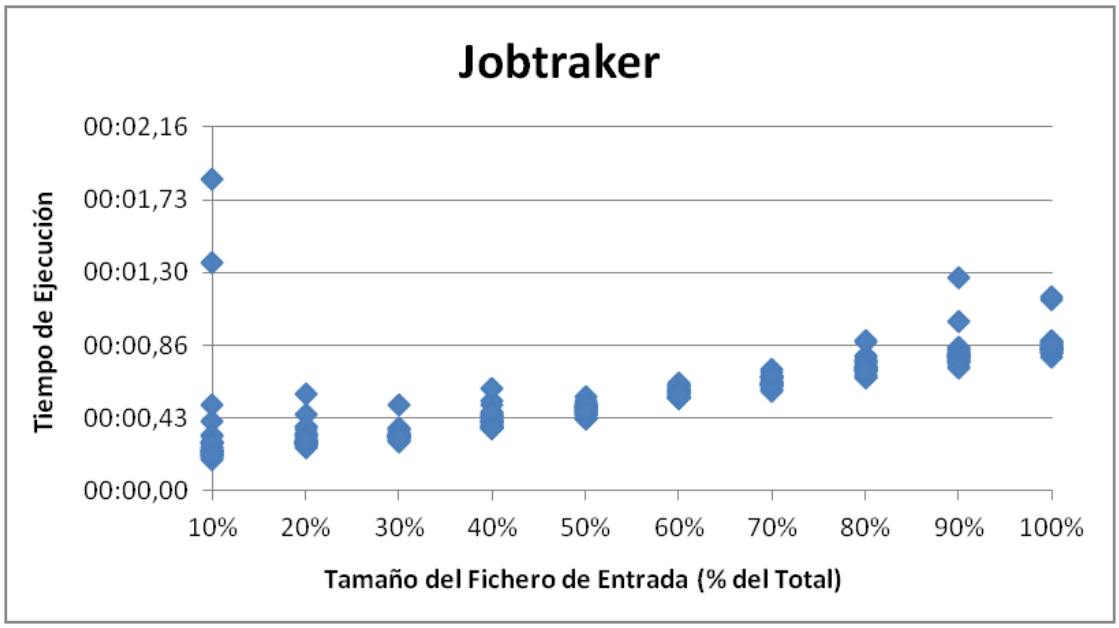


Figura 4.17: Total de los datos del proceso JobTracker para el Escenario 2

NameNode

Tamaño	Media	Desv. Típ	Intervalo de conf (95%)	Intervalo Inferior	Intervalo Superior
10%	00:00,16	00:00,04	00:00,01	00:00,15	00:00,18
20%	00:00,21	00:00,04	00:00,01	00:00,19	00:00,22
30%	00:00,25	00:00,08	00:00,03	00:00,22	00:00,27
40%	00:00,28	00:00,02	00:00,01	00:00,27	00:00,29
50%	00:00,34	00:00,02	00:00,01	00:00,33	00:00,35
60%	00:00,40	00:00,02	00:00,01	00:00,39	00:00,41
70%	00:00,46	00:00,03	00:00,01	00:00,45	00:00,47
80%	00:00,53	00:00,05	00:00,02	00:00,51	00:00,54
90%	00:00,57	00:00,02	00:00,01	00:00,57	00:00,58
100%	00:00,62	00:00,02	00:00,01	00:00,61	00:00,63

Tabla 4-10: Datos del proceso NameNode para el Escenario 2

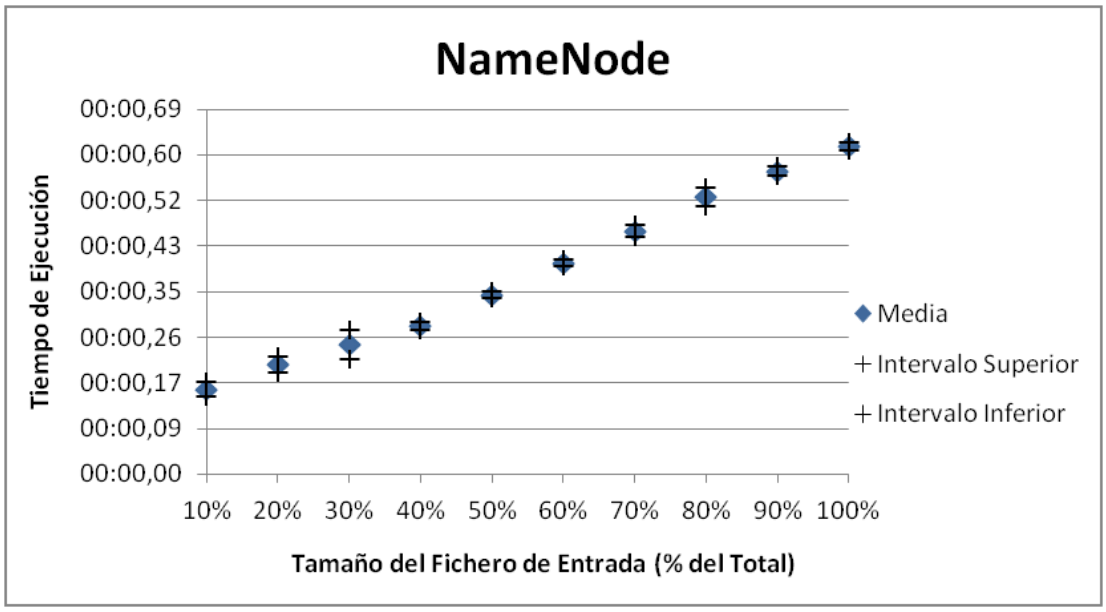


Figura 4.18: Representación de los Datos del proceso NameNode para el Escenario 2

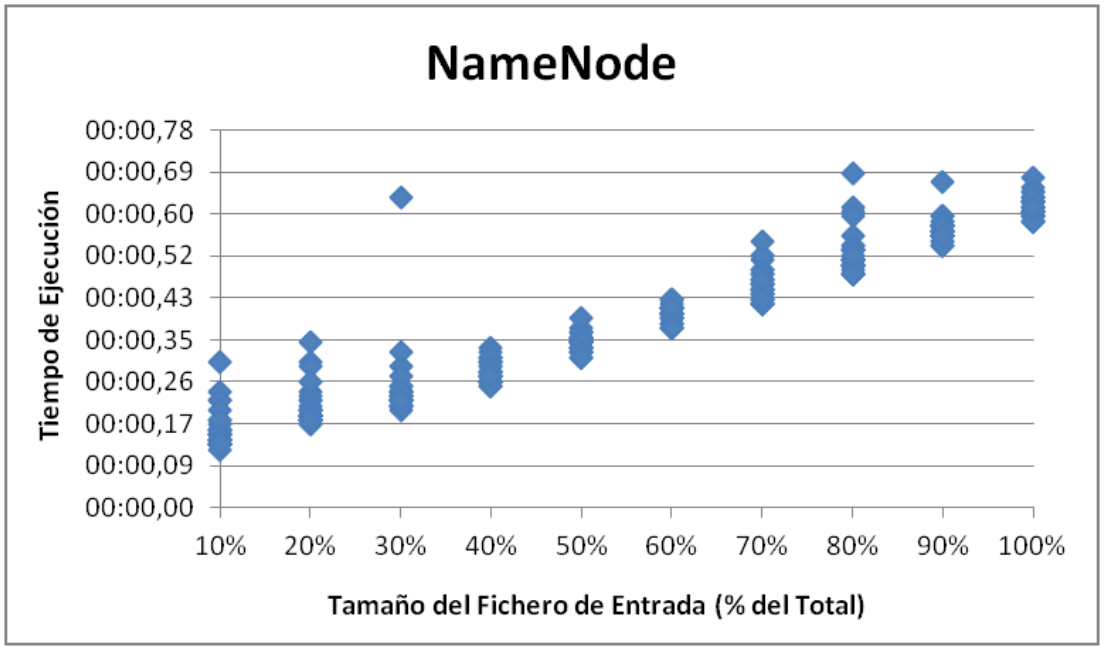


Figura 4.19: Total de los datos del proceso NameNode para el Escenario 2

SecondaryNameNode

Tamaño	Media	Desv. Típ	Intervalo de conf (95%)	Intervalo Inferior	Intervalo Superior
10%	00:00,03	00:00,06	00:00,02	00:00,01	00:00,05
20%	00:00,05	00:00,10	00:00,03	00:00,01	00:00,08
30%	00:00,04	00:00,02	00:00,01	00:00,03	00:00,05
40%	00:00,06	00:00,08	00:00,03	00:00,03	00:00,09
50%	00:00,06	00:00,03	00:00,01	00:00,05	00:00,07
60%	00:00,07	00:00,02	00:00,01	00:00,06	00:00,07
70%	00:00,08	00:00,01	00:00,01	00:00,07	00:00,08
80%	00:00,09	00:00,02	00:00,01	00:00,08	00:00,09
90%	00:00,10	00:00,02	00:00,01	00:00,09	00:00,10
100%	00:00,11	00:00,02	00:00,01	00:00,11	00:00,12

Tabla 4-11: Datos del proceso SecondaryNameNode para el Escenario 2

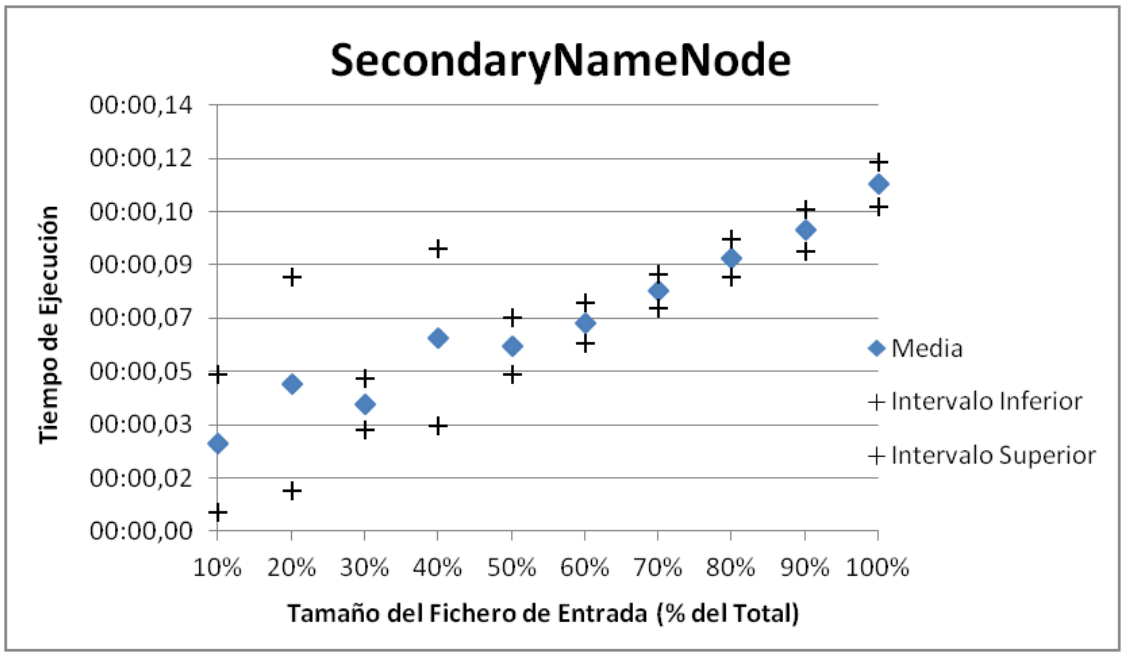


Figura 4.20: Representación de los Datos del proceso SecondaryNameNode para el Escenario 2

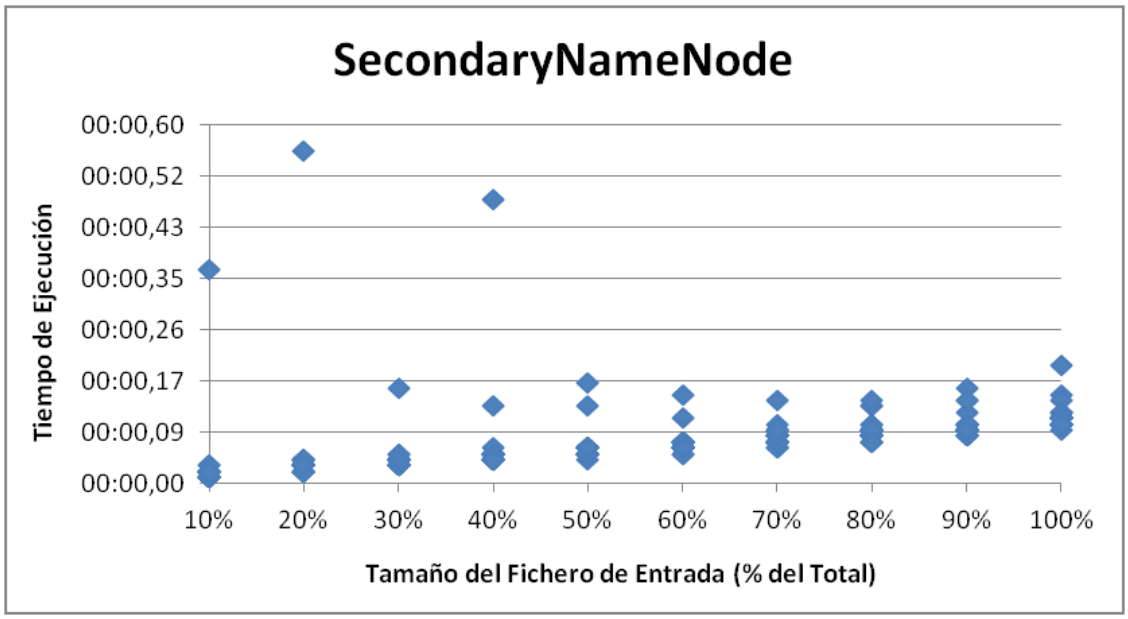


Figura 4.21: Total de los datos del proceso SecondaryNameNode para el Escenario 2

Espacio en disco

Tamaño	Media
10%	929133
20%	1825318
30%	2726699
40%	3632558
50%	4513059
60%	5708247
70%	6636507
80%	7533884
90%	8437599
100%	9341866

Tabla 4-12: Ocupación HDFS. Escenario 2

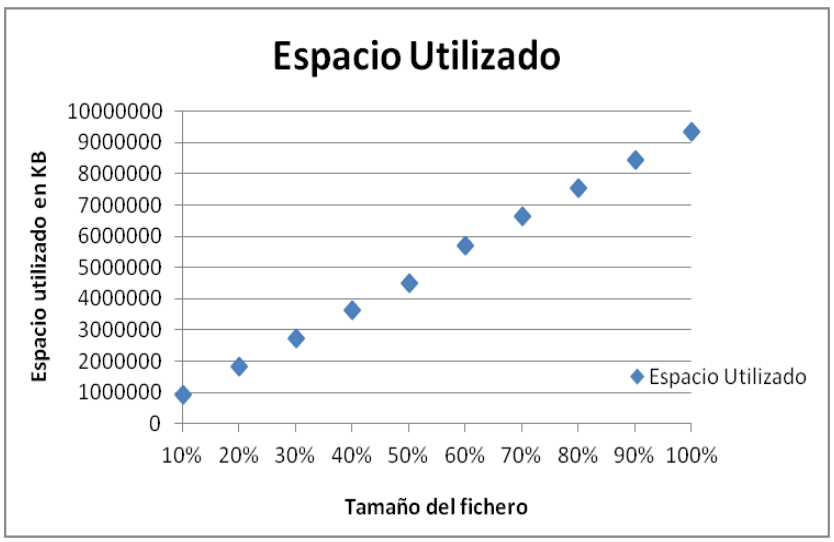


Figura 4.22: Evolución de la ocupación de espacio en disco. Escenario 2

4.2.3. Tres Nodos

Esta ejecución ha sido realizada en un *cluster* con tres nodos. Las máquinas utilizadas tiene las siguientes direcciones IP: master: 163.117.141.188, slave01: 163.117.141.189, slave02: 163.117.141.190. La máquina master hace de maestro y esclavo a la vez.

Tiempos

DataNode

Tamaño	Media	Desv. Típ	Intervalo de conf (95%)	Intervalo Inferior	Intervalo Superior
10%	00:05,14	00:02,39	00:00,86	00:04,29	00:06,00
20%	00:09,18	00:04,71	00:01,69	00:07,49	00:10,87
30%	00:19,43	00:06,00	00:02,15	00:17,29	00:21,58
40%	00:20,37	00:10,25	00:03,67	00:16,70	00:24,04
50%	00:28,68	00:11,20	00:04,01	00:24,67	00:32,68
60%	00:35,77	00:14,19	00:05,08	00:30,69	00:40,85
70%	00:44,58	00:16,97	00:06,07	00:38,50	00:50,65
80%	00:43,34	00:20,71	00:07,41	00:35,92	00:50,75
90%	00:54,43	00:14,92	00:05,34	00:49,09	00:59,76
100%	00:58,37	00:18,14	00:06,49	00:51,88	01:04,86

Tabla 4-13: Datos del proceso DataNode para el Escenario 3

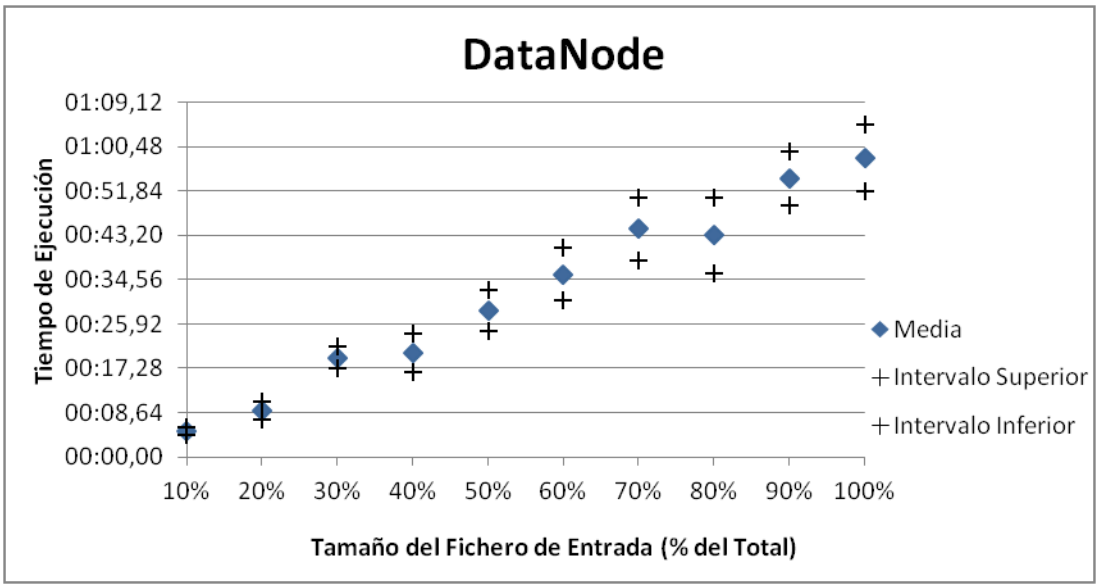


Figura 4.23: Representación de los Datos del proceso DataNode para el Escenario 3

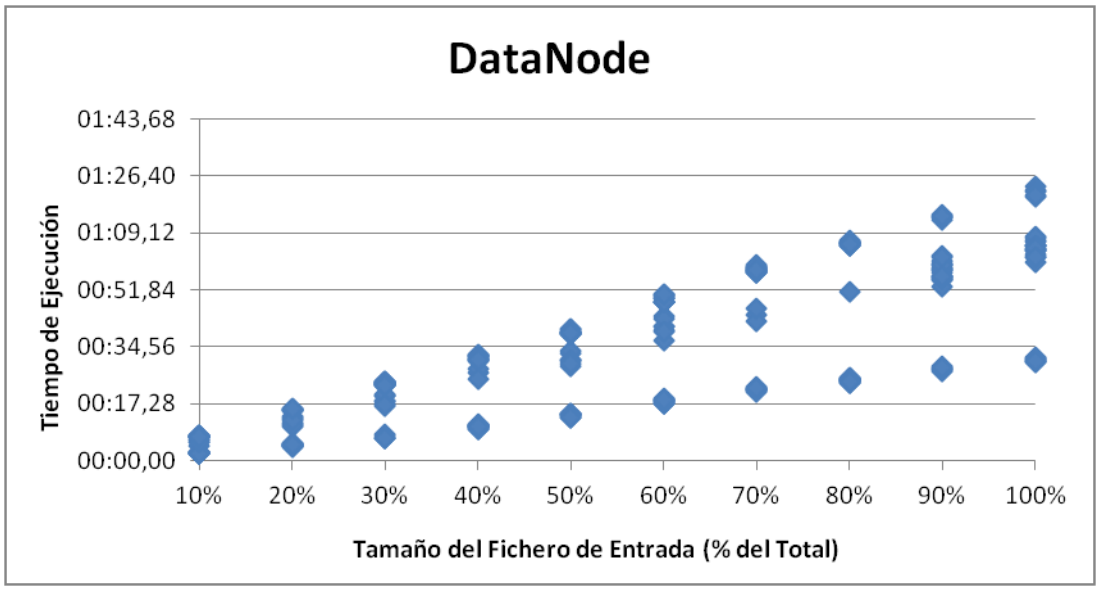


Figura 4.24: Total de los datos del proceso DataNode para el Escenario 3

TaskTracker

Tamaño	Media	Desv. Típ	Intervalo de conf (95%)	Intervalo Inferior	Intervalo Superior
10%	00:00,82	00:00,38	00:00,13	00:00,68	00:00,95
20%	00:01,17	00:00,16	00:00,06	00:01,11	00:01,22
30%	00:01,80	00:00,25	00:00,09	00:01,72	00:01,89
40%	00:02,27	00:00,30	00:00,11	00:02,17	00:02,38
50%	00:02,95	00:00,45	00:00,16	00:02,79	00:03,11
60%	00:03,62	00:00,52	00:00,19	00:03,43	00:03,80
70%	00:04,35	00:00,71	00:00,25	00:04,10	00:04,60
80%	00:04,74	00:00,87	00:00,31	00:04,43	00:05,05
90%	00:05,48	00:00,65	00:00,23	00:05,25	00:05,71
100%	00:05,90	00:00,88	00:00,32	00:05,58	00:06,21

Tabla 4-14: Datos del proceso TaskTracker para el Escenario 3

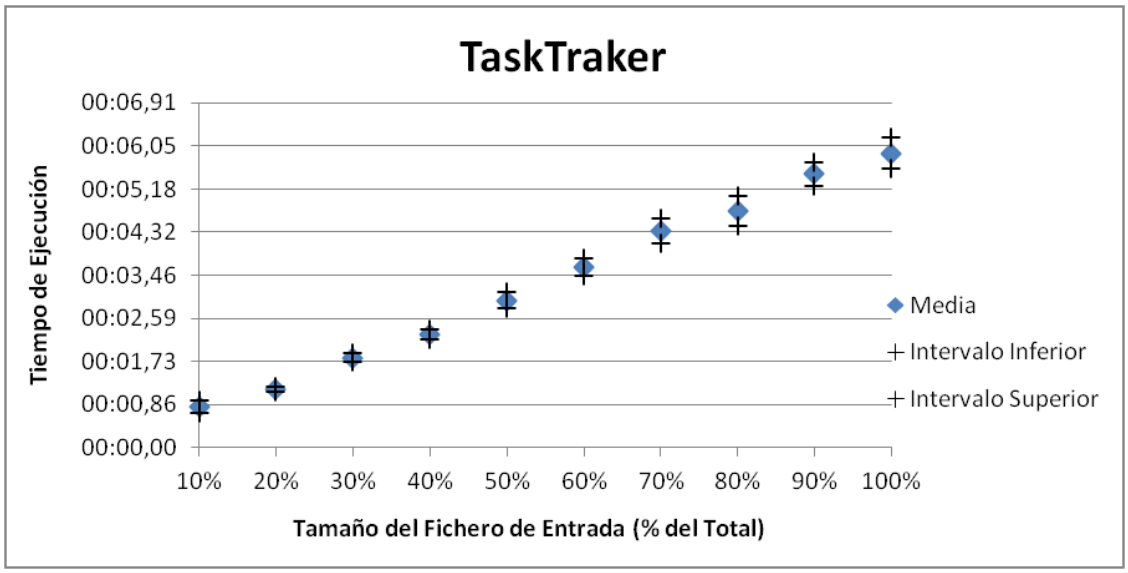


Figura 4.25: Representación de los Datos del proceso TaskTracker para el Escenario 3

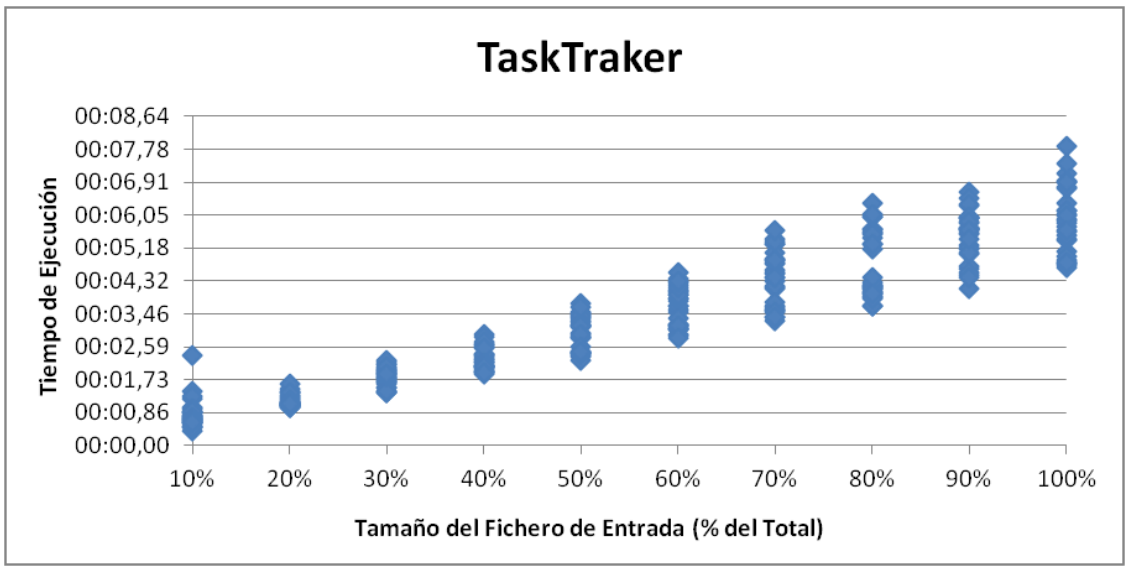


Figura 4.26: Total de los datos del proceso TaskTracker para el Escenario 3

JobTraker

Tamaño	Media	Desv. Típ	Intervalo de conf (95%)	Intervalo Inferior	Intervalo Superior
10%	00:00,38	00:00,31	00:00,11	00:00,27	00:00,49
20%	00:00,40	00:00,07	00:00,02	00:00,38	00:00,43
30%	00:00,51	00:00,04	00:00,02	00:00,49	00:00,52
40%	00:00,57	00:00,04	00:00,01	00:00,56	00:00,59
50%	00:00,67	00:00,07	00:00,02	00:00,65	00:00,70
60%	00:00,82	00:00,06	00:00,02	00:00,79	00:00,84
70%	00:00,95	00:00,08	00:00,03	00:00,92	00:00,97
80%	00:01,10	00:00,39	00:00,14	00:00,96	00:01,24
90%	00:01,13	00:00,08	00:00,03	00:01,10	00:01,16
100%	00:01,23	00:00,14	00:00,05	00:01,18	00:01,27

Tabla 4-15: Datos del proceso JobTracker para el Escenario 3

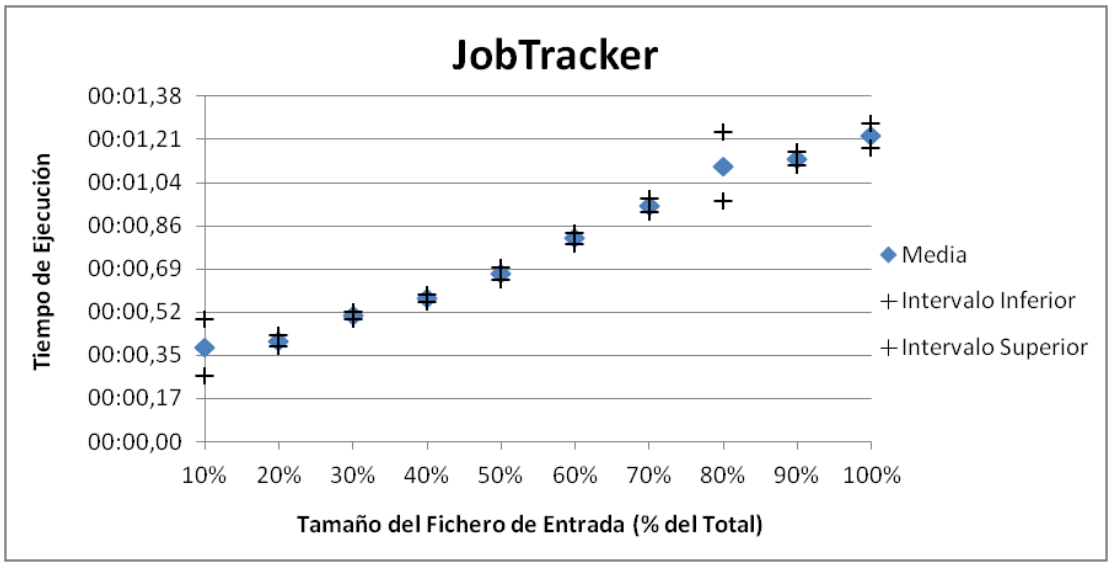


Figura 4.27: Representación de los Datos del proceso JobTracker para el Escenario 3

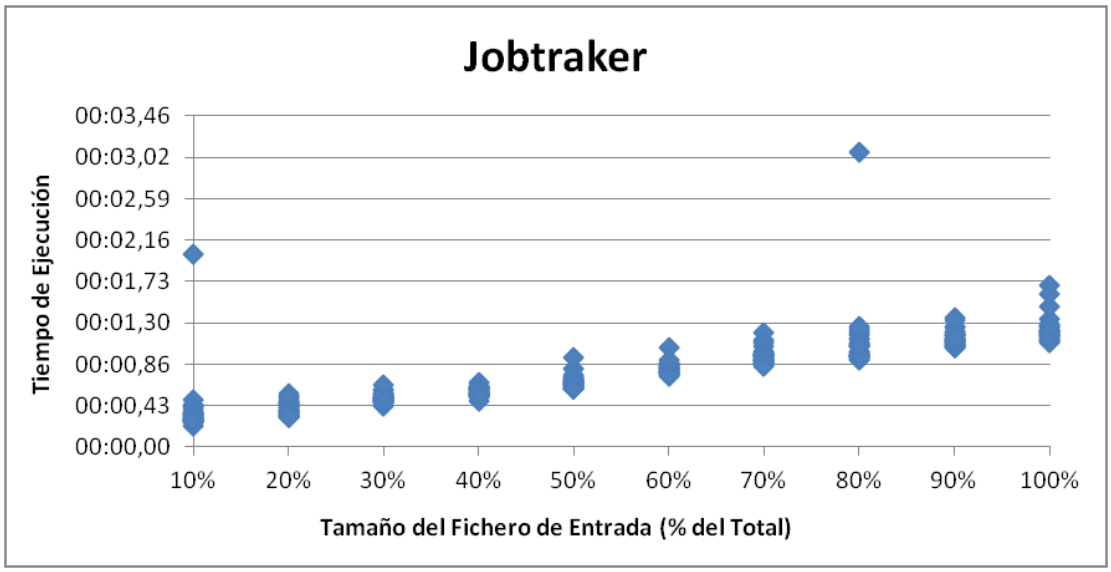


Figura 4.28: Total de los datos del proceso JobTracker para el Escenario 3

NameNode

Tamaño	Media	Desv. Típ	Intervalo de conf (95%)	Intervalo Inferior	Intervalo Superior
10%	00:00,20	00:00,04	00:00,01	00:00,19	00:00,22
20%	00:00,26	00:00,05	00:00,02	00:00,24	00:00,27
30%	00:00,32	00:00,08	00:00,03	00:00,29	00:00,35
40%	00:00,36	00:00,03	00:00,01	00:00,35	00:00,37
50%	00:00,43	00:00,04	00:00,01	00:00,42	00:00,45
60%	00:00,51	00:00,04	00:00,01	00:00,50	00:00,52
70%	00:00,46	00:00,09	00:00,03	00:00,43	00:00,49
80%	00:00,44	00:00,03	00:00,01	00:00,43	00:00,45
90%	00:00,47	00:00,03	00:00,01	00:00,46	00:00,48
100%	00:00,50	00:00,02	00:00,01	00:00,49	00:00,51

Tabla 4-16: Datos del proceso NameNode para el Escenario 3

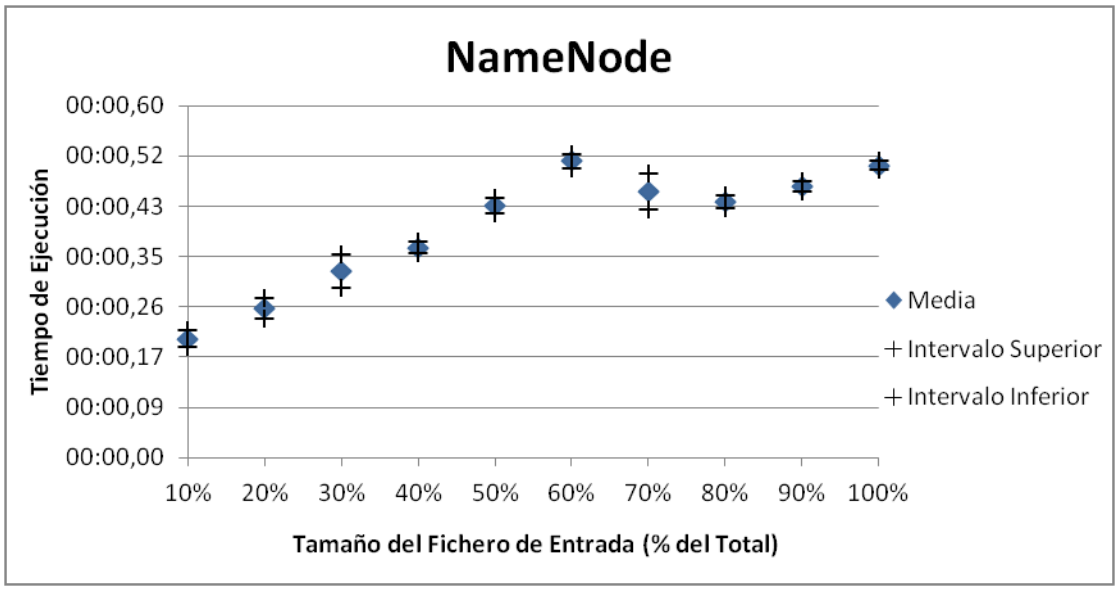


Figura 4.29: Representación de los Datos del proceso NameNode para el Escenario 3

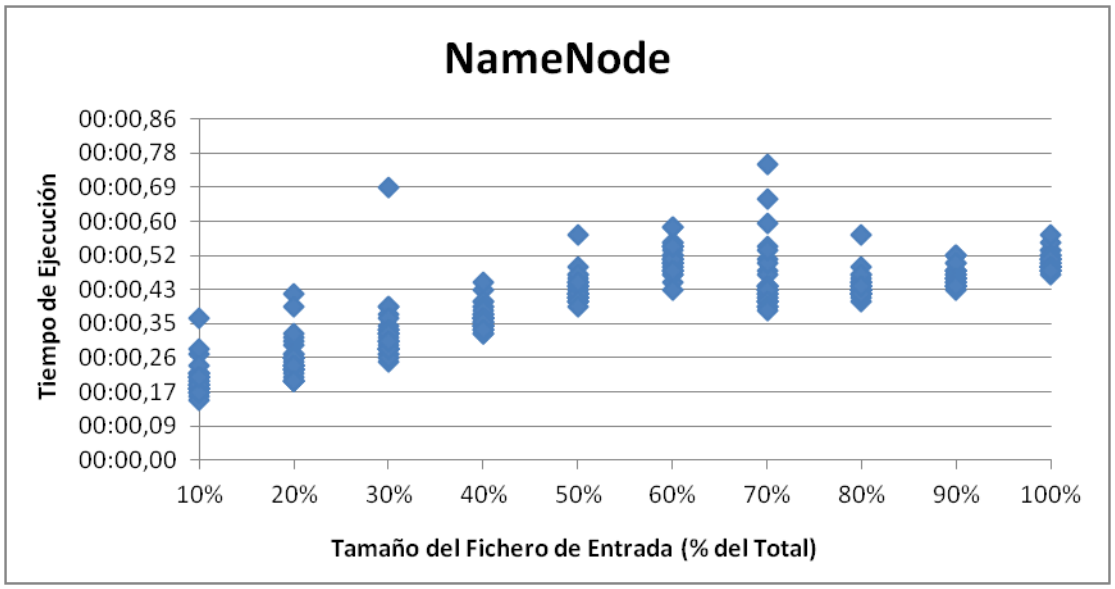


Figura 4.30: Total de los datos del proceso NameNode para el Escenario 3

SecondaryNameNode

Tamaño	Media	Desv. Típ	Intervalo de conf (95%)	Intervalo Inferior	Intervalo Superior
10%	00:00,03	00:00,05	00:00,02	00:00,01	00:00,05
20%	00:00,06	00:00,10	00:00,04	00:00,02	00:00,09
30%	00:00,05	00:00,02	00:00,01	00:00,05	00:00,06
40%	00:00,08	00:00,07	00:00,03	00:00,05	00:00,10
50%	00:00,07	00:00,02	00:00,01	00:00,07	00:00,08
60%	00:00,09	00:00,03	00:00,01	00:00,08	00:00,10
70%	00:00,11	00:00,02	00:00,01	00:00,10	00:00,11
80%	00:00,12	00:00,02	00:00,01	00:00,11	00:00,13
90%	00:00,12	00:00,02	00:00,01	00:00,12	00:00,13
100%	00:00,14	00:00,02	00:00,01	00:00,13	00:00,15

Tabla 4-17: Datos del proceso SecondaryNameNode para el Escenario 3

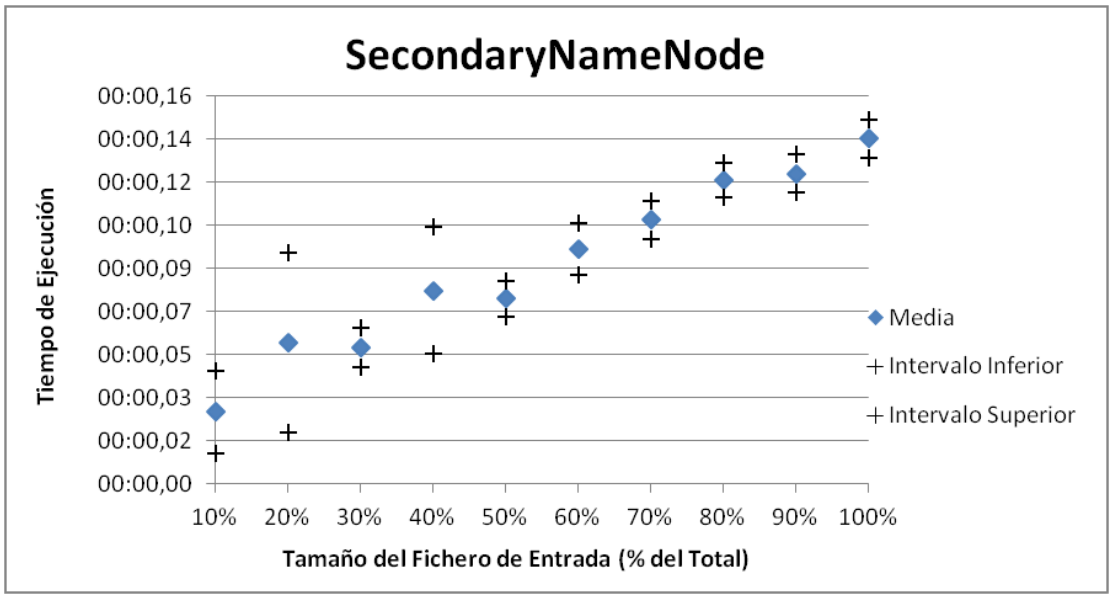


Figura 4.31: Representación de los Datos del proceso SecondaryNameNode para el Escenario 3

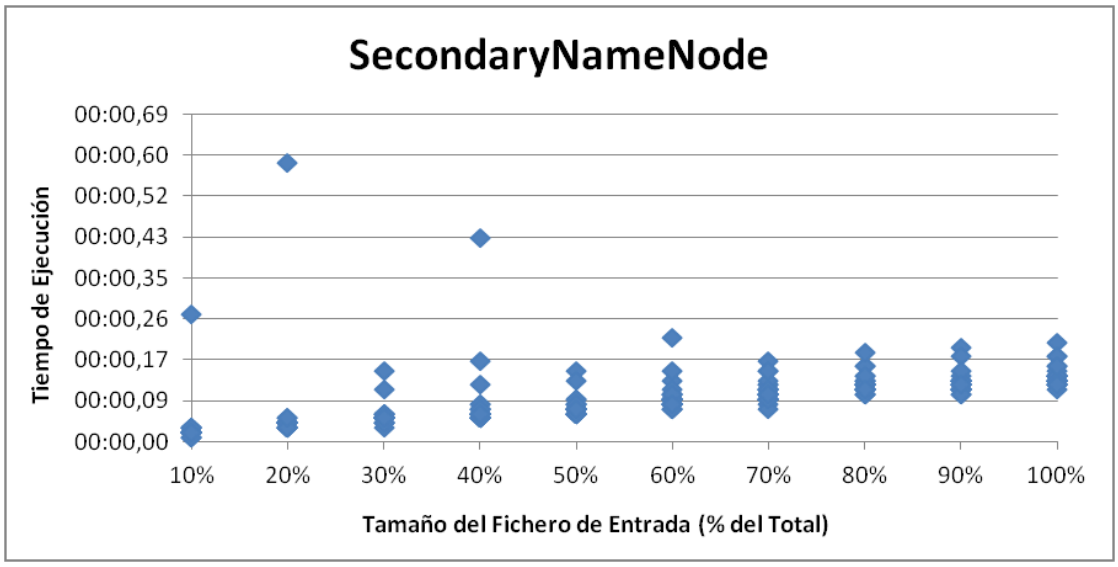


Figura 4.32: Total de los datos del proceso SecondaryNameNode para el Escenario 3

Espacio en disco

Tamaño	Media
10%	929159
20%	1825338
30%	2726735
40%	3632839
50%	4513095
60%	5708277
70%	6636548
80%	7533921
90%	8437626
100%	9341890

Tabla 4-18: Ocupación HDFS. Escenario 3

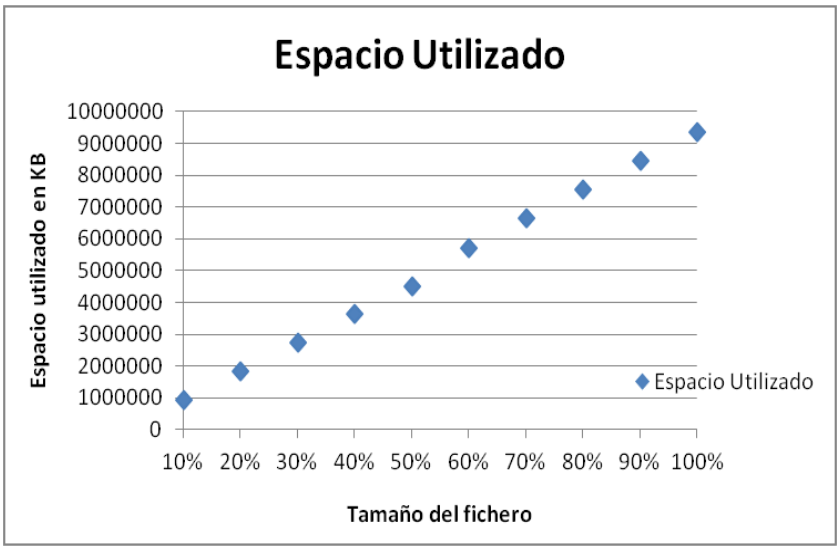


Figura 4.33: Evolución de la ocupación de espacio en disco. Escenario 3

4.2.4. Cuatro Nodos

Esta ejecución ha sido realizada en un *cluster* con cuatro nodos. Las máquinas utilizadas tiene las siguientes direcciones IP: master: 163.117.141.188, slave01: 163.117.141.189, slave02: 163.117.141.190, slave03: 163.117.141.191. La máquina master hace de maestro y esclavo a la vez.

Tiempos

DataNode

Tamaño	Media	Desv. Típ	Intervalo de conf (95%)	Intervalo Inferior	Intervalo Superior
10%	00:04,09	00:01,66	00:00,59	00:03,50	00:04,68
20%	00:11,33	00:03,35	00:01,20	00:10,13	00:12,53
30%	00:16,80	00:05,31	00:01,90	00:14,90	00:18,70
40%	00:22,20	00:05,46	00:01,96	00:20,24	00:24,15
50%	00:24,49	00:06,91	00:02,47	00:22,02	00:26,96
60%	00:31,76	00:11,07	00:03,96	00:27,80	00:35,72
70%	00:37,82	00:12,02	00:04,30	00:33,51	00:42,12
80%	00:48,74	00:10,53	00:03,77	00:44,98	00:52,51
90%	00:51,13	00:15,75	00:05,64	00:45,49	00:56,76
100%	00:55,06	00:17,54	00:06,28	00:48,79	01:01,34

Tabla 4-19: Datos del proceso DataNode para el Escenario 4

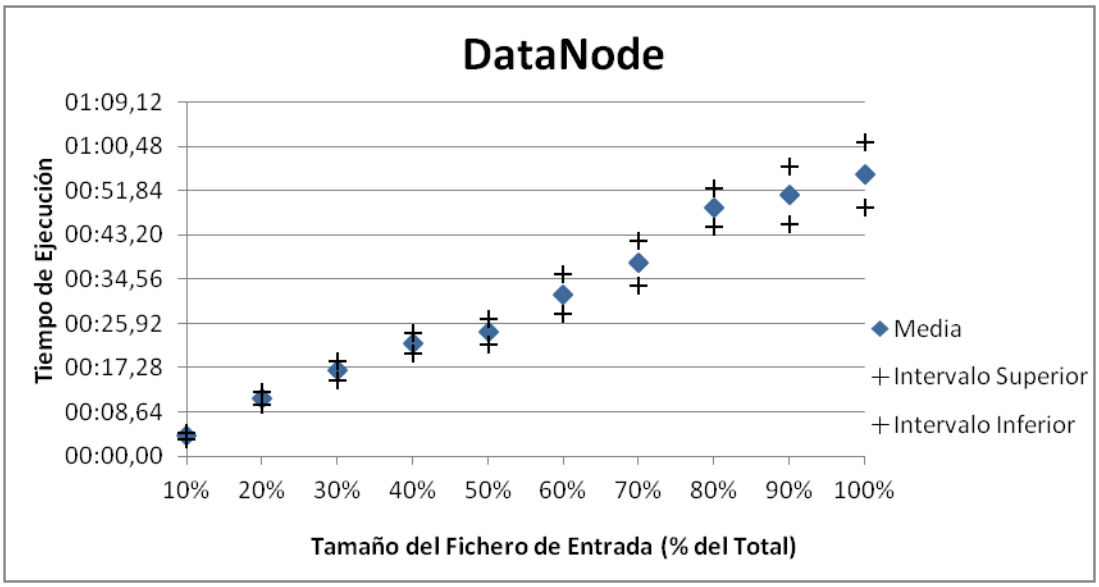


Figura 4.34: Representación de los Datos del proceso DataNode para el Escenario 4

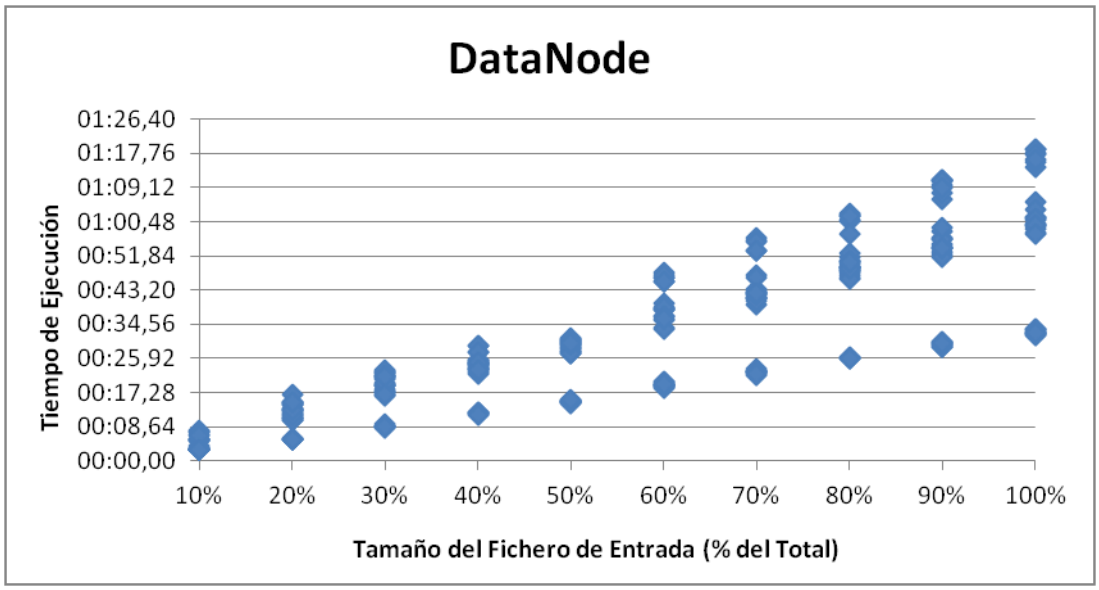


Figura 4.35: Total de los datos del proceso DataNode para el Escenario 4

TaskTracker

Tamaño	Media	Desv. Típ	Intervalo de conf (95%)	Intervalo Inferior	Intervalo Superior
10%	00:00,76	00:00,34	00:00,12	00:00,64	00:00,88
20%	00:01,03	00:00,26	00:00,09	00:00,94	00:01,13
30%	00:01,37	00:00,18	00:00,06	00:01,31	00:01,44
40%	00:01,71	00:00,23	00:00,08	00:01,63	00:01,79
50%	00:01,96	00:00,22	00:00,08	00:01,88	00:02,04
60%	00:02,59	00:00,46	00:00,16	00:02,43	00:02,75
70%	00:02,88	00:00,43	00:00,15	00:02,72	00:03,03
80%	00:03,47	00:00,50	00:00,18	00:03,29	00:03,65
90%	00:03,78	00:00,78	00:00,28	00:03,51	00:04,06
100%	00:04,21	00:00,80	00:00,29	00:03,93	00:04,50

Tabla 4-20: Datos del proceso TaskTracker para el Escenario 4

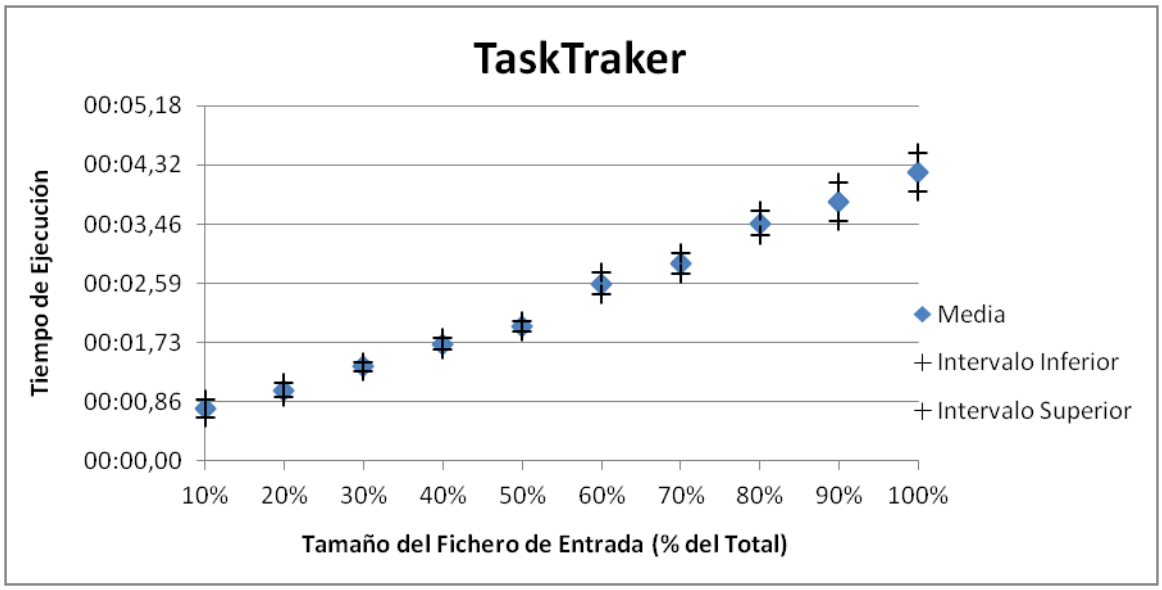


Figura 4.36: Representación de los Datos del proceso TaskTracker para el Escenario 4

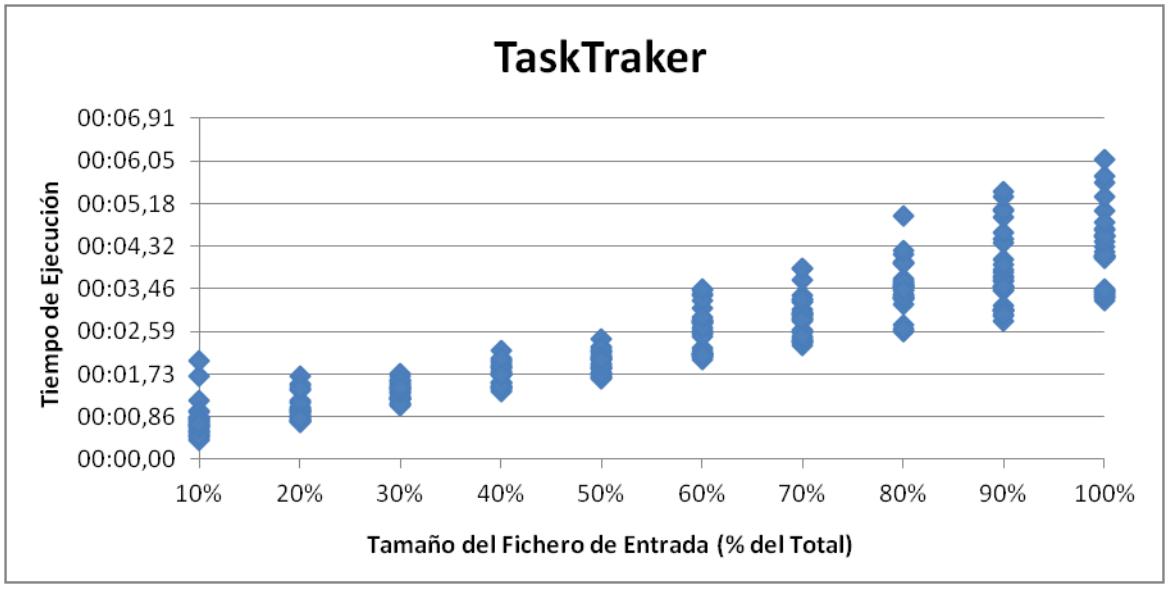


Figura 4.37: Total de los datos del proceso TaskTracker para el Escenario 4

JobTraker

Tamaño	Media	Desv. Típ	Intervalo de conf (95%)	Intervalo Inferior	Intervalo Superior
10%	00:00,38	00:00,31	00:00,11	00:00,26	00:00,49
20%	00:00,43	00:00,08	00:00,03	00:00,40	00:00,46
30%	00:00,57	00:00,29	00:00,11	00:00,47	00:00,68
40%	00:00,53	00:00,07	00:00,02	00:00,51	00:00,56
50%	00:00,51	00:00,03	00:00,01	00:00,50	00:00,51
60%	00:00,65	00:00,12	00:00,04	00:00,60	00:00,69
70%	00:00,79	00:00,29	00:00,10	00:00,68	00:00,89
80%	00:00,76	00:00,09	00:00,03	00:00,73	00:00,79
90%	00:00,86	00:00,10	00:00,03	00:00,83	00:00,90
100%	00:00,92	00:00,14	00:00,05	00:00,87	00:00,97

Tabla 4-21: Datos del proceso JobTracker para el Escenario 4

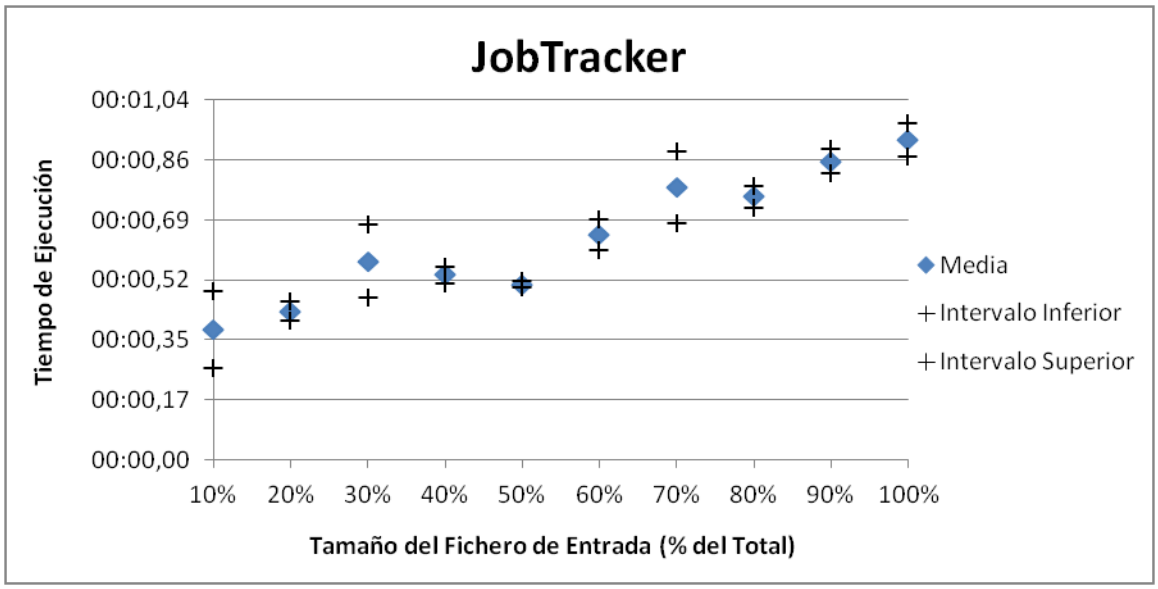


Figura 4.38: Representación de los Datos del proceso JobTracker para el Escenario 4

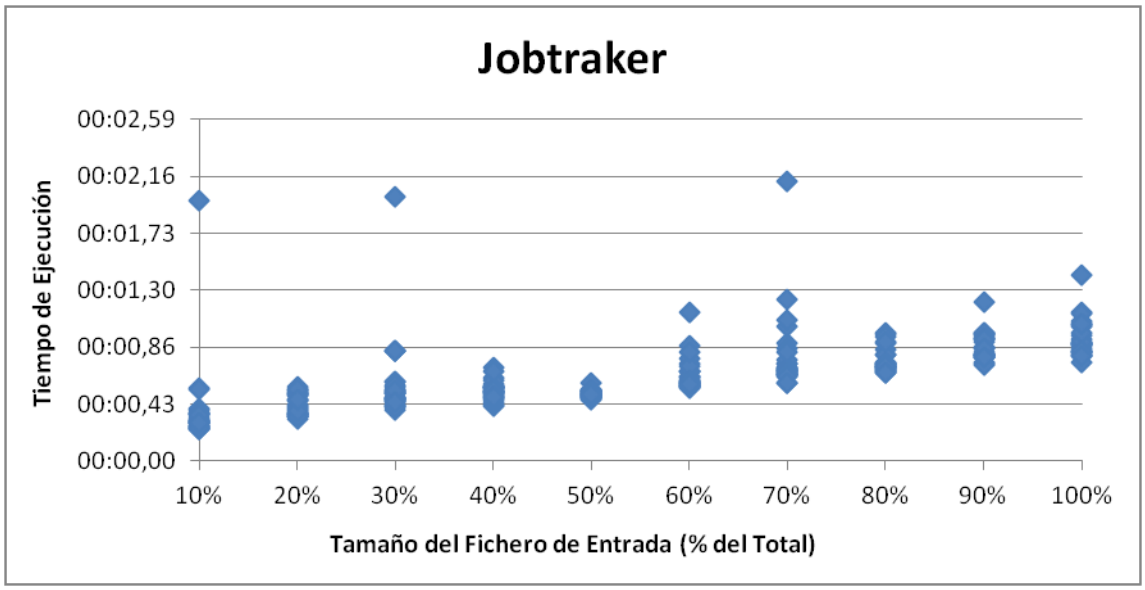


Figura 4.39: Total de los datos del proceso JobTracker para el Escenario 4

NameNode

Tamaño	Media	Desv. Típ	Intervalo de conf (95%)	Intervalo Inferior	Intervalo Superior
10%	00:00,22	00:00,05	00:00,02	00:00,21	00:00,24
20%	00:00,29	00:00,07	00:00,03	00:00,26	00:00,31
30%	00:00,36	00:00,08	00:00,03	00:00,33	00:00,39
40%	00:00,36	00:00,05	00:00,02	00:00,34	00:00,38
50%	00:00,35	00:00,06	00:00,02	00:00,33	00:00,37
60%	00:00,40	00:00,08	00:00,03	00:00,37	00:00,43
70%	00:00,44	00:00,04	00:00,01	00:00,42	00:00,45
80%	00:00,50	00:00,03	00:00,01	00:00,49	00:00,51
90%	00:00,55	00:00,06	00:00,02	00:00,53	00:00,57
100%	00:00,58	00:00,04	00:00,01	00:00,57	00:00,60

Tabla 4-22: Datos del proceso NameNode para el Escenario 4

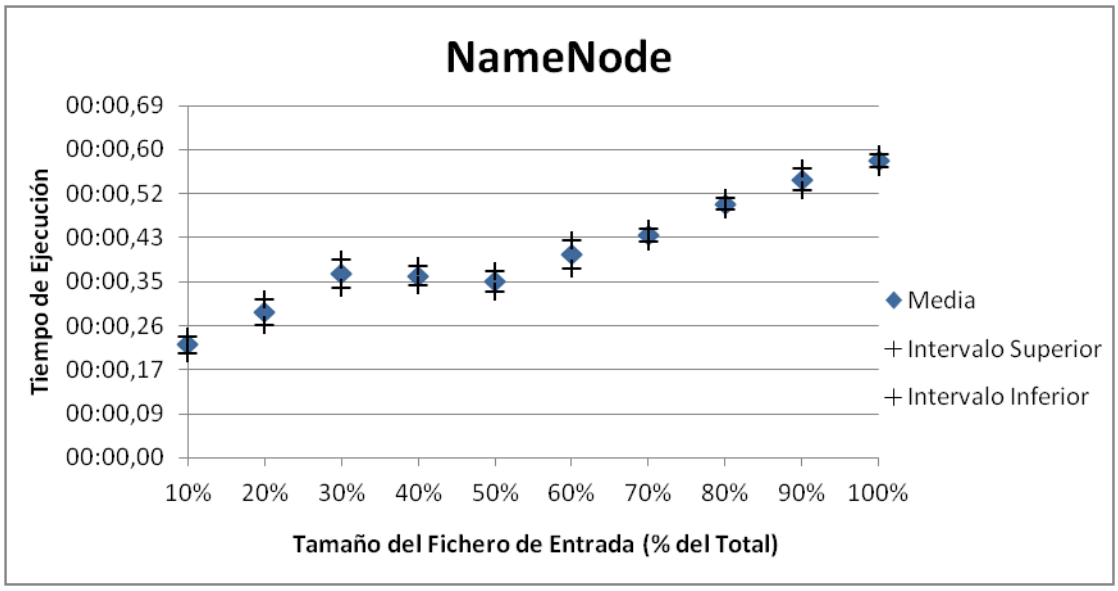


Figura 4.40: Representación de los Datos del proceso NameNode para el Escenario 4

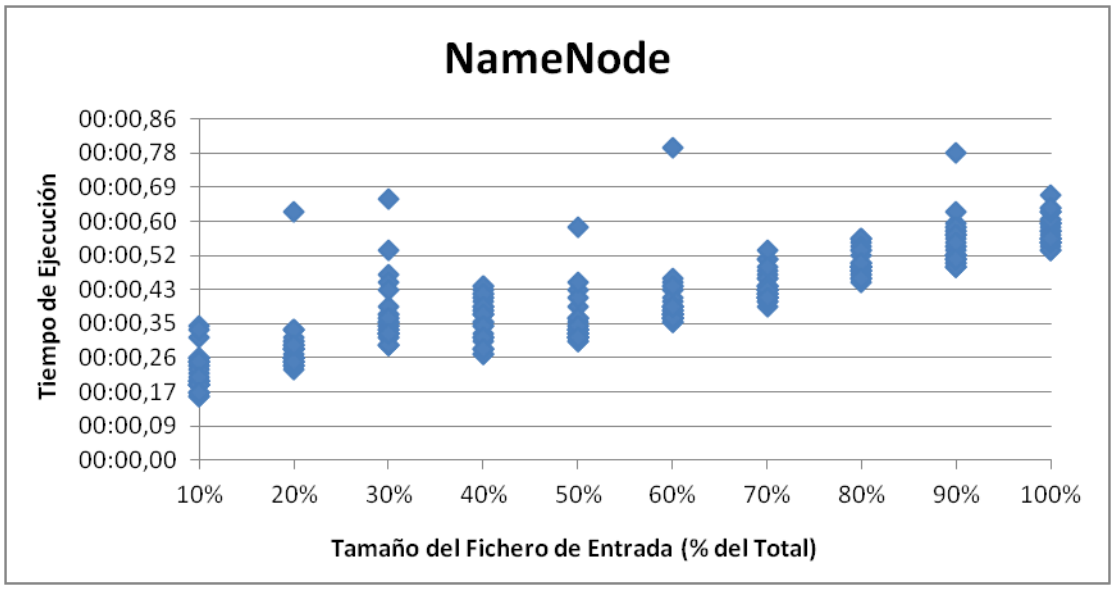


Figura 4.41: Total de los datos del proceso NameNode para el Escenario 4

SecondaryNameNode

Tamaño	Media	Desv. Típ	Intervalo de conf (95%)	Intervalo Inferior	Intervalo Superior
10%	00:00,02	00:00,01	00:00,00	00:00,02	00:00,02
20%	00:00,05	00:00,11	00:00,04	00:00,01	00:00,09
30%	00:00,06	00:00,03	00:00,01	00:00,05	00:00,07
40%	00:00,06	00:00,05	00:00,02	00:00,04	00:00,08
50%	00:00,06	00:00,02	00:00,01	00:00,05	00:00,07
60%	00:00,08	00:00,03	00:00,01	00:00,07	00:00,09
70%	00:00,09	00:00,02	00:00,01	00:00,09	00:00,10
80%	00:00,11	00:00,02	00:00,01	00:00,10	00:00,11
90%	00:00,12	00:00,02	00:00,01	00:00,11	00:00,12
100%	00:00,13	00:00,03	00:00,01	00:00,12	00:00,14

Tabla 4-23: Datos del proceso SecondaryNameNode para el Escenario 4

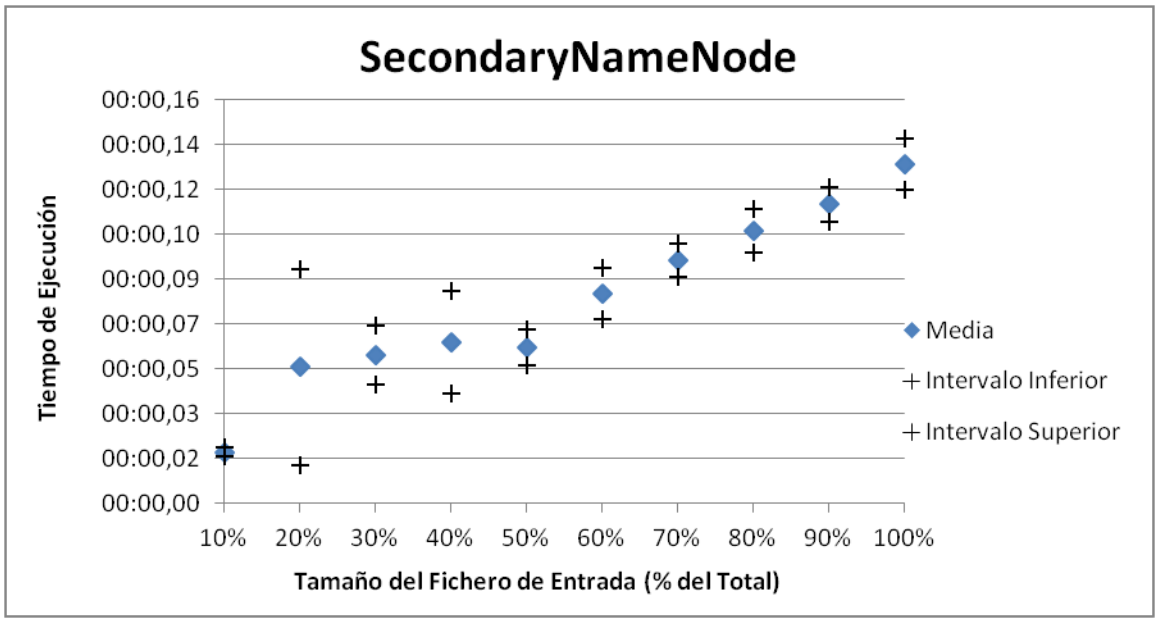


Figura 4.42: Representación de los Datos del proceso SecondaryNameNode para el Escenario 4

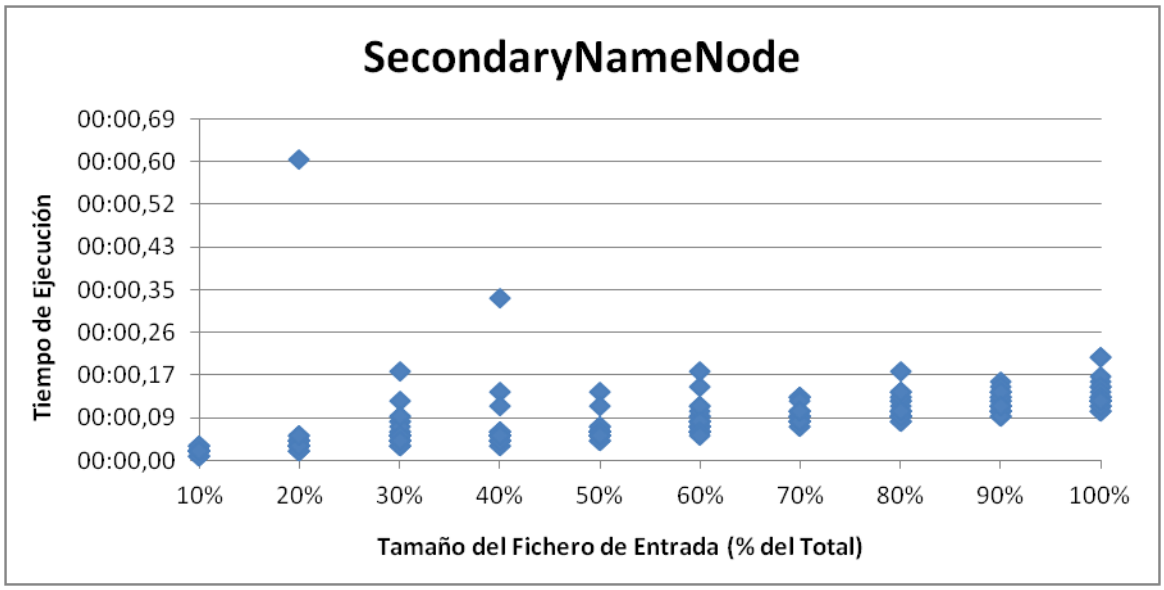


Figura 4.43: Total de los datos del proceso SecondaryNameNode para el Escenario 4

Espacio en disco

Tamaño	Media
10%	929159
20%	1825334
30%	2726734
40%	3632835
50%	4513097
60%	5708280
70%	6636555
80%	7533921
90%	8437635
100%	9341897

Tabla 4-24: Ocupación HDFS. Escenario 4

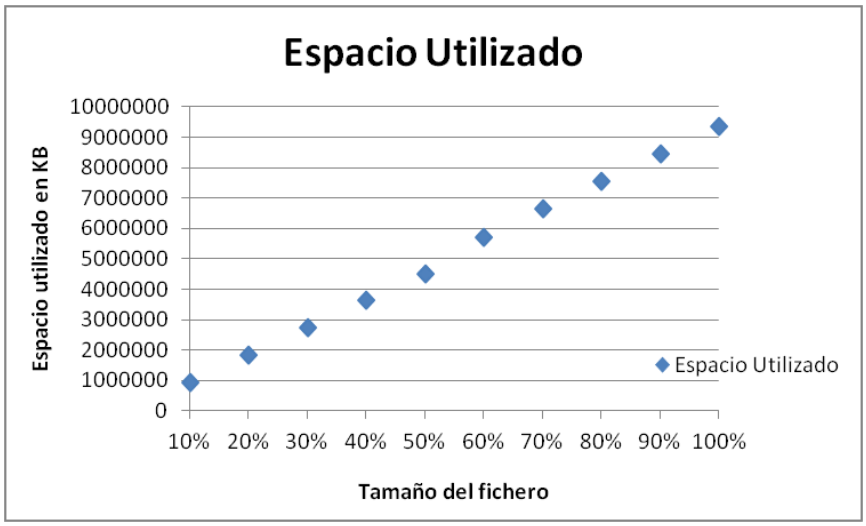


Figura 4.44: Evolución de la ocupación de espacio en disco. Escenario 4

4.3. Interpretación de los Resultados

A continuación se muestra una tabla resumen con los resultados medios obtenidos para las ejecuciones del 100% del fichero de entrada:

Escenario	Disco Duro (GB)	Tiempo de los Procesos (mm:ss,dd)				
		DataNode	TaskTracker	JobTracker	NameNode	SecondaryNameNode
1 Nodo	~ 8,9	00:25,88	00:12,06	00:01,03	00:00,68	00:00,20
2 Nodos	~ 8,9	00:44,27	00:06,62	00:00,87	00:00,62	00:00,11
3 Nodos	~ 8,9	00:58,37	00:05,90	00:01,23	00:00,50	00:00,14
4 Nodos	~ 8,9	00:55,06	00:04,21	00:00,92	00:00,58	00:00,13

Tabla 4-25: Resumen de los resultados para tamaño de fichero del 100%

Una cosa que llama la atención es que la ocupación del HDFS es más o menos la misma siempre, cercana a 8,9GB. Esto es así puesto que lo que realmente se mide en las ejecuciones es el tamaño que ocupa el directorio `/tmp/hadoop-mamenpala/dfs` justo después de cada ejecución. En este directorio se encuentran únicamente los datos del HDFS, en donde se alojan los ficheros de entrada y salida, y dos ficheros de logs que contienen información de la ejecución. Ya que el fichero de entrada total ocupa unos 5,55GB y su fichero resultante ocupa unos 3,27GB, la suma de estos dos ficheros es 8,82GB. La razón por la que cada escenario ocupa prácticamente el mismo valor es obvia. En todos los escenarios se ha realizado la misma prueba con el mismo fichero de entrada, obteniendo así el mismo fichero de salida. Las pequeñas variaciones en el tamaño del espacio en disco utilizado en cada escenario se deben a los dos ficheros de logs que contienen, ya que su información contiene datos distintos en cada ejecución, como son las fechas y horas en las que se realizan.

Respecto a la interpretación de los tiempos de los procesos podemos sacar las siguientes conclusiones:

- Que el tiempo de los procesos DataNode sea mucho mayor que el resto de los procesos es algo razonable, ya que este proceso es el que se

ocupa del acceso a disco, siendo esta la tarea que consume más tiempo que cualquiera de las otras.

- El segundo proceso que más tarda con respecto a los otros es el TaskTracker, ya que es el que se ocupa de realizar la tarea propiamente dicha.
- Sin embargo, este proceso TaskTracker tarda menos cuantos más nodos haya, ya que se reparten la tarea entre todos los nodos.
- El proceso DataNode tarda menos cuando hay un solo nodo, ya que no hay acceso a los datos en discos remotos. Esto se puede comprobar fácilmente observando las gráficas. En la gráfica que muestra los tiempos del proceso DataNode del escenario 2 se observan claramente las ejecuciones en las que intervenía el disco remoto. En los demás escenarios también se observa una clara diferencia entre las ejecuciones en las que intervenía el disco local del master y en las que intervenían los remotos de los esclavos.
- El tercer proceso que más tiempo consume es el JobTracker, que es el que se encarga de rastrear y organizar los trabajos entre todos los nodos.
- Teniendo en cuenta los resultados, en los tiempos de los demás procesos no se pueden sacar demasiadas conclusiones respecto de si se realizan con más o menos nodos, ya que estos tiempos son muy cortos.

4.4. Resultados

A título informativo, se muestran los datos más relevantes obtenidos para cada una de las pruebas.

4.4.1. Prueba 1

En esta prueba se han recogido los datos del fichero de salida con el número de enlaces que aparecen en una página. En la Figura 4.45 se puede observar que la página de Wikipedia que más enlaces tiene es la entrada de "Russell Berman" con 27129 enlaces hacia otras páginas. A continuación se muestra una gráfica con las páginas que más enlaces tienen:

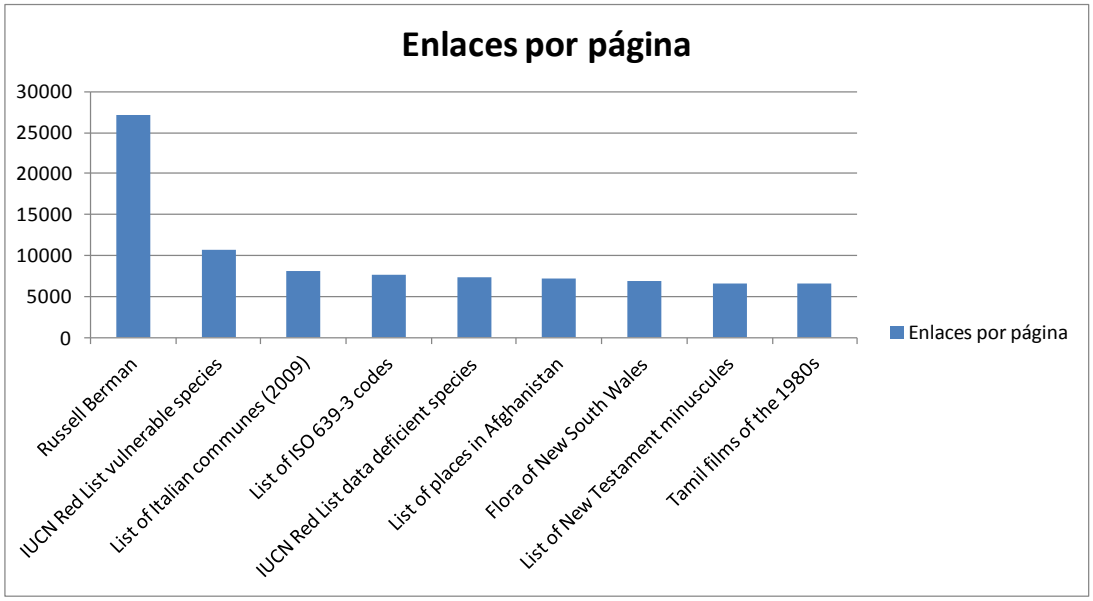


Figura 4.45: Prueba 1 - Ranking de Enlaces por Página

De forma general se muestra una gráfica con la totalidad de los datos en la que se puede observar una tendencia de descenso rápida en la cantidad de enlaces por página.

Figura 4.46: Prueba 1 - Total de Enlaces por Página

Para que se vean más claros estos resultados a continuación se representan estos mismos datos en escala logarítmica.

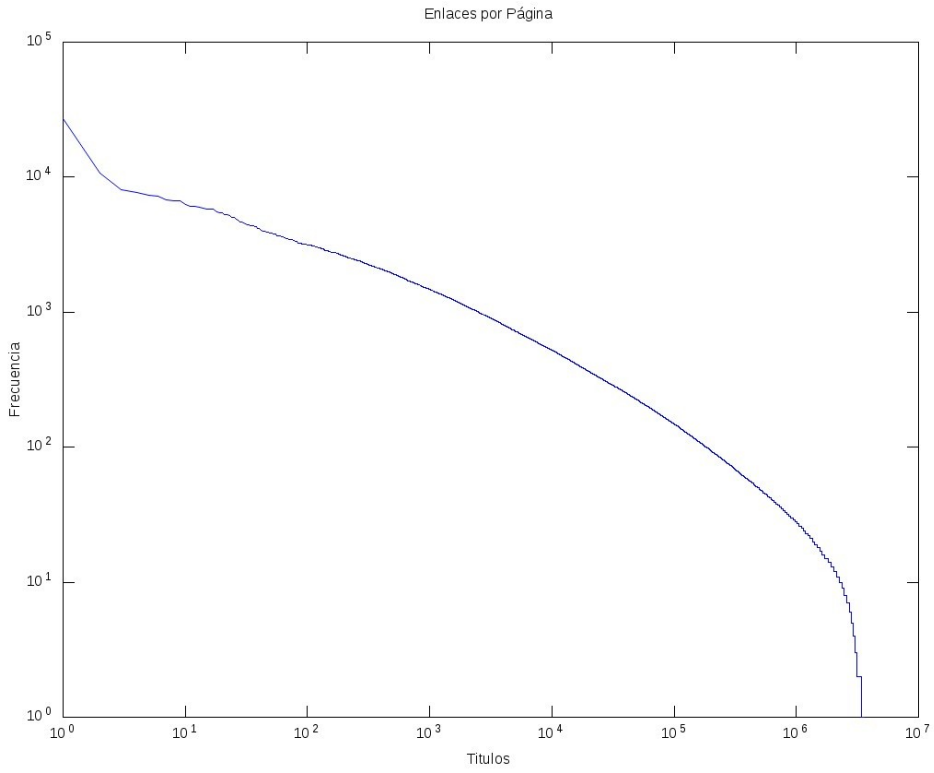


Figura 4.47: Prueba 1 - Total de Enlaces por Página (Escala logarítmica)

4.4.2. Prueba 2

En esta prueba se han recogido los datos del fichero de salida con el número de destinos que apuntan a una página. En la Figura 4.48 se puede observar que la página de Wikipedia a la que más enlaces apuntan es la entrada de "United States" con 492014 enlaces desde otras páginas. A modo anecdótico, la página de España en la Wikipedia ocupa el puesto 18 en este ranking de destinos desde otras páginas con 72722 enlaces. A continuación se muestra una gráfica con las páginas que más enlaces apuntan hacia ellas:

Figura 4.48: Prueba 2 - Ranking de destinos hacia una página

De forma general se muestra una gráfica con la totalidad de los datos en la que se puede observar de nuevo una tendencia de descenso rápida en la cantidad de enlaces que apuntan a una página, teniendo alguna de las páginas una gran cantidad de enlaces.

Figura 4.49: Prueba 2 - Total de destinos hacia una página

Para que se vean más claros estos resultados a continuación se representan estos mismos datos en escala logarítmica.

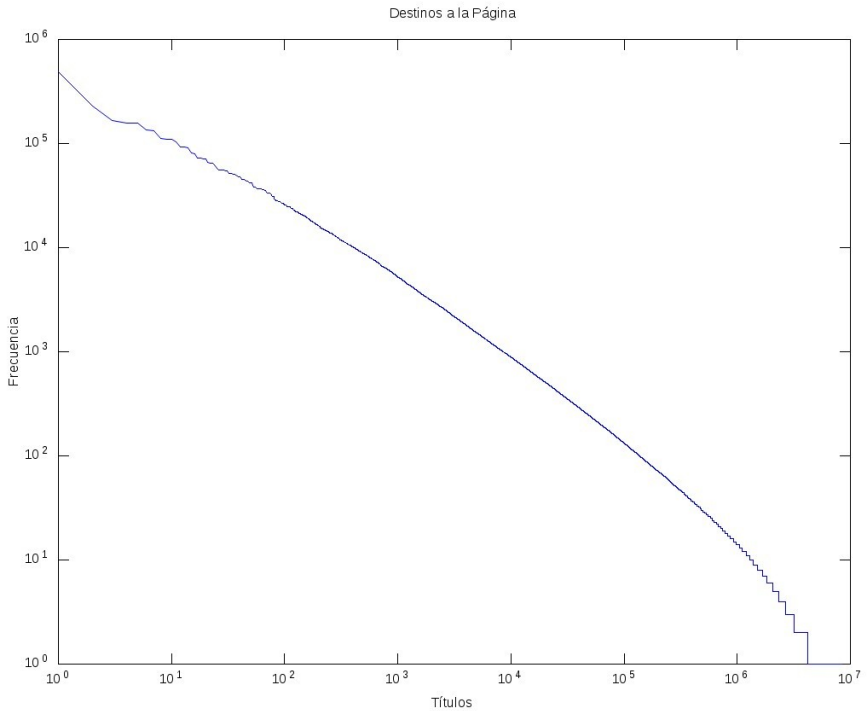


Figura 4.50: Prueba 2 - Total de destinos hacia una página (Escala logarítmica)

4.4.3. Prueba 3

En esta prueba se han recogido los datos del fichero de salida con el número de apariciones de un determinado anchor en una página. En la Figura 4.51 se puede observar que la página de Wikipedia en la que más veces aparece un enlace es "List of Tipula Species" con el enlace "Alexander". Aparece 1351 veces. A continuación se muestra una gráfica con el ranking de número de apariciones de un determinado anchor en una página:

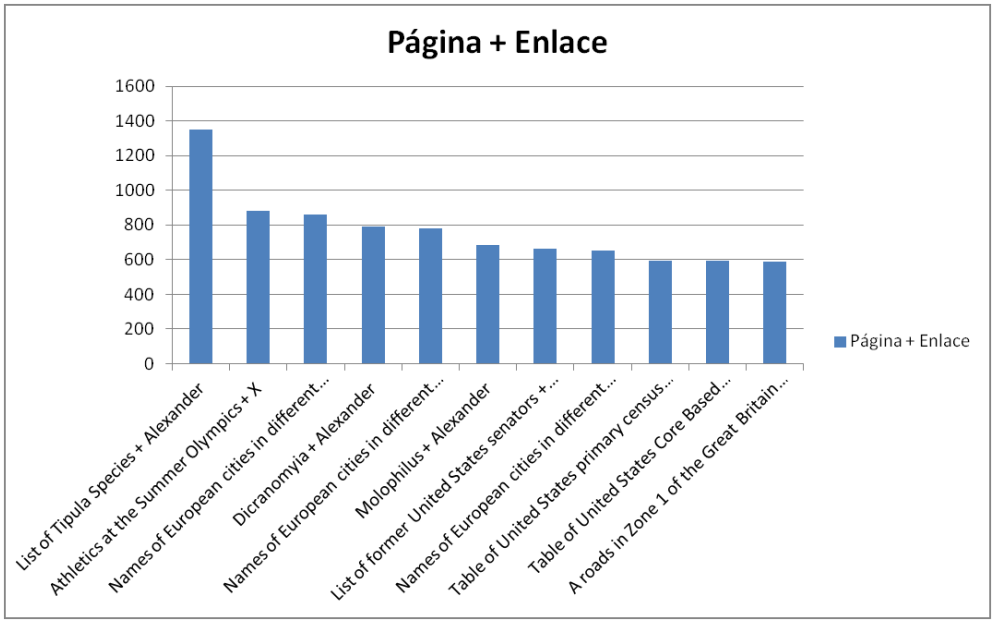


Figura 4.51: Prueba 3 - Ranking de un anchor en una página

De forma general se muestra una gráfica con la totalidad de los datos en la que se puede observar de nuevo una tendencia de descenso rápida en el número de apariciones de un determinado anchor en una página.

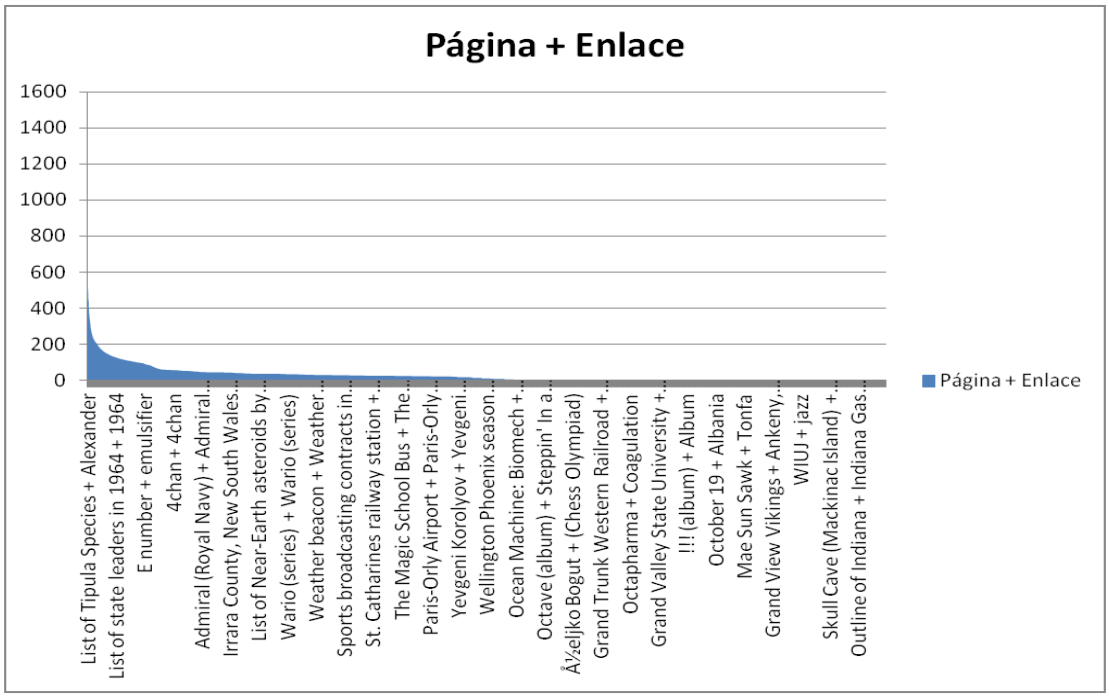


Figura 4.52: Prueba 3 - Total de un anchor en una página

Para que se vean más claros estos resultados a continuación se representan estos mismos datos en escala logarítmica.

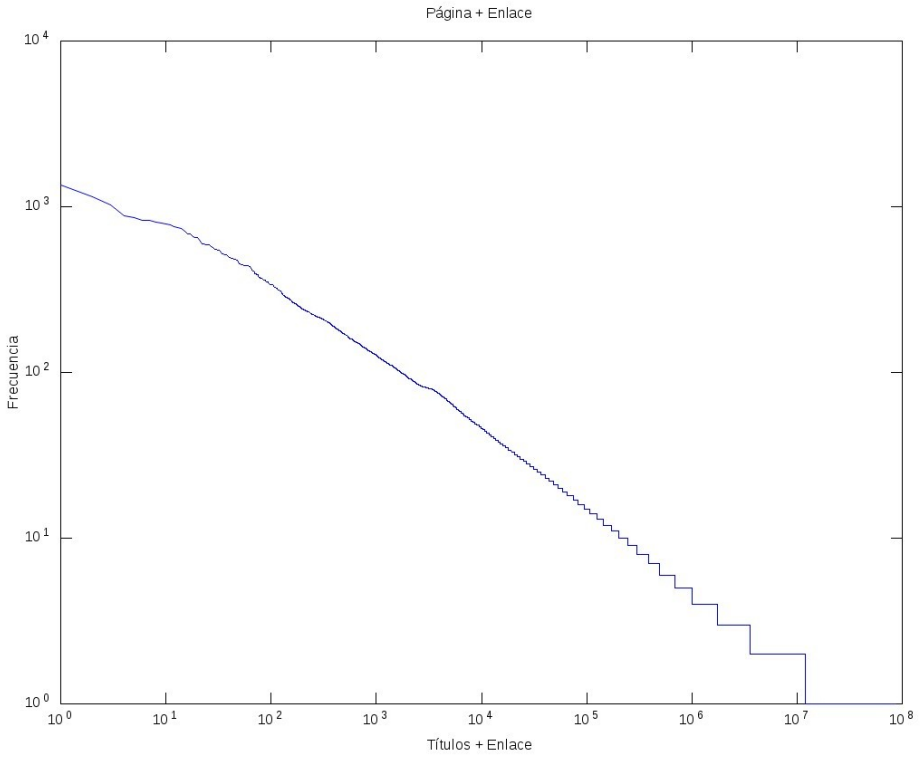


Figura 4.53: Prueba 3 - Total de un anchor en una página (Escala logarítmica)

4.4.4. Prueba 4

En esta prueba se han recogido los datos del fichero de salida con el número de apariciones de un determinado destino en una página. En la Figura 4.54 se puede observar que la página de Wikipedia en la que más veces aparece un destino es "List of Tipula Species" con el destino "Charles Paul Alexander". Aparece 1351 veces. Como se puede ver coincide con el valor de la prueba anterior. Esto es debido a que normalmente el anchor es el mismo para un determinado destino. A continuación se muestra una gráfica con el ranking de número de apariciones de un determinado destino en una página:

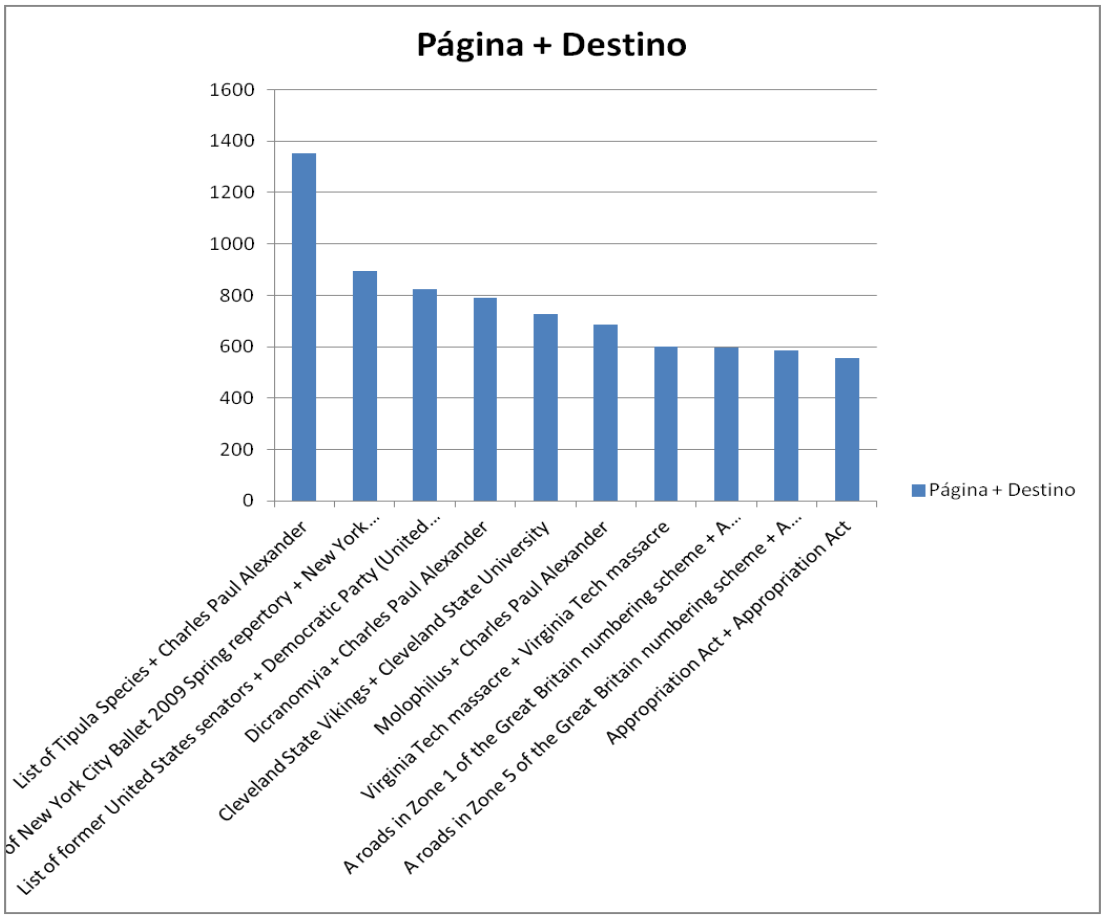


Figura 4.54: Prueba 4 - Ranking de un destino en una página

De forma general se muestra una gráfica con la totalidad de los datos en la que se puede observar de nuevo una tendencia de descenso rápida en el número de apariciones de un determinado destino en una página.

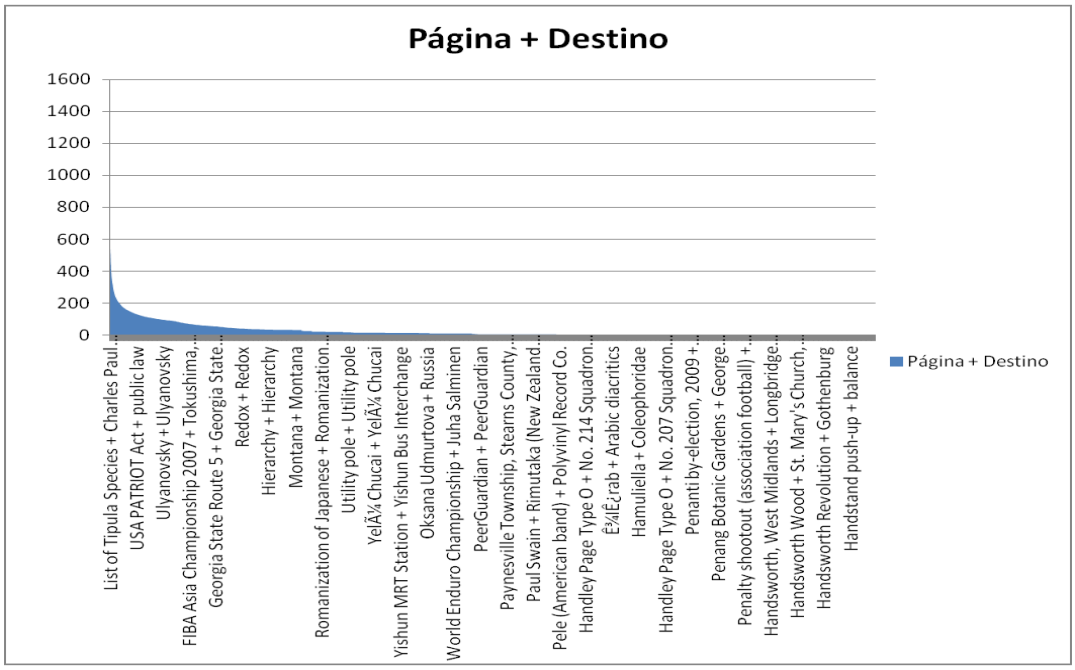


Figura 4.55: Prueba 4 - Total de un destino en una página

Para que se vean más claros estos resultados a continuación se representan estos mismos datos en escala logarítmica.

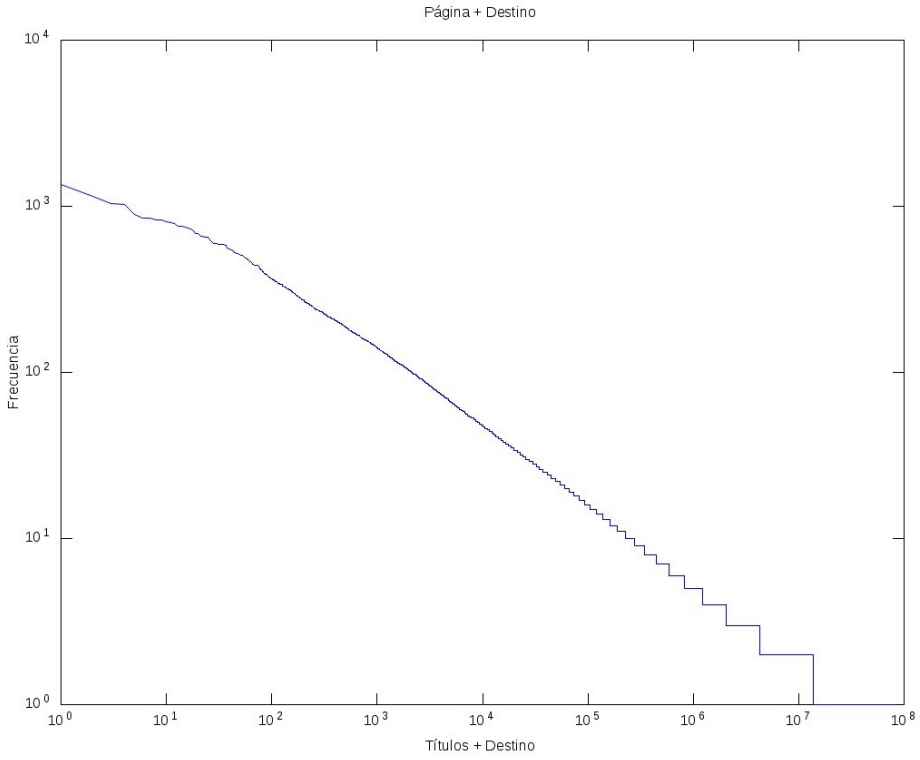


Figura 4.56: Prueba 4 - Total de un destino en una página (Escala logarítmica)

4.4.5. Prueba 5

En esta prueba se han recogido los datos del fichero de salida con el número de apariciones de un determinado anchor asociado a un destino. En la Figura 4.57 se puede observar que el anchor que más veces apunta a un destino es "United States" con el destino "United States". Aparece 279792 veces. A continuación se muestra una gráfica con el ranking de anchors que más veces apunta a un destino:

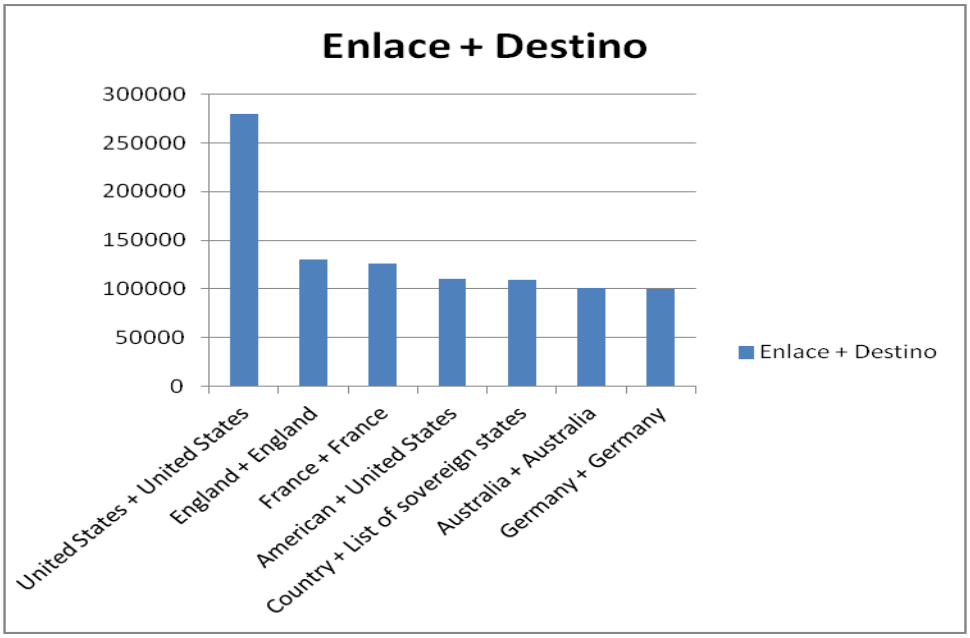


Figura 4.57: Prueba 5 - Ranking de un anchor en un destino

De forma general se muestra una gráfica con la totalidad de los datos en la que se puede observar de nuevo una tendencia de descenso rápida en el número de apariciones de un determinado anchor asociado a un destino.

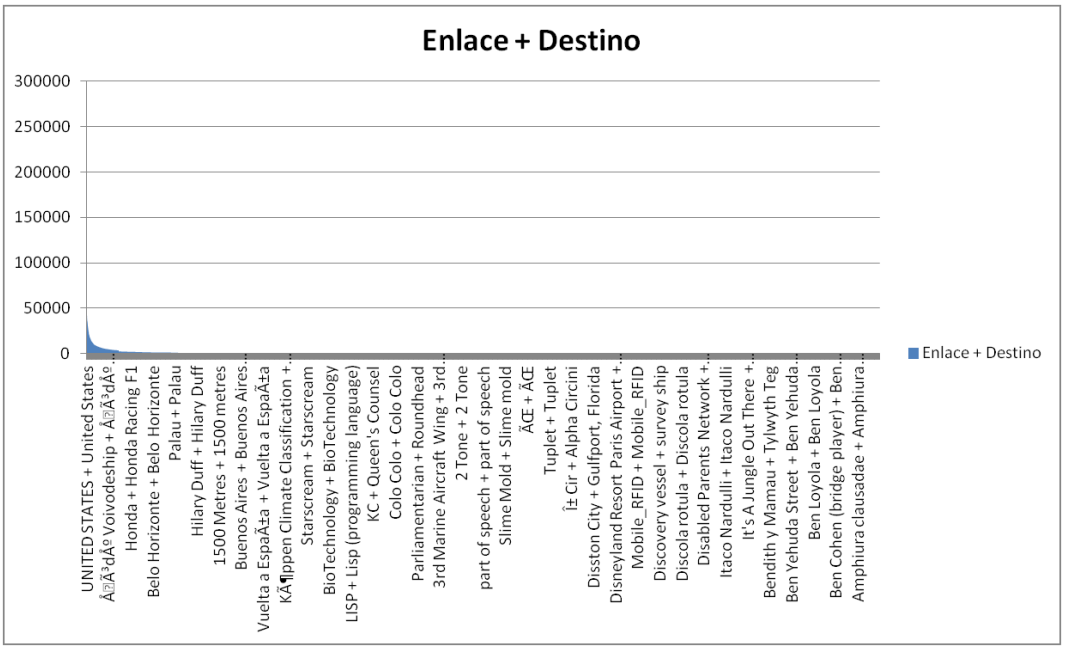


Figura 4.58: Prueba 5 - Total de un anchor en un destino

Para que se vean más claros estos resultados a continuación se representan estos mismos datos en escala logarítmica.

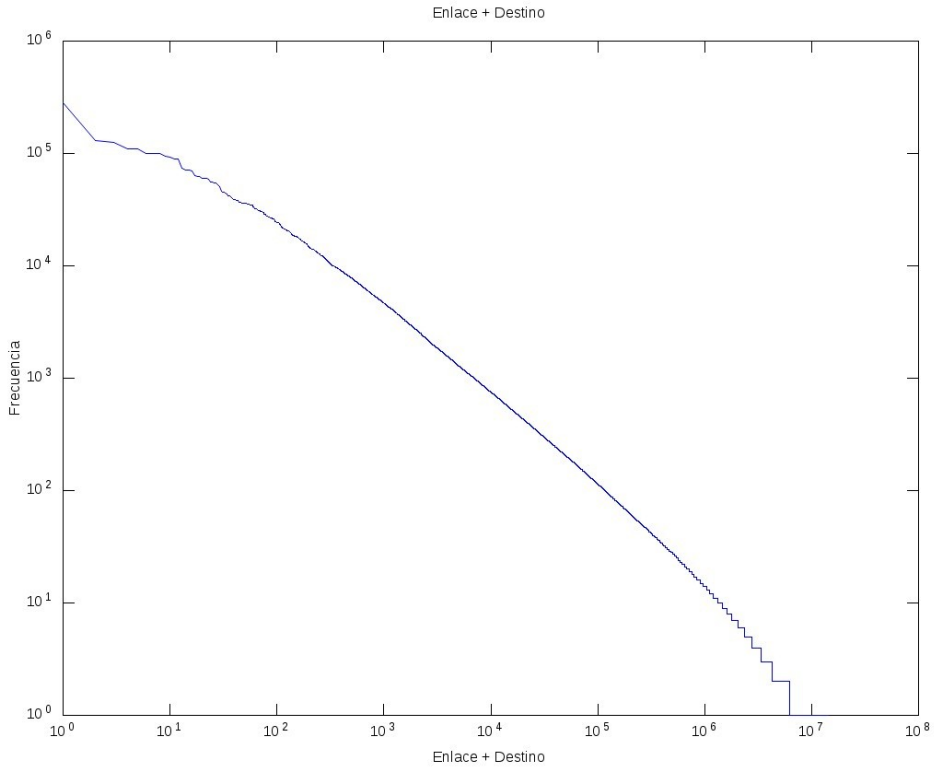


Figura 4.59: Prueba 5 - Total de un anchor en un destino (Escala logarítmica)

4.5. Conclusiones de los resultados

Al representar todos los datos anteriores en escala logarítmica el resultado se aproxima a una recta, indicando que los datos analizados siguen una distribución de tipo potencial, similar a una Zipf [19]. Esto es un resultado esperable, puesto que este tipo de distribuciones es común en el mundo web [20].

5. CONCLUSIONES Y LÍNEAS FUTURAS

5.1. Conclusiones

En este capítulo se muestran las conclusiones a las que se ha llegado tras la realización del presente proyecto. Durante el desarrollo del mismo, se han encontrado diversos problemas que se han tenido que resolver, investigando y profundizando sobre las distintas tecnologías que se han utilizado finalmente, obteniendo así una visión más clara sobre cada uno de los aspectos que han intervenido.

Observando los resultados obtenidos que se muestran en el apartado 4.2 del proyecto, podemos concluir que el procesamiento de archivos pequeños (en el orden de los MB) retarda la tarea de ejecución debido a que Hadoop está desarrollado para ser más eficiente manipulando archivos de gran tamaño (del orden de GB o TB). Estos tiempos podrían mejorarse aumentando el tamaño del *cluster*.

Y como última muestra de dificultad asociada al proyecto, cabe destacar el costoso trabajo de generar las gráficas a partir de los ficheros de salida obtenidos, ya que estos ficheros son de gran tamaño y en alguna de las pruebas llegaba a unos 86 millones de líneas. Con lo cual ningún programa de

generación de gráficas al uso, ni la memoria de los ordenadores soportaba el tamaño de dichos ficheros. Finalmente se consiguieron realizar dichas gráficas con el programa de Microsoft PowerPivot para Excel 2010 [21], que es una aplicación que puede trabajar con grandes cantidades de datos sin necesidad de grandes requerimientos hardware. También se utilizó la aplicación Octave para generar las gráficas ajustadas por Zipf.

A continuación se muestra una interpretación de los resultados obtenidos.

5.2. Líneas futuras

Como objetivos a largo plazo se podría crear una aplicación con una interfaz gráfica que permita realizar la instalación del *cluster* más automáticamente. Es decir, una aplicación que permita configurar el número de máquinas en el *cluster*, una clase *map*, una clase *reduce* y especificar los parámetros generales para poder lanzar un trabajo sobre un entorno distribuido de datos, aprovechando la capacidad de procesamiento que ofrezca éste.

Un trabajo muy interesante podría estar orientado a hacer las pruebas de Hadoop aumentando el tamaño de los datos, o aumentando el número de ficheros de entrada, ya que en este caso se han hecho las pruebas con un único fichero de entrada. De este modo se podrá comprobar si con un número muy elevado de ficheros o con ficheros mucho más grandes la herramienta sigue aumentado su eficacia a la hora de realizar ejecuciones más rápidamente.

Otros trabajos futuros podrían también estar orientados a hacer las pruebas de la herramienta Hadoop con mayor número de nodos en el *cluster*. Así, se podrá comprobar si con un número muy elevado de nodos la herramienta sigue aumentado su eficacia a la hora de realizar ejecuciones más rápidamente.

También se podría realizar un trabajo en el cual se realicen operaciones en el código Java que requieran un consumo de recursos mayor y así comparar con el presente proyecto y determinar si la complejidad del código importa para que el tiempo y los recursos del sistema aumenten.

ANEXO A: PRESUPUESTO

En este anexo se detallará el presupuesto necesario para poder realizar este proyecto. Para calcular los costes del proyecto se tendrá en cuenta tanto el coste material como el trabajo del personal que ha participado en su desarrollo.

A.1. Costes de Personal

A continuación se expondrá el coste generado por el personal encargado del desarrollo del proyecto, que en este caso se trata de dos personas (alumna y tutor). Los costes de personal incluyen los honorarios de un Ingeniero Técnico de Telecomunicación en Telemática encargado del desarrollo del proyecto y un Ingeniero de Telecomunicación encargado de la supervisión del proyecto. Se considerará un coste estimado de 65 €/hora como salario de un Ingeniero Técnico de Telecomunicación, un coste de 95 €/hora como salario de un Ingeniero de Telecomunicación.

La duración de este proyecto ha sido de aproximadamente 4 meses. Suponiendo 20 días laborables al mes, se obtiene un total de 80 días de realización del proyecto. Con una jornada laboral de 8 horas diarias, la realización del proyecto ha requerido de 640 horas, de las cuales 580 horas corresponden al desarrollo del proyecto y 60 horas corresponden a la supervisión. En la Tabla A-0-1 se puede ver el resumen de costes directos de personal.

Apellidos y nombre	Categoría	Horas	Coste/Hora (€)	Coste Total (€)
Fernández García, Norberto	Ingeniero de Telecomunicación	60	95,00	5.700,00
Palacios Díaz-Zorita, MªCarmen	Ingeniero Técnico de Telecomunicación	580	65,00	37.700,00
			Total	43.400,00

Tabla A-0-1: Presupuesto - Costes Directos de Personal

A.2. Costes de Materiales

Los materiales empleados durante la realización del proyecto han sido los siguientes:

- 4 ordenadores de sobremesa, con sistema operativo Linux, valorados en 800€ cada uno. Total 3200€.
- 1 router para interconectar los ordenadores valorado en 60€.

El coste detallado de estos materiales puede verse en la Tabla A-0-2:

Descripción	Coste (€)	% Uso dedicado proyecto	Dedicación (meses)	Periodo de depreciación	Coste imputable [d]
Ordenador1	800	100	5	60	66,67
Ordenador2	800	100	5	60	66,67
Ordenador3	800	100	5	60	66,67
Ordenador4	800	100	5	60	66,67
Router	60	100	5	60	5,00
				Total	271,67

Tabla A-0-2: Presupuesto - Costes Directos de los Materiales

[d] Fórmula de cálculo de la Amortización: $\frac{A}{B} x C x D$

A = nº de meses desde la fecha de facturación en que el equipo es utilizado

B = periodo de depreciación (60 meses)

C = coste del equipo (sin IVA)

D = % del uso que se dedica al proyecto (habitualmente 100%)

A.3. Presupuesto Total

El presupuesto final para la realización de este proyecto está formado por los costes de material y de personal presentados anteriormente. Además se añade un 20% de costes indirectos al total del prepuesto. En estos costes indirectos se incluye la conexión a internet, costes de luz, etc. Como se puede ver en la Tabla A-0-3, el presupuesto total asciende a **52.406,00 €.**

Tipo de Coste	Coste
Costes de Personal	43.400,00 €
Costes de Materiales	271,67 €
Costes Indirectos (20%)	8.734,33 €
Coste Total	**52.406,00 €**

Tabla A-0-3: Presupuesto Total

ANEXO B: INSTALACIÓN Y CONFIGURACIÓN DE HADOOP EN UN SOLO NODO

B.1. Instalación

La configuración descrita aquí es una instancia HDFS con un *NameNode* y un *DataNode* individuales y un *cluster* MapReduce, con un *JobTracker* y un *TaskTracker* único.

Requisitos necesarios para la Instalación de Hadoop en Linux:

- Java SE 1.6 de Sun preferiblemente.
- ssh deben de estar instalado y su demonio sshd debe estar ejecutándose para utilizar los scripts Hadoop que manejan a distancia demonios Hadoop.

Descargar la última versión disponible de Hadoop del sitio de Apache http://hadoop.apache.org/common/releases.html.

Una vez descargado Hadoop es necesario descomprimirlo y colocarlo en alguna ubicación dentro de nuestro sistema de archivos al que tengamos acceso.

B.3. Configuración

B.3.1. Hadoop

Una vez copiados los archivos de Hadoop en nuestra ubicación destino, es necesario indicarle la ruta donde está instalado Java en las variables de entorno. Esto se hace en el archivo `conf/hadoop-env.sh` quedando como:

```
export JAVA_HOME=/usr/lib/jvm/java-6-sun
export PATH=/usr/lib/jvm/java-6-sun/bin:${PATH}
export HADOOP_HOME=/home/apt/mamenpala/Desktop/hadoop-0.20.1
export PATH=$PATH:$HADOOP_HOME/bin
```

Ahora hay que configurar el directorio donde Hadoop almacena sus archivos de datos, los puertos que escucha, etc. El programa de instalación utilizará el sistema de archivos distribuidos Hadoop, HDFS, a pesar de que el "*cluster*" sólo contiene un equipo local. Para hacer esto hay que modificar los tres ficheros de configuración:

En `conf/core-site.xml`:

```
<?xml version="1.0"?>
<?xml-stylesheet type="text/xsl" href="configuration.xsl"?>

<!-- Put site-specific property overrides in this file. -->

<configuration>
	<property>
		<name>fs.default.name</name>
		<value>hdfs://localhost/</value>
	</property>
</configuration>
```

En `conf/mapred-site.xml`:

```
<?xml version="1.0"?>
<?xml-stylesheet type="text/xsl" href="configuration.xsl"?>

<!-- Put site-specific property overrides in this file. -->
<configuration>
	<property>
		<name>mapred.job.tracker</name>
		<value>localhost:8021</value>
	</property>
</configuration>
```

En `conf/hdfs-site.xml`:

```
<?xml version="1.0"?>
<?xml-stylesheet type="text/xsl" href="configuration.xsl"?>

<!-- Put site-specific property overrides in this file. -->

<configuration>
       <property>
              <name>dfs.replication</name>
              <value>1</value>
       </property>
</configuration>
```

Por defecto, Hadoop viene configurado para ejecutarse en modo no distribuido, es decir, como un proceso simple de Java lo cual es muy útil para la depuración de programas. Ahora la instalación de Hadoop está lista para ejecutarlo en modo local.

B.3.2. Configuración de SSH

Hadoop requiere el acceso SSH para gestionar sus nodos, es decir, las máquinas remotas, más el equipo local si desea utilizar Hadoop en ella.

En primer lugar, hay que generar una clave SSH:

```
ssh-keygen -t rsa -P '' -f ~/.ssh/id_rsa
```

En segundo lugar, hay que permitir el acceso SSH a la máquina local con esta nueva clave creada:

```
cat ~/.ssh/id_rsa.pub >> ~/.ssh/authorized_keys
```

El último paso es probar la configuración de SSH conectándose a la máquina local. Este paso también es necesario para guardar la huella digital de la máquina local de acogida en el fichero `known_hosts`.

```
ssh localhost
```

B.4. Puesta en marcha

El primer paso para la puesta en marcha de la instalación de Hadoop es formatear el sistema de archivos que está implementado en la parte superior del sistema de archivos local del *cluster* (que incluye sólo el equipo local). Se necesita hacer esto la primera vez que se configura un *cluster* de Hadoop. **No se debe formatear un sistema de archivos ejecutando Hadoop, esto hará que todos los datos sean borrados.**

Para formatear el sistema de archivos (que simplemente inicializa el directorio especificado por la variable `dfs.name.dir)`, hay que ejecutar el comando:

```
hadoop namenode –format
```

El siguiente paso el lanzar el *cluster* en el único nodo:

```
<HADOOP_HOME>/bin/start-dfs.sh
<HADOOP_HOME>/bin/start-mapred.sh
```

Ahora podemos acceder a la interfaz web del *JobTracker* y del *NameNode* en el puerto *50030* y *50070* respectivamente.

B.5. Ejecución de un trabajo MapReduce

Para simular el funcionamiento de Hadoop usaremos un ejemplo con un programa de Java llamado WordCount. WordCount es una sencilla aplicación que cuenta el número de ocurrencias de cada palabra en un fichero de entrada. Las entradas y las salidas son los archivos de texto. Cada línea del fichero de salida contiene una palabra y el recuento de la frecuencia con que se produjo.

El código de la aplicación es el siguiente [22]:

```
01  package org.myorg;
02
03  import java.io.IOException;
04  import java.util.*;
05
06  import org.apache.hadoop.fs.Path;
07  import org.apache.hadoop.conf.*;
08  import org.apache.hadoop.io.*;
09  import org.apache.hadoop.mapreduce.*;
10  import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
11  import org.apache.hadoop.mapreduce.lib.input.TextInputFormat;
12  import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
13  import org.apache.hadoop.mapreduce.lib.output.TextOutputFormat;
14
15  public class WordCount {
16
17   public static class Map extends Mapper<LongWritable, Text, Text,
    IntWritable> {
18      private final static IntWritable one = new IntWritable(1);
19      private Text word = new Text();
20
21      public void map(LongWritable key, Text value, Context context)
    throws IOException, InterruptedException {
22          String line = value.toString();
23          StringTokenizer tokenizer = new StringTokenizer(line);
24          while (tokenizer.hasMoreTokens()) {
25              word.set(tokenizer.nextToken());
26              context.write(word, one);
27          }
28      }
29   }
30
31   public static class Reduce extends Reducer<Text, IntWritable, Text,
    IntWritable> {
32
33      public void reduce(Text key, Iterable<IntWritable> values, Context
    context)
34        throws IOException, InterruptedException {
35          int sum = 0;
36          for (IntWritable val : values) {
37              sum += val.get();
38          }
39          context.write(key, new IntWritable(sum));
40      }
41   }
42
43   public static void main(String[] args) throws Exception {
44      Configuration conf = new Configuration();
45
46          Job job = new Job(conf, "wordcount");
47
48      job.setOutputKeyClass(Text.class);
49      job.setOutputValueClass(IntWritable.class);
50
51      job.setMapperClass(Map.class);
52      job.setReducerClass(Reduce.class);
53
54      job.setInputFormatClass(TextInputFormat.class);
55      job.setOutputFormatClass(TextOutputFormat.class);
56
57      FileInputFormat.addInputPath(job, new Path(args[0]));
58      FileOutputFormat.setOutputPath(job, new Path(args[1]));
59
60      job.waitForCompletion(true);
61   }}
```

B.6. Uso

Suponiendo HADOOP_HOME es la raíz de la instalación y HADOOP_VERSION es la versión instalada Hadoop, hay que compilar WordCount.java y crear un .jar. Para ello:

```
mkdir wordcount_classes
cd Desktop/org.myorg
javac -cp ${HADOOP_HOME}/hadoop-0.20.1-core.jar -d wordcount_classes WordCount.java
jar -cvf /home/apt/mamenpala/Desktop/org.myorg/wordcount.jar -C wordcount_classes/ .
```

Suponiendo que:

`/user/mamenpala/wordcount/input` → directorio de entrada en HDFS

`/user/mamenpala/wordcount/output` → directorio de salida en HDFS

En este ejemplo he utilizado dos ficheros de entrada:

```
/user/mamenpala/wordcount/input/file01
/user/mamenpala/wordcount/input/file02
```

Con el siguiente texto:

```
En file01: Hello World Bye World
En file02: Hello Hadoop Goodbye Hadoop
```

Una vez tenemos los ficheros creados, los copiamos al sistema de archivos de Hadoop:

```
cd ${HADOOP_HOME}
bin/hadoop dfs -ls /tmp/hadoop-mamenpala
bin/hadoop dfs -mkdir -p wordcount/input
bin/hadoop dfs -ls /user
bin/hadoop dfs -touchz /user/mamenpala/wordcount/input/file01
bin/hadoop dfs -touchz /user/mamenpala/wordcount/input/file02
cd bin
./hadoop dfs -rm  wordcount/input/file0*
./hadoop dfs -copyFromLocal
/home/apt/mamenpala/Desktop/wordcount/input/file01 wordcount/input
./hadoop dfs -copyFromLocal
/home/apt/mamenpala/Desktop/wordcount/input/file02 wordcount/input
./hadoop dfs -copyFromLocal
/home/apt/mamenpala/Desktop/org.myorg/wordcount.jar .
./hadoop dfs -copyFromLocal /home/apt/mamenpala/Desktop/org.myorg/ .
```

Ejecutamos la aplicación:

```
./hadoop jar /home/apt/mamenpala/Desktop/org.myorg/wordcount.jar
org.myorg.WordCount /user/mamenpala/wordcount/input
/user/mamenpala/wordcount/output
```

Por pantalla debería aparecer:

```
11/10/29 20:05:14 WARN mapred.JobClient: Use GenericOptionsParser for parsing
the arguments. Applications should implement Tool for the same.
11/10/29 20:05:14 INFO input.FileInputFormat: Total input paths to process : 1
11/10/29 20:05:14 INFO mapred.JobClient: Running job: job_201110291953_0001
11/10/29 20:05:15 INFO mapred.JobClient:  map 0% reduce 0%
11/10/29 20:05:28 INFO mapred.JobClient:  map 20% reduce 0%
11/10/29 20:05:33 INFO mapred.JobClient:  map 40% reduce 0%
11/10/29 20:05:34 INFO mapred.JobClient:  map 60% reduce 0%
11/10/29 20:05:36 INFO mapred.JobClient:  map 70% reduce 0%
11/10/29 20:05:37 INFO mapred.JobClient:  map 90% reduce 6%
11/10/29 20:05:42 INFO mapred.JobClient:  map 100% reduce 6%
11/10/29 20:05:46 INFO mapred.JobClient:  map 100% reduce 26%
11/10/29 20:05:49 INFO mapred.JobClient:  map 100% reduce 66%
11/10/29 20:05:52 INFO mapred.JobClient:  map 100% reduce 70%
11/10/29 20:05:55 INFO mapred.JobClient:  map 100% reduce 73%
11/10/29 20:05:58 INFO mapred.JobClient:  map 100% reduce 76%
11/10/29 20:06:01 INFO mapred.JobClient:  map 100% reduce 80%
11/10/29 20:06:04 INFO mapred.JobClient:  map 100% reduce 83%
11/10/29 20:06:07 INFO mapred.JobClient:  map 100% reduce 87%
11/10/29 20:06:10 INFO mapred.JobClient:  map 100% reduce 90%
11/10/29 20:06:13 INFO mapred.JobClient:  map 100% reduce 94%
11/10/29 20:06:19 INFO mapred.JobClient:  map 100% reduce 100%
11/10/29 20:06:21 INFO mapred.JobClient: Job complete: job_201110291953_0001
11/10/29 20:06:21 INFO mapred.JobClient: Counters: 17
11/10/29 20:06:21 INFO mapred.JobClient:   Job Counters
11/10/29 20:06:21 INFO mapred.JobClient:     Launched reduce tasks=1
11/10/29 20:06:21 INFO mapred.JobClient:     Launched map tasks=10
11/10/29 20:06:21 INFO mapred.JobClient:     Data-local map tasks=10
11/10/29 20:06:21 INFO mapred.JobClient:   FileSystemCounters
11/10/29 20:06:21 INFO mapred.JobClient:     FILE_BYTES_READ=705847904
11/10/29 20:06:21 INFO mapred.JobClient:     HDFS_BYTES_READ=616677623
11/10/29 20:06:21 INFO mapred.JobClient:     FILE_BYTES_WRITTEN=1062929530
11/10/29 20:06:21 INFO mapred.JobClient:     HDFS_BYTES_WRITTEN=326199370
11/10/29 20:06:21 INFO mapred.JobClient:   Map-Reduce Framework
11/10/29 20:06:21 INFO mapred.JobClient:     Reduce input groups=7729009
11/10/29 20:06:21 INFO mapred.JobClient:     Combine output records=15254874
11/10/29 20:06:21 INFO mapred.JobClient:     Map input records=10893877
11/10/29 20:06:21 INFO mapred.JobClient:     Reduce shuffle bytes=320265048
11/10/29 20:06:21 INFO mapred.JobClient:     Reduce output records=7729009
11/10/29 20:06:21 INFO mapred.JobClient:     Spilled Records=22983886
11/10/29 20:06:21 INFO mapred.JobClient:     Map output bytes=490785157
11/10/29 20:06:21 INFO mapred.JobClient:     Combine input records=18419739
11/10/29 20:06:21 INFO mapred.JobClient:     Map output records=10893877
11/10/29 20:06:21 INFO mapred.JobClient:     Reduce input records=7729012
```

Para ver el fichero de salida obtenido ejecutamos:

```
./hadoop dfs -cat /user/mamenpala/wordcount/output/part-r-00000
```

Se copia el fichero obtenido al sistema de ficheros local para que sea más cómodo de ver:

```
./hadoop dfs -copyFromLocal wordcount/output/part-r-00000
/home/apt/mamenpala/Desktop/wordcount/output
```

Los resultados deberían ser:

```
Bye 1
Goodbye 1
Hadoop 2
Hello 2
World 2
```

Finalmente, para detener todos los demonios ejecutándose en nuestra máquina ejecutamos el comando:

```
<HADOOP_HOME>/bin/stop-dfs.sh
<HADOOP_HOME>/bin/stop-mapred.sh
```

ANEXO C: DESARROLLO CON ECLIPSE

El desarrollo del código Java fue escrito con la ayuda del compilador para Java Eclipse, el cual puede ser descargado desde su sitio oficial.

Una vez descargado el paquete es necesario descomprimirlo y ejecutarlo, ya que este no necesita ningún tipo de instalación [23].

C.1. Plugin de Hadoop para ECLIPSE

La instalación del plugin de Hadoop e integración con el entorno Eclipse es muy sencilla:

- Una vez descargado el plugin de Hadoop, se debe ir a la carpeta de instalación de Eclipse y copiar el plugin dentro del directorio plugins dentro de la misma.
- Iniciar Eclipse e ir al menú Windows -> Show View -> Map Reduce Tools y seleccione Map Reduce Servers y aceptar la selección.

El plugin de Hadoop mostrara una interfaz en la cual se puede configurar los datos de conexión hacia un servidor Hadoop habilitado.

C.2. Configuración del entorno de desarrollo

Después de descargar e instalar eclipse, procedemos con la instalación del plugin de Hadoop:

1. Establecer las variables de entorno JAVA_HOME y HADOOP_HOME, con la ruta a la carpeta raíz de Java y Hadoop respectivamente.

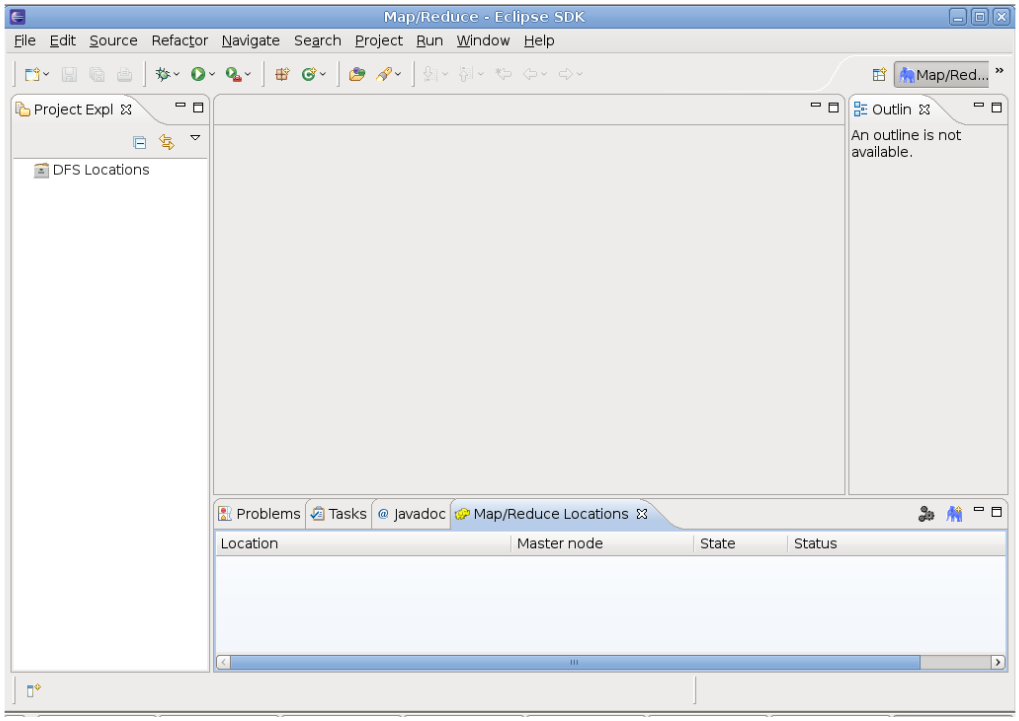
2. Copiar el plugin de Hadoop a la carpeta de plugins de eclipse.

```
cp $HADOOP_HOME/contrib/eclipse-plugin/hadoop-0.20.2-eclipse-
plugin.jar PATH_TO/eclipse/plugins/
```

3. Iniciamos Eclipse.

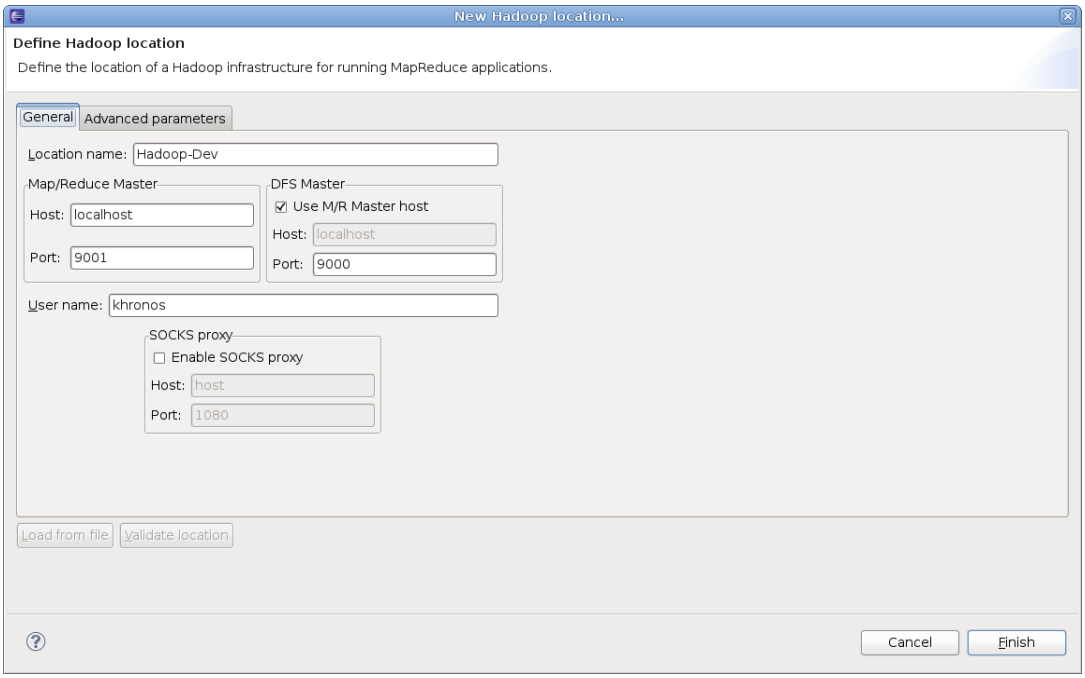
4. Abrimos la perspectiva Map/Reduce. *Window->Open Perspective->Others->Map/Reduce*

5. Entramos en la vista MapReduce Locations.



6. Clic derecho sobre la lista MapReduce Locations, seleccionar **New Hadoop location**.

7. Le damos un nombre a la nueva ubicación, y configuramos el host y puerto tanto del MapReduce maestro (*JobTracker*) cómo del dfs.master (*NameNode*).

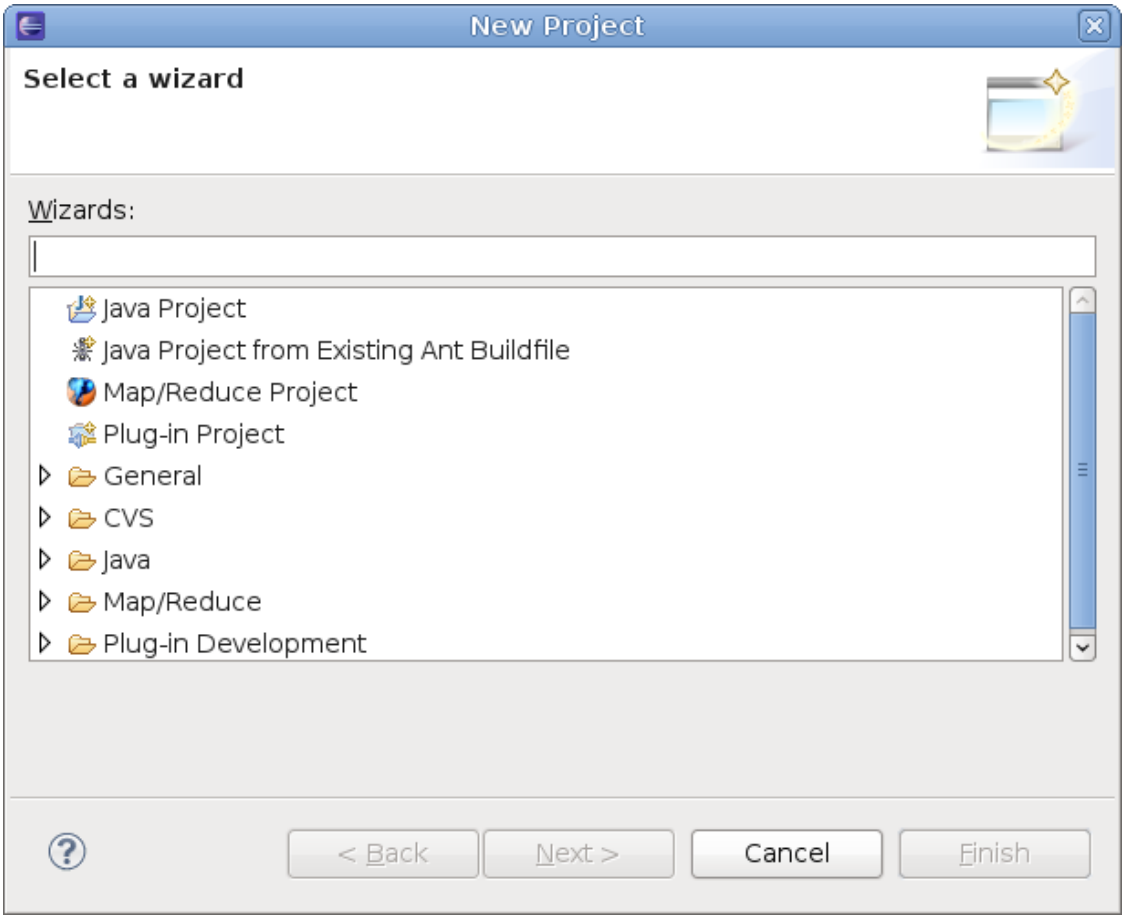


8. Ahora en el explorador de proyectos podremos ver los archivos de nuestro sistema de archivos distribuido en DFS Locations.

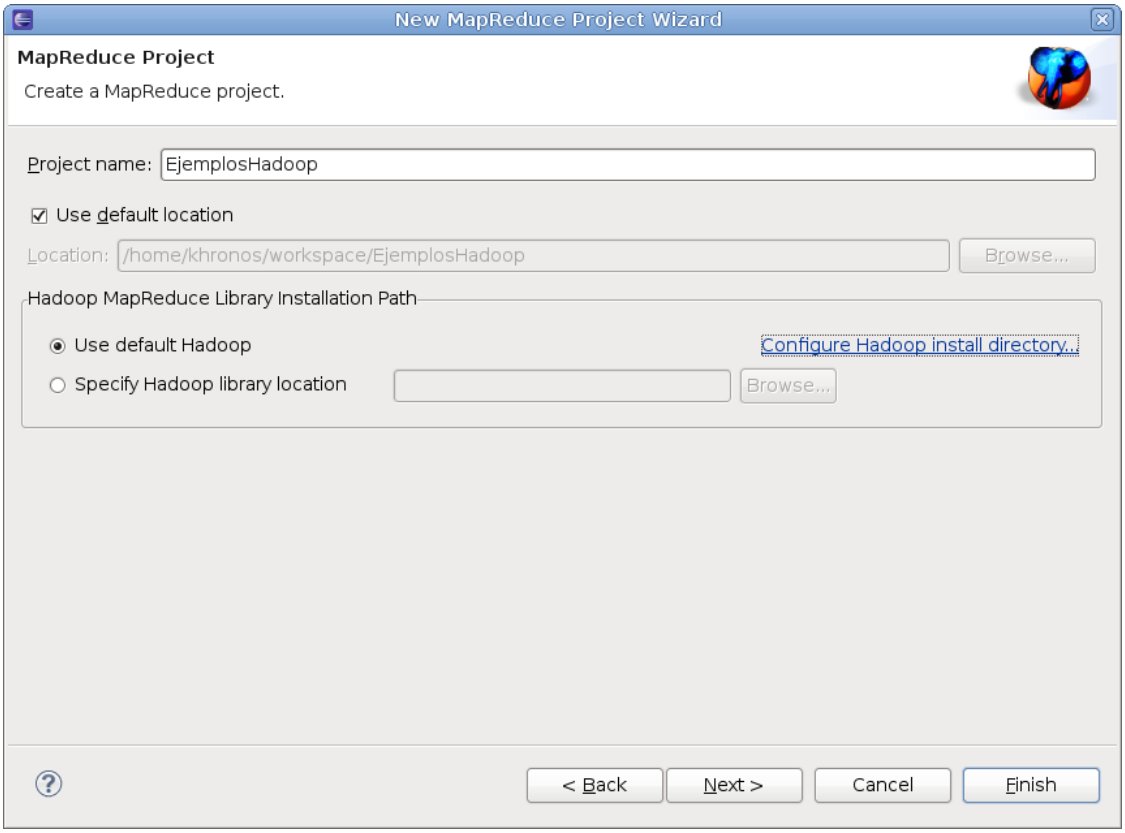
Ahora tenemos listo nuestro ambiente de desarrollo, a continuación crearemos una aplicación MapReduce.

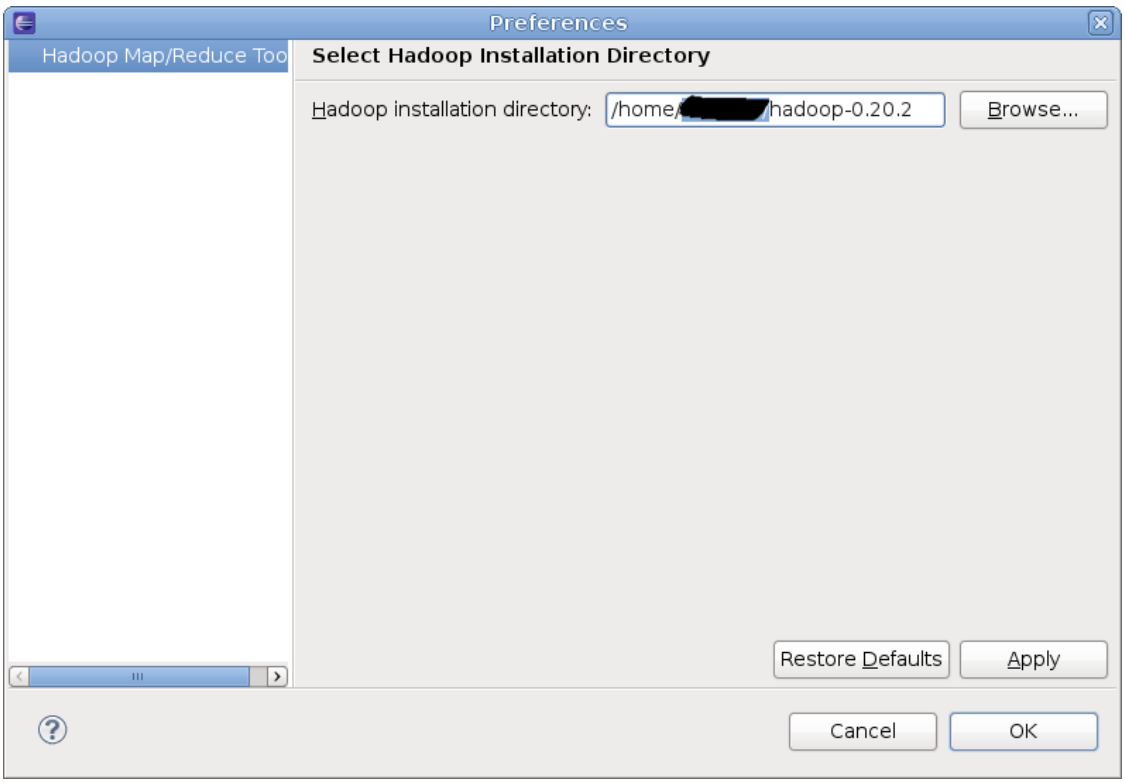
C.3. Crear una nueva aplicación

El plugin de Hadoop nos crea un nuevo tipo de proyecto en eclipse y tres tipos de archivo Mapper, Reducer y MapReduce Driver. Para crear nuestra nueva aplicación crearemos un proyecto MapReduce:



En la siguiente ventana seleccionamos el nombre del proyecto y configuramos el directorio de la instalación de Hadoop a utilizar.


New MapReduce Project Wizard
MapReduce Project
Create a MapReduce project.
Project name: EjemplosHadoop
Use default location
Location: /home/khronos/workspace/EjemplosHadoop
Browse...
Hadoop MapReduce Library Installation Path
Use default Hadoop
Configure Hadoop install directory...
Specify Hadoop library location
Browse...
< Back
Next >
Cancel
Finish


Preferences
Hadoop Map/Reduce Too
Select Hadoop Installation Directory
Hadoop installation directory: /home/[redacted]hadoop-0.20.2
Browse...
Restore Defaults
Apply
Cancel
OK

Continuamos con el asistente hasta finalizar.

Ahora crearemos la primera aplicación, utilizando las facilidades que nos da el plugin:

1. Creamos un nuevo Mapper: file->new->Mapper

```
01  package ejemplos.wordcount;
02
03  import java.io.IOException;
04  import java.util.*;
05  import org.apache.hadoop.io.*;
06  import org.apache.hadoop.mapred.MapReduceBase;
07  import org.apache.hadoop.mapred.Mapper;
08  import org.apache.hadoop.mapred.OutputCollector;
09  import org.apache.hadoop.mapred.Reporter;
10
11  public class WordCountMapper extends MapReduceBase implements Mapper
    <LongWritable, Text, Text, IntWritable>  {
      private final static IntWritable one = new IntWritable(1);
12    private Text word = new Text();
      public void map(LongWritable key, Text value, OutputCollector<Text,
13  IntWritable> output, Reporter reporter) throws IOException {
14      String line = value.toString();
        StringTokenizer tokenizer = new StringTokenizer(line);
        while (tokenizer.hasMoreTokens()) {
15         word.set(tokenizer.nextToken());
16         output.collect(word, one);
17      }
18    }
19  }
```

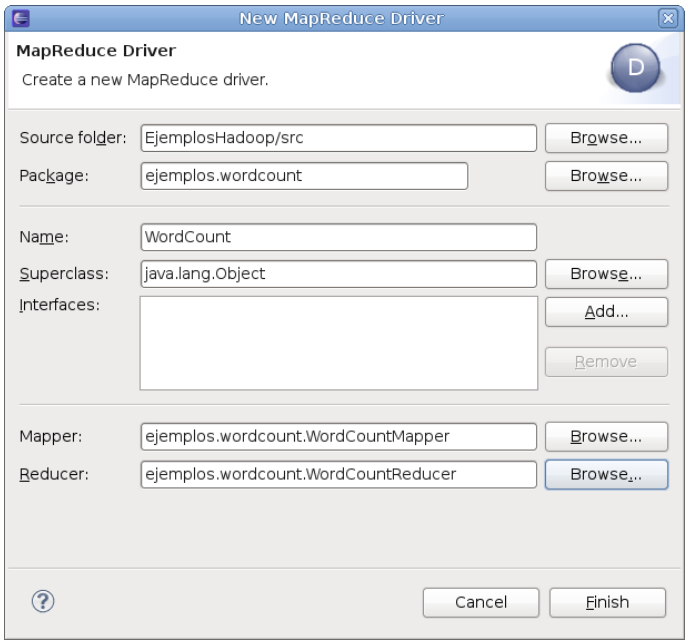
2. Si el IDE no encuentra la librería de Hadoop, agregar hadoop<version>-core.jar al proyecto.

3. Creamos un nuevo Reducer: file->new->Reducer

```
01  package ejemplos.wordcount;
02
03  import java.io.IOException;
04  import java.util.Iterator;
05
06  import org.apache.hadoop.io.*;
07  import org.apache.hadoop.mapred.MapReduceBase;
08  import org.apache.hadoop.mapred.OutputCollector;
09  import org.apache.hadoop.mapred.Reducer;
10  import org.apache.hadoop.mapred.Reporter;
11
12  public class WordCountReducer extends MapReduceBase implements
    Reducer<Text, IntWritable, Text, IntWritable> {
13
14      public void reduce(Text key, Iterator<IntWritable> values,
    OutputCollector<Text, IntWritable> output, Reporter reporter) throws
```

```
IOException {
15          int sum = 0;
16          while (values.hasNext()) {
17              sum += values.next().get();
18          }
19          output.collect(key, new IntWritable(sum));
20      }
21  }
```

4. Creamos el MapReduce Driver, seleccionando las clases que generamos anteriormente como Mapper y Reducer.



Modificamos las propiedades del Job:

```
01  package ejemplos.wordcount;
02
03  import org.apache.hadoop.fs.Path;
04  import org.apache.hadoop.io.*;
05  import org.apache.hadoop.mapred.*;
06
07  public class WordCount {
08
09      public static void main(String[] args) {
10          JobClient client = new JobClient();
11          JobConf conf = new
    JobConf(ejemplos.wordcount.WordCount.class);
12          conf.setJobName("wordcount");
13
14          //  specify output types
15          conf.setOutputKeyClass(Text.class);
16          conf.setOutputValueClass(IntWritable.class);
17
```

```
18          //  specify input and output Format and DIRECTORIES (not
    files)
19          conf.setInputFormat(TextInputFormat.class);
20          conf.setOutputFormat(TextOutputFormat.class);
21
22          FileInputFormat.setInputPaths(conf, new Path("input"));
            FileOutputFormat.setOutputPath(conf, new Path("output"));
23
            conf.setMapperClass(ejemplos.wordcount.WordCountMapper.class)
24  ;
25                      conf.setCombinerClass(ejemplos.wordcount.WordCoun
    tReducer.class);
26          conf.setReducerClass(ejemplos.wordcount.WordCountReducer.clas
    s);
27
            client.setConf(conf);
28          try {
29              JobClient.runJob(conf);
30          } catch (Exception e) {
31              e.printStackTrace();
32          }
33      }
34  }
35
36
```

5. Subimos algunos datos al sistema de archivos distribuido, lo podemos hacer desde el DFS location en el project explorer o desde una terminal ejecutando:

```
$HADOOP_HOME/bin/hadoop fs -put conf input
```

6. Ahora ejecutamos la aplicación, si el plugin funciona se puede ejecutar haciendo clic derecho sobre la clase que actúa como MapReduce *Driver* y *Run As->Run on Hadoop* aparecerá una ventana preguntando en cual ubicación de Hadoop ejecutar la aplicación. Si no funciona, generamos el jar y lo ejecutamos:

```
$HADOOP_HOME/bin/hadoop jar wordcount.jar input output
```

Cuando termine el trabajo podremos ver la salida en la carpeta Output del DFS, tanto a través del plugin (probablemente necesite un *refresh* o un *reconect* para actualizarse) como por línea de comandos:

```
$HADOOP_HOME/bin/hadoop fs -ls output
$HADOOP_HOME/bin/hadoop fs -cat output/part*
```

Bibliografía

1. **John F. Gantz, Christopher Chute, Alex Manfrediz, Stephen Minton, David.** The diverse and exploding. [En línea] 2008. [Citado el: 07 de 06 de 2011.] http://www.emc.com/collateral/analyst-reports/diverse-exploding-digital-universe.pdf.

2. **White, Tom.** *Hadoop: The Definitive Guide.* s.l. : O'Reilly, 2009. ISBN: 978-0-596-52197-4.

3. European Organization for Nuclear Research. [En línea] [Citado el: 24 de 07 de 2011.] http://public.web.cern.ch/public/en/lhc/Computing-en.html.

4. **Maldonado, Luis Daniel Soto.** BigData: La base de datos relacional no lo es todo . [En línea] 17 de 05 de 2011. [Citado el: 22 de 07 de 2011.] http://www.sg.com.mx/content/view/1179.

5. **Jeffrey Dean, Sanjay Ghemawat.** MapReduce: Simplified Data Processing on Large *Clusters*. *Google Labs.* [En línea] 4 de Diciembre de 2004. [Citado el: 24 de 07 de 2011.] http://labs.google.com/papers/mapreduce.html.

6. **Sanjay Ghemawat, Howard Gobioff, Shun-Tak Leung.** The Google File System. *Google Labs.* [En línea] 19 de 10 de 2003. [Citado el: 26 de 07 de 2011.] http://labs.google.com/papers/gfs.html.

7. Wikipedia - MapReduce. [En línea] [Citado el: 24 de 08 de 2011.] http://en.wikipedia.org/wiki/MapReduce.

8. Massive Information processing - Concurrency and Fault-Tolerance: The Google Approach. [En línea] [Citado el: 02 de 09 de 2011.]

http://www.cis.temple.edu/~ingargio/cis307/readings/MapReduce/MapReduce.html.

9. **Jeffrey Cohen, Brian Dolan, Mark Dunlap, Joseph M. Hellerstein, Caleb Welton.** Berkeley Database Group. [En línea] [Citado el: 22 de 07 de 2011.] http://db.cs.berkeley.edu/jmh/papers/madskills-032009.pdf.

10. Temoa. *Portal de Recursos Educativos Abiertos.* [En línea] [Citado el: 22 de 07 de 2011.] http://www.temoa.info/es/node/60377.

11. Evaluating MapReduce for Multi-core and Multiprocessor Systems. [En línea] [Citado el: 27 de 08 de 2011.] http://csl.stanford.edu/~christos/publications/2007.cmp_mapreduce.hpca.pdf.

12. Applications of Map-Reduce. [En línea] [Citado el: 27 de 09 de 2011.] http://web.cs.wpi.edu/~cs4513/d08/OtherStuff/MapReduce-TeamC.ppt.

13. **Gottfrid, Derek.** New York Times. *Self-Service, Prorated Supercomputing Fun!* [En línea] New York Times, 1 de 11 de 2007. [Citado el: 15 de 07 de 2011.] http://open.blogs.nytimes.com/2007/11/01/self-service-prorated-super-computing-fun/?scp=1&sq=self%20service%20prorated&st=cse.

14. **Jose María Ruíz.** BigTable. *Linux-magazine.* [En línea] [Citado el: 31 de Julio de 2011.] http://www.linux-magazine.es/issue/39/047-050_PythonLM39.pdf.

15. Wikipedia - Hadoop. [En línea] [Citado el: 24 de 07 de 2011.] http://es.wikipedia.org/wiki/Hadoop.

16. Apache Software Foundation. *Apache Hadoop.* [En línea] [Citado el: 25 de 08 de 2011.] http://apachefoundation.wikispaces.com/Apache+Hadoop.

17. Apache Hadoop and HBase. [En línea] [Citado el: 30 de 07 de 2011.] http://nosql.mypopescu.com/post/1473423255/apache-hadoop-and-hbase.

18. Hadoop Wiki - PoweredBy. [En línea] [Citado el: 19 de 09 de 2011.] http://wiki.apache.org/hadoop/PoweredBy.

19. Wikipedia. *Zipf's law.* [En línea] [Citado el: 10 de 11 de 2011.] http://en.wikipedia.org/wiki/Zipf%27s_law.

20. *Power-Law Distribution of the World Wide Web.* **Huberman, Lada A. Adamic y Bernardo A.** 5461, s.l. : Science, 24 de March de 2000, Science, Vol. 287, pág. 2115. ISSN 0036-8075 (print), 1095-9203 (online) .

21. PowerPivot para Excel 2010. [En línea] [Citado el: 29 de 10 de 2011.] http://office.microsoft.com/es-es/excel/caracteristicas-y-ventajas-de-powerpivot-para-excel-2010-HA101810445.aspx.

22. Hadoop Wiki - WordCount. [En línea] [Citado el: 10 de 10 de 2010.] http://wiki.apache.org/hadoop/WordCount.

23. **Ariza, Christian.** *Mordiendo Hadoop: Desarrollo de aplicaciones MapReduce.* [En línea] [Citado el: 25 de 11 de 2010.] http://www.christian-ariza.net/techstuff/mordiendo-hadoop-desarrollo-de-aplicaciones-mapreduce.