2875: INTEGRACIÓN DE HADOOP CON PLANIFICADORES BATCH

Memoria del Proyecto Fin de Carrera de Ingeniería en Informática realizado por

José Fidel Díaz Cañizares

y dirigido por

Porfidio Hernández Budé

Bellaterra, Junio de 2011

Universitat Autònoma de Barcelona

etse Escola Tècnica Superior d'Enginyeria

El sotasignat, Porfidio Hernández Budé

Professor/a de l'Escola Tècnica Superior d'Enginyeria de la UAB,

CERTIFICA:

Que el treball a què correspon aquesta memòria ha estat realitzat sota la seva direcció per en José Fidel Díaz Cañizares

I per tal que consti firma la present.

Signat: Porfidio Hernández Budé

Bellaterra, Juny de 2011

Agradecimientos

Deseo expresar mi agradecimiento al Dr. Porfidio Hernández Budé por la dedicación y ayuda prestada a lo largo del todo el proyecto. También por la pasión que desprende en sus explicaciones. Las charlas que hemos podido mantener en su caluroso despacho de la ETSE, han contribuido a aumentar, si cabe, la motivación por una temática que, ya de por sí, me resultaba atractiva.

Y como no, a mi mujer, Lidia, y a mis hijos, Pau y Joel, por la compresión mostrada y por el tiempo que les he robado en estos meses que ha durado la elaboración de este proyecto.

Índice general

Índice de figuras

Índice de tablas

Capítulo 1. Introducción

1.1. Presentación del problema

En 1965, el cofundador de Intel, Gordon Moore predijo que el número de transistores en un circuito integrado se duplicaría aproximadamente cada 18 meses (Moore, 1965) [1]. Posteriormente esta ley sería interpretada de diversas maneras: la potencia de los microprocesadores se duplica cada 18 meses, la potencia de cómputo se duplica cada 18 meses, el precio de la computación se reduce cada 18 meses.

El volumen de información digital está creciendo probablemente más rápido de lo que la ley de Moore predijo. En plena era de la información, cada día se genera una cantidad ingente de datos que hace necesario medirlos a escala de cientos de petabytes. Esto incluye transacciones electrónicas, publicaciones y medios de comunicación gestionados por órganos gubernamentales y organizaciones comerciales, contenidos sociales creados por millones de usuarios y los resultados de grandes experimentos científicos. A continuación se muestran algunos casos que muestran el crecimiento comentado:

- Google pasó de procesar 100 terabytes de datos por día en el año 2004 a procesar 20 petabytes en el año 2008 [2].
- Los usuarios de Facebook generan cada día 15 terabytes de nueva información [3].
- eBay dispone de dos *datawarehouses* de 2 y 6,5 petabytes y los usuarios de este portal generan diariamente 50 terabytes de nueva información [4].
- El acelerador de partículas de Ginebra (LHC) genera unos 15 petabytes de datos al año [5].
- El futuro telescopio LSST (*Large Synoptic Survey Telescope)* que se está construyendo en Chile y que entrará en funcionamiento en el año 2012, está previsto que genere medio petabyte de información cada mes correspondiente a las imágenes generadas con su cámara de 3,2 giga pixeles [6].

Se estima que la cantidad de información digital creada en el año 2011 será de aproximadamente 1.800 exabytes y se calcula que el volumen de información se verá multiplicado por 10 cada cinco años [7]. En cambio, las tasas de transferencia de la información, ya sea de discos físicos o de redes

de interconexión, o bien no están a la altura de tal evolución o bien requieren de una fuerte inversión que hace que sea difícilmente asumible por una organización. Tanto el crecimiento exponencial de la información digital como los nuevos retos de almacenamiento masivo de datos, están cambiando la arquitectura de los nuevos supercomputadores y el modo en que son utilizados para acelerar el procesamiento de la información. Estos cambios están centrados en una nueva forma de computación de alto rendimiento, que pone énfasis en los datos como foco central del sistema. Dicho sistema es responsable de la adquisición, modificación, intercambio y archivado de los datos y son conocidos como sistemas de Computación Intensiva de Datos o DIC (*Data-Intensive Computing*).

A un ritmo tan asombroso de crecimiento de la información, no es extraño que muchas organizaciones tengan la necesidad de procesar y analizar, cada vez con mayor rapidez, el creciente volumen de datos, con el fin de obtener una mayor ventaja competitiva o acelerar el descubrimiento científico.

Uno de los enfoques más adecuados para hacer frente a los problemas que requieren de gran procesamiento de información, es el paradigma de diseño algorítmico “divide y vencerás”, un concepto fundamental en la informática, que consiste en resolver un problema a partir de la solución de sub-problemas del mismo tipo pero de menor tamaño. Si los sub-problemas son todavía relativamente grandes, se aplica de nuevo esta técnica hasta alcanzar sub-problemas lo suficientemente pequeños para ser solucionados directamente. En la medida que los sub-problemas son independientes, pueden ser abordados en paralelo por diferentes hilos de ejecución en el núcleo de un procesador, en los núcleos de un procesador multi-núcleo, en los procesadores de una máquina o por varias máquinas en un clúster. Los resultados intermedios de cada hilo son combinados para generar el resultado final.

Los sistemas de Cómputo de Altas Prestaciones (HPC) tradicionales presentan una arquitectura como la que se muestra en la Figura 1.1. Estos sistemas disponen de nodos de cómputo con almacenamiento local mínimo y nodos de almacenamiento, enlazados entre sí mediante una interconexión de alta capacidad como Gigabit Ethernet o InfiniBand. Los nodos de cómputo suelen estar conectados a un sistema de ficheros de alto rendimiento como Lustre o GPFS (*General Parallel File System* de IBM)

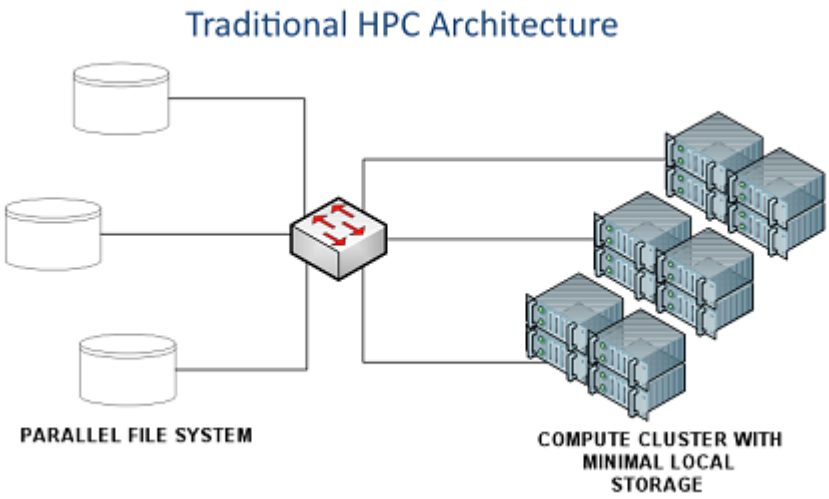


Figura 1.1. Arquitectura HPC tradicional.

El desarrollo de la programación paralela ha permitido que varios procesadores puedan trabajar juntos para resolver una tarea común, donde cada procesador trabaja en una porción del problema y donde los procesos pueden intercambiarse datos a través de la memoria o a través de paso de mensajes. De esta manera, es posible resolver problemas que por su envergadura no podrían ser resueltos por una CPU o en un tiempo razonable. Modelos como el paso de mensajes (*message passing*) o de memoria compartida (*shared memory*), mostrados de forma esquemática en la Figura 1.2, son ampliamente utilizados en entornos HPC.

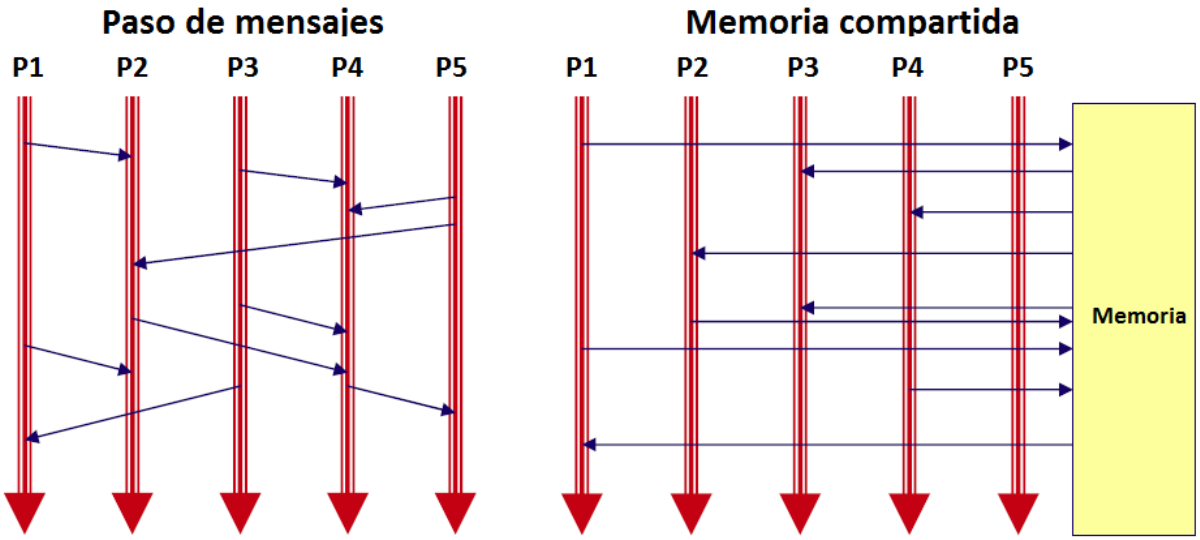


Figura 1.2. Modelos de programación paralela.

En los entornos tradicionales de programación paralela, el programador debía abordar parte (o todas) las tareas relacionadas a continuación:

- Determinar el modo en que se debe descomponer el problema para que las tareas más pequeñas se puedan ejecutar en paralelo.

- Asignar tareas a los nodos de cómputo a través de un número potencialmente elevado de máquinas.
- Asegurarse que los nodos de cómputo reciben la información que necesitan.
- Coordinar la sincronización entre los diferentes procesos.
- El modo en que los procesos deben compartir los resultados generados.
- Controlar errores de software y fallos de hardware.

Al tener que abordar aspectos de bajo nivel en el desarrollo, el resultado era aplicaciones que estaban fuertemente acopladas al hardware sobre el cual se habían diseñado y, como consecuencia, poco portables. A lo largo de los años, se han desarrollado diferentes interfaces de programación con el objetivo de ocultar detalles sobre la comunicación y sincronización de procesos. Algunos de los más extendidos son:

- OpenMP, ofrece un estándar de programación paralela sobre sistemas de memoria compartida portable y permite añadir concurrencia a los programas sobre la base del modelo de ejecución fork-join [8].
- Interfaz de Paso de Mensajes (MPI), es una especificación para programación de paso de mensajes, estándar y portable, creada para permitir desarrollar aplicaciones que puedan ser migradas a diferentes computadores paralelos [9].

Sin embargo, incluso con estas extensiones, los desarrolladores siguen teniendo dificultades para sacar el máximo partido a las arquitecturas actuales mediante el uso de estos paradigmas.

Con MPI el usuario debe codificar todo el paralelismo de su aplicación, desde iniciar el entorno paralelo, crear los grupos de procesos, distribuir los datos y tareas entre ellos, especificar todas las comunicaciones, y así hasta la finalización del entorno paralelo. Esto requiere que, para crear programas paralelos en MPI, se precisen profundos conocimientos de paralelismo y, en algunos casos, de la arquitectura paralela sobre la cual se está desarrollando. Todo esto origina que los programas paralelos que se obtienen en MPI, incluso para los casos más simples, raramente se asemejan a su homólogo secuencial.

OpenMP también presenta algunas desventajas, como la dificultad en ajustar los parámetros para obtener un alto rendimiento o la escalabilidad, por tratarse de un paradigma diseñado específicamente para ser ejecutados en máquinas de memoria compartida.

En el año 2004, Google introduce un nuevo paradigma de programación paralela, denominado MapReduce, enfocado al procesamiento de grandes conjuntos de datos [10]. En este modelo, el desarrollador crea una función denominada *map*, que procesa un conjunto de datos, típicamente bloques de un fichero, para generar un conjunto intermedio de duplas <clave, valor>, y una función denominada *reduce* que combina todos los valores asociados a una misma clave intermedia.

Los programas escritos mediante este modelo, son automáticamente paralelizados y ejecutados en un clúster, dado que las primitivas *map* y *reduce* se pueden ejecutar de manera paralela. En tiempo de ejecución, el sistema se encarga de detalles como el particionado de los datos de entrada, planificar la ejecución de los programas en las diferentes máquinas del clúster, gestionar los fallos que se puedan producir en las máquinas y gestionar la comunicación entre las máquinas. Este modelo permite a programadores, sin ninguna experiencia en sistemas paralelos o distribuidos, utilizar de manera sencilla los recursos de un gran sistema distribuido.

En la práctica, muchas de las aplicaciones intensivas de datos no son muy exigentes en lo que respecta a la potencia del procesador, por lo que la separación entre el cómputo y el almacenamiento que se produce en los sistemas HPC tradicionales acaba provocando un cuello de botella en la red.
Una manera de resolver esta problemática es acercar el proceso o las tareas de cómputo a los datos tanto como sea posible. Para ello, se aumenta la capacidad de almacenamiento local del nodo de cómputo y, por tanto, se minimiza el tráfico en la red de interconexión. En consecuencia, la máquina se acaba convirtiendo en un nodo de cómputo y en un nodo de almacenamiento (arquitectura conocida con el nombre de *shared-nothing)*, tal y como se puede observar en la Figura 1.3.

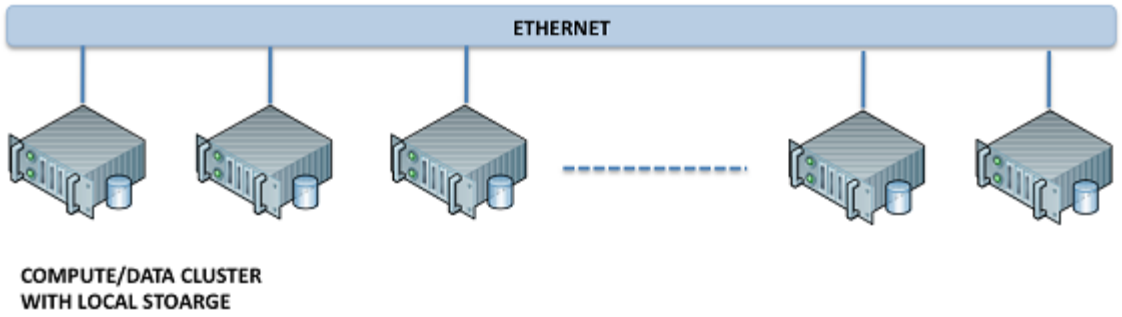


Figura 1.3. Arquitectura shared-nothing.

La distribución del almacenamiento en los diferentes nodos de cómputo hace necesaria la existencia de un sistema de ficheros distribuido como HDFS (*Hadoop Distributed File System*) [11] o GFS (*Google File System*) [12]. La idea principal consiste en dividir la información en bloques y replicarlos a lo largo de los discos locales de los nodos del clúster. Este tipo de sistema de ficheros adopta una arquitectura master/worker, en la cual el *master* mantiene meta-información (estructura de directorios, mapeo de bloques que forman un fichero, la localización de los bloques y permisos de acceso) y los *workers* gestionan los bloques de información. Uno de los puntos clave del modelo MapReduce consiste en ejecutar la tarea en el nodo donde reside la información que ésta necesita procesar, minimizando de esta manera la transferencia de información entre nodos. En caso que el nodo se encuentre ocupado, la tarea se derivará a otro procesador libre y el sistema de ficheros distribuido se encargará de proporcionar la información necesaria a dicha tarea.

MapReduce ha ganado un enorme interés en los últimos años, convirtiéndose rápidamente en un modelo muy popular de computación distribuida debido a su simplicidad y escalabilidad a un bajo

coste. MapReduce saltó al primer plano de la computación distribuida gracias al éxito que tuvo Google al reescribir el sistema de indexación, que genera las estructuras de datos usadas por su servicio de búsqueda, para adaptarlo a este nuevo modelo de programación.

Esta popularidad ha contribuido a la aparición de diferentes implementaciones algunas de las cuales se describen a continuación:

- Ejemplos de implementaciones para un solo nodo multinúcleo:
  - **Mars**. Se trata de una herramienta que permite desarrollar aplicaciones intensivas de datos y cómputo sobre procesadores gráficos (GPUs). Mars oculta la complejidad que conlleva la programación de una GPU mediante una interface simple permitiendo al desarrollador escribir su código sin conocimientos de la arquitectura del procesador ni de la API gráfica [13].
  - **Metis**. Se trata de una librería MapReduce optimizada para arquitecturas multinúcleo. La alta eficiencia de Metis se basa en el uso de tablas hash para almacenar las duplas intermedias <clave, valor> [14]. Metis organiza las duplas almacenadas en cada uno de los slots de la tabla hash como un árbol B+
  - **Phoenix**. Se trata de una implementación de MapReduce para memoria compartida. Puede ser usado tanto en procesadores multinúcleo como en multiprocesadores de memoria compartida [15].

- Ejemplos de implementaciones para clúster:
  - Google MapReduce. Implementación propietaria de Google.
  - Apache Hadoop. Implementación de código abierto del modelo MapReduce.
  - Microsoft Dryad. Iniciativa de Microsoft Research concebido para ser utilizado en entornos Windows Server y SQL Server [16].

Hadoop es, sin duda, uno de los proyectos que más acogida ha tenido entre la comunidad de software libre y, es también, uno de los responsables del auge de la computación distribuida en entornos no HPC. Hadoop es una plataforma basada en Java y orientada a aplicaciones distribuidas con un uso intensivo de datos, aunque en realidad se puede hablar de todo un ecosistema de proyectos alrededor del núcleo de Hadoop. Ese núcleo está compuesto por un sistema de ficheros distribuido (HDFS), una interface para el desarrollo de software mediante el modelo MapReduce (Hadoop Mapreduce) y un componente software que ejecuta, en un clúster, los programas que han sido creados mediante la interface Hadop Mapreduce. En el apartado 1.4 se profundiza acerca de la arquitectura de Hadoop.

Hadoop se ha convertido en una herramienta imprescindible, en las grandes empresas de internet, para el análisis de las enormes cantidades de datos que generan. Yahoo! encabezó el desarrollo de Hadoop y es el principal contribuidor del proyecto Apache. Fue la primera empresa en desarrollar

parte de su infraestructura crítica con esta tecnología. Actualmente posee un clúster con capacidad para almacenar 14 petabytes, aunque el mayor clúster Hadoop que se conoce es el *datawarehouse* de Facebook, siendo las características de dicho clúster las siguientes [17]:

- 21 Pb de almacenamiento en un único clúster HDFS.
- 2.000 máquinas.
- 12 TB por máquina.
- 1.200 máquinas de 8 núcleos + 800 máquinas de 16 núcleos
- 32 GB de RAM por máquina.
- 15 tareas MapReduce por máquina.

El gráfico de la parte superior de la Figura 1.4, muestra la evolución de las búsquedas relacionadas con la palabra Hadoop. El gráfico de la parte inferior, muestra la evolución de noticias y artículos relacionados con Hadoop que han aparecido en los últimos años. Ambos gráficos proporcionan una muestra del interés que ha despertado este tema en la comunidad en los últimos años, reflejando su popularidad.

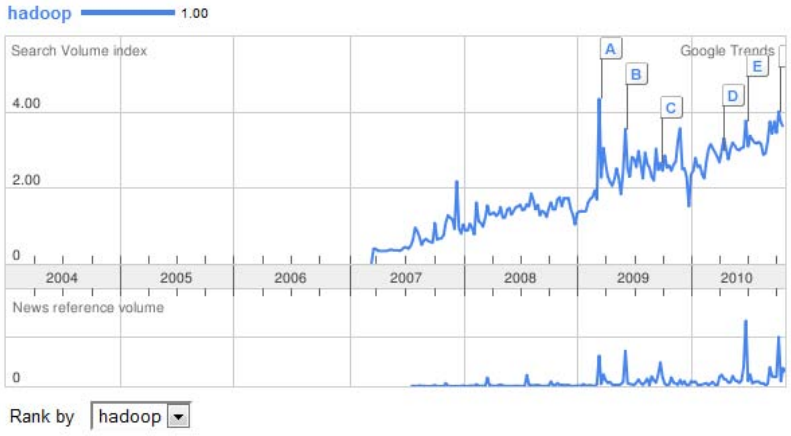


Figura 1.4. Evolución de las búsquedas de Hadoop en Google.

En entornos HPC, la ejecución de aplicaciones paralelas requiere una especial atención debido a que este tipo de aplicaciones están formadas por un gran número de tareas. Cada tarea se asigna a un único nodo de cómputo y, durante la ejecución de la aplicación, los nodos están constantemente comunicándose entre ellos. La forma en la cual las tareas son asignadas a los procesadores se denomina mapping. Debido a que el mapping afecta al tiempo de ejecución, el planificador debe realizar la asignación de tareas de forma cuidadosa. El planificador debe asegurarse que los nodos seleccionados para ejecutar una aplicación paralela están conectados por una red de interconexión rápida para minimizar la sobrecarga asociada a la comunicación entre nodos. Algunos de los gestores de recursos más utilizados en entornos HPC son PBS (Portable Batch System), Torque [18] o SGE (Sun Grid Engine) [19].

Cada vez es mayor el número de aplicaciones desarrolladas en el ámbito científico, como en la Bioinformática[20][21] o en las Geociencias[22], escritas bajo el modelo MapReduce empleando herramientas de código abierto como Apache Hadoop. No obstante, la ejecución de aplicaciones MapReduce en un entorno HPC clásico, utilizando herramientas como Hadoop, ha resultado ser una tarea ardua y costosa debido a que Hadoop posee su propio gestor de recursos, encargándose él mismo de gestionar la ejecución y seguimiento de tareas. Esta dificultad, ha llevado a muchos usuarios a crear un clúster físico para gestionar y mantener sus propias instancias de Hadoop.

No obstante, muchos usuarios de entornos HPC sólo tienen acceso a recursos del tipo HPC. Por este motivo, han surgido nuevas líneas de trabajo que han derivado en la implementación de herramientas que permiten generar un clúster Hadoop bajo demanda en entornos HPC, como HoD [23] y myHadoop [24]. A través del sistema estándar de procesamiento por lotes instalado en este tipo de entornos, se realiza la petición de recursos para la generación de un clúster Hadoop de forma dinámica. Una vez se ha ejecutado el trabajo, se liberan los recursos de la misma manera que se haría para cualquier trabajo ejecutado. De esa manera, se provee a los usuarios de entornos HPC una forma sencilla de ejecutar aplicaciones desarrolladas bajo el modelo MapReduce.

1.2. Planificadores batch

La planificación consiste en el despliegue de las tareas, en las que se divide un trabajo, sobre los nodos de cómputo del sistema, atendiendo a las necesidades de recursos y a la dependencia entre dichas tareas.

Un sistema de gestión de recursos administra la carga, previniendo que unos trabajos compitan con otros por los recursos del sistema. Por lo general, un sistema de gestión de recursos cuenta con un administrador de recursos y un planificador de tareas, como se muestra en la Figura 1.5. La mayoría de los administradores de recursos tienen un planificador de tareas interno, por lo que el administrador del sistema suele sustituir el planificador externo por el planificador interno para mejorar la funcionalidad. En cualquier caso, el planificador se comunica con el gestor de recursos para obtener información acerca de las colas, la carga de trabajo en los nodos de cómputo y la disponibilidad de recursos, para tomar decisiones de programación de trabajos.

Por lo general, el administrador de recursos ejecuta varios demonios en el nodo maestro del clúster y en los nodos de cómputo. La administración de recursos también establece un sistema de colas para los trabajos que envían los usuarios al clúster, pudiendo dichos usuarios realizar consultas al administrador de recursos para determinar el estado de sus trabajos. Además, un administrador de recursos mantiene una lista de recursos de cómputo disponibles, y reporta el estado de los trabajos

previos enviados por el usuario. El administrador de recursos ayuda organizar los trabajos enviados dependiendo de la prioridad, los recursos solicitados, y la disponibilidad de dichos recursos.

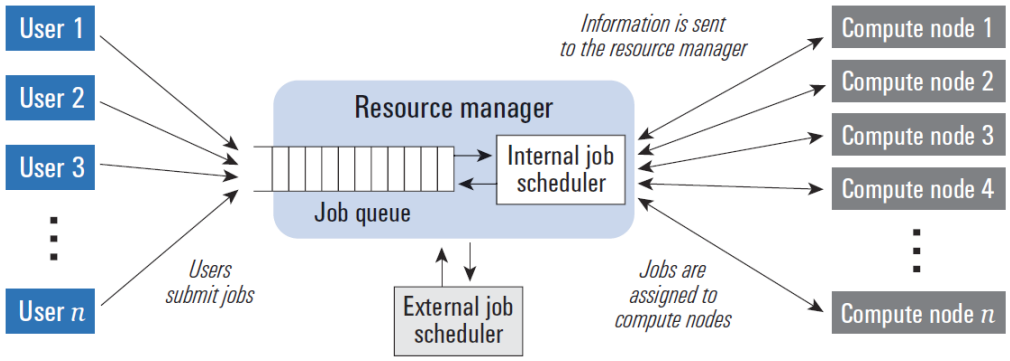


Figura 1.5. Sistema de gestión de recursos típico.

El planificador recibe del gestor de recursos, de forma periódica, peticiones acerca del estado de la cola de trabajos y los recursos disponibles, realizando una planificación que establece el orden en el cual se ejecutarán los trabajos pendientes. Esto se hace manteniendo, al mismo tiempo, la prioridad de los trabajos, de conformidad con las políticas que ha establecido el administrador respecto a la cantidad de recursos y el tiempo que un trabajo puede utilizarlos. En base a esa información, el planificador decide en qué nodo de cómputo y en qué momento un trabajo será ejecutado.
La planificación de un trabajo puede hacerse de dos formas:

- Planificación estática: Antes de que el trabajo entre en máquina, se determina dónde y cuándo se va a ejecutar las tareas asociadas a dicho trabajo. Propio de sistemas HPC.
- Planificación dinámica: Una vez desplegado un trabajo, y de acuerdo al comportamiento del sistema, se puede revisar el despliegue inicial. Propio de sistemas distribuidos como Hadoop.

En la planificación estática, el planificador selecciona un trabajo de la cola y, si hay recursos disponibles, lo pone en ejecución, de lo contrario espera. Para poder tomar las decisiones correspondientes a la política del planificador, éste debe disponer de información sobre los trabajos a ejecutar: número de tareas a ejecutar, prioridad, relación entre ellas, estimación de consumo de recursos, estimación del tiempo de ejecución de cada tarea, etc.

Para la selección del siguiente trabajo a ejecutar de la cola de trabajos existen diferentes políticas:

- **FCFS** (First-Come-First.Serve): Se respeta el orden de llegada de los trabajos.
- **SJF** (Shorted-Job-First): El trabajo más pequeño en primer lugar, medido en recursos que necesita, uso del procesador o tiempo estimado de ejecución.
- **LJF** (Longest-Job-First): Al contrario que SJF, se ejecuta el trabajo más grande en primer lugar.

- **Basada en prioridades**: administrativamente se define unos criterios de prioridad, que pueden contemplar la facturación del coste de los recursos a emplear, el número de trabajos enviados, el trabajo más urgente en primer lugar (EDF, Earliest-Deadline-First), etc.

Sobre las políticas comentadas anteriormente, se pueden aplicar otras técnicas, como el backfilling, para adelantar trabajos. En el caso que el trabajo seleccionado no tiene recursos para ejecutarse, se busca otro trabajo en la cola que demande menos recursos y pueda entrar en ejecución. De esta manera se aprovecha mejor el sistema.

Dado que el backfilling puede provocar que trabajos que demanden muchos recursos nunca se ejecuten, se aplica el backfilling con reserva. Si la tarea seleccionada no se puede ejecutar, se estima cuándo podrá ser ejecutada. En base a esa estimación, se dejan entrar trabajos que demanden menos recursos, siempre y cuando finalicen antes del inicio estimado para la tarea que demandaba mayor número de recursos. De esta manera, aumenta el aprovechamiento del sistema sin retrasar indefinidamente a los trabajos grandes.

La planificación estática, decide si un proceso se ejecuta en el sistema o no pero, una vez lanzado, no se realiza ningún seguimiento del mismo. En cambio, la planificación dinámica:

- Evalúa el estado del sistema y toma decisiones correctivas.
- Resuelve problemas debidos a la paralelización del problema (desequilibrio entre las tareas).
- Reacciona ante fallos en los nodos del sistema (caídas o fallos parciales).
- Permite un uso no dedicado o exclusivo del sistema.

La planificación dinámica requiere, monitorizar el sistema y aplicar políticas de gestión de trabajos, para balancear la carga entre los nodos de cómputo:

- Load Sharing: busca que el estado de los procesadores no sea diferente, por lo que, si una tarea está esperando a ser servida en otro procesador y existe un procesador ocioso, se realiza la transferencia del trabajo.
- Load Balancing: el objetivo buscado es que la carga de los procesadores sea igual. Normalmente la carga varía durante la ejecución de un trabajo, por lo que no realiza la transferencia de tareas en forma tan continua como en Load Sharing.

1.3. Paradigma MapReduce

El modelo de programación MapReduce fue introducido por primera vez por Google a finales del año 2004 [10]. El objetivo era crear un framework adaptado a la computación paralela que permitiera

procesar y generar grandes colecciones de datos sobre máquinas genéricas, sin la necesidad de utilizar supercomputadores o servidores dedicados, y que fuera fácilmente escalable.

En esencia, el modelo que propone MapReduce es bastante sencillo. El programador debe encargarse de implementar dos funciones, *map* y *reduce*, que serán aplicadas a todos los datos de entrada. Tareas como particionar los datos de entrada, despliegue de maestro y trabajadores, esquema de asignación de trabajos a los trabajadores, comunicación y sincronización entre procesos, tratamiento de caídas de procesos, quedan a cargo del entorno de ejecución, liberando de esa manera al programador.

Las funciones *map* y *reduce* están definidas ambas con respecto a datos estructurados en duplas <clave, valor>. La función *map* es aplicada en paralelo sobre cada dupla de entrada (k1,v1), generando una lista intermedia de duplas <clave, valor> (lista(k2,v2)). La función *reduce*, es aplicada en paralelo sobre cada grupo de duplas con la misma clave intermedia (k2), tomando los valores (v2) contenidos en el grupo para generar como salida una nueva lista de duplas <clave, valor> (lista(k3,v3)). Entre las fases *map* y *reduce*, existe una operación de agrupación que junta todos los pares con la misma clave de todas las listas, creando un grupo por cada una de las diferentes claves generadas. Las duplas generadas por la función *reduce,* son grabadas de forma permanente en el sistema de ficheros distribuido. Por tanto, la ejecución de la aplicación acaba produciendo r ficheros, donde r es el número de tareas *reduce* ejecutadas. La Figura 1.6 muestra una visión simplificada del paradigma de programación MapReduce.

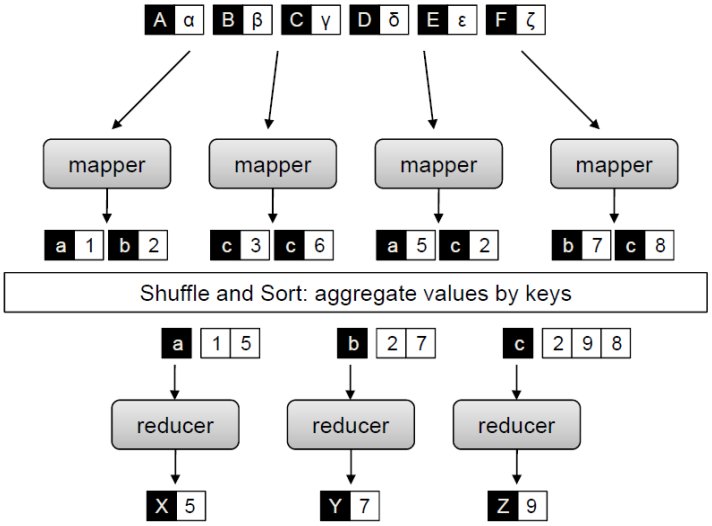


Figura 1.6. Visión simplificada del modelo MapReduce.

Una de las ideas más importantes que hay detrás de MapReduce, es separar el qué se distribuye del cómo. Un trabajo MapReduce está formado por el código asociado a las funciones *map* y *reduce,* junto con parámetros de configuración. El desarrollador envía el trabajo al planificador y el entorno de ejecución se encarga de todo lo demás, gestionando el resto de aspectos relacionados con la ejecución distribuida del código, como son:

- Planificación. Cada aplicación MapReduce es dividida en unidades más pequeñas denominadas tareas. Dado que el número de tareas puede ser superior al de los nodos del clúster, es necesario que el planificador mantenga una cola de tareas y vaya haciendo un seguimiento del progreso de las tareas en ejecución, asignando tareas de la cola de espera a medida que los nodos van quedando disponibles.
- Co-ubicación datos/código. Una de las ideas clave de MapReduce es mover el código y no los datos. Esta idea está ligada con la planificación y depende, en gran medida, del diseño del sistema de ficheros distribuido. Para lograr la localidad de datos, el planificador intentará ejecutar la tarea en el nodo que contiene un determinado bloque de datos necesario para la tarea.
- Sincronización. En MapReduce, la sincronización se consigue mediante una “barrera”. Una “barrera”, es un mecanismo de sincronización entre procesos que espera a que todos los procesos de un lado de la barrera terminen antes que empiecen los procesos del otro lado. Esto significa que la fase *map* debe terminar antes que empiece la fase *reduce*.
- Manejo de errores y fallos. El entorno de ejecución debe cumplir con todas las tareas mencionadas en los puntos anteriores, en un entorno donde los errores y los fallos son la norma. MapReduce fue diseñado expresamente para ser ejecutados en servidores de gama baja, por lo que el entorno de ejecución debe ser especialmente resistente. En grandes clústeres de máquinas, los errores de disco y de RAM son comunes.

Anteriormente, se ha presentado una visión simplificada de MapReduce. Existen dos elementos adicionales que completan el modelo de programación, denominados *combiner* y *partitioner*. La Figura 1.7 muestra una visión completa de MapReduce.

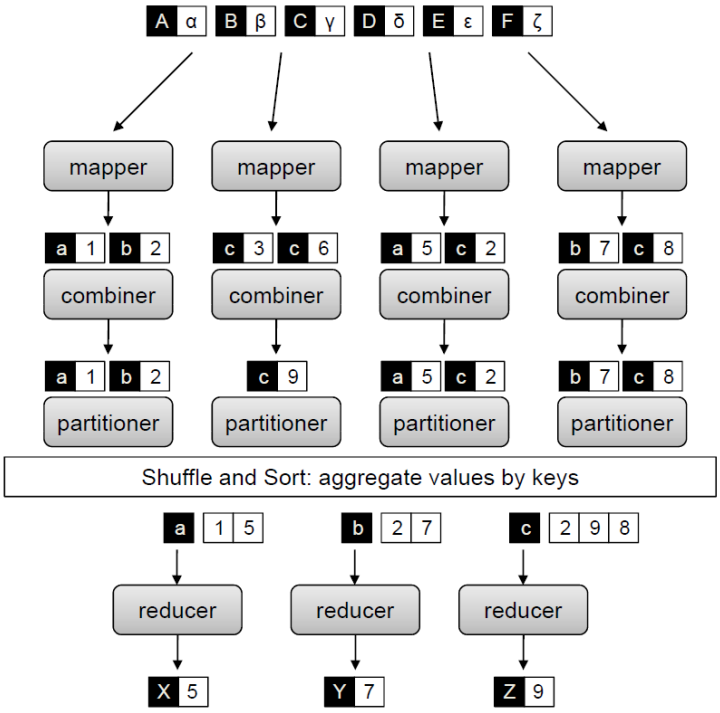


Figura 1.7. Visión expandida del modelo MapReduce.

El *partitioner* es el responsable de dividir el espacio de claves intermedias y asignar duplas <clave, valor> intermedias a las tareas *reduce*. Dicho de otro modo, el *partitioner* especifica la tarea a la cual debe asignarse una dupla <clave, valor>. Para ello, calcula el valor hash de la clave y obtiene el módulo del valor en base al número de tareas *reduce* a ejecutar. El objetivo es asignar el mismo número de claves a cada *reduce*. Sin embargo, dado que el *partitioner* sólo tiene en cuenta la clave y no los valores, puede haber grandes diferencias en el volumen de duplas <clave, valor> enviadas a cada *reduce*, dado que diferentes claves pueden tener diferente número de valores asociados.

El combiner representa un punto de optimización en el modelo MapReduce. En algunos casos, existe una repetición significativa en las claves intermedias producidas por cada *map*. Todas esas repeticiones deberían ser enviadas, a través de la red, al nodo asignado para procesar una tarea *reduce* para una clave dada. Este modelo se muestra del todo ineficiente, por lo que el desarrollador tiene la opción de definir una función *combiner.* Esta función le permite fusionar, parcialmente, los datos antes de ser enviados a través de la red. La función *combiner* se ejecuta en cada nodo donde se procesa una tarea *map*. Típicamente, se utiliza el mismo código para implementar la función *combiner* y la función *reduce*. La diferencia radica en el modo en que MapReduce gestiona la salida de la función. La salida de la función *reduce* se escribe en el sistema de ficheros de forma permanente. La salida de la función *combiner* se escribe en un fichero intermedio que será enviado a una tarea *reduce*.

1.4. Hadoop

Hadoop es un proyecto de la *Apache Software Fundation* [25] que aglutina diferentes subproyectos, donde se desarrolla software de código abierto. Proporciona un framework, escrito en Java, sobre el cual desarrollar aplicaciones distribuidas que requieren un uso intensivo de datos y de alta escalabilidad.

Se presenta como una solución para los programadores sin experiencia en desarrollo de aplicaciones para entornos distribuidos, dado que oculta la implementación de detalles propios de estos sistemas: paralelización de tareas, administración de procesos, balanceo de carga y tolerancia a fallos.

Hadoop está inspirado en las publicaciones de Google sobre el modelo de programación MapReduce [10] y sobre su sistema de ficheros distribuido denominado GFS (Google File System) [12]. Por tanto, Hadoop implementa, entre otras cosas, el paradigma MapReduce y un sistema de ficheros distribuido denominado HDFS (Hadoop Distributed File System).

1.4.1. HDFS

HDFS es un sistema de ficheros pensado para almacenar grandes cantidades de información, del orden de terabytes o petabytes, tolerante a fallos y diseñado para ser instalado en máquinas de bajo coste. La información es dividida en bloques, que son almacenados y replicados en los discos locales de los nodos del clúster. Tiene muchas similitudes con otros sistemas de ficheros distribuidos, pero es diferente en varios aspectos. Una diferencia notable es, que está pensado para aplicaciones que siguen un modelo de una sola escritura y muchas lecturas, permitiendo relajar los requisitos de control de concurrencia, simplificando la coherencia de los datos, proporcionando como consecuencia un acceso de alto rendimiento. Otra cualidad única de HDFS es que parte de la suposición que, por lo general, es mejor ubicar la lógica de procesamiento cerca de los datos en lugar de mover los datos al espacio de aplicación.

HDFS se compone de un grupo de nodos interconectados, donde residen los archivos y directorios. Presenta una arquitectura *master/worker* basada en un único nodo maestro, denominado NameNode, que maneja el espacio de nombres del sistema y regula el acceso de los clientes a los ficheros, redirigiéndolos a los nodos de datos que contienen la información, denominados DataNodes, que son los encargados de gestionar el almacenamiento en los discos locales del propio nodo. La Figura 1.8 muestra de forma gráfica la arquitectura HDFS.

Tanto el NameNode como los DataNodes son componentes software, diseñados para funcionar, de manera desacoplada, en máquinas genéricas a través de sistemas operativos heterogéneos. HDFS ha sido construido utilizando el lenguaje de programación Java, por lo tanto, cualquier máquina que

soporte dicho lenguaje puede ejecutar HDFS. Una instalación típica consta de una máquina dedicada, donde se ejecutará el NameNode, y en cada una de las máquinas restantes que constituyen el clúster, se ejecutará un DataNode.

Una aplicación cliente, que desea leer un fichero en HDFS, debe contactar primero con el NameNode, para determinar en lugar en el cual está almacenada la información que requiere. En respuesta al cliente, el NameNode retorna el identificador del bloque más relevante y el nodo en el cual está almacenado. A continuación, el cliente contacta con el DataNode para recuperar la información requerida. Los bloques se encuentran almacenados en el sistema de ficheros local de la máquina, y el HDFS se encuentra en la parte superior de la pila del sistema operativo estándar (por ejemplo Linux). Una característica importante del diseño de este sistema de ficheros es, que la información nunca se mueve al NameNode. Toda la transferencia de información se produce directamente entre los clientes y los nodos de datos. La comunicación con el NameNode sólo implica transferencia de meta-información.

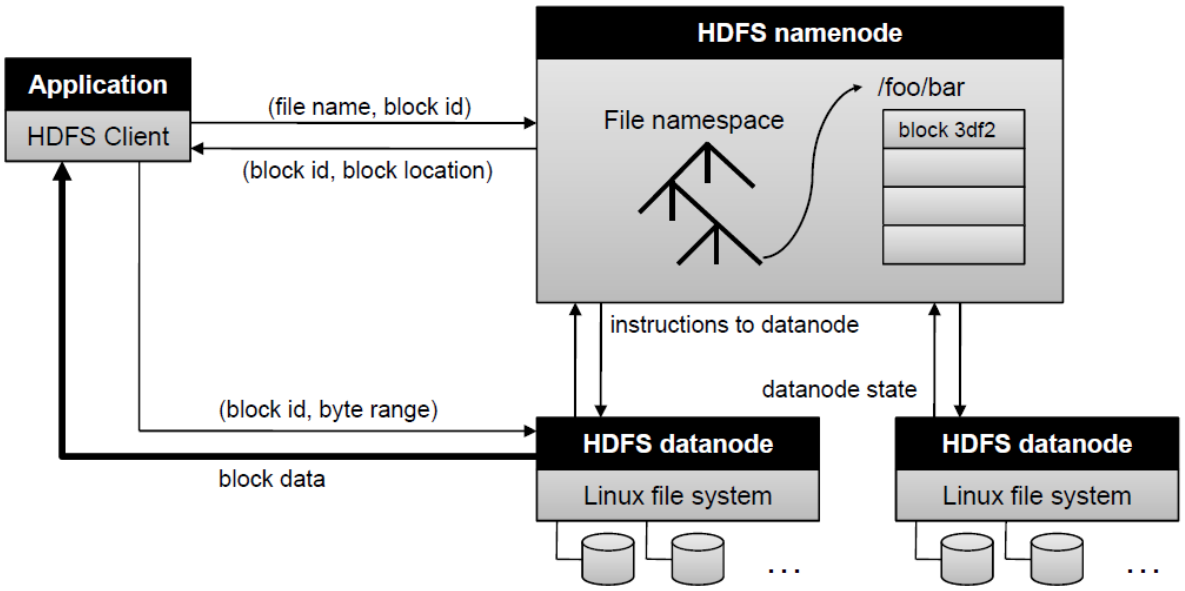


Figura 1.8. Esquema general arquitectura HDFS.

Los DataNodes están periódicamente facilitando información de estado al NameNode, dado que este último no puede conectarse directamente con el primero. El Namenode simplemente se limita a responder a las peticiones realizadas por el DataNode, el cual mantiene un servidor de sockets abierto mediante el cual un cliente u otro DataNode puede leer o escribir información. El NameNode conoce el host o puerto para este servidor, proporcionando dicha información a los clientes interesados en contactar con un nodo de datos concreto.

HDFS presenta una estructura jerárquica de ficheros tradicional, en la cual un usuario puede crear directorios y almacenar ficheros bajo ellos. La jerarquía del espacio de nombres del sistema de

ficheros es similar a otros sistemas de ficheros existentes, permitiendo al usuario cambiar el nombre, la ubicación o eliminar ficheros.

Para que el sistema de ficheros sea tolerante a fallos, HDFS replica los bloques de ficheros. Una aplicación puede especificar el número de réplicas para un fichero en el momento que es creado, pudiendo ser cambiado en cualquier momento. El NameNode toma las decisiones relativas a la replicación de bloques.

Uno de los objetivos principales de HDFS es dar soporte a ficheros de gran tamaño. El tamaño típico de un bloque de fichero en HDFS es 64Mb o 128Mb. Un fichero está compuesto de uno o varios bloques de 64/128Mb, y HDFS trata de colocar cada bloque en nodos de datos separados, distribuyendo la información a lo largo del clúster.

Los bloques no siempre pueden ser colocados de manera uniforme en los nodos de datos, lo que significa que el espacio disponible por uno o más nodos de datos puede estar infrautilizado. Otro caso común, que provoca que la distribución de los datos entre los diferentes nodos no esté balanceada, es la adición de nodos de datos al clúster. HDFS proporciona rebalanceo de bloques de datos utilizando diferentes modelos. Un modelo permite mover los bloques de un nodo de datos a otro, de forma automática, si el espacio libre en un nodo cae demasiado. Otro modelo permite crear, dinámicamente, réplicas adicionales para un determinado fichero, si se produce un aumento repentino de la demanda, rebalanceando otros bloques en el clúster. HDFS también proporciona comandos que permite realizar tareas de reajuste de forma manual.

Existe un elemento adicional denominado Secondary NameNode, cuyo objetivo es realizar periódicamente puntos de control (checkpoints) sobre los cambios que se van realizando en el sistema de ficheros. El uso de este elemento ha sido marcado como obsoleto justo cuando se estaba finalizando el proyecto, proponiéndose otros sistemas para realizar puntos de control [26]. Se menciona este elemento dado que aparece en algunas de las ilustraciones que se muestran en los siguientes apartados.

1.4.2. Hadoop MapReduce

Hadoop proporciona un entorno de ejecución orientado a aplicaciones desarrolladas bajo el modelo de programación MapReduce. Bajo este modelo, la ejecución de una aplicación presenta dos etapas:

- Map: donde se realiza la ingestión y la transformación de los datos de entrada, en la cual los registros de entrada pueden ser son procesados en paralelo.
- Reduce: fase de agregación o resumen, donde todos los registros asociados entre sí deben ser procesados juntos por una misma entidad.

La idea principal sobre la cual gira el entorno de ejecución Hadoop MapReduce es, que la entrada puede ser dividida en fragmentos y, cada fragmento, puede ser tratado de forma independiente por una tarea *map*. Los resultados de procesar cada fragmento, pueden ser físicamente divididos en grupos distintos. Cada grupo se ordena y se pasa a una tarea *reduce*.

Una tarea *map* puede ejecutarse en cualquier nodo de cómputo del clúster, y múltiples tareas *map* pueden ejecutarse en paralelo en el clúster. La tarea *map* es responsable de transformar los registros de entrada en duplas <clave, valor>. La salida de todos los *map* se dividirá en particiones, y cada partición será ordenada por clave. Habrá una partición por cada tarea *reduce*. Tanto la clave como los valores asociados a la clave son procesados por la tarea *reduce*, siendo posible que varias tareas *reduce* se ejecuten en paralelo.

En la Figura 1.9, se muestra de forma esquemática el ciclo de ejecución de una aplicación en Hadoop. El desarrollador únicamente deberá proporcionar al framework Hadoop cuatro funciones (entre paréntesis se indica la correspondencia en la Figura 1.9): la función que lee los registros de entrada y los transforma en duplas <clave, valor> (RecordReader), la función map (Mapper), la función reduce (Reducer), y la función que transforma las duplas <clave, valor> generadas por la función reduce en registros de salida (RecordWriter).

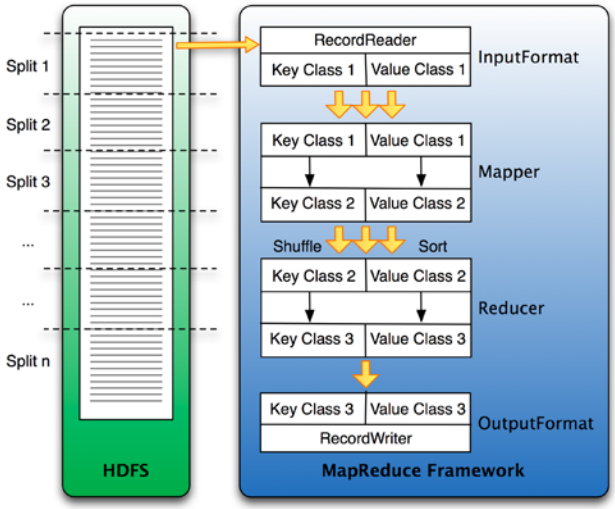


Figura 1.9. Modelo MapReduce en Hadoop.

El entorno de ejecución Hadoop Mapreduce está formado por dos componentes principales: JobTracker y TaskTracker, ambos codificados en Java. Al igual que en el HDFS, el entorno de ejecución presenta una arquitectura cliente/servidor. En un clúster hay un único JobTracker, siendo su labor principal la gestión de los TaskTrackers, entre los que distribuye los trabajos MapReduce que recibe. Los TaskTrackers son los encargados de ejecutar las tareas *map*/*reduce*. En un clúster típico,

se ejecuta un TaskTracker por nodo de cómputo. En la Figura 1.10 se muestra de forma esquemática la interacción entre JobTracker y TaskTracker.

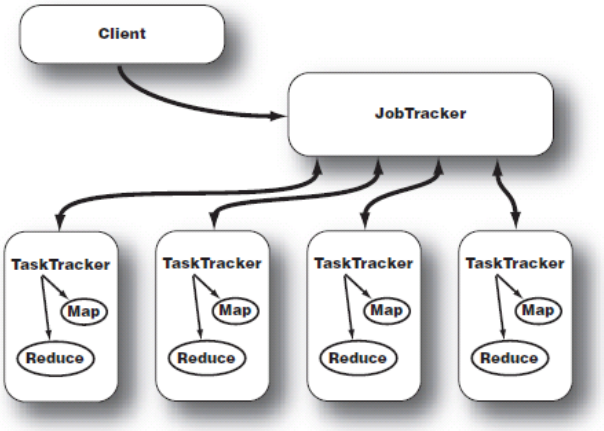


Figura 1.10. Interacción entre JobTracker y TaskTracker.

El JobTracker es el enlace entre la aplicación y Hadoop. Una vez se envía un trabajo al clúster, el JobTracker determina el plan de ejecución en base a los ficheros a procesar, asigna nodos de cómputo a las diferentes tareas, y supervisa todas las tareas que se están ejecutando. En el caso de fallar una tarea, el JobTracker relanza la tarea, posiblemente en un nodo diferente, existiendo un límite predefinido de intentos en el caso que dicha tarea falle de forma reiterada. Como se ha comentado anteriormente, sólo se ejecuta un JobTracker por clúster Hadoop.

Cada TaskTracker es responsable de ejecutar las tareas que el JobTracker le ha asignado. Cada TaskTracker puede ejecutar varias tareas *map*/*reduce* en paralelo. El JobTracker crea instancias Java separadas para la ejecución de cada tarea. De esa manera se garantiza que, en caso de fallar la ejecución de una tarea, no afecta al resto de tareas ni tampoco al propio JobTracker. En la Figura 1.11 se muestra el esquema de ejecución de aplicaciones en un entorno Hadoop.

El TaskTracker se comunica con el JobTracker a través de un protocolo de latidos (heartbeat protocol). En esencia, el latido es un mecanismo que utiliza el TaskTracker para anunciar su disponibilidad en el clúster. El protocolo de latido es lo que permite saber al JobTracker que el TaskTracker está vivo. Además de anunciar su disponibilidad, el protocolo de latidos también incluye información sobre el estado del TaskTracker. Los mensajes de latido indican si el TaskTracker está listo para la siguiente tarea o no. Cuando el JobTracker recibe un mensaje de latido de un TaskTracker, declarando que está listo para la siguiente tarea, el JobTracker selecciona el siguiente trabajo disponible en la lista de prioridades y determine qué tarea es la más apropiada para el TaskTracker.

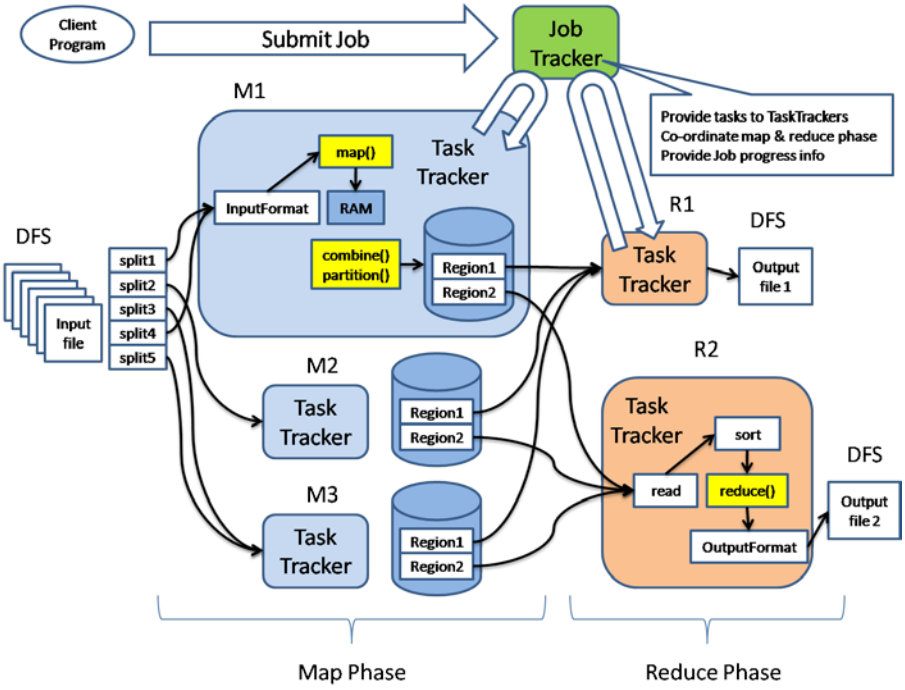


Figura 1.11. Esquema simplificado de ejecución de aplicaciones en Hadoop.

Los TaskTrackers están limitados por el número de tareas que pueden ejecutar. Por ejemplo, un TaskTracker puede ser configurado para procesar dos tareas *map* y dos tareas *reduce* de forma paralela en un momento dado. Por esta razón, el JobTracker tiene que averiguar qué tipo de tarea debe asignar al TaskTracker. Si tiene disponibilidad para ejecutar tarea map, se le asigna una tarea *map*, de lo contrario se le asigna una tarea *reduce*. El número de tareas que un TaskTracker puede ejecutar al mismo tiempo, depende de la cantidad de núcleos y de la memoria disponible en el nodo donde se está ejecutando.

Hadoop trata de ser eficiente en el tratamiento de las tareas, considerando la localidad de los datos de la tarea a procesar. A cada tarea *map* de un trabajo Hadoop, se le asigna una parte de los datos de entrada, pudiendo estar ubicados dichos datos en cualquier parte del HDFS del clúster. El planificador, inicialmente intentará que una tarea se ejecute en el nodo que contienen los datos a nivel local. Cuando los datos a procesar están almacenados de manera local en el nodo donde se ejecuta la tarea, el TaskTracker se libera de tener que descargar los datos necesarios de un nodo remoto. La localidad de los datos asegura un mejor rendimiento y eficiencia. Cuando una tarea no puede ser asignada a un nodo que contiene los datos a nivel local, Hadoop intentará asignar la tarea al nodo más cercano a los datos. En este contexto, el nodo más cercano sería un nodo en el mismo *rack* donde se almacenan los datos. Por último, si no se puede asignar la tarea a un nodo del mismo *rack*, entonces la alternativa es encontrar un nodo en otro *rack*.

El número de tareas map a ejecutar depende de varios factores: el número de tareas map especificado por el programador sirve como referencia para el entorno de ejecución, pero el número de tareas que finalmente ejecutará tendrá en cuenta tanto el número de ficheros a procesar como el

número de bloques HDFS que ocupan dichos ficheros. En cambio, el número de tareas reduce es igual al número de tareas especificado por el programador.

El JobTracker mantiene una cola de trabajos enviados para su ejecución en el clúster y realiza un seguimiento del progreso de cada trabajo. Los clientes pueden configurar los trabajos en Hadoop con un nivel de prioridad que representa la importancia del trabajo respecto a otros trabajos en la cola.

Por defecto, FIFO es la política de planificación que se utiliza en Hadoop para priorizar los trabajos de la cola. Dado que este tipo de planificación puede no ser muy eficiente en entornos productivos, a partir de la versión 0.19 de Hadoop, el planificador se sacó fuera de núcleo para, de esta manera, permitir el uso de un planificador alternativo en el caso que fuera necesario. Dos de los más extendidos son Fair Scheduler y Capacity Scheduler.

Fair Scheduler [27] fue desarrollado por Facebook. El objetivo del planificador es, proporcionar una rápida respuesta a trabajos pequeños y calidad de servicio (QoS) para trabajos de producción. Se basa en tres conceptos básicos: los trabajos se agrupan en pools, cada pool tiene asignada una porción del clúster mínima garantizada y el exceso de capacidad se distribuyen entre los trabajos. Los trabajos que están sin categorizar van a un pool por defecto.

Capacity Scheduler [28] fue desarrollado por Yahoo y presenta algunas similitudes con Fair Scheduler. Existen una serie de colas al estilo de Fair Scheduler, que pueden estar organizadas de forma jerárquica y cada cola tiene una capacidad asignada. Cada cola está planificada siguiendo una política FIFO con prioridades. Este tipo de planificador permite a los usuarios simular un clúster MapReduce separado, con una planificación FIFO en cada uno.

En la Figura 1.12 se muestra el esquema detallado de ejecución de una aplicación desarrollada bajo el paradigma MapReduce. Como se puede observar, además de las fases *map* y *reduce*, existen otras fases a tener en cuenta a la hora de ejecutar una aplicación en Hadoop.

A continuación, se describe cada una de las fases que presenta cada proceso. Todo y que la mayoría de fases son funciones internas del propio framework de ejecución, tienen asociados una serie de parámetros de configuración que pueden afectar al rendimiento del propio clúster. La configuración de dichos parámetros puede ser realizada por el propio programador o por el administrador del sistema.

Realizar la sintonización de los parámetros de configuración no es una tarea sencilla, dado que Hadoop presenta más de 165 parámetros ajustables. Además, un único parámetro puede tener efectos importantes en el rendimiento global del sistema. Por ese motivo, en la descripción que se realiza de cada una de las fases, se indican los parámetros asociados. De esta manera es posible formarse una idea de la dificultad que entraña la sintonización del sistema.

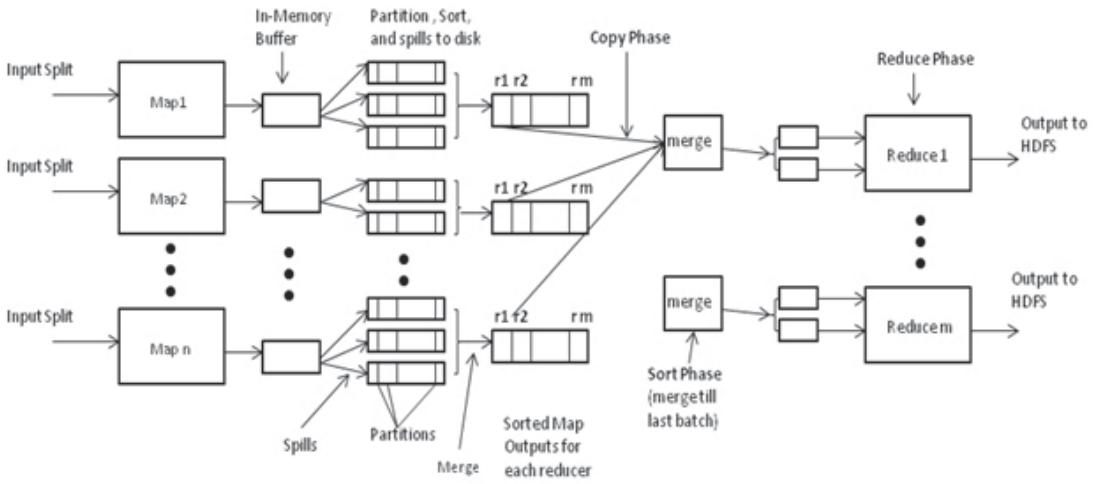


Figura 1.12. Esquema detallado de ejecución de aplicaciones en Hadoop.

- Tarea *map*. Presenta las siguientes fases:

  - **Procesamiento map**: a cada tarea *map* (Map 1, Map 2, …, Map n) se le asigna un bloque del fichero (Input Split). Cada bloque es dividido en duplas del tipo <clave_registro, registro>. La función *map* es invocada para cada dupla. La información generada por la función *map* es escrita en un buffer de memoria circular asociado con cada tarea *map* (In Memory Buffer). El tamaño del buffer viene fijado por la propiedad *io.sort.mb*, siendo el tamaño por defecto de 100Mb.
  - **Spill**: cuando el buffer llega al porcentaje de ocupación fijado por la propiedad *io.sort.spill.percent* (por defecto 0,80 que corresponde a un 80%), un proceso ejecutado en segundo plano realiza el volcado del contenido del buffer a disco (Partitions). Mientras el volcado tiene lugar, la tarea *map* sigue escribiendo en el buffer mientras éste tenga espacio libre. El volcado se realiza siguiendo una política *round-robin* en un subdirectorio específico creado para el trabajo en ejecución. Este subdirectorio reside bajo el directorio especificado en la propiedad *mapred.local.dir*. Se crea un nuevo fichero cada vez que el buffer alcanza el porcentaje de ocupación fijado. A estos ficheros se les denomina ficheros de *spill*.
  - **Particionamiento**: antes de escribir a disco, un proceso en segundo plano divide la información en tantas particiones (r1, r2, …, rm) como tareas *reduce* se han programado para el trabajo en curso.
  - **Sorting**: Se realiza en memoria una ordenación por clave de cada partición. En el caso que la aplicación hubiera definido una función *combiner*, ésta procesará la salida generada por la ordenación. El propósito de la función *combiner* es reducir el volumen de datos a entregar a la fase Reduce agrupando los registros que comparten la misma clave.
  - **Merge:** antes de que la tarea *map* finalice, los diferentes ficheros de *spill* generados son intercalados generándose un fichero de salida ordenado. La propiedad *io.sort.factor* controla el máximo número de ficheros a fusionar a la vez, que por defecto es 10.

- Tarea *reduce*: Los ficheros particionados generados en la fase *map* son proporcionados a las tareas *reduce* (Reduce 1, …, Reduce m) vía HTTP. El número de hilos usados para servir dichos ficheros viene fijado por la propiedad *tasktracker.http.threads* y por defecto está fijado a 40 hilos para cada TaskTracker. Presenta las siguientes fases:

    - **Copy**: Tan pronto como las tareas map finalizan la información correspondiente a cada tarea *reduce* es copiada. La fase *reduce* tiene un número de hilos que realizan dichas copias. Dicho número está fijado por la propiedad *mapred.reduce.parallel.copies* (por defecto 5). La información de salida que genera el *map* es copiada a un buffer del TaskTracker controlado por la propiedad *mapred.job.shuffle.input.buffer.percent* (por defecto 0,70 que corresponde al 70%) que indica la proporción de memoria *heap* utilizada para tal propósito. Cuando dicho buffer llega al porcentaje de ocupación indicado por la propiedad *mapred.job.shuffle.merge.percent* (por defecto 0,66 o 66%), o se ha llegado al número máximo de ficheros de salida de tareas *map* escritos en el búfer, fijado por la propiedad *mapred.job.shuffle.merge.percent* (por defecto 1.000), son fusionados y volcados a disco. Dado que las copias se acumulan en disco, un proceso en segundo plano los fusiona y ordena en ficheros más grandes ahorrando tiempo en subsiguientes fusiones.
    - **Merge**: Cuando todos los ficheros de salida de las tareas *map* han sido copiados, son intercalados manteniendo la ordenación con la que se han generado en el proceso *map*. Este proceso se realiza en varias iteraciones, cuyo número viene dado por el resultado de dividir el número de ficheros *map* por la propiedad *io.sort.factor* (por defecto 10). Por ejemplo, si se generaron 40 ficheros en la fase *map*, se realizarán cuatro iteraciones. En la primera se intercalarán 10 ficheros en uno sólo y así sucesivamente. Al finalizar se habrán obtenido cuatro ficheros que se pasan directamente a la fase *reduce*.
    - **Reduce**: Se invoca a la función *reduce* por cada clave extraída del fichero generado en la fase Merge. La salida de esta fase puede ser grabada directamente en el HDFS, o bien a través de una función auxiliar para grabar la información con un formato determinado. Se genera un fichero en el HDFS por tarea *reduce* ejecutada.

1.5. Virtualización

La virtualización es la introducción de una capa abstracción, dispuesta en una máquina física, que permite la emulación de múltiples instancias o máquinas sobre un mismo hardware, denominadas máquinas virtuales. La virtualización no es algo nuevo, es una tecnología que lleva bastante tiempo en desarrollo y en uso, si bien es cierto, que los avances en los desarrollos, la evolución del hardware

y la necesidad de una rapidez en despliegue de aplicaciones junto con un abaratamiento en los costes, han potenciado mucho el uso de este tipo de tecnologías.

Esta tecnología permite la separación del hardware y el software, lo cual posibilita a su vez que múltiples sistemas operativos, aplicaciones o plataformas de cómputo se ejecuten simultáneamente en una sola máquina física según sea el caso de aplicación (Figura 1.13).

Figura 1.13. Diferentes sistemas operativos sobre una misma máquina física.

Hay varias formas de ver o catalogar la virtualización, pero en general se trata de uno de estos dos casos: virtualización de plataforma o virtualización de recursos:

- **Virtualización de plataforma**: se trata de simular una máquina real con todos sus componentes y prestarle todos los recursos necesarios para su funcionamiento. En general, hay un software anfitrión, que es el que controla que las diferentes máquinas virtuales sean atendidas correctamente, y que está ubicado entre el hardware y las máquinas virtuales. Dentro de este esquema caben la mayoría de las formas de virtualización más conocidas, incluidas la virtualización de sistemas operativos, la virtualización de aplicaciones y la emulación de sistemas operativos.
- **Virtualización de recursos**: permite agrupar varios dispositivos para que sean vistos como uno solo, o al revés, dividir un recurso en múltiples recursos independientes. Generalmente se aplica a medios de almacenamiento. También existe una forma de virtualización de recursos muy popular que no es sino las redes privadas virtuales o VPN, abstracción que permite a un PC conectarse a una red corporativa a través de la Internet como si estuviera en la misma sede física de la compañía.

En los siguientes apartados se comentan algunos de los tipos de virtualización más extendidos.

1.5.1. Emulación

Una aplicación simula el hardware completo, permitiendo la ejecución de sistemas operativos sin necesidad de ser modificado.

La ejecución se hace bajo el control de un emulador que simula el sistema completo, incluyendo la ejecución de las instrucciones a nivel de CPU (Figura 1.14). El emulador simula la ejecución de código binario para una CPU concreta, en un sistema real que usa un procesador y un juego de instrucciones diferente al del sistema emulado.

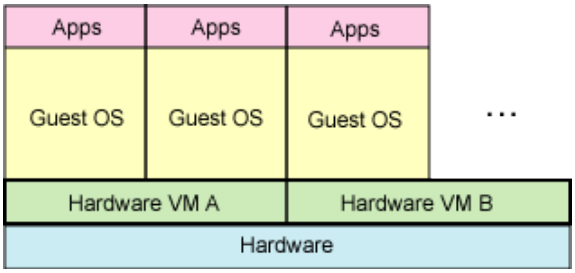


Figura 1.14. Modelo de virtualización mediante emulación.

Este tipo de virtualización es la más costosa y la menos eficiente, ya que obliga a simular completamente el comportamiento de la plataforma hardware a emular, e implica también que cada instrucción que se ejecute en estas plataformas sea traducida al hardware real.

Sin embargo, la emulación tiene características interesantes, como poder ejecutar un sistema operativo diseñado para una plataforma concreta sobre otra plataforma, sin tener que modificarlo, o en el desarrollo de firmware para dispositivos hardware, donde se pueden comenzar estos desarrollos sin tener que esperar a tener disponible el hardware real.

Uno de los ejemplos más destacados de la actualidad es QEMU[29]. QEMU permite, entre otras cosas, emular diferentes plataformas hardware como x86, x86-64, PowerPC, SPARC o MIPS. Otros ejemplos de virtualización mediante emulación son Bochs[30] o MAME[31].

1.5.2. Virtualización completa

Este tipo de sistemas utiliza una máquina virtual que hace de intermediaria entre el sistema huésped y el hardware real (Figura 1.15). El software de virtualización es conocido generalmente como monitor de máquina virtual (VMM, Virtual Machine Monitor) o hipervisor (hypervisor).

En este tipo de sistemas, el hipervisor se encarga de emular un sistema completo y analiza dinámicamente el código que el sistema huésped quiere ejecutar, reemplazando las instrucciones críticas (las que hace falta virtualizar) por nuevas secuencias de instrucciones que tienen el efecto deseado en el hardware virtual.

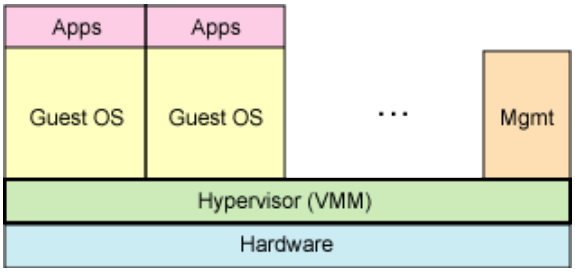


Figura 1.15. Modelo de virtualización completa.

Su principal ventaja reside en que los sistemas operativos pueden ejecutarse sin ninguna modificación sobre la plataforma, aunque como inconveniente frente a la emulación, el sistema operativo debe estar soportado en la arquitectura virtualizada.

En lo que respecta al rendimiento, éste es significativamente mayor que en la emulación, pero menor que en una plataforma nativa, debido a la monitorización y la mediación del hipervisor. Sin embargo, recientes incorporaciones técnicas en las plataformas x86 hechas por Intel y AMD, como son Intel VT[32] y AMD-V[33], han permitido que soluciones basadas en la virtualización completa se acerquen prácticamente al rendimiento nativo. La virtualización que hace uso de estas técnicas, se conoce como virtualización completa asistida por hardware.

Ejemplos de este tipo de virtualización son VMware[34], VirtualBox[35] o KVM[36].

1.5.3. Paravirtualización.

En este tipo de sistemas, sólo se ejecuta un núcleo (sistema anfitrión) y, este núcleo, crea entornos de ejecución que las aplicaciones ven como máquinas virtuales (Figura 1.16).

La paravirtualización surgió como una forma de mejorar la eficiencia de las máquinas virtuales y acercarlo al rendimiento nativo. Se basa en que los sistemas virtualizados (huesped) deben estar basados en sistemas operativos, especialmente modificados, para ejecutarse sobre un hipervisor. De esta forma, no es necesario que el hipervisor monitorice todas las instrucciones, sino que los sistemas operativos huesped y anfitrión colaboran en dicha tarea.

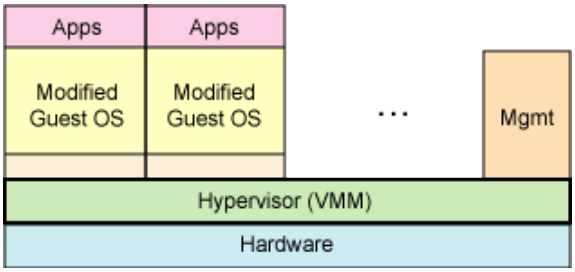


Figura 1.16. Modelo de paravirtualización.

Este es uno de los modelos de virtualización más económicos, dado que no necesita apoyo hardware ni hace falta supervisar el código a bajo nivel, pero tiene el inconveniente que sólo permite ejecutar entornos virtuales para la misma CPU y sistema operativo. Además, dado que sólo hay un núcleo, por lo que si este núcleo tiene un problema, todas las máquinas virtuales se ven afectadas. Otro problema que presenta este modelo es la necesidad de modificar el sistema operativo del huésped. No siempre se puede disponer del código fuente, por ejemplo, por tratarse de sistemas propietarios.

Ejemplos de este tipo de virtualización son XEN[37] o UML (User-Mode Linux)[38].

1.6. Objetivos

Como se ha comentado en el apartado 1.1, existen diversos paradigmas desde los cuales paralelizar un problema para ejecutarlo en un entorno HPC. No hay un paradigma mejor que otro sino que todos son válidos. Dependiendo del tipo de tarea a realizar, un paradigma se adaptará mejor que otro. Por tanto, es cada vez más habitual la necesidad de ejecutar en un entorno HPC aplicaciones programadas en diferentes paradigmas.

Por tanto, se trata de hacer convivir aplicaciones bajo diferentes modelos de programación paralela (por ejemplo MapReduce, OpenMP o MPI) sin necesidad de reservar una parte sustancial del clúster para un modelo concreto de forma estática, disponiendo de esta manera de toda la potencia de cálculo que brinda el clúster para cualquiera de los modelos, si el trabajo a ejecutar así lo requiriera.

En consecuencia, un primer objetivo del proyecto estriba en analizar soluciones que permitan la generación de un clúster Hadoop de forma dinámica, utilizando planificadores batch estándar como PBS, Torque o SGE. Por tanto, se trata de herramientas que ante la necesidad de ejecutar una aplicación programada bajo el paradigma MapReduce, puedan crear e inicializar un clúster Hadoop sobre los recursos que el planificador le hubiera asignado, ejecutar la aplicación correspondiente y liberar los recursos una vez finalizada la ejecución de la aplicación.

Las herramientas a analizar son HoD y myHadoop. De este primer análisis, se debe obtener el modo en que se deben instalar y configurar dichas herramientas. También se debe extraer conclusiones

acerca de las diferentes posibilidades que brindan, en cuanto a entornos a disponer para la ejecución de aplicaciones DIC.

Un segundo objetivo del proyecto es medir las prestaciones de los clúster Hadoop generados de forma dinámica respecto a un clúster Hadoop dedicado. Determinar las diferentes alternativas que brindan las herramientas seleccionadas, para poder adaptar el clúster en función del tipo de trabajo MapReduce a procesar, debe formar parte de este segundo objetivo.

1.7. Metodología

Para abordar el presente proyecto, se ha optado por un enfoque metodológico basado en el modelo en espiral. Se trata de un modelo de ciclo de vida donde las actividades de este modelo son una espiral, cada bucle es una actividad. Las actividades no están fijadas a priori, sino que las siguientes se eligen en función del análisis de riesgo, comenzando por el bucle interior.
Se ha optado por este enfoque debido al desconocimiento del alumno en los temas relacionados tanto con el paradigma de programación MapReduce como en lo referente al framework Hadoop.

En la Figura 1.17 se muestra la aproximación que se ha seguido para abordar el proyecto. Cada iteración comienza con una reunión con el director del proyecto, en la cual se trazan las líneas básicas a seguir. Se planifica la carga de trabajo a realizar en esa iteración para, posteriormente, comenzar con el análisis de las materias a abordar en dicha iteración. Dado que en este proyecto se están evaluando herramientas, cada iteración tiene asociada una herramienta a evaluar. Por ese motivo se define una etapa de instalación y configuración. Una vez la herramienta está operativa, se procede a realizar la experimentación y se evalúa su rendimiento en base a los resultados obtenidos.

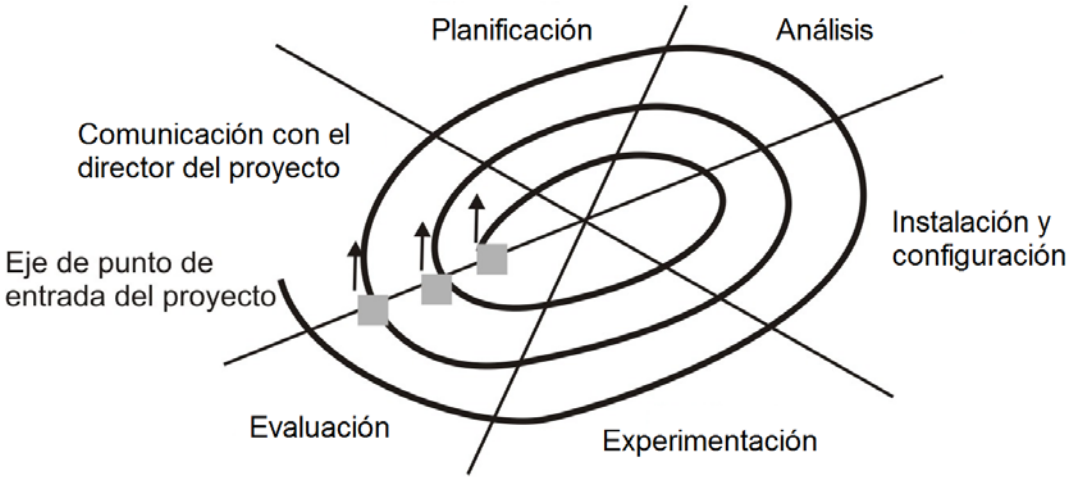


Figura 1.17. Modelo en espiral.

Se realizan las siguientes iteraciones:

- Primera iteración: se realiza el análisis del paradigma de programación MapReduce y del framework Hadoop. Se procede a instalar dicho framework y se realiza la experimentación y la evaluación de los resultados. Finalmente se presentan los resultados en la reunión de seguimiento al director del proyecto.
- Segunda iteración: Se analiza el framework myHadoop, se procede a su instalación y experimentación, evaluando los resultados obtenidos, y se presentan al director del proyecto.
- Tercera iteración: Se analiza el framework HoD repitiendo los pasos comentados en la iteración anterior.
- Cuarta iteración: Una vez analizados todos los frameworks objeto de estudio, y conocidas sus características y comportamiento, se traza un plan de pruebas para realizar una comparativa de rendimiento entre los tres frameworks. Se ejecuta el plan de pruebas y se analizan los resultados, que son la base de las conclusiones a extraer en el presente proyecto.

1.8. Estudio de viabilidad

Para poder desarrollar el proyecto y poder cumplir con los objetivos previstos se requiere de los siguientes elementos:

- Hardware: equipo necesario para implementar el entorno de pruebas sobre el cual se realizará la experimentación.
    - **PCs**: Tanto el alumno como la propia Universidad disponen de equipos sobre los cuales se puede construir un clúster para realizar la experimentación. Además, el Departamento de Arquitectura de Computadores y Sistemas Operativas de la UAB dispone de un clúster IBM de 32 nodos y clúster Dell de 8 nodos.
    - **LAN**: Tanto el alumno como la UAB disponen de la infraestructura necesaria para la interconexión de los diferentes nodos del clúster.
    - **Conexión a internet**: Necesaria tanto para obtener la información necesaria en la fase de análisis como para obtener el software necesario para crear los entornos de prueba. Tanto el alumno como la UAB dispone de conexión a internet.
- Software:
    - **Apache Hadoop**: se trata de un software libre y por tanto no es necesaria la adquisición de licencia.
    - **HoD**: viene incluido en la distribución de Apache Hadoop.

- **myHadoop**: se trata de un software libre.
- **Linux**: Se opta por Ubuntu Linux como sistema operativo a instalar en los nodos del entorno de pruebas por tratarse de un software que se distribuye bajo licencia GNU GPL.
- **Monitorización**: se estima suficiente las herramientas que proporciona Linux y el propio Apache Hadoop para medir el rendimiento de las aplicaciones ejecutadas en la fase de experimentación.
- **Torque**: gestor de recursos necesario para realizar pruebas con HoD y myHadoop. Se trata de un software libre, por tanto no es necesaria la adquisición de licencia.
- **Virtualización**: La primera etapa del proyecto se realizará sobre un entorno virtual por lo que será necesario disponer de un software de virtualización. Oracle VirtualBox es un software libre distribuido bajo licencia GNU GPL. También existe una versión libre del software de virtualización VMware.

A nivel temporal, para las diferentes etapas del proyecto se estima la siguiente carga de trabajo:

- **Análisis (64h)**:
  - Analizar el paradigma MapReduce (8h)
  - Analizar el framework Apache Hadoop (12h)
  - Definir entorno de pruebas (8h)
  - Analizar framework myHadoop (12h).
  - Analizar framework HoD (12h).
  - Definir metodología de pruebas (12h).

- **Instalación y configuración (28h)**:
  - Generación entorno de pruebas (4h).
  - Tres entornos a instalar y configurar: Hadoop, HoD y myHadoop. Se estima 8 horas por cada entorno.

- **Experimentación (104h)**:
  - Pruebas unitarias para testear la instalación y configuración, permitiendo familiarizarse con el entorno. Se estima 8 horas por entorno, lo que hace un total de 24 horas.
  - Ejecución del plan de pruebas en clúster Hadoop dedicado (16h).
  - Ejecución del plan de pruebas en clúster myHadoop (32h).
  - Ejecución del plan de pruebas en clúster HoD (32h).

- **Conclusiones (20h)**: análisis de los resultados obtenidos en la experimentación asociada al plan de pruebas definido. Total horas: 20h

- **Documentación (100h)**: Se debe realizar el informe previo (20h) y la memoria del proyecto (80h). Total horas: 100h.

La fecha de inicio del proyecto es 28/febrero/2011 y la fecha de finalización prevista 17/junio/2011, lo que hacen un total de 16 semanas. La dedicación prevista del alumno es de unas 21 horas semanales (de lunes a jueves 4 horas por día y 5 horas el viernes), lo que da una capacidad bruta de 336 horas (16 semanas * 21 horas/semana).

Las tareas relacionadas anteriormente suman un total de 316 horas. Se deben sumar además, las horas dedicadas a las reuniones de seguimiento con el director del proyecto. Se estima una reunión semanal de una hora, lo que hace un total de 16 horas adicionales. En total, se deben dedicar al proyecto 332 horas, que está en línea con la dedicación prevista por el alumno.

A nivel técnico, todos los elementos propuestos anteriormente son compatibles entre sí, por lo que se estima que el proyecto es técnicamente viable.

A nivel económico, el hecho que no sea necesario realizar inversión alguna para la adquisición ya sea de software o de hardware, hacen lógicamente viable el proyecto en el aspecto económico.

A nivel temporal, dado que la planificación encaja con la disponibilidad del alumno, el proyecto se considera viable.

1.9. Planificación temporal

En la Figura 1.18 se muestra un diagrama de Gantt con la planificación detallada prevista para el proyecto:

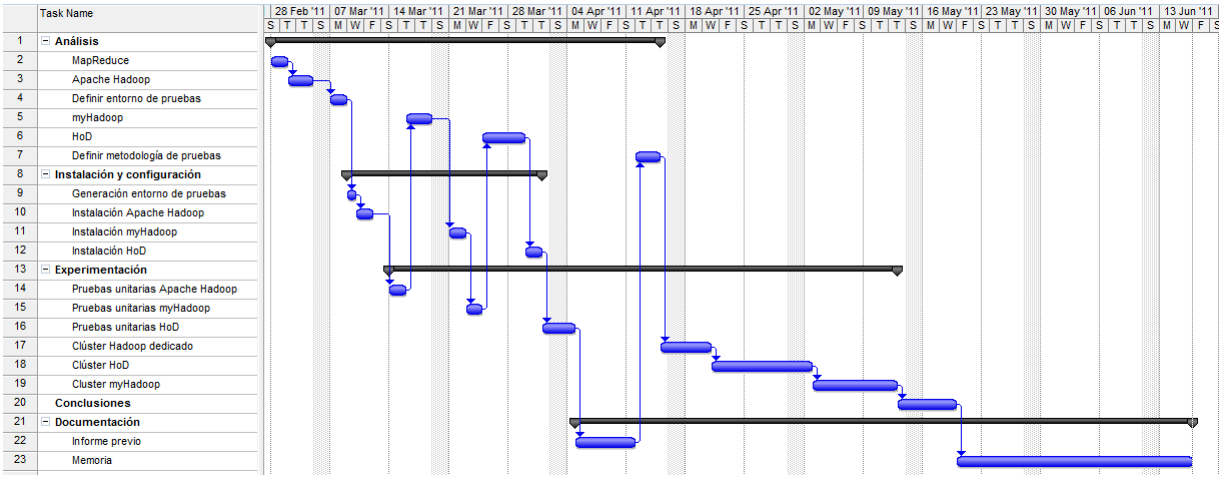


Figura 1.18. Planificación detallada.

A continuación se describen las diferentes fases en las cuales se ha dividido el proyecto (entre paréntesis se indica el número de fila al cual hacen referencia en el diagrama de Gantt):

- Fase I: Análisis:
  - Análisis de la documentación existente para comprender el paradigma de programación MapReduce (2).
  - Análisis del framework Apache Hadoop para familiarizarse con los elementos que la componen (3).
  - Definición del entorno de pruebas sobre el cual se realizará la experimentación (4).
  - Análisis de la documentación asociada al framework myHadoop (5).
  - Análisis de la documentación asociada al framework HoD (6).
  - Definición de la metodología de pruebas donde se definirá las aplicaciones a testear, el volumen que debe procesar cada aplicación y los parámetros a medir (7).

- Fase II: Instalación y configuración.
  - Generación del entorno de pruebas (9)
  - Instalación y configuración del framework Hadoop en el entorno de pruebas (10).
  - Instalación y configuración del framework myHadoop en el entorno de pruebas (11).
  - Instalación y configuración del framework HoD en el entorno de pruebas (12).

- Fase IV: Experimentación:
  - Pruebas unitarias en el clúster creado con el framework Apache Hadoop para familiarizarse con el entorno (14).
  - Pruebas unitarias en el clúster creado con el framework myHadoop para familiarizarse con el entorno (15).
  - Pruebas unitarias en el clúster creado con el framework HoD para familiarizarse con el entorno (16).
  - Ejecución de pruebas en clúster Hadoop y toma de métricas (17).
  - Ejecución de pruebas en clúster HoD y toma de métricas (18).
  - Ejecución de pruebas en myHadoop y toma de métricas (19).

- Fase V: Análisis de métricas y conclusiones:
  - Análisis de las métricas obtenidas en cada una de las fases anteriores y extracción de conclusiones (20)

- Fase VI: Documentación:
  - Informe previo (22)
  - Memoria del proyecto (23)

1.10. Estructura del documento

La presente memoria está organizada en cuatro capítulos, más las referencias y anexos, donde se pretende plasmar el trabajo realizado. El contenido de cada capítulo es el siguiente:

- **Capítulo 1: Introducción**. Se realiza una introducción a la problemática que plantea las aplicaciones intensivas de datos para, posteriormente, dar una visión general de las técnicas que se están utilizando hoy en día para resolver dicha problemática. Se introduce también la base teórica sobre la cual se sustenta el proyecto, donde se habla de los planificadores batch, del paradigma de programación MapReduce, del framework Hadoop y, finalmente, de la virtualización dado que el entorno de pruebas estará formado por máquinas virtuales. Seguidamente se describen los objetivos marcados para este proyecto y se muestra de manera general la metodología a seguir para conseguir los objetivos y la planificación prevista.

- **Capítulo 3: Análisis.** Se realiza un análisis de los frameworks HoD y myHadoop, los cuales permiten generar un clúster Hadoop de forma dinámica sobre un clúster físico. También se analizan los entornos de prueba que, en base a dichos frameworks, se crearán y sobre los cuales se realizará la experimentación.

- **Capítulo 3: Experimentación.** En este capítulo se describen las aplicaciones seleccionadas para realizar las pruebas, la metodología empleada y los resultados obtenidos en base a la ejecución del plan de pruebas trazado..

- **Capítulo 4: Conclusiones**. Se presentan las conclusiones extraídas en base al análisis de los resultados obtenidos, fruto de la experimentación realizada en el capítulo anterior. También se describe la evolución del proyecto, donde se comentan los problemas surgidos y las acciones correctivas realizadas. Se proponen posibles líneas de trabajo y se finaliza valorando el trabajo desde un punto de vista personal.

- **Anexos**. En los apéndices se recoge, tanto información pormenorizada de los resultados obtenidos en las diferentes pruebas realizadas, como información detallada respecto a la preparación de cada uno de los entornos que se han utilizado en el presente proyecto.

Capítulo 2. ANÁLISIS

Hadoop proporciona una herramienta denominada Hadoop on Demand (HoD)[23] que permite generar y gestionar instancias Hadoop MapReduce y HDFS, en un clúster compartido con otras modalidades de procesamiento paralelo. Requiere de la instalación de un gestor de recursos, como puede ser Torque [18], y es necesario que tenga configurados al menos tres nodos para que sea posible ejecutar HoD.

Aunque Hadoop puede estar preinstalado en cada uno de los nodos del clúster, no es un requisito indispensable ya que es posible pasarle a la herramienta HoD un fichero de instalación de Hadoop. HoD desplegará Hadoop en un directorio temporal y, una vez finalizados los trabajos sobre el clúster generado de forma dinámica, la instalación será eliminada.

Por otro lado, myHadoop [24] es un framework de código abierto que permite configurar un clúster Hadoop bajo demanda en HPC tradicionales a través de planificadores batch estándar (p.e. Torque, PBS o SGE), sin necesidad de disponer de permisos de root. Por tanto, se trata de una herramienta que se ajusta perfectamente a los objetivos del proyecto y por tanto a considerar en los análisis a realizar.

A nivel de sistema de ficheros, tanto HoD como myHadoop permiten trabajar:

- con un HDFS no persistente, esto es, se crea en el proceso de arranque del clúster y es eliminado, por el proceso de liberación de recursos, una vez se finaliza el trabajo con el clúster Hadoop temporal. En este caso, si es necesario salvar los resultados generados por los trabajos ejecutados en el clúster temporal, deberán ser grabados en el sistema de ficheros local antes de ejecutar el proceso de liberación de recursos.

- con un HDFS persistente, creado con anterioridad y que no se destruye cuando el clúster es eliminado, por lo que no es necesario realizar actividades adicionales para salvar los resultados.

En los siguientes apartados, se analiza con más detalle las herramientas HoD y myHadoop, y se proporcionan detalles acerca de la arquitectura y configuración de cada uno de los entornos sobre los

cuales se realiza la experimentación. Cabe señalar que, los entornos de pruebas se crean sobre la base de un clúster de cuatro nodos.

2.1. Hadoop dedicado

En el clúster de cuatro nodos creado para las pruebas, se procede a instalar y configurar Hadoop de tal modo que el entorno de pruebas pasa a convertirse en un clúster Hadoop dedicado. La Figura 2.1, muestra una visión general de la estructura que presenta el clúster una vez finalizada el proceso de instalación y configuración, detallado de forma más amplia en el apéndice C.

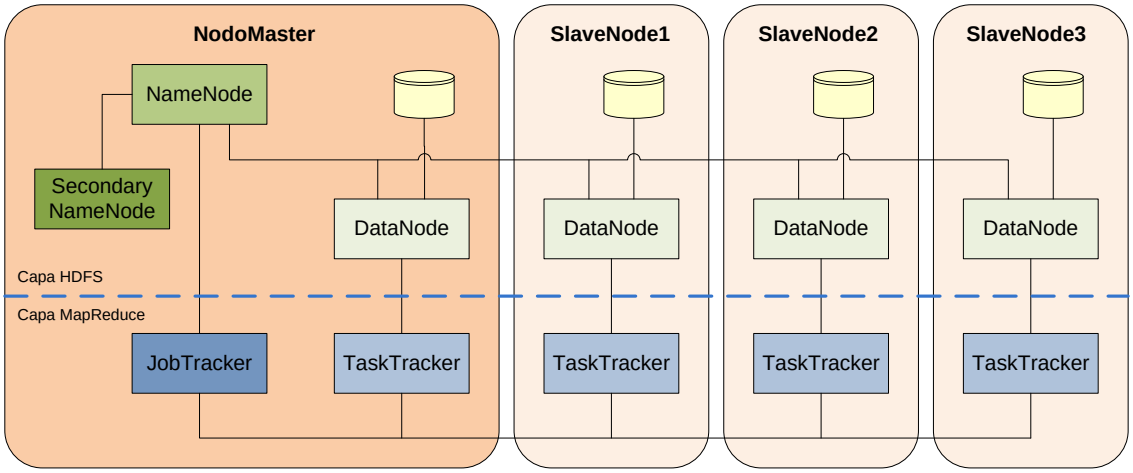


Figura 2.1. Esquema entorno Hadoop dedicado.

El nodo denominado NodoMaster, se configura como nodo maestro, en el cual se ejecutarán el NameNode y el JobTracker. Este nodo también se configura como nodo de cómputo y, por tanto, se ejecutan los demonios TaskTracker y DataNode, al igual que en el resto de nodos del clúster.

Se utiliza la configuración por defecto que el proceso de instalación genera, a excepción del número de réplicas a realizar de cada fichero. Por defecto, el número de réplicas que se realiza de un fichero es tres. El entorno de pruebas se configura para que no se realice réplicas.

Hadoop proporciona una interfaz de usuario web que ofrece sobre información su estado. Por defecto, se puede acceder al puerto 50070 del nodo donde se ejecuta el NameNode para el sistema de ficheros HDFS, y en el puerto 50030 del nodo donde se ejecuta el JobTracker para los trabajos que se hayan enviado y se encuentren ejecutando.

En el caso del clúster de pruebas, la información correspondiente al estado del HDFS se obtiene accediendo a la URL http://nodomaster:50070. En la Figura 2.2 se muestra la información que proporciona la interface web de HDFS, donde se puede obtener por ejemplo, información acerca del

número de nodos y la capacidad de almacenamiento disponible, el total de espacio ocupado o acceso a los ficheros de log del sistema.

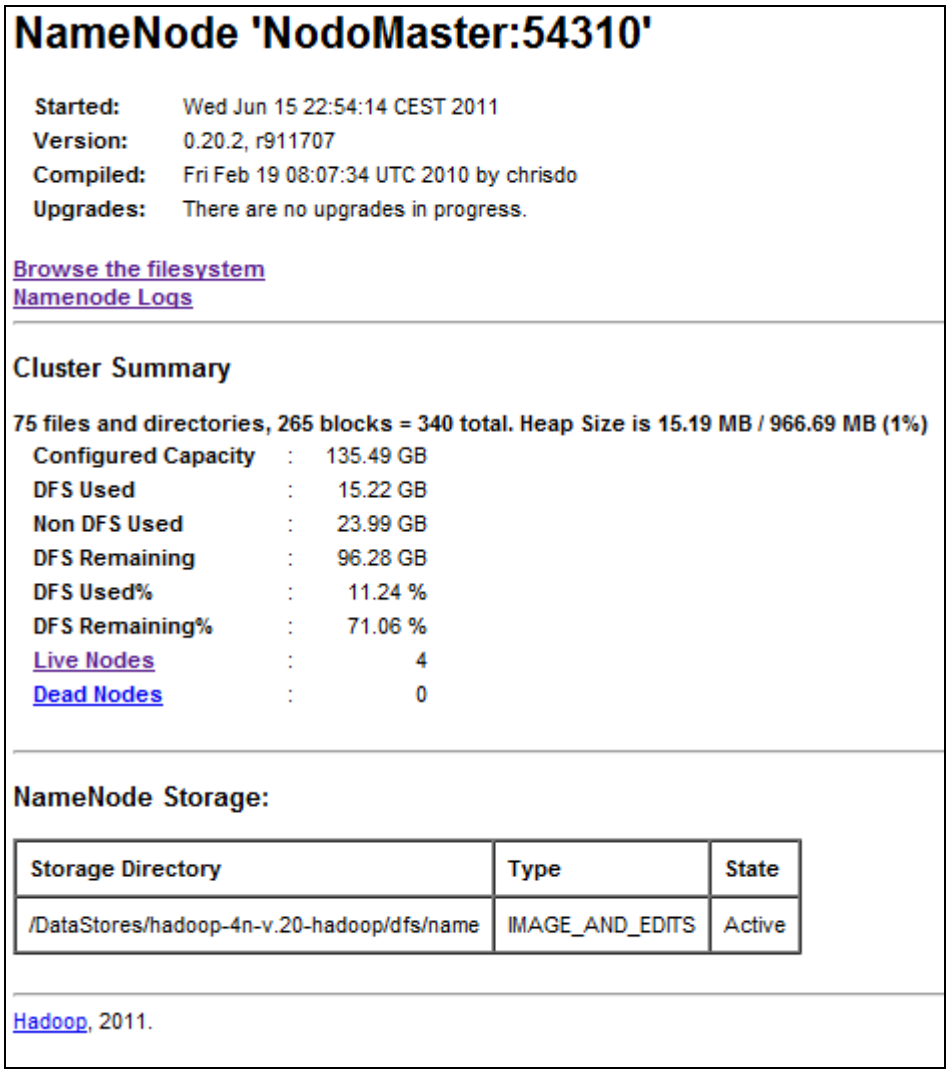
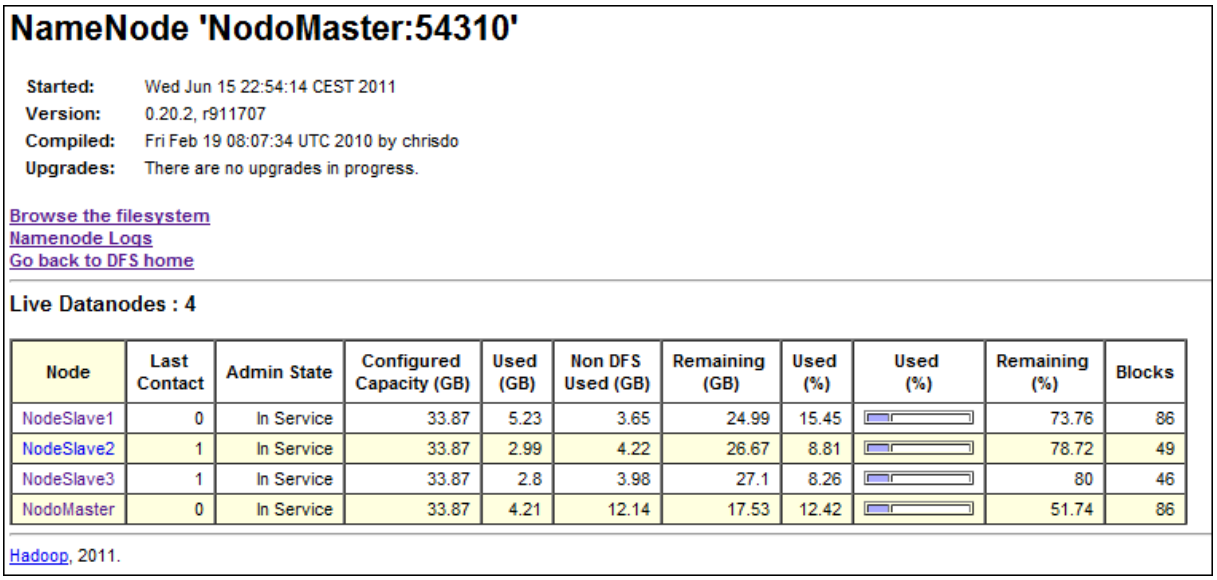


Figura 2.2. Interface web de HDFS.

Desde el enlace asociado a la opción “Live Nodes” de la pantalla mostrada en la Figura 2.2, se muestra información detallada de cada uno de los nodos de almacenamiento. Nombre de la máquina, capacidad total de almacenamiento, espacio en uso o número de bloques almacenados son algunos de los parámetros que se pueden consultar, tal y como se puede observar en la Figura 2.3.



NameNode 'NodoMaster:54310'

Started: Wed Jun 15 22:54:14 CEST 2011
Version: 0.20.2, r911707
Compiled: Fri Feb 19 08:07:34 UTC 2010 by chrisdo
Upgrades: There are no upgrades in progress.

Browse the filesystem
Namenode Logs
Go back to DFS home

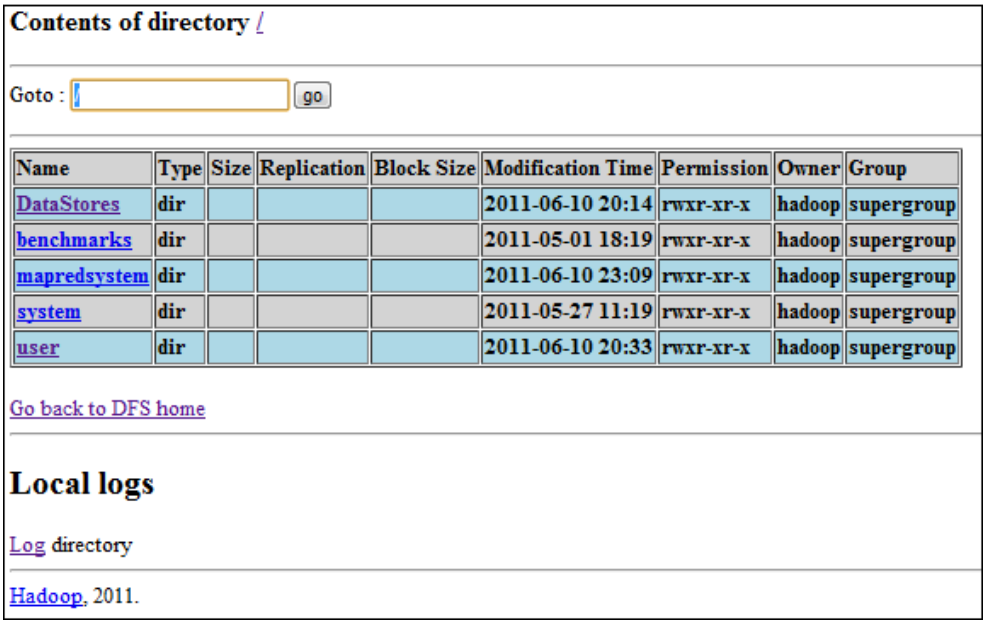
Live Datanodes : 4

Node	Last Contact	Admin State	Configured Capacity (GB)	Used (GB)	Non DFS Used (GB)	Remaining (GB)	Used (%)	Used (%)	Remaining (%)	Blocks
NodeSlave1	0	In Service	33.87	5.23	3.65	24.99	15.45		73.76	86
NodeSlave2	1	In Service	33.87	2.99	4.22	26.67	8.81		78.72	49
NodeSlave3	1	In Service	33.87	2.8	3.98	27.1	8.26		80	46
NodoMaster	0	In Service	33.87	4.21	12.14	17.53	12.42		51.74	86

Hadoop, 2011.

Figura 2.3. Información del estado de los nodos de almacenamiento en Hadoop.

Pulsando sobre el enlace asociado a cualquiera de los nodos visualizados, mostrará una pantalla con el contenido del sistema de ficheros, siendo posible navegar e incluso visualizar el contenido de los ficheros (ver Figura 2.4).



Contents of directory /

Goto : / go

Name	Type	Size	Replication	Block Size	Modification Time	Permission	Owner	Group
DataStores	dir				2011-06-10 20:14	rwxr-xr-x	hadoop	supergroup
benchmarks	dir				2011-05-01 18:19	rwxr-xr-x	hadoop	supergroup
mapredsystem	dir				2011-06-10 23:09	rwxr-xr-x	hadoop	supergroup
system	dir				2011-05-27 11:19	rwxr-xr-x	hadoop	supergroup
user	dir				2011-06-10 20:33	rwxr-xr-x	hadoop	supergroup

Go back to DFS home

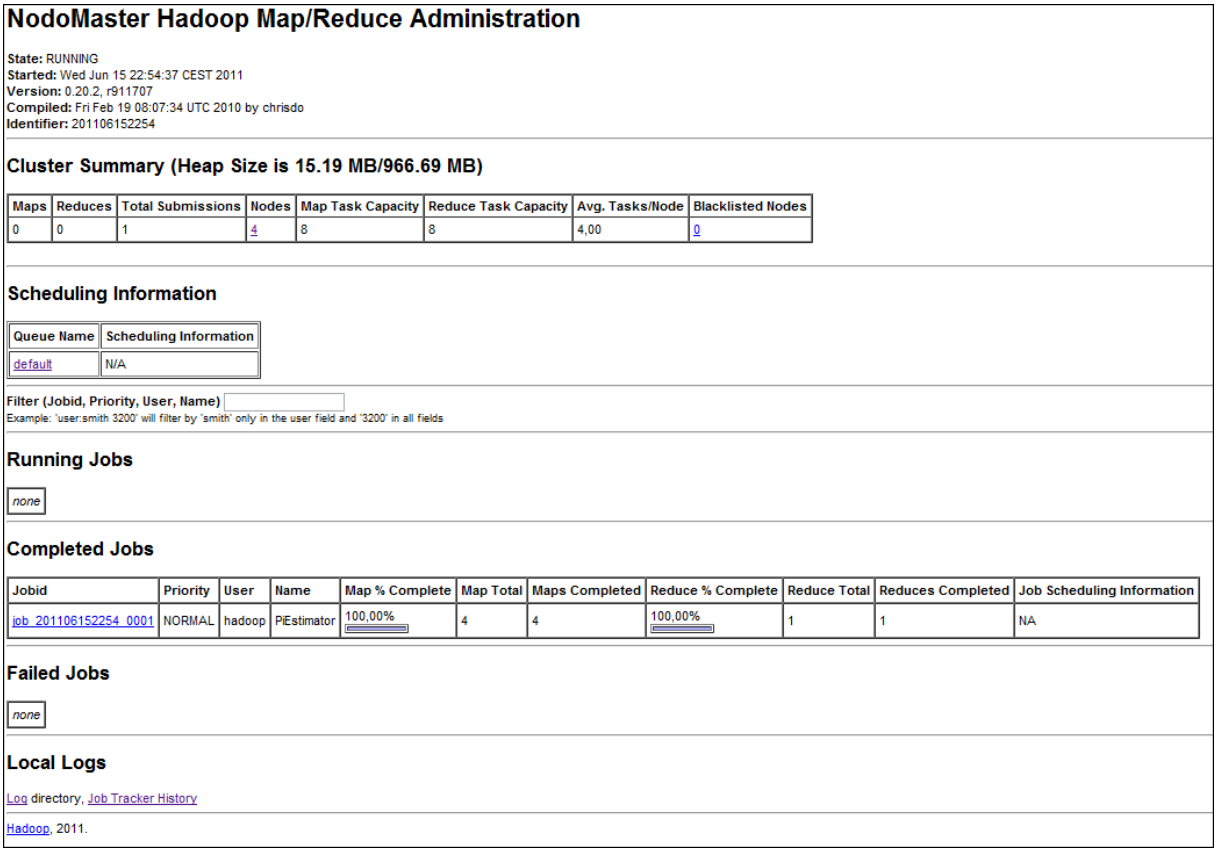
Local logs

Log directory

Hadoop, 2011.

Figura 2.4. Navegación por el sistema de ficheros en Hadoop.

El estado del entorno de ejecución se puede obtener accediendo a la url http://nodomaster:50030. En la página que se muestra en dicha dirección (Figura 2.5), se puede obtener información relativa al número de nodos de cómputo disponibles, el número de tareas map/reduce que puede ejecutar el clúster de forma simultánea, consultar el estado de los trabajos enviados al clúster o los ficheros de log correspondientes al entorno de ejecución.



NodoMaster Hadoop Map/Reduce Administration

State: RUNNING
Started: Wed Jun 15 22:54:37 CEST 2011
Version: 0.20.2, r911707
Compiled: Fri Feb 19 08:07:34 UTC 2010 by chrisdo
Identifier: 201106152254

Cluster Summary (Heap Size is 15.19 MB/966.69 MB)

Maps	Reduces	Total Submissions	Nodes	Map Task Capacity	Reduce Task Capacity	Avg. Tasks/Node	Blacklisted Nodes
0	0	1	4	8	8	4,00	0

Scheduling Information

Queue Name	Scheduling Information
default	N/A

Filter (Jobid, Priority, User, Name)
Example: 'user:smith 3200' will filter by 'smith' only in the user field and '3200' in all fields

Running Jobs

none

Completed Jobs

Jobid	Priority	User	Name	Map % Complete	Map Total	Maps Completed	Reduce % Complete	Reduce Total	Reduces Completed	Job Scheduling Information
job_201106152254_0001	NORMAL	hadoop	PiEstimator	100,00%	4	4	100,00%	1	1	NA

Failed Jobs

none

Local Logs

Log directory, Job Tracker History

Hadoop, 2011.

Figura 2.5. Interface web entorno Hadoop MapReduce.

Cada nodo del clúster está configurado para ejecutar un máximo de dos tareas map/reduce de forma concurrente. Por ese motivo, el clúster puede ejecutar un máximo de 8 tareas map/reduce.

2.2. HoD

Hadoop on Demand (HoD) forma parte del proyecto Apache Hadoop. HoD es un sistema que permite provisionar un clúster Hadoop virtual sobre un clúster físico. Utiliza el gestor de recursos Torque para realizar la asignación de nodos que formarán el clúster virtual. Sobre los nodos asignados, HoD puede iniciar los demonios del entorno de ejecución MapReduce y del sistema de ficheros HDFS. También genera, de forma automática, los ficheros de configuración necesarios tanto para los demonios Hadoop como para los clientes. HoD tiene la capacidad de distribuir Hadoop sobre los nodos del clúster virtual. En resumen, HoD hace que sea fácil para los programadores y los administradores, la preparación de forma ágil un entorno Hadoop. También es una herramienta muy útil para los desarrolladores que necesitan compartir un clúster físico para probar sus propias versiones de Hadoop.

Tanto Hadoop como la herramienta HoD deben estar instalados en aquellos nodos desde los cuales el usuario puede lanzar trabajos al clúster. A través del cliente HoD, el usuario creará un clúster

Hadoop de forma dinámica y, mediante el cliente Hadoop, se enviarán aplicaciones MapReduce al nuevo clúster. Básicamente el proceso se resume en los siguientes pasos:

- El usuario a través del cliente HoD realiza la petición de reserva de un número determinado de nodos del clúster físico para crear un clúster Hadoop sobre ellos.
- El cliente HoD utiliza la interface del gestor de recursos para enviar un trabajo, denominado RingMaster, para solicitar el número de nodos indicado por el usuario. RingMaster se ejecuta en uno de los nodos reservados.
- Los nodos son reservados y RingMaster es ejecutado en uno de esos nodos.
- RingMaster lanza un proceso, denominado HodRing, en cada uno de los nodos reservados.
- Los procesos HodRing se comunican con el RingMaster para obtener los comandos Hadoop a lanzar en cada uno de los nodos. Se trata de lanzar los demonios Hadoop (NameNode, JobTracker, DataNode y TaskTracker) e informar al RingMaster de su estado.
- Los ficheros de configuración necesarios para la ejecución de un clúster Hadoop, son generados por la propia herramienta HoD, o proporcionados por el usuario en el momento de realizar la llamada inicial.
- El cliente HoD mantiene la comunicación con el RingMaster para determinar la localización de los demonios JobTracker y NameNode.
- En este punto el usuario ya está en disposición de enviar trabajos MapReduce al clúster Hadoop creado al vuelo.
- Finalizados los trabajos MapReduce se procede a matar los demonios Hadoop y a liberar los recursos asignados.

HoD necesita un mínimo de tres nodos para crear un clúster Hadoop, dado que en uno de los nodos se ejecutará el NameNode y en otro de los nodos el JobTracker. El resto de nodos serán usados como nodos de cómputo, por tanto, se ejecutarán los demonios DataNode y TaskTracker en cada uno de ellos.

En el momento de crear el clúster Hadoop de forma dinámica, HoD puede utilizar una versión preinstalada de Hadoop en los nodos asignados, o instalar Hadoop, como parte del proceso de inicialización, en base a un fichero de instalación que se le proporciona a la herramienta en el momento de ser invocada. En el caso de utilizar un fichero de instalación, deberá residir en un sistema de ficheros compartido y accesible por todos los nodos. En el momento de redactar la presente memoria HoD solamente soportaba NFS.

En lo que respecta al HDFS, un clúster Hadoop creado mediante HoD puede ser configurado en modo no persistente, por lo que el sistema de ficheros será destruido una vez finalice la ejecución del clúster, o en modo persistente especificando la dirección y el puerto en el que reside el NameNode del HDFS externo.

HoD automáticamente elimina aquellos clústeres que, tras un periodo dado (típicamente 60 minutos, aunque puede ser ajustado por el administrador), no han ejecutado ningún trabajo. Cada clúster creado por HoD consta de un monitor, que constantemente comprueba si se están ejecutando trabajos en el clúster. Si detecta que no hay trabajos en ejecución durante un período determinado, automáticamente desasigna su propio clúster y por lo tanto libera los nodos que no están siendo utilizados.

La Figura 2.6 muestra una visión general del entorno configurado para las pruebas con la herramienta HoD y HDFS no persistente.

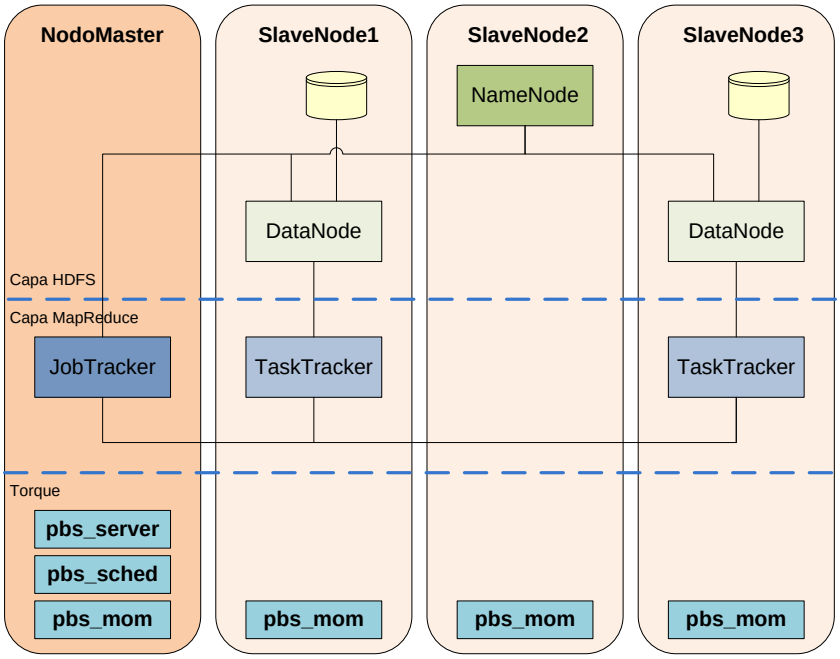


Figura 2.6. Esquema clúster Hadoop generado con HoD.

Como se comentó anteriormente en el apartado 2.2, HoD requiere de Torque para poder configurar un clúster Hadoop de forma dinámica. Por lo tanto, se ha procedido a instalar Torque tal y como se muestra en la Figura 2.6, esto es, en el nodo NodoMaster se instala el servidor batch pbs (pbs_server) y el planificador (pbs_sched). En todos los nodos se instala el mini-servidor pbs de ejecución batch, de esta manera Torque ve a todos los nodos como nodos de cómputo.

La configuración que muestra la Figura 2.6 tanto en la capa HDFS como en la capa MapReduce, es una de las posibles opciones. HoD asigna una máquina para el JobTracker, una máquina para el NameNode y el resto de máquinas son nodos de cómputo, donde se arranca los demonios DataNode y TaskTracker en cada una de ellas. La configuración de clúster que genera HoD es más propia de un clúster de altas prestaciones, donde existe un número elevado de nodos, y por tanto se hace necesario dedicar una máquina para la gestión del HDFS y otra para la gestión de trabajos.

La capacidad de cómputo que presenta el clúster generado por HoD de forma dinámica, es de 4 tareas *map*/*reduce* ejecutándose de forma concurrente en el clúster, es decir, 2 tareas *map*/*reduce* por nodo.

Como se expuso en el apartado 2.2, HoD puede utilizar una versión preinstalada de Hadoop, existente en cada uno de los nodos, o puede instalar Hadoop antes de inicializar el clúster mediante un fichero de instalación que se le proporciona a HoD. Por tanto, para el modo no persistente, en la práctica se crean dos entornos para realizar pruebas, uno con Hadoop presintalado en todos los nodos y otro en el que HoD debe realizar la instalación de Hadoop.

Para realizar las pruebas con HoD y HDFS persistente, se configura el entorno de pruebas tal y como se muestra en la Figura 2.7. Se crea el sistema de ficheros persistente utilizando los cuatro nodos del clúster, siguiendo el modelo comentado para el entorno Hadoop dedicado (apartado 2.1), y se configura HoD para indicarle la máquina en la que se está ejecutando el NameNode.

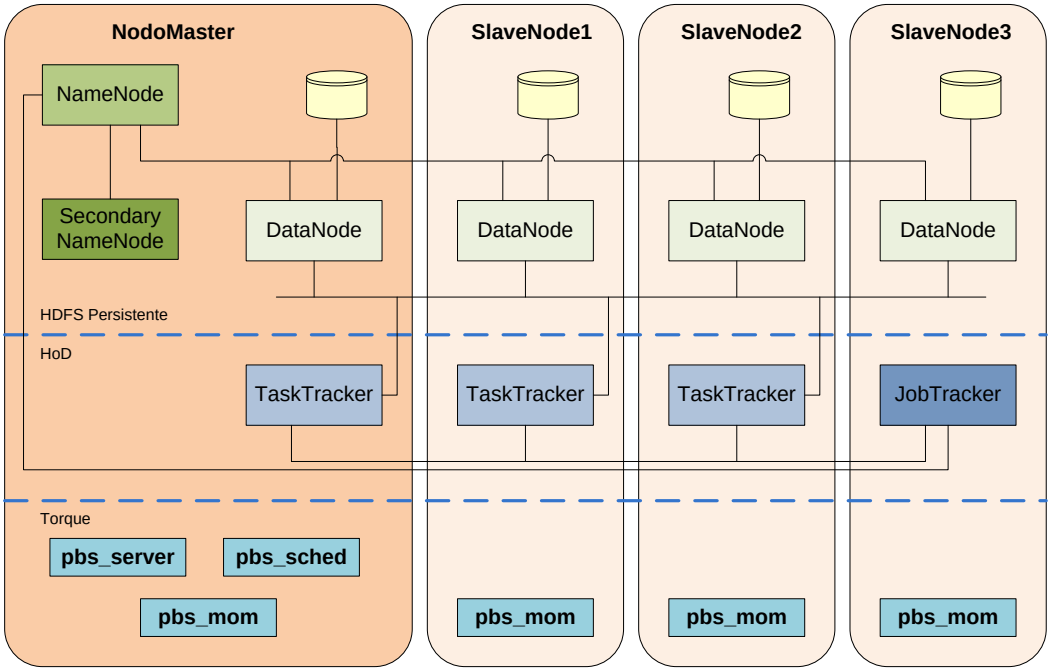


Figura 2.7. Esquema clúster Hadoop generado con HoD con HDFS persistente.

Dado que en el modo persistente HoD no tiene que asignar una máquina para el NameNode, el nodo disponible pasa a ser un nodo de cómputo. Por tanto, en el modo persistente el clúster generado de forma dinámica presenta tres nodos de cómputo, siendo capaz de ejecutar hasta 6 tareas *map*/*reduce* de forma concurrente.

2.3. myHadoop

myHadoop permite configurar un clúster Hadoop bajo demanda sobre un HPC tradicional. En primer lugar myHadoop solicita un conjunto de nodos al sistema de gestión de recursos nativo. Sobre el conjunto de nodos proporcionados designa el nodo maestro y los nodos esclavos, genera los ficheros de configuración necesarios y ejecuta los demonios Hadoop apropiados en cada uno de los nodos. A continuación el usuario puede ejecutar sus trabajos sobre el clúster y una vez finalizados se procede a parar todos los demonios Hadoop y a liberar los recursos asignados.

myHadoop permite que múltiples usuarios ejecuten trabajos Hadoop dentro del mismo HPC. Esto no implica que deban usar la misma instancia Hadoop, sino que garantiza que diferentes configuraciones de Hadoop no interferirán entre sí. Por tanto, cada usuario tiene la posibilidad de configurar su clúster Hadoop en base a sus necesidades, teniendo en cuenta además que no requiere de grandes cambios en la configuración del sistema ni disponer de permisos de root.

Como ya se ha comentado anteriormente, myHadoop puede ser configurado en dos modos: persistente y no persiste. En el caso del modo no persistente se utilizará el almacenamiento local de cada nodo para implementar el HDFS. En el caso del modo persistente, el HDFS estará alojado en un sistema de ficheros compartido como Lustre, GPFS o NFS. La Figura 2.8 muestra una visión general de la arquitectura de myHadoop.

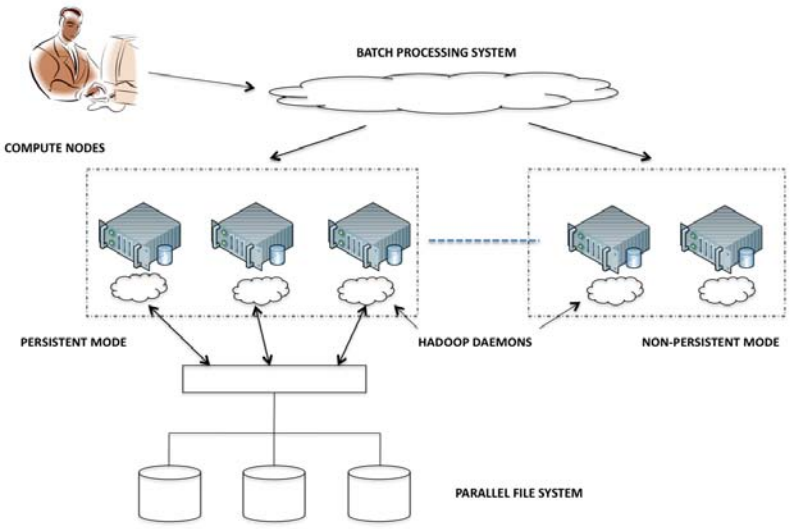


Figura 2.8. Visión general arquitectura myHadoop.

El framework myHadoop requiere que:

- todos los nodos candidatos a ser usados por myHadoop tengan instalado Apache Hadoop versión 0.20.2.

- Torque, PBS o SGE como gestor de recursos del HPC.
- Sistema de ficheros compartido entre los nodos para almacenar la configuración del clúster que se creará bajo demanda.

myHadoop proporciona una serie de scripts que automatizan tareas como: la inicialización y configuración del clúster, la liberación de recursos una vez se han finalizado los trabajos en el clúster dinámico.

myHadoop toma como nodo maestro el primero de los nodos proporcionado por el gestor de recursos. Es en el nodo maestro donde myHadoop procede a ejecutar los demonios NameNode, SecondaryNameNode y JobTracker. En todos los nodos del clúster, incluyendo el nodo maestro, se arrancan los demonios TaskTracker y DataNode. Por lo tanto, todos los nodos del clúster generado de forma dinámica son nodos de cómputo.

La Figura 2.9 muestra una visión general del entorno configurado para las pruebas con la herramienta myHadoop y HDFS no persistente.

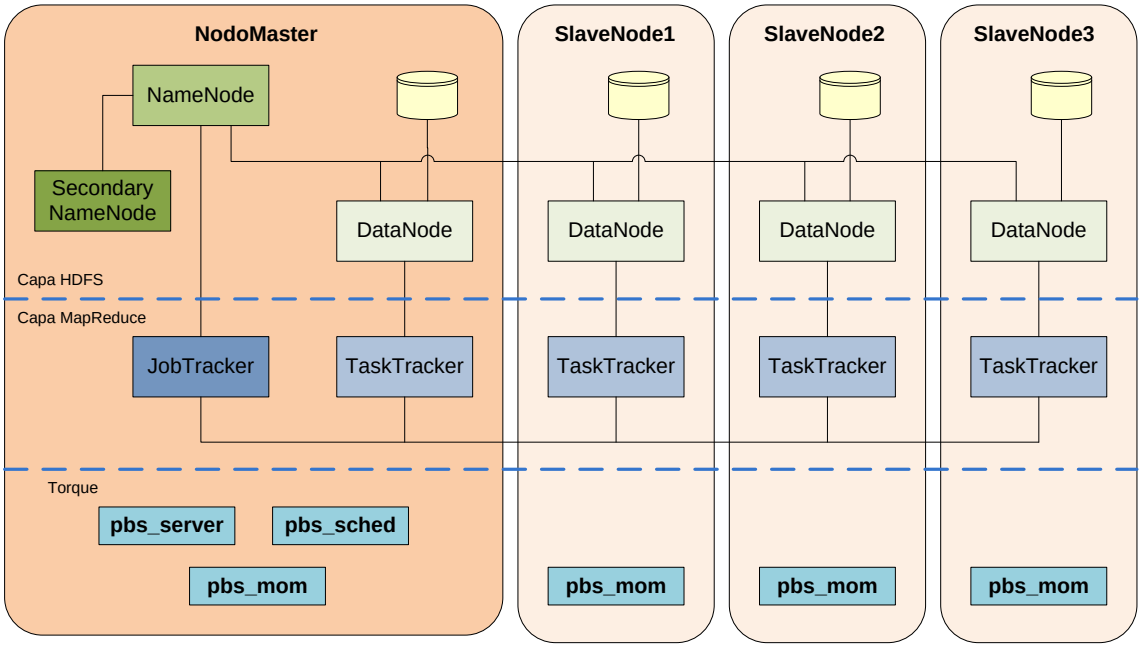


Figura 2.9. Esquema clúster Hadoop generado con myHadoop.

La estructura de clúster Hadoop es la misma que la expuesta para el clúster Hadoop dedicado. La diferencia estriba en la necesidad de disponer de Torque para arrancar los demonios Hadoop en cada uno de los nodos.

En el caso del entorno myHadoop con HDFS persistente, se crea un sistema de ficheros en la máquina NodoMaster y se comparte vía NFS con el resto de nodos del clúster. Bajo el sistema de

ficheros compartido se crea el HDFS. La Figura 2.10 muestra de manera gráfica la configuración del entorno de pruebas.

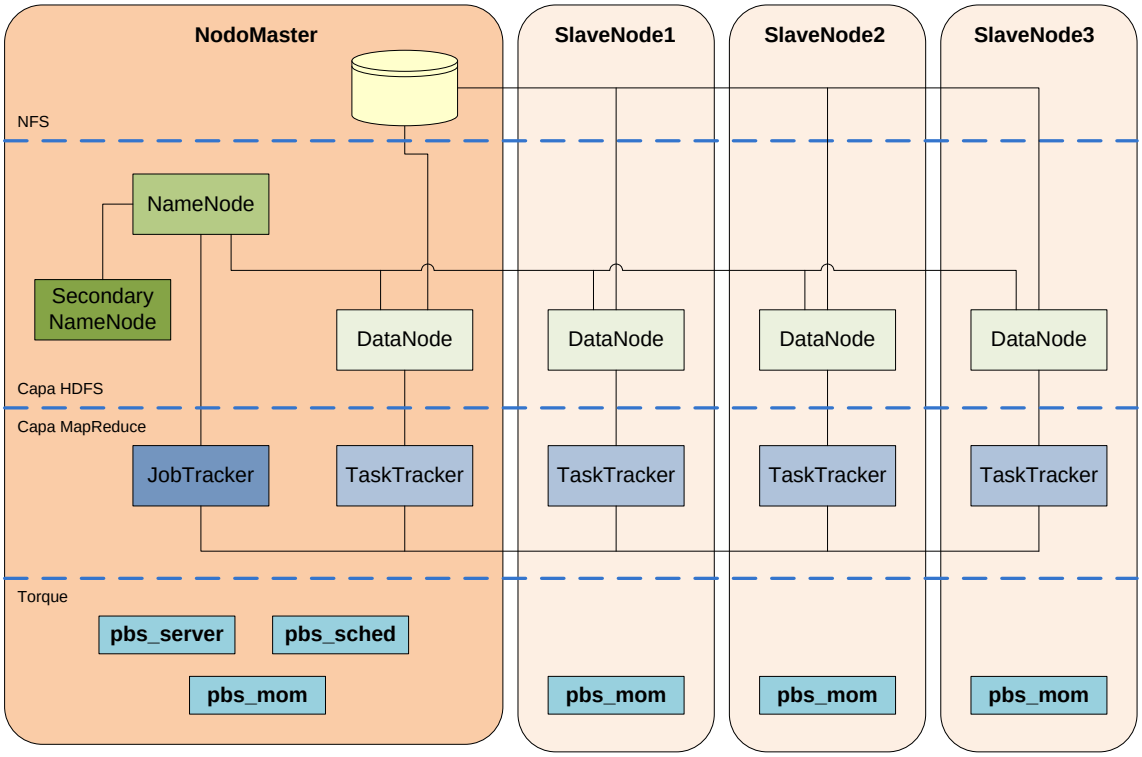


Figura 2.10. Esquema clúster Hadoop generado con myHadoop con HDFS persistente.

Tanto en la versión no persistente como en la persistente, el clúster se configura con la misma capacidad de cómputo que para el clúster Hadoop dedicado, es decir, un máximo de 2 tareas *map/reduce* por nodo ejecutándose concurrentemente y HDFS sin réplicas de bloques.

Capítulo 3. Experimentación

3.1. Entornos

Para realizar las pruebas se opta por crear un clúster de cuatro nodos donde cada nodo corresponde a una máquina virtual (VM). Todas las VM se montarán sobre una única máquina física que presenta las siguientes características:

- Procesador: Intel® Core™ i7 920 2,67Ghz
- Memoria RAM: 6 Gb
- Disco duro: 1 TB SATA/300 7200 RPM
- Sistema operativo: Windows 7 Enterprise 64 bits
- Software de virtualización: Oracle VM VirtualBox 3.2

Cada una de las VM presenta las siguientes características:

- Procesador: 1 CPU a 2046 Mhz
- Memoria RAM: 1 Gb
- Disco duro: 80Gb
- Sistema operativo: Ubuntu Server 10.04.1 LTS

La Figura 3.1 muestra una visión general de la configuración del entorno experimental sobre el cual se realizarán las pruebas del presente proyecto:

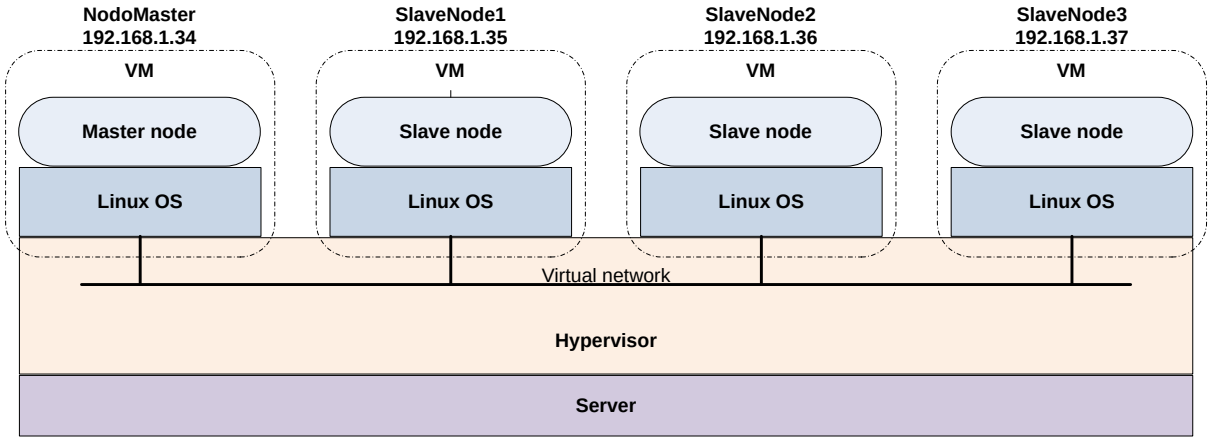


Figura 3.1. Esquema entorno experimental.

Para simplificar el despliegue de Hadoop, se emplea la virtualización dado que ofrece algunas ventajas. Aunque el rendimiento puede no ser del todo ventajoso en este entorno, con el uso de la virtualización es posible crear una instalación de Hadoop y, a continuación, clonarla para crear los demás nodos. Por esta razón, en la Figura 3.1 el clúster Hadoop aparece ejecutándose como máquinas virtuales en el contexto de un hipervisor en un único host.

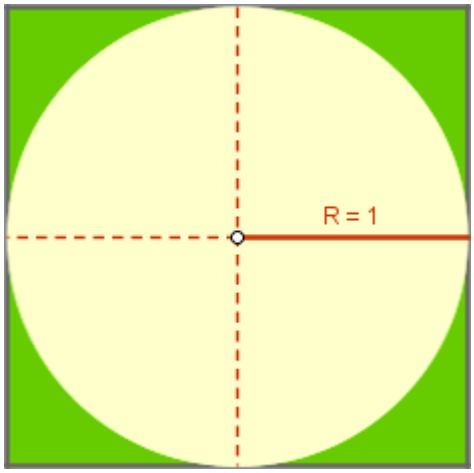
3.2. Aplicaciones

La propia distribución de Hadoop contiene una serie de micro-benchmarks ampliamente usados por la comunidad para evaluar el rendimiento de Hadoop. Alguno de estos benchmarks forman parte de la suite HiBench [39], un conjunto de programas, que incluye tanto micro-benchmarks sintéticos como aplicaciones reales, utilizados para evaluar el rendimiento de un clúster Hadoop en términos de velocidad, rendimiento, ancho de banda del HDFS y utilización de recursos del sistema. Tres de estos benchmarks, Sort (I/O bound), WordCount (I/O bound - CPU bound) y PiEstimator (CPU bound) han sido utilizados en el proyecto para evaluar el rendimiento de un clúster Hadoop bajo demanda frente a un clúster Hadoop dedicado.

3.2.1. PiEstimator

Se trata de un benchmark que implementa el método de Monte Carlo para estimar el valor del número π. Se le llamó así en referencia al Casino de Monte Carlo por ser la capital del juego de azar y por tratarse la ruleta de un generador simple de números aleatorios.

El primer paso consiste en inscribir un círculo de radio R en el interior de un cuadrado de lado 2R tal y como muestra la Figura 3.2:



$As = 4R^2$

$R^2 = As/4$

$Ac = \pi R^2$

$\pi = Ac / R^2$

$R = 1$

$\pi = 4 * Ac / As$

Figura 3.2. Método de Monte Carlo.

El área del cuadrado se calcula mediante la fórmula $As = (2R)^2$ o $4R^2$ y el área del círculo mediante la fórmula $Ac = \pi R^2$. Si R equivale a una unidad, se puede deducir que la relación entre el área del círculo y la del cuadrado es de π/4, siguiendo la manipulación algebraica mostrada en la Figura 3.2.

En base al resultado obtenido anteriormente, se puede inferir que generando N números aleatorios dentro del área del cuadrado, aproximadamente N* π/4 de esos puntos estarán dentro del círculo.
En la aplicación PiEstimator, cada tarea *map* (el número de tareas *map* es proporcionado por el usuario) genera de forma aleatoria, en un cuadrado de lado 2R, el número de puntos especificado como parámetro y cuenta cuantos puntos están dentro del círculo (M). El resultado es transferido a la fase *reduce*. Una única tarea *reduce* recoge los resultados generados por todas las tareas *map* y estima el valor de π en base a la fórmula $\pi = 4 * M'/N'$, donde M' corresponde al sumatorio de puntos generados dentro del círculo y N' corresponde al número de puntos totales generados en la fase *map*.

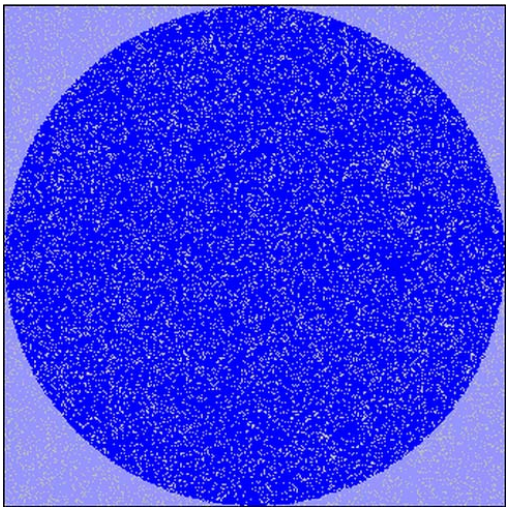


Figura 3.3. Resultado de aplicar el método de Monte Carlo.

La Figura 3.3 muestra de forma gráfica el resultado de aplicar el método de Monte Carlo. Se generan puntos en el interior del cuadrado de forma aleatoria. Una vez se han generado todos los puntos, se procede a contar cuántos han quedado dentro del círculo inscrito en el interior del cuadrado y cuántos fuera para realizar los cálculos comentados anteriormente.

3.2.2. Sort

Este benchmark realiza la ordenación del contenido de un directorio de entrada dejando el resultado en el directorio de salida, ambos especificados como parámetros. La Figura 3.4 muestra el esquema general de ejecución de la aplicación Sort.

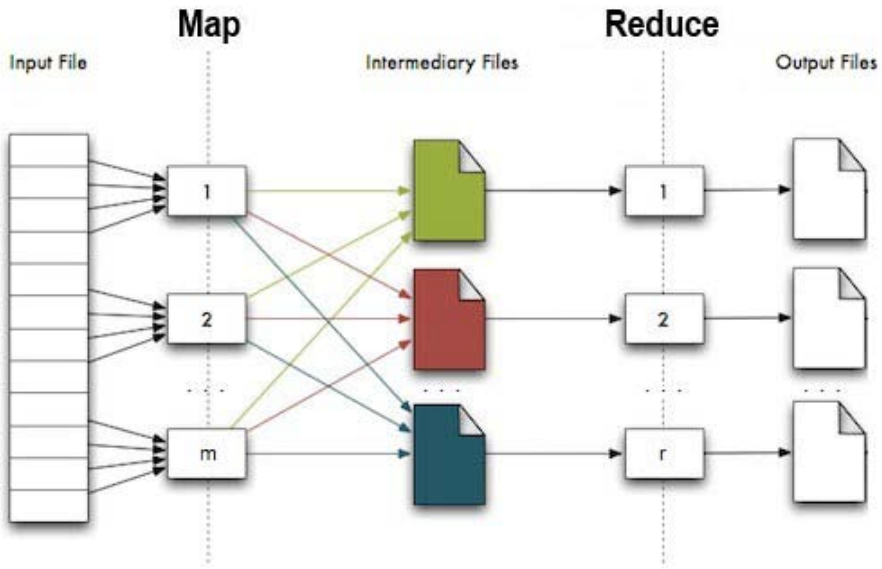


Figura 3.4. Visión general ejecución aplicación Sort.

El número de *maps* viene dado por el número de bloques que presenta el fichero a ordenar. El tamaño de bloque para el HDFS de los diferentes entornos de pruebas está fijado a 67108864 bytes (64 Mb) mediante la propiedad *dfs.block.size*.

El número de *reduce* se calcula en base a la siguiente fórmula:

```
0,90 * (número de nodos) * mapred.tasktracker.reduce.tasks.maximum
```

La propiedad *mapred.tasktracker.reduce.tasks.maximum* viene fijada por defecto a 2 en los diferentes entornos de prueba sobre los cuales se testeará la aplicación.

3.2.3. WordCount

Este benchmark cuenta el número de ocurrencias de una palabra en un fichero dado y genera un fichero con el resultado que contiene duplas del tipo <palabra, número ocurrencias>. La Figura 3.5 muestra el esquema general de ejecución de la aplicación WordCount.

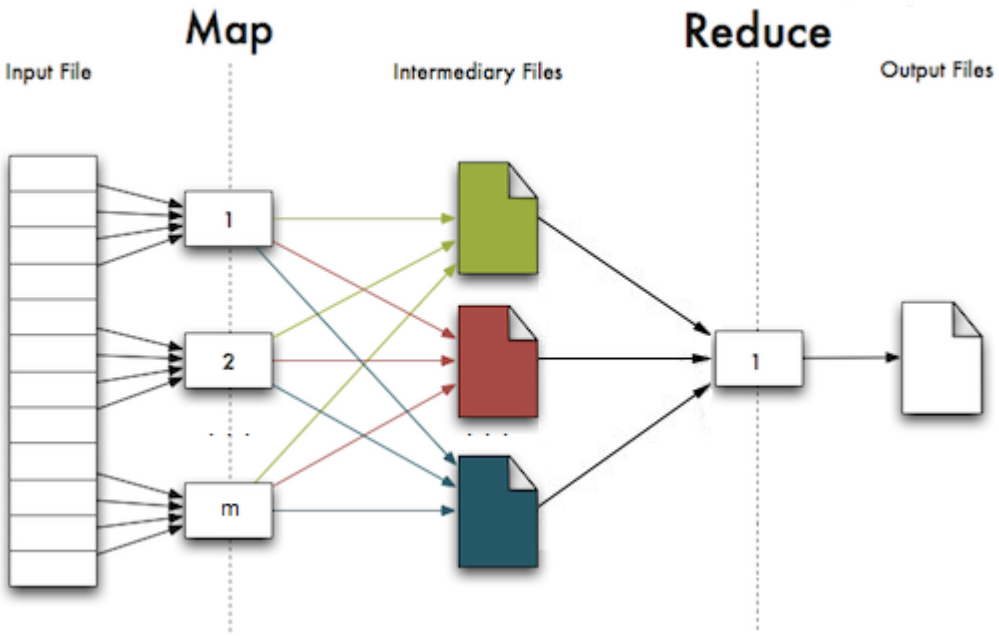


Figura 3.5. Esquema general ejecución WordCount.

El esquema de ejecución es similar al de la aplicación Sort con la diferencia que únicamente se ejecuta una tarea *reduce*:

- A cada tarea *map* se le asigna un bloque del fichero a procesar.
- Cada tarea *map* lee el bloque asignado y realiza una llamada a la función *map* por cada palabra leída, generando una dupla del tipo <clave, 1> donde clave corresponde a la propia palabra.
- Al contrario que la aplicación Sort, la aplicación WordCount tiene definida una función *combiner*. Tras la ordenación del fichero particionado, el fichero resultante es procesado por dicha función agrupando todas las duplas con la misma clave. Como resultado se genera un fichero con duplas del tipo <clave, contador> donde contador corresponde al resultado de contar todas las ocurrencias para la misma clave. De esta manera se reduce el tamaño del fichero a escribir a disco y su vez el volumen de información a transferir a la fase *reduce*.
- Finalmente la fase *reduce* agrupa todas las duplas por clave y suma los contadores asociados obteniéndose una dupla final del tipo <palabra, número ocurrencias> que es volcada al fichero de salida.

3.3. Metodología de pruebas.

Para la ejecución de los test de pruebas, se han desarrollado una serie de scripts que realizan las siguientes tareas:

- Inicialización del clúster, que consiste en realizar la reserva de nodos, arranque de demonios Hadoop, generación y formateo del sistema de ficheros HDFS (sólo entornos no persistentes).
- Copia de ficheros de pruebas del sistema de ficheros local al HDFS del clúster. Esta tarea sólo se realizará en aquellos entornos con HDFS configurado en modo no persistente. En el caso de entornos con HDFS persistente, los ficheros ya estarán disponibles en el propio HDFS.
- Ejecución de la aplicación correspondiente: PiEstimator, Sort y WordCount.
- Copia de los resultados desde el HDFS al sistema de ficheros local. Esta tarea sólo se realizará en aquellos entornos con HDFS no persistente debido a que el sistema de ficheros se destruye.
- Proceso de parada del clúster, que consiste en la eliminación de los demonios Hadoop, eliminación de los ficheros asociados al HDFS, eliminación de logs, y ficheros temporales.

El objetivo es la medición del tiempo requerido por cada una de las tareas comentadas. Para determinar el tiempo que requiere cada tarea para completarse, la ejecución de cada script se precederá del comando *time* de Unix, tomándose el valor retornado en *real* (elapsed time) de la salida que genera dicho comando. A continuación se muestra el formato de script utilizado para realizar los test de pruebas:

```
#!/bin/bash
#Variables entorno
export ...
#Inicio prueba
echo "====> Iniciando cluster"
time ./start_cluster.sh
echo "====> Cluster Inicializado"
echo "====> Copiando ficheros al HDFS"
time ./copy_files.sh
echo "====> Copiados ficheros al HDFS"
echo "====> Ejecutando test"
time hadoop --config ${HADOOP_CONF_DIR} jar
${HADOOP_HOME}/hadoop-*-examples.jar <test> <parameters>
echo "====> test ejecutado"
echo "====> Copiando resultados al filesystem local"
time ./copy_output.sh
echo "====> Copiados resultados al filesystem local "
echo "====> Liberando cluster"
time ./stop_cluster.sh
echo "====> Cluster liberado"
```

En amarillo se ha resaltado la invocación de los scripts de inicialización, copias desde y hacia el HDFS, ejecución de las aplicaciones de prueba, y el proceso de parada del clúster.

Cada una de las aplicaciones se ejecuta con diferentes cargas de trabajo, lo que permite evaluar el rendimiento de cada uno de los entornos en función de dichas cargas y, por tanto, extraer conclusiones al respecto.

La aplicación PiEstimator recibe como parámetros el número de tareas *map* que se desea ejecutar y el número de puntos aleatorios que generada cada *map*. En las pruebas realizadas se ha ejecutado la aplicación con los siguientes parámetros:

- 4 tareas *map* generando cada una cien millones de puntos, obteniéndose un valor estimado de π = 3.1415933.
- 4 tareas *map* generando cada una mil millones de puntos, obteniéndose un valor estimado de π = 3,141592634.
- 4 tareas *map* generando cada una diez mil millones de puntos, obteniéndose un valor estimado de π = 3,1415926651.

La aplicación Sort recibe como parámetros la ruta del directorio donde residen los ficheros a ordenar y la ruta donde se debe almacenar el resultado obtenido. En las pruebas realizadas se han utilizado tres directorios con 1Gb (16 bloques), 2Gb (32 bloques) y 4Gb (64 bloques) de contenido. Los ficheros a ordenar han sido generados mediante la utilidad RandomWriter, proporcionada por el propio framework de Hadoop para generar ficheros de prueba para la aplicación Sort.

La aplicación WordCount recibe como parámetros la ruta del directorio donde residen los ficheros a procesar y la ruta donde se debe almacenar el resultado. Al igual que en la aplicación Sort, se han utilizado tres directorios con un volumen de información a procesar de 0.98Gb (16 bloques), 2,05Gb (34 bloques) y 4,1Gb (68 bloques) de contenido. Los ficheros a procesar han sido generados mediante la utilidad RandomTextWriter, proporcionada por el propio framework de Hadoop para generar ficheros de prueba de la aplicación WordCount.

En resumen, para cada uno de los benchmarks se realizan tres test de pruebas, con diferente carga de trabajo, y cada test se ejecuta 10 veces. Para obtener el valor medio y la desviación estándar de cada test, se eliminan el mayor y el menor tiempo de ejecución, realizándose el cálculo con las ocho ejecuciones restantes.

En el caso de las pruebas a realizar en modo no persistente, tanto con el clúster generado con HoD como con el generado con myHadoop, una vez el clúster se encuentre inicializado y el HDFS operativo, se procederá a copiar los ficheros a procesar por las aplicaciones Sort y WordCount del sistema de ficheros local del nodo NodoMaster al HDFS del clúster inicializado. Esos ficheros habrán

sido extraídos del clúster Hadoop y copiados al sistema de ficheros local antes de comenzar las pruebas. De esa manera, se garantiza que cada aplicación realiza las pruebas con los mismos ficheros en cada uno de los entornos a evaluar.

3.4. Resultados obtenidos

En el presente apartado, se muestran los resultados obtenidos con cada test de pruebas en cada uno de los entornos creados para el presente proyecto.

Las pruebas se han realizado sobre los entornos de experimentación explicados en el Capítulo 2: clúster Hadoop dedicado, clúster generado mediante el framework myHadoop y clúster generado mediante el framework HoD. Tal y como se comentaba en dicho capítulo, myHadoop y HoD presentaban algunas alternativas a la hora de inicializar un clúster, alternativas que fueron analizadas en el Capítulo 2 dado que también era objetivo del proyecto.

A modo de resumen, los entornos sobre los cuales se han realizado pruebas son (entre paréntesis se indica la etiqueta que se utiliza en las diferentes gráficas para identificar el entorno):

- Clúster Hadoop dedicado (**Hadoop**).

- Clúster generado mediante myHadoop, para el cual se han analizado dos modos:
  - Clúster generado con myHadoop y HDFS no persistente (**myHadoop**).
  - Clúster generado con myHadoop y HDFS persistente (**myHPersist**).

- Clúster generado mediante HoD, para el cual se han analizado tres modos:
  - Clúster generado con HoD, HDFS no persistente y Hadoop preinstalado en cada uno de los nodos (**HoD**).
  - Clúster generado con HoD, HDFS no persistente e instalación de Hadoop como parte de proceso de inicialización (**HoDTar**).
  - Clúster generado con HoD, HDFS persistente y Hadoop preinstalado en cada uno de los nodos (**HoDPersist**).

Por tanto, se ha realizado la experimentación sobre seis entornos: un clúster Hadoop dedicado, dos clústeres Hadoop generados mediante el framework myHadoop y tres clústeres generados mediante el framework HoD. En todos los casos, el clúster sobre el cual se han realizado las pruebas estaba compuesto de cuatro nodos.

Antes de comenzar con el análisis, señalar que en el anexo A se dispone de información detallada de los resultados obtenidos de cada una de las pruebas realizadas. Esto es debido al gran volumen de información que han generado las pruebas. Cabe recordar que para las pruebas se han utilizado las aplicaciones Sort, WordCount y PiEstimator. Cada aplicación se ha testeado tres veces con diferente tipo de carga y cada test estaba compuesto por 10 ejecuciones. Esto hace un total de:

*(6 entornos) *(3 aplicaciones) *(3 volúmenes de carga por aplicación) *(10 ejecuciones por volumen de carga) = 540 pruebas.*

La Figura 3.6 muestra un gráfico que muestra el tiempo de ejecución obtenido para cada uno de los test en cada uno de los entornos comentados anteriormente. Señalar que los resultados del entorno Hadoop dedicado se utilizan para tener una referencia con la cual comparar el resto de entornos de pruebas.

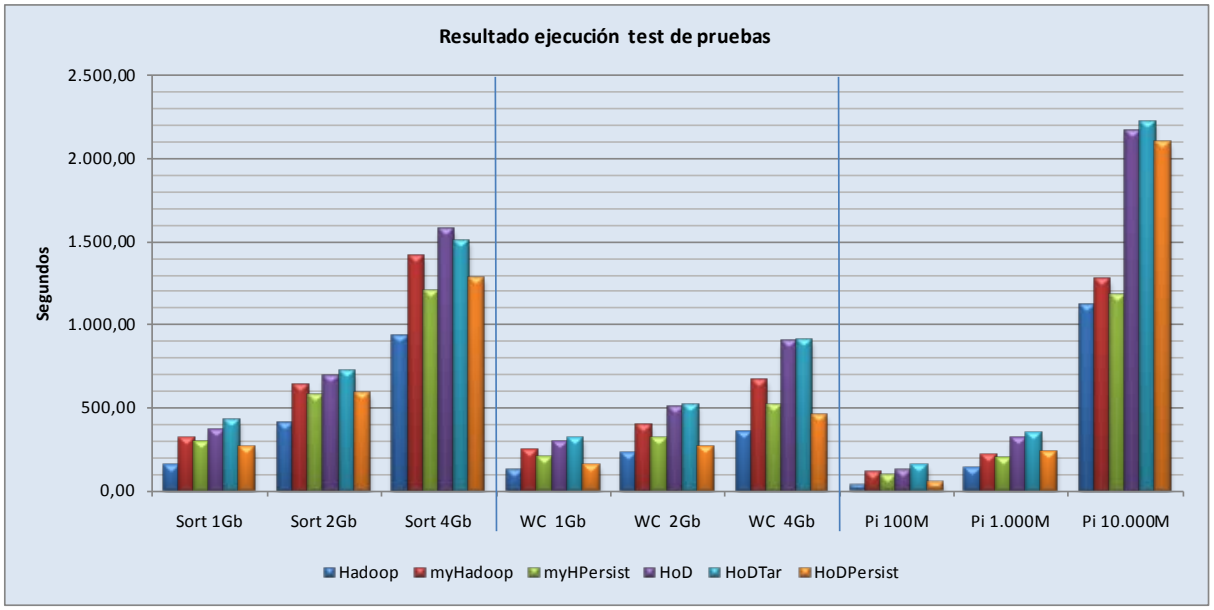


Figura 3.6. Resultados obtenidos en la ejecución de los test de pruebas.

Del gráfico de la Figura 3.6 se puede inferir que los entornos que han ofrecido un peor rendimiento son HoD y HoDTar. El comportamiento de estos dos entornos es, de alguna manera, el esperado dado que HoD arranca dos TaskTracker y dos DataNodes en el clúster de cuatro nodos que genera. Por ese motivo, se ven fuertemente penalizados en los test asociados a la aplicación PiEstimator, por tratarse de una aplicación de uso intensivo de CPU.

En los test con un uso intensivo de disco, aplicaciones Sort y WordCount, HoD y HoDTar no se ven penalizadas en la misma medida que en los test de la aplicación PiEstimator. Esto es debido al modelo que se ha seguido para crear los entornos de prueba, donde únicamente se utiliza un disco de almacenamiento. Con este modelo, el disco se convierte en un cuello de botella, por lo que los

entornos generados con myHadoop, donde se arranca un TaskTracker en cada uno de los cuatro nodos del clúster, se ven limitados por las prestaciones del almacenamiento físico.

Los entornos que muestran un mejor rendimiento en los test con uso intensivo de disco son los que presentan HDFS persistente: myHPersist y HoDPersist. Además, myHPersist se muestra como el entorno que mejor rendimiento ofrece en aplicaciones intensivas de CPU.

¿Qué origina la diferencia de tiempo requerido para ejecutar los test en el entorno Hadoop dedicado y el resto de entornos? Los entornos Hadoop generados de forma dinámica, presentan una serie de fases adicionales que no son necesarias en el entorno Hadoop dedicado: inicialización del clúster, copia del fichero a procesar al HDFS y copia de los resultados generados por la aplicación al sistema de ficheros local (excepto en los entornos con HDFS persistente), y fase de liberación de recursos.

La Figura 3.7 muestra un gráfico con el tiempo requerido por cada entorno para ejecutar cada una de las aplicaciones. Evaluando únicamente el tiempo requerido para ejecutar una aplicación, el entorno que mejor se comporta es myHadoop, moviéndose en tiempos de ejecución similares a los obtenidos en el clúster Hadoop dedicado. En algunos casos incluso los tiempos obtenidos son mejores, resultados atribuibles a que se trata de una instancia “fresca” de Hadoop. En cambio, los entornos con HDFS persistente se muestran penalizados, dado que exigen de un mayor tránsito de datos por la red fruto de que el almacenamiento no reside en el nodo de cómputo. De alguna manera, los entornos con HDFS persistente rompen con la idea original de MapReduce de acercar el cómputo a los datos.

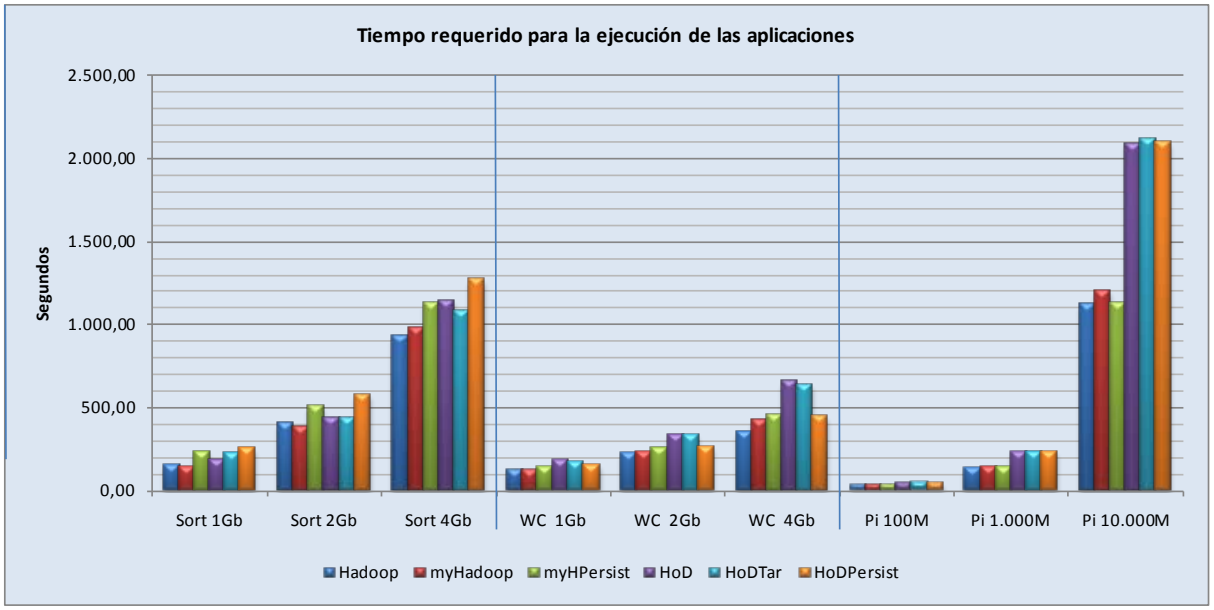


Figura 3.7. Tiempo requerido para la ejecución de las aplicaciones.

En la Figura 3.8 se muestra el tiempo requerido por el resto de fases en cada uno de los entornos de prueba.

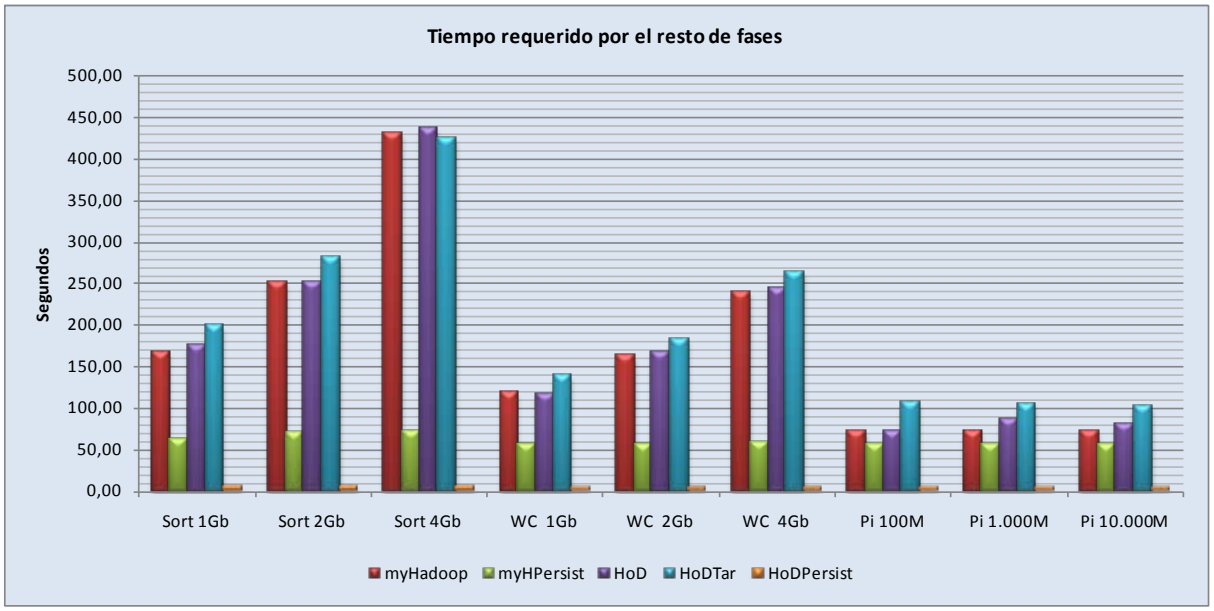


Figura 3.8. Tiempo requerido por las fases de inicialización, copia y liberación de recursos.

De la gráfica de la Figura 3.8 se desprende que, el entorno que menor tiempo necesita para realizar el resto de fases de los test de pruebas es el HoDPersist. Esto es debido a que el HDFS se encuentra ya levantado y listo para atender peticiones, por lo tanto se ahorra el tiempo necesario para arrancar los demonios NameNode y DataNodes, y el tiempo que necesitan para sincronizarse. En el caso de myHPersist, a pesar de tratarse también de un sistema de ficheros persistente, se debe encargar de levantar los demonios NameNode y DataNodes, y éstos deben sincronizarse entre sí.

Los entornos con HDFS no persistente, se ven fuertemente penalizados,, en los test Sort y WordCount, por el hecho de tener que copiar los ficheros de prueba que deben procesar las aplicaciones, del sistema de ficheros local al HDFS del clúster, y por tener que copiar el resultado generado al sistema de ficheros local.

La Figura 3.9 muestra el tiempo requerido por cada fase en los diferentes test ejecutados en cada uno de los entornos de prueba. Aquí se puede observar el impacto que representa, tanto en los test asociados a la aplicación Sort como en los asociados a la aplicación WordCount, el hecho de tener que realizar la copia de los ficheros a procesar del sistema de ficheros local al HDFS. En el caso de los test del Sort, el impacto es doble dado que el tamaño de los ficheros resultantes, en su conjunto, es igual al tamaño de los datos de entrada.

Mención especial merece la fase de inicialización del clúster. El tiempo requerido para completar dicha fase se mantiene prácticamente constate en cada entorno para cada test ejecutado.

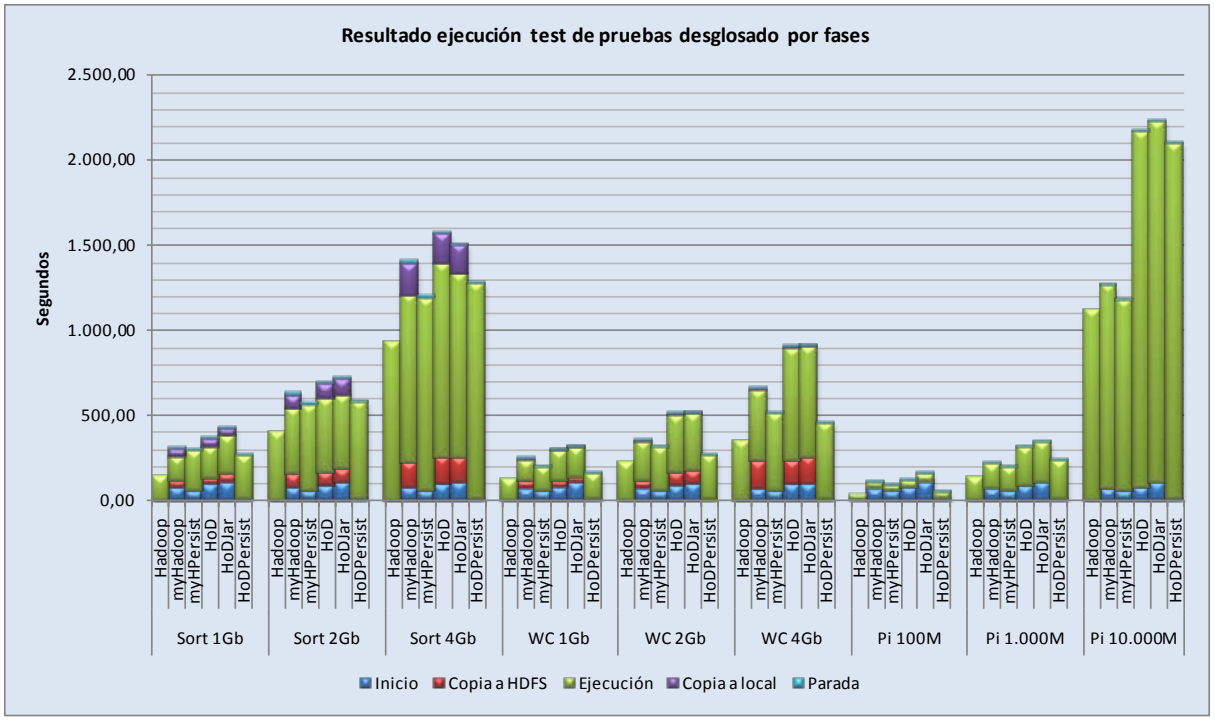


Figura 3.9. Tiempo requerido por cada fase para la ejecución de las aplicaciones.

En el caso de los entornos con HDFS no persistente, una vez se han arrancando los demonios Hadoop en los nodos asignados, se debe introducir una espera para que se complete la sincronización entre el NameNode y los DataNodes. Si se procede a ejecutar un trabajo antes de que la sincronización se produzca, fallará por no hallarse el HDFS disponible [40]. De forma empírica, se ha determinado que el tiempo que se requiere para dicha sincronización está en un horquilla entre los 10 y los 50 segundos. Por tanto, se ha añadido un *sleep* de 60 segundos para garantizar que el HDFS estará operativo en el momento de ejecutar un trabajo. Esos 60 segundos están contabilizados dentro de la fase de inicialización.

En el caso del entorno myHPersist, no es necesario introducir el *sleep* de 60 segundos sino que, mediante el comando Hadoop *safemode wait* [41], se realiza la pausa hasta que el NameNode y los DataNodes quedan sincronizados.

Cuando se inicia la ejecución del NameNode, su primera tarea es reconstruir en memoria una imagen con la meta información del sistema de ficheros. Una vez creado, el NameNode está en disposición de atender peticiones. Sin embargo, el NameNode se está ejecutando en modo seguro (*safemode*), lo que significa que sólo ofrece una vista de sólo lectura del sistema de ficheros a los clientes.

La información correspondiente a la ubicación de los bloques en el sistema de ficheros, no se conserva de forma persistente en el NameNode sino que reside en los DataNodes. Durante el funcionamiento normal del sistema, el NameNode posee un mapa en memoria con la localización de los bloques. Cuando el sistema se inicializa, el modo seguro es necesario para dar tiempo a los DataNodes a comunicar al NameNode la lista de los bloques que maneja.

Se abandona el modo seguro cuando se ha alcanzado un porcentaje mínimo de réplica sobre el NameNode, más una extensión de tiempo. El porcentaje queda fijado mediante la propiedad *dfs.safemode.threshold.pct* (por defecto 99,99%) y la extensión de tiempo mediante la propiedad *dfs.safemode.extension* (por defecto 30 segundos).

Cuando se inicializa un clúster con un HDFS recién formateado, el NameNode no entra en modo seguro por no existir bloques en el sistema. Por ese motivo, no se ha podido utilizar el modo seguro en aquellos entornos con HDFS no persistente, siendo necesario recurrir al comando *sleep* de Unix.

En el caso del entorno HoDPersist, no es necesario introducir ninguna pausa dado que el HDFS se encuentra ya activo en el momento que se inicializa el clúster dinámico.

Capítulo 4. Conclusiones

El siguiente capítulo se inicia exponiendo las conclusiones extraídas sobre las diferentes soluciones analizadas, en base a los resultados obtenidos en la experimentación realizada. A continuación se pasa a realizar un análisis de la evolución del proyecto para comentar los diferentes problemas acontecidos a lo largo del mismo y las acciones correctivas que se han llevado a cabo. Para finalizar, se proponen una serie de trabajos futuros, con el objetivo de completar el análisis realizado en el presente proyecto, y se realiza una valoración personal del proyecto y del trabajo realizado.

4.1. Conclusiones del trabajo realizado

Tanto HoD como myHadoop se han mostrado como herramientas eficaces que permiten la generación de un clúster bajo demanda de manera relativamente simple. Su instalación es relativamente sencilla y el proceso de configuración tampoco requiere dedicarle un gran esfuerzo, si se está familiarizado con la configuración de un entorno Hadoop.

A la hora de elegir una u otra herramienta, se debe tener en cuenta la configuración de clúster que generan ambas:

- myHadoop genera un clúster donde un nodo, denominado nodo maestro, concentra el NameNode y JobTracker y donde todos los nodos, incluido el nodo maestro, son nodos de cómputo.

- HoD genera un tipo de clúster más orientado a altas prestaciones, dado que utiliza un nodo para ejecutar el NameNode, otro nodo para el JobTracker y el resto de nodos son nodos de cómputo. Por tanto, las aplicaciones a ejecutar deben ser exigentes en cuanto a carga de trabajo para mantener ocupados tanto al NameNode como al JobTracker. Esta configuración de clúster también puede ser adecuada en entornos donde los nodos disponen de poco procesador y/o memoria, donde difícilmente podrían convivir en un mismo nodo NameNode, JobTracker y los demonios asociados a los nodos de cómputo (TaskTracker y DataNode), como ocurre en myHadoop.

El uso de estas herramientas requerirá de una rigurosa planificación del espacio en disco asignado a los usuarios que lancen trabajos con HoD o myHadoop:

- En el caso de usar un HDFS persistente, se debe tener en cuenta asignar espacio suficiente al sistema de ficheros local sobre el cual se soporta el HDFS, para poder almacenar tanto la información final que generará cada aplicación como la información intermedia que genera la fase map. El uso de este tipo de sistema de ficheros tiene una implicación adicional, se están utilizando recursos del HPC de forma permanente. Por tanto, es vital que se encuentre bien dimensionado para garantizar un buen aprovechamiento de esos recursos.

- En el caso de usar un HDFS no persistente, sirven las premisas comentadas para el HDFS persistente. Se debe dimensionar de forma adecuada el sistema de ficheros sobre el cual residirán los directorios de trabajo del clúster dinámico.

A modo de ejemplo, la Tabla 4.1 muestra el número de bytes leídos y escritos por la aplicación Sort cuando procesa un fichero de 4Gb.

	Counter	Map	Reduce	Total
FileSystemCounters	FILE_BYTES_READ	0	4.285.568.259	4.285.568.259
	HDFS_BYTES_READ	4.295.151.293	0	4.295.151.293
	FILE_BYTES_WRITTEN	4.285.581.911	4.285.568.259	8.571.150.170
	HDFS_BYTES_WRITTEN	0	4.294.104.442	4.294.104.442

Tabla 4.1. Contadores de bytes leídos/escritos para Sort con fichero de 4Gb.

El fichero de entrada consta de 4.295.151.293 bytes (HDFS_BYTES_READ), la fase map escribe 4.285.581.911 bytes (FILE_BYTES_WRITTEN) de ficheros temporales y la fase reduce escribe 4.294.104.442 bytes en el HDFS. Resumiendo, para ejecutar la aplicación Sort con un fichero de 4Gb, el HDFS necesita aproximadamente 12Gb de espacio: 4Gb para el fichero de entrada, 4Gb para los ficheros temporales y 4Gb para el fichero resultante. Cuando finaliza la ejecución de la aplicación, los ficheros temporales son eliminados pero deben tenerse en cuenta a la hora de dimensionar el espacio del sistema de ficheros.

En el caso de utilizar un HDFS no persistente, aún se requerirá de más espacio en disco. El fichero a clasificar debe residir en el sistema de ficheros local y el fichero resultante debe copiarse, del HDFS del clúster dinámico al sistema de ficheros local, tras ejecutarse la aplicación y antes de que se ejecute la fase de liberación de recursos. Por tanto, requerirá de 4Gb para el fichero de entrada, más 4Gb correspondientes a la copia de éste en el HDFS, más 4Gb para el fichero resultante que reside en el HDFS, más 4Gb correspondientes a la copia del fichero resultante al sistema de ficheros local. En resumen, se requieren aproximadamente 16Gb disponibles para completar con éxito todo el proceso.

El uso de un clúster Hadoop bajo demanda implica la existencia de fases adicionales que no se presentan en un clúster Hadoop dedicado, como son: inicialización, copia de los ficheros a procesar y copia de los ficheros resultantes si no se dispone de un HDFS dedicado, y liberación de recursos. Por lo tanto, a la hora de ejecutar un trabajo con este tipo de clúster, el tiempo requerido por las fases comentadas debería ser poco significativo respecto al tiempo requerido para ejecutar dicho trabajo.

Para finalizar, comentar que HoD sólo soporta Torque/PBS como gestor de recursos mientras que myHadoop, además de Torque/PBS, también soporta SGE. Aunque esto pueda ser un impedimento para el uso de estas herramientas, al tratase de soluciones de código libre, es posible modificarlas para dar soporte al gestor de recursos instalado en el HPC que se disponga. Así, se podrá hacer uso de las mismas, beneficiándose de la posibilidad de utilizar un framework Hadoop sin necesidad de reservar una parte significativa del clúster de forma estática para poder ejecutar tareas MapReduce.

4.2. Desarrollo del proyecto

En la etapa de análisis, en lo concerniente al estudio de las diferentes soluciones que permiten generar un clúster Hadoop bajo demanda, ha sido del todo fundamental la orientación del director del proyecto para llevar a un buen puerto esta primera etapa.

Existe un volumen importante de documentación tanto de MapReduce como de Hadoop, así como multitud de líneas de trabajo en diferentes aspectos relacionados con la computación en un clúster Hadoop. Por tanto, estar al día de los trabajos que se publican para conocer las diferentes líneas de trabajo existentes, se muestra fundamental. Una vez guiado, el alumno posee la base del hilo argumental sobre el cual se basará el resto del análisis.

Las reuniones de seguimiento, se han mostrado como una herramienta eficaz para hacer que el enfoque sea el adecuado, corrigiéndose posibles desviaciones o conceptos entendidos de forma errónea por parte del alumno. También han servido para que el alumno realice sus propias propuestas en base a la investigación realizada.

Respecto al resto del proyecto, básicamente se han producido dos problemas:

- En la instalación y configuración de Hadoop en el clúster dedicado, donde se constató que el NameNode y los DataNodes necesitaban varios minutos para sincronizarse. Se analizaron los logs que genera Hadoop hasta determinar que el problema se producía en el servidor HTTP Jetty que Hadoop lleva embebido. En el repositorio de incidencias del propio Apache Hadoop, se encontró un caso abierto donde se comentaba esta circunstancia [42], donde se ofrecía

una solución al problema. Esto supuso una desviación de 16 horas respecto a la planificación inicial.

- En la fase de experimentación, una vez obtenidos los resultados y viendo el peso que presentaban las fases de inicialización y copia de ficheros de y hacia el HDFS, se decidió realizar una nueva rueda de pruebas con mayor carga para ver cómo evolucionaba el peso de las fases comentadas respecto al tiempo requerido para ejecutarse la aplicación. Esto supuso una carga adicional de 40 horas.

Al estar la aplicación bastante ajustada, el alumno decidió aumentar la dedicación prevista tanto para resolver el problema de configuración como la nueva rueda de pruebas en la fase de experimentación. Por tanto, se dedicaron horas del fin de semana para poder cumplir con las fechas inicialmente previstas.

Por tanto, la dedicación total del proyecto ha sido de 332h + 16h + 40h = 388horas.

4.3. Futuros trabajos

Se proponen dos líneas de trabajo para completar el análisis iniciado por el presente proyecto.

Una primera línea consiste en realizar la experimentación con HoD y myHadoop en un clúster físico para evaluar el impacto que tiene la utilización de discos separados y la red de interconexión en un clúster generado de forma dinámica. En el apartado 3.4, cuando se hablaba de los resultados obtenidos, se comentaba que el disco de almacenamiento había sido un cuello de botella. La disponibilidad de discos separados debe redundar en un mejor rendimiento. Por otro lado, al existir distancia física, será necesario realizar transmisión de información entre los diferentes nodos a través de la red que los une, produciéndose una pérdida de rendimiento respecto al clúster virtual utilizado en este proyecto.

Una segunda línea consiste en evaluar el rendimiento de un clúster Hadoop bajo demanda generado mediante Oracle Grid Engine. Se trata de un producto comercial de Oracle que permite generar un clúster Hadoop de forma dinámica y que es una evolución de la línea de trabajo marcada por SGE, que hasta la versión 6.2 era un proyecto de código abierto. La adquisición de SUN por parte de Oracle hizo que esta última incluyera este proyecto en su suite de productos. En el momento de elaborar esta memoria, había aparecido un nuevo proyecto de código abierto denominado Open Grid Scheduler que continúa con la línea de desarrollo marcada por el antiguo SGE. Por tanto, sería una solución a evaluar si el proyecto se encuentra suficientemente maduro.

4.4. Valoración personal

Siempre me han producido una especial fascinación todo lo relacionado con la ejecución paralela. Arquitecturas, paradigmas de programación, soluciones para aprovechar al máximo los recursos de una red compartida, etc.

Esta fascinación, me llevó a buscar en la lista de proyectos ofertados por la UAB alguno que tuviera relación con este "mundillo". No eran pocos los proyectos que se ofertaban relacionados con la paralelización, pero me llamó especialmente la atención uno con la palabra "Hadoop". Dado que no sabía de qué se trataba, me pasé varios días buscando por internet para entender de qué se trataba. Al final, la decisión estaba tomada, me gustaba todo lo que había visto y ese tenía que ser mi proyecto. La suerte me acompañó.

Ahora, en el momento de hacer balance, siento que el proyecto ha cubierto sobradamente las expectativas que tenía depositadas. Este proyecto me ha permitido:

- Conocer un nuevo paradigma de programación: MapReduce.
- Trabajar con Hadoop, herramienta que no conocía y permite generar un clúster de manera sencilla. Instalarlo, configurarlo, sintonizarlo son algunas de las tares que he podido realizar en el proyecto.
- Ampliar mis conocimientos sobre los sistemas de ficheros distribuidos a través de HDFS.
- Experimentar la ejecución de aplicaciones paralelas en un clúster.
- Ampliar conocimientos en lo que respecta a planificadores batch y algunas de las soluciones que se plantean para un mejor aprovechamiento de los recursos disponibles.

El conjunto de herramientas que se están creando alrededor de Hadoop y que empresas como Google, Yahoo, Facebook o eBay, lo utilicen como parte de la infraestructura sobre la cual ejecutan sus procesos críticos, convierten a Hadoop un producto muy interesante y atractivo sobre el que adquirir conocimientos.

Referencias

1. Moore, Gordon E. Cramming more components onto integrated circuits. Electronics Magazine, 1965.

2. Niall Kennedy's Weblog. [Online] [Cited: 13 junio 2011.] http://www.niallkennedy.com/blog/2008/01/google-mapreduce-stats.html

3. DBMS2. [Online] 11 2009. [Cited: 13 junio 2011.] http://www.dbms2.com/2009/05/11/facebook-hadoop-and-hive/

4. DBMS2. [Online] 30 04 2009. [Cited: 13 06 2011.] http://www.dbms2.com/2009/04/30/ebays-two-enormous-data-warehouses/

5. European Organization for Nuclear Research. [Online] [Cited: 13 06 2011.] http://public.web.cern.ch/public/en/lhc/Computing-en.html

6. Becla, Jacek, et al. Designing a multi-petabyte database for LSST. Stanford Linear Accelerator Center : SLAC Publications, 2006. SLAC-PUB-12292.

7. Gantz, John F., et al. The Diverse and Exploding Digital Universe (An Updated Forecast of Worldwide Information Growth Through 2011). s.l. : IDC, 2008.

8. OpenMP. [Online] [Cited: 13 06 2011.] http://openmp.org/wp/

9. Message Passing Interface Forum. [Online] [Cited: 13 06 2011.] http://www.mpi-forum.org/

10. Dean, Jeffrey and Ghemawat, Sanjay. MapReduce: Simplified Data Processing on Large Clusters. San Francisco, California : 6th Symposium on Operating System Design and Implementation (OSDI 2004), 2004.

11. Apache Hadoop. HDFS Architecture Guide. [Online] [Cited: 13 06 2011.] http://hadoop.apache.org/common/docs/current/hdfs_design.html

12. Ghemawat, Sanjay, Gobioff, Howard and Leung, Shun-Tak. The Google File System. Bolton Landing, New York, USA : Google, 2003.

13. He, Bingsheng, et al. Mars: A MapReduce Framework on Graphics Processors.

14. Metis. [Online] [Cited: 13 06 2011.] http://pdos.csail.mit.edu/metis/

15. The Phoenix System for MapReduce Programming. [Online] http://mapreduce.stanford.edu/

16. Microsoft Research - Dryad. [Online] [Cited: 13 06 2011.]
http://research.microsoft.com/en-us/projects/dryad/

17. HDFS. [Online] [Cited: 13 06 2011.]
http://hadoopblog.blogspot.com/2010/05/facebook-has-worlds-largest-hadoop.html

18. Cluster Resources – Torque Resource Manager. [Online]. [Cited: 13 06 2011.]
http://www.clusterresources.com/products/torque-resource-manager.php

19. Oracle – Oracle Grid Engine. [Online]. [Cited: 13 06 2011.]
http://www.sun.com/software/sge/

20. B. Langmead, M. Schatz, J. Lin, M. Pop, and S. Salzberg. Searching for SNPs with cloud computing. Genome Biol, 10(11):R134, 2009.

21. T. Gunarathne, T. Wu, J. Qiu, and G. Fox. Cloud computing paradigms for pleasingly parallel biomedical applications. In 19th ACM Intl Symp on High Perf Dist Comp, pages 460–469. ACM, 2010.

22. S. Krishnan, C. Baru, and C. Crosby. Evaluation of MapReduce for Gridding LIDAR Data. In 2nd IEEE Intl Conf on Cloud Comp Tech and Science, 2010.

23. Apache Hadoop. [Online] http://hadoop.apache.org/common/docs/r0.20.2/hod_user_guide.html

24. Krishnan, Sriram, Tatineni, Mahidhar and Baru, Chaitanya. myHadoop - Hadoop-on-Demand on Traditional HPC Resources. San Diego : San Diego Supercomputer Center, 2010.

25. Apache Hadoop. [Online] [Cited: 13 06 2011.] http://hadoop.apache.org/

26. HDFS Users Guide - Secondary NameNode. [Online] [Cited: 13 06 2011.]
http://hadoop.apache.org/common/docs/current/hdfs_user_guide.html#Secondary+NameNode

27. Fair Scheduler Guide. [Online] [Cited: 13 06 2011.]
http://hadoop.apache.org/common/docs/r0.20.2/fair_scheduler.html

28. Capacity Scheduler Guide. [Online] [Cited: 13 06 2011.]
http://hadoop.apache.org/common/docs/r0.20.2/capacity_scheduler.html

29. QEMU open source processor emulator. [Online] [Cited: 13 06 2011.]
http://wiki.qemu.org/Main_Page

30. Bochs - think inside the bochs. [Online] [Cited: 13 06 2011.]
http://bochs.sourceforge.net/

31. MAME – Multiple Arcade Machine Emulator. [Online] [Cited: 13 06 2011.]
http://mamedev.org/about.html

32. Intel – Intel Virtualization Technology. [Online] [Cited: 13 06 2011.]
http://www.intel.com/technology/virtualization/technology.htm

33. AMD – AMD Virtualization Technology. [Online] [Cited: 13 06 2011.]
http://sites.amd.com/us/business/it-solutions/virtualization/Pages/amd-v.aspx

34. WMware – Aspectos básicos de la virtualización. [Online] [Cited: 13 06 2011.]
http://www.vmware.com/es/virtualization/what-is-virtualization.html

35. VirtualBox. [Online] [Cited: 13 06 2011.]
http://www.virtualbox.org/

36. KVM. [Online] [Cited: 13 06 2011.]
http://www.linux-kvm.org/page/Main_Page

37. XEN. [Online] [Cited: 13 06 2011.]
http://www.xen.org/

38. The User-mode Linux Kernel Home Page. [Online] [Cited: 13 06 2011.]
http://user-mode-linux.sourceforge.net/

39. Agrawal, Divyakant, Candan, K. Selçuk and Li, Wen-Syan. New frontiers in information and Sofware as Services: Service and Application Design Challenges in the cloud. s.l. : Springer; Lecture notes in business information processing [LNBIP] 74, 2011.

40. HOD User Guide. [Online] [Cited: 13 06 2011.]
http://hadoop.apache.org/common/docs/r0.20.2/hod_user_guide.html#Hadoop+DFSClient+Warns+with+a%0A++NotReplicatedYetException

41. Apache Hadoop – Hadoop DFS User Guide - Safemode. [Online] [Cited: 13 06 2011.]
http://hadoop.apache.org/common/docs/r0.17.2/hdfs_user_guide.html#Safemode

42. The Apache Software Foundation. [Online] [Cited: 13 06 2011.]
https://issues.apache.org/jira/browse/HADOOP-6882

ANEXOS

A.Resultado detallado de las pruebas

En el siguiente apartado se muestra de forma detallada los tiempos obtenidos en cada una de las pruebas realizadas en los diferentes entornos que se han testeado a lo largo del proyecto. Esta información complementa a la presentada en el apartado 3.4.

Clúster Hadoop dedicado

La Tabla A.1 muestra el número de tareas ejecutadas y los tiempos obtenidos para la aplicación Sort en función del volumen procesado:

Aplicación Sort				
Volumen de datos a clasificar	**Map**	**Reduce**	**Tiempo medio ejecución**	**Desviación estándar**
1Gb (16 bloques de 64Kb)	16	7	148,26	8,80
2Gb (32 bloques de 64Kb)	32	7	401,14	15,79
4Gb (64 bloques de 64Kb)	64	7	926,27	19,35

Tabla A.1. Resultados obtenidos para la aplicación Sort.

La Tabla A.2 muestra el número de tareas ejecutadas y los tiempos obtenidos para la aplicación WordCount en función del volumen procesado:

Aplicación WordCount				
Volumen de datos a procesar	**Map**	**Reduce**	**Tiempo medio ejecución**	**Desviación estándar**
1Gb (16 bloques de 64Kb)	16	1	123,75	1,53
2Gb (34 bloques de 64Kb)	34	1	221,72	2,51
4Gb (68 bloques de 64Kb)	68	1	351,21	3,17

Tabla A.2. Resultados obtenidos para la aplicación WordCount.

La Tabla A.3 muestra el número de tareas ejecutadas y los tiempos obtenidos para la aplicación WordCount en función del volumen procesado:

Aplicación PiEstimator				
Volumen de puntos a procesar	Map	Reduce	Tiempo medio ejecución	Desviación estándar
100.000.000 por map	4	1	32,09	0,35
1.000.000.000 por map	4	1	131,99	1,48
10.000.000.000 por map	4	1	1.116,50	6,41

Tabla A.3. Resultados obtenidos para la aplicación PiEstimator.

Clúster HoD

La Tabla A.4 muestra el número de tareas ejecutadas y los tiempos obtenidos para la aplicación Sort en función del volumen procesado.

Aplicación Sort				
Volumen de datos a clasificar	Map	Reduce	Tiempo medio ejecución	Desviación estándar
1Gb (16 bloques de 64Kb)	16	3	362,31	26,46
2Gb (32 bloques de 64Kb)	32	3	688,10	63,13
4Gb (64 bloques de 64Kb)	64	3	1.571,31	160,16

Tabla A.4. Resultados obtenidos para la aplicación Sort.

La Tabla A.5 se muestra el tiempo invertido en cada fase.

Prueba	Inicio	Copia ficheros al HDFS	Ejecución	Copia resultados a local	Parada
Sort 1Gb	92,13	36,55	186,52	46,14	0,98
Sort 2Gb	86,66	78,48	435,54	85,63	1,78
Sort 4Gb	92,29	163,91	1.134,89	178,43	1,79

Tabla A.5. Segundos invertidos por cada fase en pruebas con Sort.

La Figura A.1 muestra la distribución porcentual del consumo para cada una de las fases de la Tabla A.5.

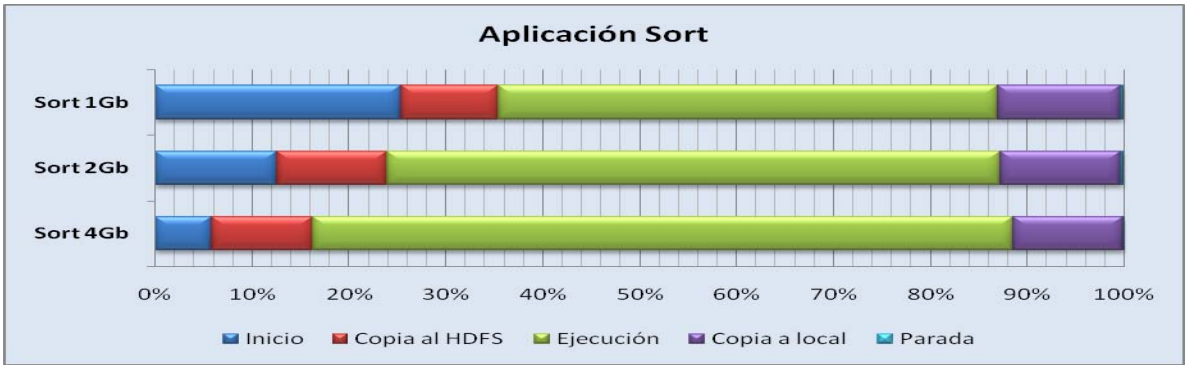


Figura A.1. Distribución porcentual del consumo obtenida para Sort.

La Tabla A.6 muestra los resultados obtenidos para la aplicación WordCount en base a los diferentes volúmenes procesados:

Aplicación WordCount				
Volumen de datos a procesar	Map	Reduce	Tiempo medio ejecución	Desviación estándar
1Gb (16 bloques de 64Kb)	16	1	296,54	14,75
2Gb (34 bloques de 64Kb)	34	1	504,81	19,67
4Gb (68 bloques de 64Kb)	68	1	900,79	27,08

Tabla A.6. Resultados obtenidos para la aplicación WordCount.

La Tabla A.7 muestra el tiempo invertido en cada fase.

Prueba	Inicio	Copia ficheros al HDFS	Ejecución	Copia resultados a local	Parada
WC 1Gb	77,70	36,55	180,10	1,82	0,36
WC 2Gb	89,11	75,47	337,10	2,83	0,30
WC 4Gb	92,12	145,57	656,95	5,12	1,02

Tabla A.7. Segundos invertidos por cada fase en pruebas con WordCount.

En la Figura A.2 se muestra la distribución porcentual del consumo para cada una de las fases de la Tabla A.7.

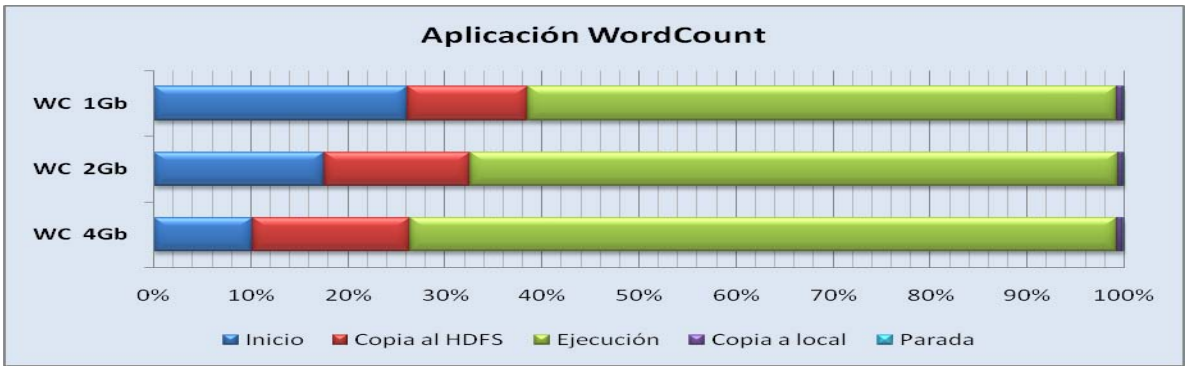


Figura A.2. Distribución porcentual del consumo obtenida para WordCount.

La Tabla A.8 muestra los resultados obtenidos para la aplicación PiEstimator en base a los diferentes volúmenes procesados por cada map:

Aplicación PiEstimator				
Volumen de puntos a procesar	Map	Reduce	Tiempo medio ejecución	Desviación estándar
100 millones por map	4	1	117,37	8,46
1.000 millones por map	4	1	316,27	11,85
10.000 millones por map	4	1	2.164,83	27,86

Tabla A.8. Resultados obtenidos para la aplicación PiEstimator.

La Tabla A.9 muestra el tiempo invertido en cada fase.

Prueba	Inicio	Copia ficheros al HDFS	Ejecución	Copia resultados a local	Parada
Pi 100M	72,13	N/A	45,11	N/A	0,13
Pi 1.000M	86,37	N/A	229,77	N/A	0,13
Pi 10.000M	80,61	N/A	2.084,08	N/A	0,13

Tabla A.9. Segundos invertidos por cada fase en pruebas con PiEstimator.

En la Figura A.3 se muestra la distribución porcentual del consumo para cada una de las fases reflejadas en la Tabla A.9.

Figura A.3. Distribución porcentual del consumo obtenida para PiEstimator.

A continuación se procede a presentar los resultados para un clúster generado con HoD, donde el proceso de inicialización procede a desplegar un tarball con los ficheros necesarios para crear una instalación temporal de Hadoop en cada uno de los nodos asignados. Al igual que en el clúster anterior, se emplea un HDFS no persistente.

La Tabla A.10 muestra los resultados obtenidos para la aplicación Sort:

Aplicación Sort				
Volumen de datos a clasificar	Map	Reduce	Tiempo medio ejecución	Desviación estándar
1Gb (16 bloques de 64Kb)	16	3	424,07	36,54
2Gb (32 bloques de 64Kb)	32	3	713,55	84,64
4Gb (64 bloques de 64Kb)	64	3	1.504,87	174,18

Tabla A.10. Resultados obtenidos para la aplicación Sort.

La Tabla A.11 muestra los resultados obtenidos para la aplicación WordCount:

Aplicación WordCount				
Volumen de datos a procesar	Map	Reduce	Tiempo medio ejecución	Desviación estándar
1Gb (16 bloques de 64Kb)	16	1	309,98	7,45
2Gb (34 bloques de 64Kb)	34	1	511,68	18,84
4Gb (68 bloques de 64Kb)	68	1	902,88	44,28

Tabla A.11. Resultados obtenidos para la aplicación WordCount.

La Tabla A.12 muestra los resultados obtenidos para la aplicación PiEstimator:

Aplicación PiEstimator				
Volumen de puntos a procesar	Map	Reduce	Tiempo medio ejecución	Desviación estándar
100.000.000 por map	4	1	153,05	4,38
1.000.000.000 por map	4	1	339,30	4,99
10.000.000.000 por map	4	1	2.220,42	21,47

Tabla A.12. Resultados obtenidos para la aplicación PiEstimator.

Para finalizar con las pruebas sobre HoD se presentan los resultados sobre un clúster con HDFS persistente.

La Tabla A.13 muestra los resultados obtenidos para la aplicación Sort.

Aplicación Sort				
Volumen de datos a clasificar	Map	Reduce	Tiempo medio ejecución	Desviación estándar
1Gb (16 bloques de 64Kb)	16	5	259,12	24,53
2Gb (32 bloques de 64Kb)	32	5	581,76	20,41
4Gb (64 bloques de 64Kb)	64	5	1.275,86	32,12

Tabla A.13. Resultados obtenidos para la aplicación Sort.

La Tabla A.14 muestra los resultados obtenidos para la aplicación WordCount:

Aplicación WordCount				
Volumen de datos a procesar	Map	Reduce	Tiempo medio ejecución	Desviación estándar
1Gb (16 bloques de 64Kb)	16	1	151,88	3,66
2Gb (34 bloques de 64Kb)	34	1	266,37	3,62
4Gb (68 bloques de 64Kb)	68	1	451,13	2,69

Tabla A.14. Resultados obtenidos para la aplicación WordCount.

La Tabla A.15 muestra los resultados obtenidos para la aplicación PiEstimator:

Aplicación PiEstimator				
Volumen de puntos a procesar	Map	Reduce	Tiempo medio ejecución	Desviación estándar
100.000.000 por map	4	1	49,69	0,57
1.000.000.000 por map	4	1	236,37	1,44
10.000.000.000 por map	4	1	2.096,20	14,34

Tabla A.15. Resultados obtenidos para la aplicación PiEstimator.

No se presentan resultados desglosados puesto que el tiempo de ejecución representa entre el 97 y el 99,5% del tiempo total para el caso del Sort y WordCount y en el caso de la aplicación PiEstimator representa entre el 92% para la ejecución con menor volumen y el 99,8% para la ejecución con mayor volumen. Destacar que en esta modalidad no es necesario introducir el sleep de 60 segundos en la fase de inicialización del clúster.

Clúster myHadoop

En el siguiente apartado se presentan los resultados obtenidos para las diferentes modalidades de clúster myHadoop analizadas:

- Clúster myHadoop con HDFS no persistente.
- Clúster myHadoop con HDFS persistente.

Se procede a iniciar la presentación de resultados para un clúster myHadoop con un HDFS no persistente. La Tabla A.16 muestra los resultados obtenidos para la aplicación Sort.

Aplicación Sort				
Volumen de datos a clasificar	Map	Reduce	Tiempo medio ejecución	Desviación estándar
1Gb (16 bloques de 64Kb)	16	7	310,05	9,01
2Gb (32 bloques de 64Kb)	32	7	639,50	31,45
4Gb (64 bloques de 64Kb)	64	7	1.409,10	74,42

Tabla A.16. Resultados obtenidos para la aplicación Sort.

La Tabla A.17 se muestra el tiempo invertido en cada fase.

Prueba	Inicio	Copia ficheros al HDFS	Ejecución	Copia resultados a local	Parada
Sort 1Gb	71,38	41,63	142,98	42,63	11,44
Sort 2Gb	71,40	79,97	387,78	80,25	20,10
Sort 4Gb	71,83	150,96	978,66	187,70	19,96

Tabla A.17. Segundos invertidos por cada fase en pruebas con Sort.

La
Figura A.4 muestra la distribución porcentual del consumo para cada una de las fases de la Tabla A.17.

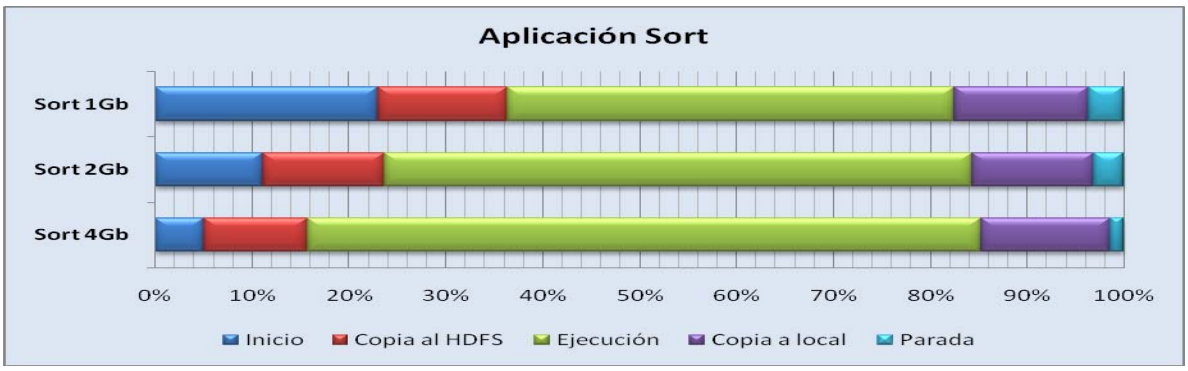


Figura A.4. Distribución porcentual del consumo obtenida para Sort.

La Tabla A.18 muestra los resultados obtenidos para la aplicación WordCount en base a los diferentes volúmenes procesados:

Aplicación WordCount				
Volumen de datos a procesar	Map	Reduce	Tiempo medio ejecución	Desviación estándar
1Gb (16 bloques de 64Kb)	16	1	243,69	3,71
2Gb (34 bloques de 64Kb)	34	1	395,24	6,26
4Gb (68 bloques de 64Kb)	68	1	661,27	7,89

Tabla A.18. Resultados obtenidos para la aplicación WordCount.

La Tabla A.19 muestra el tiempo invertido en cada fase:

Prueba	Inicio	Copia ficheros al HDFS	Ejecución	Copia resultados a local	Parada
WC 1Gb	70,96	40,83	125,48	2,43	4,00
WC 2Gb	70,93	83,92	232,70	3,39	4,30
WC 4Gb	71,06	157,72	421,04	6,62	4,83

Tabla A.19. Segundos invertidos por cada fase en pruebas con WordCount.

La Figura A.5 muestra la distribución porcentual del consumo para cada una de las fases de la Tabla A.19.

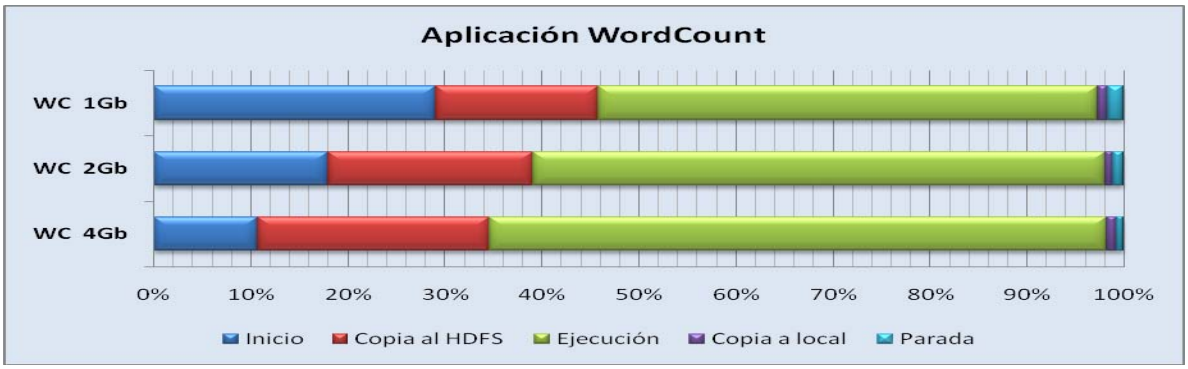


Figura A.5. Distribución porcentual del consumo obtenida para WordCount.

La Tabla A.20 muestra los resultados obtenidos para la aplicación PiEstimator en base a los diferentes volúmenes procesados por cada map:

Aplicación PiEstimator				
Volumen de puntos a procesar	Map	Reduce	Tiempo medio ejecución	Desviación estándar
100.000.000 por map	4	1	108,31	0,12
1.000.000.000 por map	4	1	212,79	1,42
10.000.000.000 por map	4	1	1.268,29	8,68

Tabla A.20. Resultados obtenidos para la aplicación PiEstimator.

La Tabla A.21 muestra el tiempo invertido en cada fase.

Prueba	Inicio	Copia ficheros al HDFS	Ejecución	Copia resultados a local	Parada
Pi 100M	70,45	N/A	35,26	N/A	2,59
Pi 1.000M	70,42	N/A	139,74	N/A	2,63
Pi 10.000M	70,42	N/A	1.195,28	N/A	2,59

Tabla A.21. Segundos invertidos por cada fase en pruebas con PiEstimator.

La Figura A.6 muestra la distribución porcentual del consumo para cada una de las fases reflejadas en la Tabla A.21.

Figura A.6. Distribución porcentual del consumo obtenida para PiEstimator.

Para finalizar con las pruebas sobre myHadoop, se presentan los resultados sobre un clúster con HDFS persistente.

La Tabla A.22 muestra los resultados obtenidos para la aplicación Sort:

Aplicación Sort				
Volumen de datos a clasificar	Map	Reduce	Tiempo medio ejecución	Desviación estándar
1Gb (16 bloques de 64Kb)	16	7	297,25	12,53
2Gb (32 bloques de 64Kb)	32	7	575,06	28.03
4Gb (64 bloques de 64Kb)	64	7	1202,22	17,90

Tabla A.22. Resultados obtenidos para la aplicación Sort.

La Tabla A.23 muestra el tiempo invertido en cada fase.

Prueba	Inicio	Copia ficheros al HDFS	Ejecución	Copia resultados a local	Parada
Sort 1Gb	55,60	N/A	234,89	N/A	6,75
Sort 2Gb	55,50	N/A	505,25	N/A	14,31
Sort 4Gb	55,79	N/A	1.129,44	N/A	16,98

Tabla A.23. Segundos invertidos por cada fase en pruebas con Sort.

La Figura A.7 muestra la distribución porcentual del consumo para cada una de las fases de la Tabla A.23.

Figura A.7. Distribución porcentual del consumo obtenida para PiEstimator.

La Tabla A.24 muestra los resultados obtenidos para la aplicación WordCount:

Aplicación WordCount				
Volumen de datos a procesar	**Map**	**Reduce**	**Tiempo medio ejecución**	**Desviación estándar**
1Gb (16 bloques de 64Kb)	16	1	198,02	1,86
2Gb (34 bloques de 64Kb)	34	1	309,86	4,12
4Gb (68 bloques de 64Kb)	68	1	512,16	3,76

Tabla A.24. Resultados obtenidos para la aplicación WordCount.

La Tabla A.25 muestra el tiempo invertido en cada fase:

Prueba	Inicio	Copia ficheros al HDFS	Ejecución	Copia resultados a local	Parada
WC 1Gb	54,64	N/A	141,58	N/A	1,80
WC 2Gb	54,76	N/A	253,20	N/A	1,91
WC 4Gb	54,91	N/A	454,35	N/A	2,91

Tabla A.25. Segundos invertidos por cada fase en pruebas con WordCount.

La Figura A.8 muestra la distribución porcentual del consumo para cada una de las fases de la Tabla A.25.

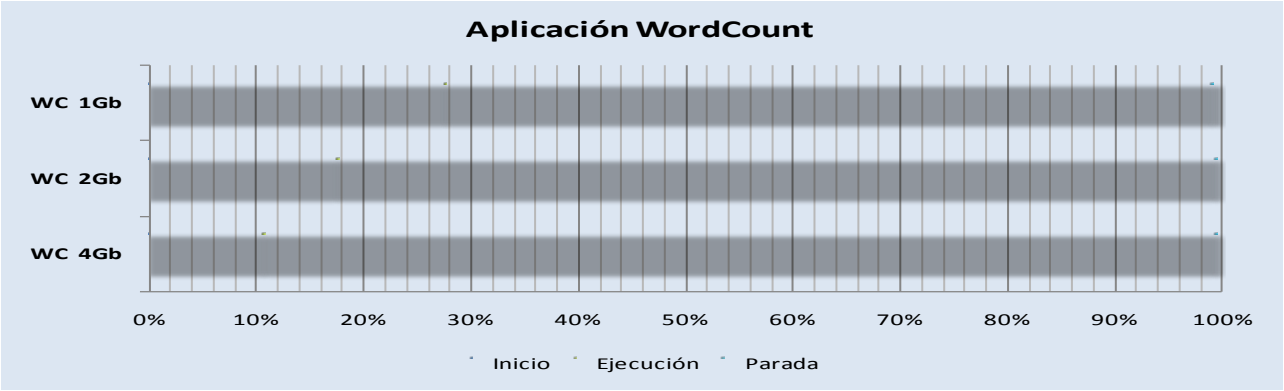


Figura A.8. Distribución porcentual del consumo obtenida para WordCount.

La Tabla A.26 muestra los resultados obtenidos para la aplicación PiEstimator:

Aplicación PiEstimator				
Volumen de puntos a procesar	Map	Reduce	Tiempo medio ejecución	Desviación estándar
100.000.000 por map	4	1	90,97	0,09
1.000.000.000 por map	4	1	194,66	2,33
10.000.000.000 por map	4	1	1.180,42	7,35

Tabla A.26. Resultados obtenidos para la aplicación PiEstimator.

La Tabla A.27 muestra el tiempo invertido en cada fase.

Prueba	Inicio	Copia ficheros al HDFS	Ejecución	Copia resultados a local	Parada
Pi 100M	54,51	N/A	35,07	N/A	1,39
Pi 1.000M	54,50	N/A	138,80	N/A	1,36
Pi 10.000M	54,50	N/A	1124,48	N/A	1,43

Tabla A.27. Segundos invertidos por cada fase en pruebas con PiEstimator.

La Figura A.9 muestra la distribución porcentual del consumo para cada una de las fases reflejadas en la Tabla A.27.

Figura A.9. Distribución porcentual del consumo obtenida para PiEstimator.

B. Preparación ambiente de pruebas

A continuación se exponen las tareas previas realizadas en los nodos del clúster de pruebas antes de proceder a la instalación de Hadoop, HoD o myHadoop.

Preparación entorno linux

Una vez instalado el sistema operativo, se ha procedido a realizar las siguientes tareas en cada uno de los nodos del clúster:

- Se agrega a Canonical Partner Repository como el repositorio por defecto de software.

```
> sudo add-apt-repository "deb http://archive.canonical.com/ lucid
partner"
```

- Se actualiza la lista de paquetes de software disponible.

```
> sudo apt-get update
```

- Se instala Java 6 dado que Hadoop necesita una JVM para ejecutarse.

```
> sudo apt-get install sun-java6-jdk
```

- Se marca Java 6 como la versión por defecto.

```
> sudo update-java-alternatives -s java-6-sun
```

- Se crea el usuario y grupo hadoop bajo el cual se realizan todas las pruebas del proyecto:

```
> sudo addgroup hadoop
> sudo adduser --ingroup hadoop hadoop
```

- Se edita el fichero */etc/hosts* para asignar a cada hostname la ip correspondiente. De esa manera, la referencia a un nodo se realizará por hostname y no por ip. Esto facilita el mantenimiento del software que se instalará posteriormente dado que, en caso de modificarse la ip de alguno de los nodos, sólo será necesario modificar dicho fichero y no los ficheros de configuración de los diferentes productos instalados. A continuación se muestra el contenido de dicho fichero:

```
> cat /etc/hosts
127.0.0.1       localhost
192.168.1.35    NodoMaster
192.168.1.36    NodeSlave1
192.168.1.37    NodeSlave2
192.168.1.38    NodeSlave3
```

- En el caso de Ubuntu, es recomendable deshabilitar el protocolo IPv6, dado que se han documentado conflictos entre Hadoop y Ubuntu en el caso de estar habilitado. Para ello se deben añadir las siguientes líneas en el fichero */etc/sysctl.conf*:

```
> cat /etc/hosts
#disable ipv6
net.ipv6.conf.all.disable_ipv6 = 1
net.ipv6.conf.default.disable_ipv6 = 1
net.ipv6.conf.lo.disable_ipv6 = 1
```

Hadoop necesita acceder vía SSH para gestionar los nodos del clúster. Para ello, en el nodo NodoMaster y conectados con el usuario hadoop, se creará una clave SSH al usuario Hadoop:

```
> ssh-keygen -t rsa -P ""
Generating public/private rsa key pair.
Enter file in which to save the key
(/home/hadoop/.ssh/id_rsa):
Your identification has been saved in
/home/hadoop/.ssh/id_rsa.
Your public key has been saved in
/home/hadoop/.ssh/id_rsa.pub.
The key fingerprint is:
96:ea:10:06:76:94:f1:94:55:2a:19:8c:a8:13:57:4c
hadoop@NodoMaster
The key's randomart image is:
+--[ RSA 2048]----+
|   =E=oo...      |
|. o.+ooo .       |
| +o . + .        |
|o. o   . .       |
| .  o   S        |
|   . . o         |
|    . .          |
|     o           |
|      .          |
+-----------------+
```

Y se copiará la clave generada al fichero de claves autorizadas de la propia máquina y al resto de nodos del clúster (el comando *ssh-copy-id* solicitará el password del usuario hadoop):

```
> cat $HOME/.ssh/id_rsa.pub >> $HOME/.ssh/authorized_keys
> ssh-copy-id -i $HOME/.ssh/id_rsa.pub hadoop@NodeSlave1
> ssh-copy-id -i $HOME/.ssh/id_rsa.pub hadoop@NodeSlave2
> ssh-copy-id -i $HOME/.ssh/id_rsa.pub hadoop@NodeSlave3
```

Se verificará, mediante el comando *ssh*, que se puede establecer la conexión con el resto de nodos:

```
hadoop@NodoMaster:~$ ssh NodeSlave1
Linux NodeSlave1 2.6.32-24-generic-pae #39-Ubuntu SMP Wed Jul 28 07:39:26
UTC 2010 i686 GNU/Linux
Ubuntu 10.04.1 LTS

Welcome to Ubuntu!
 * Documentation:  https://help.ubuntu.com/

  System information as of Tue Jun 14 20:13:22 CEST 2011

  System load:  0.45              Processes:           95
  Usage of /:   21.2% of 33.87GB  Users logged in:     1
  Memory usage: 3%                IP address for lo:   127.0.0.1
  Swap usage:   0%                IP address for eth0: 192.168.1.36

  Graph this data and manage this system at
https://landscape.canonical.com/

111 packages can be updated.
70 updates are security updates.

Last login: Tue Jun 14 20:12:49 2011 from nodomaster
hadoop@NodeSlave1:~$
```

Instalación de Python

Para poder utilizar HoD, es necesario tener instalado Python en el sistema. Antes de proceder a su instalación es necesario que exista un compilador de C. También ha sido necesario instalar la librería zlib para poder realizar las pruebas de HoD pasándole como parámetro un tarball con los ficheros de instalación de Hadoop. De lo contrario, HoD no podrá descomprimir y desplegar el tarball y fallará el proceso de generación dinámica del clúster.

Para instalar tanto el compilador C como la librería zlib en Ubuntu, se han ejecutado los siguientes comandos:

```
> sudo apt-get install build-essential gcc
> sudo apt-get install zlib1g-dev
```

A continuación se procede a instalar Python ejecutando los siguientes comandos:

```
> wget http://www.python.org/ftp/python/2.5.1/Python-2.5.1.tgz
> tar -xvzf Python-2.5.1.tgz
> cd Python-2.5.1
> ./configure --prefix=/usr/local/python2.5
> make
> make test
> sudo make install
> sudo ln -s /usr/local/python2.5/bin/python /usr/bin/python2.5
```

Instalación de Torque

Para instalar Torque, primero se debe descargar el tarball con los ficheros de instalación en un directorio temporal de la máquina NodoMaster:

```
> wget http://www.clusterresources.com/downloads/torque/torque-
3.0.0.tar.gz
```

A continuación se despliega el tarball y se ejecuta el comando *configure*:

```
> tar -xzvf torque-3.0.0.tar.gz
> cd torque-3.0.0/
> ./configure
config.status: executing depfiles commands

Building components: server=yes mom=yes clients=yes
                     gui=no drmaa=no pam=no
PBS Machine type: linux
Remote copy: /usr/bin/scp -rpB
PBS home: /var/spool/torque
Default server: NodoMaster
Unix Domain sockets: no
Tcl: disabled
Tk: disabled

configure: WARNING: This compilation has strict compiler
options enabled that cause
the build to fail if any compiler warnings are emitted.  If
this build fails
because of a harmless warning, please report the problem to
torqueusers@supercluster.org
and run configure again with --disable-gcc-warnings.

Ready for 'make'.
```

Una vez ejecutado, nos aparece un mensaje anunciando que se está en disposición de ejecutar el make que realiza la compilación de todos los componentes del paquete. Una vez realizada la compilación, se vuelve a ejecutar *make* con la opción *install* para instalar Torque en el nodo maestro:

```
> make
> sudo make install
```

A continuación se debe proceder a generar los ficheros a instalar en el resto de nodos. Para ello se ejecuta el comando *make* con la opción *packages*:

```
> make packages
Building packages from /tmp/torque-3.0.0/tpackages
rm -rf /tmp/torque-3.0.0/tpackages
mkdir /tmp/torque-3.0.0/tpackages
Building ./torque-package-server-linux-i686.sh ...
libtool: install: warning: remember to run `libtool --finish
/usr/local/lib'
Building ./torque-package-mom-linux-i686.sh ...
libtool: install: warning: remember to run `libtool --finish
/usr/local/lib'
Building ./torque-package-clients-linux-i686.sh ...
libtool: install: warning: remember to run `libtool --finish
/usr/local/lib'
Building ./torque-package-devel-linux-i686.sh ...
libtool: install: warning: remember to run `libtool --finish
/usr/local/lib'
Building ./torque-package-doc-linux-i686.sh ...
Done.

The package files are self-extracting packages that can be copied
and executed on your production machines.  Use --help for options.
```

Una vez finalizada la ejecución del comando, es posible instalar los ficheros en el resto de nodos vía ssh:

```
> for i in NodeSlave1 NodeSlave2 NodeSlave3 ; do scp torque-package-
mom-linux-i686.sh ${i}:/tmp/. ; done

> for i in NodeSlave1 NodeSlave2 NodeSlave3 ; do scp torque-package-
clients-linux-i686.sh ${i}:/tmp/. ; done

> for i in NodeSlave1 NodeSlave2 NodeSlave3 ; do ssh ${i}
/tmp/torque-package-mom-linux-i686.sh --install ; done

> for i in NodeSlave1 NodeSlave2 NodeSlave3 ; do ssh ${i}
/tmp/torque-package-clients-linux-i686.sh --install ; done
```

Una vez instalado Torque, se debe proceder a configurarlo. Primero, se debe ejecutar el siguiente comando para generar una configuración mínima:

```
> sudo pbs_server -t create
```

Seguidamente se añaden los nodos que Torque gestionará editando el fichero */var/spool/torque/server_priv/nodes*. El contenido del fichero queda como se muestra a continuación:

```
> cat /var/spool/torque/server_priv/nodes
NodoMaster
NodeSlave1
NodeSlave2
NodeSlave3
```

Finalizada la configuración, se está en disposición de ejecutar los demonios de Torque. En el nodo master se ejecutará el servidor (pbs_server) y el planificador (pbs_sched). También se ejecutará el mini-monitor de ejecución batch (pbs_mom) dado que el nodo maestro también actuará como nodo de cómputo:

```
> sudo qterm -t quick
> sudo pbs_server
> sudo pbs_sched
> sudo pbs_mom
```

En el resto de nodos se ejecutará únicamente el mini-monitor de ejecución batch (pbs_mom).

```
> sudo pbs_mom
```

Para finalizar, se configura cola de trabajos contra la cual se lanzarán los trabajos a ejecutar. A la cola se le ha llamado “batch”:

```
> sudo qmgr -c "set server scheduling=true"
> sudo qmgr -c "create queue batch queue_type=execution"
> sudo qmgr -c "set queue batch started=true"
> sudo qmgr -c "set queue batch enabled=true"
> sudo qmgr -c "set queue batch resources_default.nodes=1"
> sudo qmgr -c "set queue batch resources_default.walltime=3600"
> sudo qmgr -c "set server default_queue=batch"
```

Con el comando qstat se puede consultar el estado de la cola:

```
> qstat –q

server: NodoMaster

Queue            Memory CPU Time Walltime Node  Run Que Lm  State
---------------- ------ -------- -------- ----  --- --- --  -----
batch              --      --       --      --    0   0 --   E R
                                                ----- -----
                                                    0     0
```

C. Preparación clúster Hadoop dedicado

Instalación y configuración

La instalación de Hadoop únicamente requiere de la descarga del tarball que contiene la versión estable y desplegarlo en el directorio deseado. Este proceso se deberá repetir en cada uno de los nodos que formarán parte del clúster:

```
> wget http://apache.mirrors.tds.net//hadoop/core/stable/hadoop-
0.20.2.tar.gz

> tar xzf hadoop-0.20.2.tar.gz
```

Se crea la variable de entorno $HADOOP_HOME cuyo contenido corresponde al directorio en el cual se ha instalado Hadoop. En el presente documento se hará referencia al directorio de instalación mediante dicha variable.

En el directorio $HADOOP_HOME/conf están localizados una serie de ficheros en formato XML que permiten configurar cada uno de los elementos que forman Hadoop:

- *core-site.sh* donde se definen propiedades core de Hadoop.
- *hdfs-site.xml* donde se definen las opciones de configuración para el HDFS.
- *mapred-site.xml* donde se definen las opciones de configuración relativas a la ejecución de tareas MapReduce.

En el presente proyecto se ha utilizado la configuración por defecto que proporciona Hadoop, por lo que no se ha incorporado en los ficheros de configuración ninguna propiedad adicional, exceptuando la configuración del HDFS, que por defecto realiza tres réplicas de cada bloque grabado en el sistema de ficheros. Dicha característica (propiedad *dfs.replication*) ha sido modificada para que no realice réplicas.

En el directorio $HADOOP_HOME/conf se encuentran además los siguientes ficheros:

- *masters.* El nombre de este fichero puede inducir a error dado que en este fichero se define el nodo en el cual se ejecutará el *SecondaryNameNode.*
- *slaves* que contiene la relación de nodos en los cuales se ejecutarán tareas MapReduce.

Una vez configurado el clúster, se procede a formatear el nuevo sistema de ficheros distribuido mediante el siguiente comando:

```
> $HADOOP_HOME/bin/hadoop namenode -format
```

Inicialización del clúster

En este momento ya se está en disposición de inicializar los procesos de Hadoop mediante el siguiente comando:

```
> $HADOOP_HOME/bin/start-all.sh
```

La ejecución del comando anterior lanza los demonios Hadoop en cada uno de los nodos del clúster, como se puede apreciar en la salida que genera el comando:

```
> $HADOOP_HOME/bin/start-all.sh
hadoop@NodoMaster:~$ start-all.sh
starting namenode, logging to /usr/local/hadoop/bin/../logs/hadoop-hadoop-
namenode-NodoMaster.out
NodoMaster: starting datanode, logging to
/usr/local/hadoop/bin/../logs/hadoop-hadoop-datanode-NodoMaster.out
NodeSlave2: starting datanode, logging to
/usr/local/hadoop/bin/../logs/hadoop-hadoop-datanode-NodeSlave2.out
NodeSlave1: starting datanode, logging to
/usr/local/hadoop/bin/../logs/hadoop-hadoop-datanode-NodeSlave1.out
NodeSlave3: starting datanode, logging to
/usr/local/hadoop/bin/../logs/hadoop-hadoop-datanode-NodeSlave3.out
starting jobtracker, logging to /usr/local/hadoop/bin/../logs/hadoop-hadoop-
jobtracker-NodoMaster.out
NodeSlave3: starting tasktracker, logging to
/usr/local/hadoop/bin/../logs/hadoop-hadoop-tasktracker-NodeSlave3.out
NodoMaster: starting tasktracker, logging to
/usr/local/hadoop/bin/../logs/hadoop-hadoop-tasktracker-NodoMaster.out
NodeSlave2: starting tasktracker, logging to
/usr/local/hadoop/bin/../logs/hadoop-hadoop-tasktracker-NodeSlave2.out
NodeSlave1: starting tasktracker, logging to
/usr/local/hadoop/bin/../logs/hadoop-hadoop-tasktracker-NodeSlave1.out
```

Dado que los demonios Hadoop son procesos java, con el comando *jps* se pueden obtener los procesos java que se están ejecutando en una máquina junto con su pid:

```
> jps
1624 Jps
1033 NameNode
1189 DataNode
1297 JobTracker
1455 TaskTracker
```

Ejecución de trabajos

La ejecución de trabajos se realiza mediante el comando *hadoop*. Se le debe indicar como parámetro el jar que contiene la aplicación a ejecutar, tal y como se muestra a continuación para la aplicación Sort:

```
> $HADOOP_HOME/bin/hadoop jar /usr/local/hadoop-0.20.2/hadoop-*-
examples.jar sort Sort1Gb Sort1Gb-Out
```

Cierre del clúster

Para finalizar la ejecución del clúster y liberar los recursos asignados se debe invocar al siguiente comando:

```
> $HADOOP_HOME/bin/stop-all.sh
stopping jobtracker
NodeSlave1: stopping tasktracker
NodeSlave2: stopping tasktracker
NodeSlave3: stopping tasktracker
NodoMaster: stopping tasktracker
stopping namenode
NodeSlave3: stopping datanode
NodoMaster: stopping datanode
NodeSlave2: stopping datanode
NodeSlave1: stopping datanode
```

D. Preparación entorno HoD

Instalación y configuración

La herramienta HoD reside en el mismo tarball utilizado para la instalación de Hadoop. Se encuentra ubicada bajo el directorio contrib/hod y se puede instalar de dos modos:

- Extrayendo del tarball únicamente los ficheros ubicados en el directorio indicado y desplegándolos en los nodos desde los cuales el usuario lanzará trabajos MapReduce mediante HoD.
- Realizando la Instalación completa de Hadoop.

Dado que en el apartado anterior se ha procedido a crear un clúster Hadoop dedicado y por tanto se ha desplegado todos los ficheros contenidos en el tarball de instalación, se aprovechará la infraestructura creada para realizar las pruebas con HoD. Para facilitar la localización de la herramienta HoD se crea el enlace simbólico /usr/local/hod:

```
> ln –sf $HADOOP_HOME/contrib/hod /usr/local/hod

> ll /usr/local/hod
lrwxrwxrwx 1 hadoop hadoop 18 2011-03-12 17:33 /usr/local/hod -> hadoop/contrib/hod/
```

Para poder ejecutar HoD es requisito indispensable que los siguientes componentes se encuentren instalados en todos los nodos de cómputo:

- El gestor de recursos Torque.
- Python. HoD requiere que la versión 2.5.1 de este lenguaje de programación.

A continuación se procede configurar las siguientes variables de entorno que son requeridas por HoD para su funcionamiento:

- HOD_HOME: contiene la ruta donde reside la instalación de HoD.
- HOD_PYTHON_HOME: contiene la ruta donde reside la instalación de Python.
- HOD_CONF_DIR: contiene la ruta donde reside el fichero de configuración hodrc.
- HADOOP_CONF_DIR: contiene la ruta en la cual HoD generará los ficheros de configuración necesarios para ejecutar Hadoop.

La tabla siguiente muestra la configuración de variables de entorno utilizara para el presente proyecto:

Variable de entorno	Contenido
HOD_HOME	/usr/local/hod
HOD_PYTHON_HOME	/usr/local/python2.5/bin/python
HOD_CONF_DIR	${HOD_HOME}/conf
HADOOP_CONF_DIR	/DataStores/hod

En el directorio $HOD_CONF_DIR se encuentra el fichero *hodrc* que contiene la configuración mínima necesaria para poder ejecutar HoD. Este fichero se debe modificar en el nodo desde donde se desea ejecutar HoD, en el caso del presente proyecto se realizará desde el nodo NodoMaster. Básicamente consiste en sustituir las siguientes variables por el valor correspondiente:

- ${JAVA_HOME}: sustituir por la ruta en la que reside Java.
- ${CLUSTER_NAME}: nombre asignado al clúster en Torque (propiedad 'node property').
- ${HADOOP_HOME}: directorio en el que se encuentra instalado Hadoop.
- ${RM_QUEUE}: Cola configurada en Torque para lanzar los trabajos asociados a HoD.
- ${RM_HOME}: directorio en el que se encuentran instalados los binarios de Torque.

Finalizado el proceso de instalación y configuración, se está en disposición de iniciar la ejecución de HoD.

Para configurar el clúster en modo persistente, se debe proporcionar en la sección *[gridservice-hdfs]* del fichero de configuración *hodrc* la localización del NameNode mediante las siguientes propiedades:

- *external*: debe valer ‘True’ para indicar que el clúster estará asociado a un HDFS externo.
- *host*: la dirección de la máquina en la que se está ejecutando el NameNode.
- *fs_port*: puerto por el cual el servidor RPC embebido del NameNode atiende las peticiones procedentes de los DataNodes. Debe coincidir con el puerto indicado en la propiedad *fs.default.name* del fichero de configuración *core-site.xml* del NameNode. En caso de no indicarse puerto en la propiedad, el NameNode toma por defecto el puerto 8020.
- *info_port*: puerto por el cual el servidor web embebido del NameNode atiende las peticiones procedentes de la interface web. Debe coincidir con el puerto indicado en la propiedad *dfs.http.address* del fichero de configuración *hdfs-site.xml* del NameNode. En caso de no especificarse toma por defecto el puerto 50070.

Inicialización del clúster

En el caso que Hadoop se encuentre preinstalado en todos los nodos, la inicialización del clúster se realiza mediante la operación *allocate* del siguiente modo:

```
> ${HOD_HOME}/bin/hod allocate -d $HADOOP_CONF_DIR –n <número de nodos>
```

El parámetro –n recibe el número de nodos con el cual se desea configurar un clúster Hadoop. La ejecución del script para una configuración de cuatro nodos, configuración utilizada en las pruebas realizadas en el presente proyecto, genera una salida del tipo que se muestra a continuación donde se puede observar que se procede a ejecutar el RingMaster:

```
> ${HOD_HOME}/bin/hod allocate -d $HADOOP_CONF_DIR –n 4
Using Python: 2.5.1 (r251:54863, Mar 12 2011, 18:57:07)
[GCC 4.4.3]

[2011-05-23 20:13:54,323] DEBUG/10 hadoop:479 - account validation script is run 0
[2011-05-23 20:13:54,324] DEBUG/10 hod:326 - verify-account returned zero exit code.
[2011-05-23 20:13:54,326] DEBUG/10 hod:192 - ('NodoMaster', 37726)
[2011-05-23 20:13:54,327] DEBUG/10 hod:354 - Service Registry started.
[2011-05-23 20:13:54,328] DEBUG/10 hadoop:496 - allocate /DataStores/hod 4 4
[2011-05-23 20:13:54,330] DEBUG/10 torque:79 - ringmaster cmd: /usr/local/hadoop-
0.20.2/contrib/hod/bin/ringmaster
. . .
```

Si el proceso de arranque finaliza correctamente, la salida del comando en su parte final muestra la ubicación (nodo y puerto) del RingMaster, del NameNode y del JobTracker:

```
. . .
[2011-05-23 20:13:56,672] DEBUG/10 hadoop:545 - Ringmaster at : http://NodeSlave3:64156/
[2011-05-23 20:13:58,922] INFO/20 hadoop:554 - HDFS UI at http://NodoMaster:50177
[2011-05-23 20:14:03,969] INFO/20 hadoop:560 - Mapred UI at http://NodeSlave2:54431
[2011-05-23 20:14:03,973] INFO/20 hadoop:622 - hadoop-site.xml at /DataStores/hod
[2011-05-23 20:14:04,969] DEBUG/10 hod:597 - return code: 0

>
```

En este punto el clúster Hadoop ya está totalmente disponible dado que se han arrancando todos los demonios Hadoop y se ha formateado el HDFS, encontrándose a disposición para atender trabajos MapReduce.

HoD crea toda la infraestructura necesaria para el clúster Hadoop creado al vuelo en el directorio /tmp de cada nodo, por lo que los usuarios que ejecutan HoD deben tener permisos en el directorio especificado. Por otro lado, el sistema de ficheros en el que reside el directorio /tmp debe haber sido dimensionado adecuadamente para almacenar toda la información generada por los trabajos ejecutados en el clúster temporal. El directorio de trabajo de Hadoop puede ser modificado por cualquier otro en el fichero de configuración *hodrc*.

En el caso que los nodos no dispongan de Hadoop preinstalado, a la hora de invocar a la operación *allocate* se le debe proporcionar como argumento la ubicación del tarball que contiene los ficheros necesarios para crear una instalación temporal de Hadoop:

```
> ${HOD_HOME}/bin/hod allocate -d $HADOOP_CONF_DIR –n <número de nodos> -t
<TARBALL_BASE_DIR>
```

Mediante el argumento –t se especifica el directorio en el que reside el tarball. Dicho directorio debe estar accesible por todos los nodos por lo que en el presente proyecto ha sido creado en la máquina NodoMaster y compartido mediante NFS con el resto de nodos.

El RingMaster desplegará el tarball en el directorio de trabajo especificado en el fichero de configuración (*hodrc*). La instalación temporal será destruida una vez se concluya con la ejecución del clúster.

Ejecución de trabajos

La ejecución de trabajos MapReduce se realizará de modo similar al expuesto para el clúster Hadoop dedicado pero con la salvedad que se debe indicar el directorio donde reside la configuración del propio clúster:

```
> hadoop --config ${HADOOP_CONF_DIR} jar /usr/local/hadoop-
0.20.2/hadoop-*-examples.jar sort Sort1Gb Sort1Gb-Out
```

Cierre del clúster

Para finalizar la ejecución del clúster y liberar los recursos asignados se debe invocar a la operación *deallocate* del siguiente modo:

```
> ${HOD_HOME}/bin/hod deallocate -d $HADOOP_CONF_DIR
```

En el caso que un clúster creado con HoD se encuentre inactivo más de un ahora, automáticamente se matan los demonios Hadoop para liberar los nodos. No obstante, el usuario deberá realizar de forma explícita la operación *deallocate* para liberar todos los recursos que HoD hubiera requerido: el espacio ocupado por la instalación temporal de Hadoop, ficheros temporales correspondientes a la ejecución de trabajos, dar por finalizado el trabajo en Torque, etc. En caso contrario, al intentar ejecutar de nuevo HoD para asignar un nuevo clúster por parte del mismo usuario propietario del clúster inactivo, HoD advertirá de tal circunstancia y no permitirá generar el nuevo clúster.

E. Preparación entorno myHadoop

Instalación y configuración

La instalación de Hadoop únicamente requiere de la descarga del tarball que contiene todo lo necesario para ejecutar el framework. Este proceso se deberá repetir en cada uno de los nodos que formarán parte del clúster:

```
> wget http://sourceforge.net/projects/myhadoop/files/0.2a/myHadoop-
0.2a.tar.gz/download

> tar xzf myHadoop-0.2a.tar.gz
```

Una vez instalado myHadoop se debe modificar el script bin/setenv.sh, que reside bajo el directorio en el cual se ha realizado la instalación, para configurar las siguientes variables de entorno que son requeridas por el framework para su funcionamiento:

- MY_HADOOP_HOME: contiene la ruta donde reside la instalación de myHadoop.
- HADOOP_HOME: contiene la ruta donde reside la instalación de Hadoop.
- HADOOP_DATA_DIR: contiene la ruta en la cual reside el HDFS. La ruta indicada debe corresponder con un directorio local y dicha ruta debe existir en cada uno de los nodos del clúster.
- HADOOP_LOG_DIR: contiene la ruta en la que se almacenan los ficheros de log de Hadoop.
- HADOOP_CONF_DIR: contiene la ruta en la cual los ficheros de configuración de Hadoop serán generados por myHadoop.

La tabla siguiente muestra la configuración de variables de entorno utilizara para el presente proyecto:

Variable de entorno	Contenido
MY_HADOOP_HOME	/usr/local/myHadoop
HADOOP_HOME	/usr/local/hadoop
HADOOP_DATA_DIR	/DataStores/hadoopPBS-$USER/data
HADOOP_LOG_DIR	/DataStores/hadoopPBS-$USER/log
HADOOP_CONF_DIR	/shared/hod/config

El directorio $HADOOP_CONF_DIR debe estar accesible para todos los nodos, por lo que se opta por ubicar el directorio en la máquina NodoMaster y compartirlo con el resto de nodos mediante NFS.

Para realizar las pruebas de rendimiento se ha procedido previamente a modificar los ficheros *core-site.xml*, *hdfs-site.xml* y *mapred-site.xml* residentes en $MY_HADOOP_HOME/etc para eliminar las propiedades que por defecto tiene configuradas myHadoop con el objetivo de crear un escenario similar al clúster Hadoop dedicado del anexo D.

Inicialización del clúster

Mediante la ejecución del script $MY_HADOOP_HOME/bin/pbs-configure.sh se inicia el proceso de configuración e inicialización del clúster. Este script realiza las siguientes acciones:

- Genera los ficheros de configuración en el directorio $HADOOP_CONF_DIR.
- Borrado de los directorios $HADOOP_LOG_DIR y $HADOOP_DATA_DIR para eliminar la información correspondiente a una ejecución anterior que pudiera residir en dichos directorios.
- Creación de los directorios $HADOOP_LOG_DIR y $HADOOP_DATA_DIR para el clúster actual.

Para un clúster Hadoop en modo no persistente la invocación del script se realiza del siguiente modo:

```
> $MY_HADOOP_HOME/bin/pbs-configure.sh  -n <número de nodos> -c $HADOOP_CONF_DIR
```

El parámetro –n recibe el número de nodos con el cual se desea configurar un clúster Hadoop. La ejecución del script para una configuración de cuatro nodos, configuración utilizada en las pruebas realizadas en el presente proyecto, genera la siguiente salida:

```
> $MY_HADOOP_HOME/bin/pbs-configure.sh  -n 4 -c $HADOOP_CONF_DIR
Set up the configurations for myHadoop
Number of Hadoop nodes requested: 4
Generation Hadoop configuration in directory: /shared/hod/config
Not persisting HDFS state
Received 4 nodes from PBS
Master is: NodoMaster
Configuring node: NodoMaster
rm -rf /DataStores/hadoopPBS-hadoop/log; mkdir -p /DataStores/hadoopPBS-hadoop/log
rm -rf /DataStores/hadoopPBS-hadoop/data; mkdir -p /DataStores/hadoopPBS-hadoop/data
Configuring node: NodeSlave1
rm -rf /DataStores/hadoopPBS-hadoop/log; mkdir -p /DataStores/hadoopPBS-hadoop/log
rm -rf /DataStores/hadoopPBS-hadoop/data; mkdir -p /DataStores/hadoopPBS-hadoop/data
Configuring node: NodeSlave2
rm -rf /DataStores/hadoopPBS-hadoop/log; mkdir -p /DataStores/hadoopPBS-hadoop/log
rm -rf /DataStores/hadoopPBS-hadoop/data; mkdir -p /DataStores/hadoopPBS-hadoop/data
Configuring node: NodeSlave3
rm -rf /DataStores/hadoopPBS-hadoop/log; mkdir -p /DataStores/hadoopPBS-hadoop/log
rm -rf /DataStores/hadoopPBS-hadoop/data; mkdir -p /DataStores/hadoopPBS-hadoop/data
```

A continuación se procede al formateo del HDFS para el nuevo clúster:

```
> $HADOOP_HOME/bin/hadoop --config $HADOOP_CONF_DIR namenode –format
Format HDFS
11/05/21 19:41:37 INFO namenode.NameNode: STARTUP_MSG:
/************************************************************
STARTUP_MSG: Starting NameNode
STARTUP_MSG:   host = NodoMaster/192.168.1.34
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
************************************************************/
11/05/21 19:41:37 INFO namenode.FSNamesystem: fsOwner=hadoop,hadoop
11/05/21 19:41:37 INFO namenode.FSNamesystem: supergroup=supergroup
11/05/21 19:41:37 INFO namenode.FSNamesystem: isPermissionEnabled=true
11/05/21 19:41:37 INFO common.Storage: Image file of size 96 saved in 0 seconds.
11/05/21 19:41:37 INFO common.Storage: Storage directory /DataStores/hadoopPBS-
hadoop/data/dfs/name has been successfully formatted.
11/05/21 19:41:37 INFO namenode.NameNode: SHUTDOWN_MSG:
/************************************************************
```

En el modo persistente la invocación del script pbs-configure.sh se realiza del siguiente modo:

```
> $MY_HADOOP_HOME/bin/pbs-configure.sh  -n <número de nodos> -c $HADOOP_CONF_DIR –p
–d <HDFS_BASE_DIR>
```

Al igual que ocurre con el directorio $HADOOP_CONF_DIR, la ruta correspondiente a <HDFS_BASE_DIR> debe ser accesible por todos los nodos del clúster, por lo que se ha creado de la misma forma que el directorio de ficheros de configuración, es decir, se ha creado en la máquina NodoMaster y se ha compartido mediante NFS con el resto de nodos.

Para generar el HDFS se deben crear los subdirectorios 1,2,…,N bajo el directorio <HDFS_BASE_DIR>, siendo N el número de nodos de los que constará el clúster inicializado por myHadoop. Tras ejecutar el script pbs-configure.sh del modo indicado para el modo persistente, se procede al formateo del HDFS. El formateo sólo se debe realizar en la primera ejecución del clúster en modo persistente, omitiéndose dicho paso en el resto de ejecuciones. El modo en el que se configura el HDFS en el modo persistente obliga a utilizar siempre el mismo número de nodos a la hora de inicializar el clúster.

A continuación se muestra la salida que genera la ejecución del script pbs-configure.sh para el modo persistente donde se puede observar que se están creando enlaces simbólicos desde el sistema de ficheros local de cada uno de los nodos al directorio compartido en el que reside el HDFS:

```
> $MY_HADOOP_HOME/bin/pbs-configure.sh -n 4 -c $HADOOP_CONF_DIR -p -d
/shared/DataStores/myHadoop/data

Set up the configurations for myHadoop
Number of Hadoop nodes requested: 4
Generation Hadoop configuration in directory: /shared/hod/config
Persisting HDFS state (-p)
Using directory /shared/DataStores/myHadoop/data for persisting HDFS state
Received 4 nodes from PBS
Master is: NodoMaster
Configuring node: NodoMaster
rm -rf /DataStores/hadoopPBS-hadoop/log; mkdir -p /DataStores/hadoopPBS-hadoop/log
rm -rf /DataStores/hadoopPBS-hadoop/data; ln -s /shared/DataStores/myHadoop/data/1
/DataStores/hadoopPBS-hadoop/data
Configuring node: NodeSlave1
rm -rf /DataStores/hadoopPBS-hadoop/log; mkdir -p /DataStores/hadoopPBS-hadoop/log
rm -rf /DataStores/hadoopPBS-hadoop/data; ln -s /shared/DataStores/myHadoop/data/2
/DataStores/hadoopPBS-hadoop/data
Configuring node: NodeSlave2
rm -rf /DataStores/hadoopPBS-hadoop/log; mkdir -p /DataStores/hadoopPBS-hadoop/log
rm -rf /DataStores/hadoopPBS-hadoop/data; ln -s /shared/DataStores/myHadoop/data/3
/DataStores/hadoopPBS-hadoop/data
Configuring node: NodeSlave3
rm -rf /DataStores/hadoopPBS-hadoop/log; mkdir -p /DataStores/hadoopPBS-hadoop/log
rm -rf /DataStores/hadoopPBS-hadoop/data; ln -s /shared/DataStores/myHadoop/data/4
/DataStores/hadoopPBS-hadoop/data
```

Llegados a este punto, ya se está en disposición de iniciar los demonios de Hadoop de la manera estándar invocando al script $HADOOP_HOME/bin/start-all.sh. Una vez iniciados dichos demonios el clúster está listo para ejecutar trabajos MapReduce.

Ejecución de trabajos

La ejecución de trabajos MapReduce se realizará de modo similar al expuesto para el clúster Hadoop dedicado pero con la salvedad que se debe indicar el directorio donde reside la configuración del propio clúster:

```
> hadoop --config ${HADOOP_CONF_DIR} jar /usr/local/hadoop-
0.20.2/hadoop-*-examples.jar sort Sort1Gb Sort1Gb-Out
```

Cierre del clúster

Para finalizar la ejecución del clúster y liberar los recursos asignados se debe proceder del siguiente modo:

- Ejecución del script $HADOOP_HOME/bin/stop-all.sh para matar los demonios asociados a Hadoop (JobTracker, TaskTracker, NameNode y DataNode).
- Ejecución del script $MY_HADOOP_HOME/bin/pbs-cleanup.sh que borra los directorios creados en el sistema de ficheros local de cada uno de los nodos:

```
> $MY_HADOOP_HOME/bin/pbs-cleanup.sh -n 4
Clean up
Number of Hadoop nodes specified by user: 4
Received 4 nodes from PBS
Clean up node: NodoMaster
rm -rf /DataStores/hadoopPBS-hadoop/data /DataStores/hadoopPBS-hadoop/log
Clean up node: NodeSlave1
rm -rf /DataStores/hadoopPBS-hadoop/data /DataStores/hadoopPBS-hadoop/log
Clean up node: NodeSlave2
rm -rf /DataStores/hadoopPBS-hadoop/data /DataStores/hadoopPBS-hadoop/log
Clean up node: NodeSlave3
rm -rf /DataStores/hadoopPBS-hadoop/data /DataStores/hadoopPBS-hadoop/log
=========================================
```

Resumen

Cada vez es mayor el número de aplicaciones desarrolladas en el ámbito científico, como en la Bioinformática o en las Geociencias, escritas bajo el modelo MapReduce, empleando herramientas de código abierto como Apache Hadoop.

De la necesidad de integrar Hadoop en entornos HPC, para posibilitar la ejecutar aplicaciones desarrolladas bajo el paradigma MapReduce, nace el presente proyecto. Se analizan dos frameworks diseñados para facilitar dicha integración a los desarrolladores: HoD y myHadoop.

En este proyecto se analiza, tanto las posibilidades en cuanto a entornos que ofrecen dichos frameworks para la ejecución de aplicaciones MapReduce, como el rendimiento de los clúster Hadoop generados con HoD o myHadoop respecto a un clúster Hadoop físico.

Resum

Cada cop és més gran el número d'aplicacions desenvolupades a l'àmbit científic, com la Bioinformàtica o les Geociències, escrites sota el model MapReduce, fent servir eines de codi obert com Apache Hadoop.

De la necessitat d'integrar Hadoop en entorns HPC, per permetre l'execució d'aplicacions desenvolupades sota el paradigma MapReduce, neix el present projecte. S'analitzen dos frameworks dissenyats per facilitar aquesta integració als desenvolupadors: HoD y myHadoop.

En aquest projecte s'analitza, tant les possibilitats en quan a entorns que ofereixen aquests frameworks per l'execució d'aplicacions MapReduce, com el rendiment dels clústers Hadoop generats amb HoD o myHadoop comparat amb el rendiment d'un clúster Hadoop físic.

Abstract

A growing number of codes in scientific domain such a Bioinformatics and Geosciences are being written using open source MapReduce tools such as Apache Hadoop.

Of the need to integrate Hadoop in HPC environments, to make possible to execute applications developed under the MapReduce paradigm, born this project. Two frameworks, designed to facilitate the above mentioned integration to the developers, are analyzed: HoD and myHadoop.

In this project, we analyze the possible environments that can be generated with these frameworks, for the execution of MapReduce applications, and the performance of the Hadoop clusters generated with HoD or myHadoop in comparison with a physical Hadoop cluster.