ANÁLISIS BIOINFORMÁTICOS SOBRE LA TECNOLOGÍA HADOOP

Memoria del proyecto

de Ingeniería Técnica en

Informática de Sistemas

realizado por

Albert Gutiérrez Millà

y dirigido por

Antonio Espinosa Morales

Escola d'Enginyeria

Sabadell, junio de 2011

El abajo firmante, Antonio Espinosa Morales,

profesor de la Escola Universitària d'Informàtica de la UAB,

CERTIFICA:

Que el trabajo al que corresponde la presente memoria
ha estado realizado bajo su dirección
por *Albert Gutiérrez Millà*

Y para que conste firma la presente.
Sabadell, junio de 2011

Firma: Antonio Espinosa Morales

Título del proyecto: Análisis bioinformáticos sobre la tecnología Hadoop

Autor: Albert Gutiérrez Millà

Fecha: junio del 2011

Tutor: Antonio Espinosa Morales

Titulación: Ingeniería Informática técnica de sistemas

Palabras clave:

- Catalán: Apache Hadoop, MapReduce, sistemes distribuïts, alineament de seqüències, Smith-Waterman.
- Castellano: Apache Hadoop, MapReduce, sistemas distribuidos, alineamiento de secuencias, Smith-Waterman.
- Inglés: Apache Hadoop, MapReduce, distributed systems, sequence alignment, Smith-Waterman.

Resumen del proyecto

- Catalán: Des del principi del projecte del genoma humà i el seu éxit l'any 2001 s'han seqüenciat genomas de multitud d'espècies. La millora en les tecnologies de seqüenciació ha generat volums de dades amb un creixement exponencial. El projecte *Análisis bioinformáticos sobre la tecnología Hadoop* abarca la computació paralel·la de dades biològiques com són les seqüències d'ADN. L'estudi ha sigut guiat per la naturalesa del problema a resoldre. L'alineament de seqüències genètiques amb el paradigma MapReduce.
- Castellano: Desde el inicio del proyecto del genoma humano y su éxito en el año 2001 se han secuenciado genomas de multitud de especies. La mejora en las tecnologías de secuenciación ha generado volúmenes de datos con un crecimiento exponencial. El proyecto *Análisis bioinformáticos sobre la tecnología Hadoop* abarca la computación paralela de datos biológicos como son las secuencias de ADN. El estudio ha sido encauzado por la naturaleza del problema a resolver. El alineamiento de secuencias genéticas con el paradigma MapReduce.
- Inglés: Since the human genome project started, and its success in 2011 many species have been sequenced. The improve in sequence tecnologies have generated big ammounts of data with an exponencial growing. The project *Análisis bioinformáticos sobre la tecnología Hadoop* explains paralel computing of biological data such as DNA. The study has been moved by the origin of the problem. The genetic sequence alignment with the MapReduce paradigm.

Índice

Capítulo 1

Introducción

1.1. Motivación y objetivos

El objetivo del proyecto *Análisis bioinformáticos sobre la tecnología Hadoop* es el alineamiento de secuencias proteicas. Los genomas ocupan ficheros del orden de gigabytes. 3.200 MB en el genoma humano. Para poder analizar conjuntos genómicos comparándolos mutuamente es necesario recurrir a los sistemas distribuidos. Una única computadora no puede abarcar el cómputo.

La plataforma libre Apache Hadoop facilita la paralelización del cálculo. Analizando el modelo computacional del paradigma de programación MapReduce se pretende portar a esta tecnología los análisis genómicos. Esto permitiría aportar soluciones a la explotación surge desde el mundo de la bioinformática hacia Hadoop.

En el proyecto se estudiarán los aspectos relacionados con el problema del alineamiento de secuencias sobre Hadoop. Aspectos informáticos y biológicos. Finalmente se implementará un software donde converjan todas las temáticas estudiadas. Pretendiendo aportar así una investigación y una solución a los análisis de volúmenes de datos biológicos computacionalmente distribuidos.

1.2. Organización de la memoria

Capítulo 2: Estudio de la situación

El estudio de la situación atiende a los temas de mayor importancia del proyecto. Los sistemas distribuidos como máquinas de cómputo, el paradigma MapReduce y la tecnología Hadoop que lo implementa, y aspectos biológicos del proyecto.

Capítulo 3: Viabilidad y planificación del proyecto

El tercer capítulo describe la viabilidad de la realización del proyecto. Se exponen posibles problemas y alternativas del proyecto. Además se divide en tareas y se planifica en el tiempo.

Capítulo 4: Programación y diseño

En el cuarto capítulo se abarcan aspectos técnicos de la implementación del proyecto. Se recoge la evolución que ha sufrido el proyecto. Los errores y carencias que padecían los diseños y las mejoras que implementan su nuevas versiones.

Capítulo 5: Conclusiones

Se discuten los objetivos iniciales del proyecto y su éxito. Por otro lado se proponen cambios al proyecto final para mejorarlo y posibles ampliaciones.

Capítulo 2

Estudio de la situación

2.1. Sistemas distribuidos

2.1.1. Introducción a los sistemas distribuidos

Los sistemas distribuidos aparecieron en el contexto de las primeras redes de área local en los años 70. Es un modelo para resolver problemas de computación masiva utilizando un gran número de ordenadores organizados en una infraestructura distribuida en varios nodos. Según Andrew S. Tannenbaum [TAN08] tenemos la siguiente definición: *"Un sistema distribuido es una colección de computadoras independientes que dan al usuario la impresión de construir un único sistema coherente."*

Debido al uso cada vez más extendido de Internet los servidores web han ido cobrando más y más importancia. Su presencia es patente contemplando las tasas de datos que transfieren los grandes servidores como los de Google o Facebook. Del orden de petabytes diarios. Actualmente los sistemas distribuidos se encuentran presentes en nuestro entorno, pese a que muchos de nosotros no sepamos que son. Los podemos encontrar en un cajero, simplemente entrando en nuestro correo o buscando una palabra en un navegador conectado a Internet. Estos han madurado y poseen políticas de gestión de fallos para mejorar su eficiencia, sistemas hot plug, etc, y muestran su avance y posibilidades.

2.1.2. Estado de los sistemas distribuidos

La programación de software destinado a ejecutarse en sistemas distribuidos es una tarea considerada difícil. En parte porque en ocasiones es necesario programar a bajo nivel considerando la arquitectura hardware. Hay lenguajes específicamente diseñados para la programación distribuida como son Ada, Erlang u Oz. Dado su tamaño se implementan sistemas de alta disponibilidad y tolerancia a fallos. Para que no deje de funcionar por el error de uno de sus componentes ni haya pérdida de datos. Si un componente falla, otro lo sustituye sin que se resienta todo el sistema.

El tamaño de un sistema distribuido puede ser muy variado. Sean decenas de hosts (como una red de área local), centenas de hosts (red de área metropolitana), y miles o millones de hosts (Internet). Los sistemas distribuidos no han de estar alojados necesariamente en una misma localización como los supercomputadores. Los sistemas distribuidos se comunican a través de la red. Pueden interactuar por Internet estando cada computadora en un lugar distinto.

Los grandes servidores de Internet se encuentran repartidos por diversos puntos de la Tierra conectados a través de redes WAN. Dada la importancia creciente de la accesibilidad a los datos y a la globalización tecnológica los sistemas distribuidos son más comunes y necesarios. Sus tecnologías aún están madurando, creciendo e innovando constantemente.

Imagen 2-1. Data centers de Google en la Tierra y en EEUU

Los data centers son centros pertenecientes a organizaciones, que descentralizan los datos. Albergan información y operan con ella. En la imagen 2-1 se muestran los data centers de la compañía Google. Es patente la distribución que tienen estos y las largas distancias que tienen que recorrer los datos. La complejidad que da al haber estos que estar sincronizados. La situación de data centers en el mapa de EEUU muestran la lejanía entre ellos. En la parte izquierda de la imagen 2-1 se ve la distribución mundial

de los data centers de Google. Puede realizarse una petición a un servidor situado en París pero los datos estar contenidos en un data center en Tokio. La sincronización, y el tiempo de acceso a los datos es muy importante en el tiempo de respuesta que nota el usuario. Esta respuesta ha de ser indistinta de el sitio físico en el cual se encuentren los datos.

2.2. Cloud computing

El cloud computing (o computación en la nube), es un paradigma que permite ofrecer servicios de computación a través de Internet. La metáfora "cloud" se refiere a Internet, y a los servidores que albergan el cómputo como se muestra en la imagen 2-2. Aplicaciones cloud como Dropbox o Evernote proveen a los usuarios un servicio que les permite no guardar los datos en un lugar físico concreto sino que se albergan en la "nube". Están disponibles desde cualquier punto con acceso a Internet.

Los usuarios de una aplicación cloud acceden a sus datos y a los servicios software sin albergar físicamente los datos ni el software ejecutado. Por ejemplo, un usuario de Google Docs accede a su cuenta. Abre un archivo de texto online albergado en los servidores de Google y ejecuta un procesador de textos. El código ejecutado no está en su máquina, sino que parte se ejecuta en el navegador y parte en el servidor. Al acabar cierra el navegador, pero el archivo sigue existiendo, aunque no esté en su ordenador. La novedad y mejora que representan los sistemas cloud, es que no se mantienen los archivos ni el software en un sitio único. Se puede acceder desde cualquier punto con acceso a Internet.

Imagen 2-2. Visualización de la metáfora de la nube

Para manifestar la importancia de los datos es ilustradora la tasa de transferencia de datos: por día Google, transfiere 24 petabytes. Los datos que produce el LHC (Large Hadron Collider) son 15 petabytes al año. El Internet Archive contiene 3 petabytes de datos y crece a razón de 100 terabytes al mes. Estas empresas almacenan todos estos datos y operan con ellos en sus data centers.

Amazon Elastic Compute Cloud (Amazon EC2) es un servicio web que proporciona capacidad informática con tamaño modificable en la nube. Permite ejecutar software como servicio web. Este servicio es escalable y ejecuta Hadoop en sus sistemas. Como se muestra en la imagen 2-3 el uso de sus servidores ha seguido una tendencia exponencial en los últimos años. Las peticiones han ido aumentando. Han tenido que ir incrementando el número de nodos al mismo ritmo que crecía la demanda, exponencialmente.

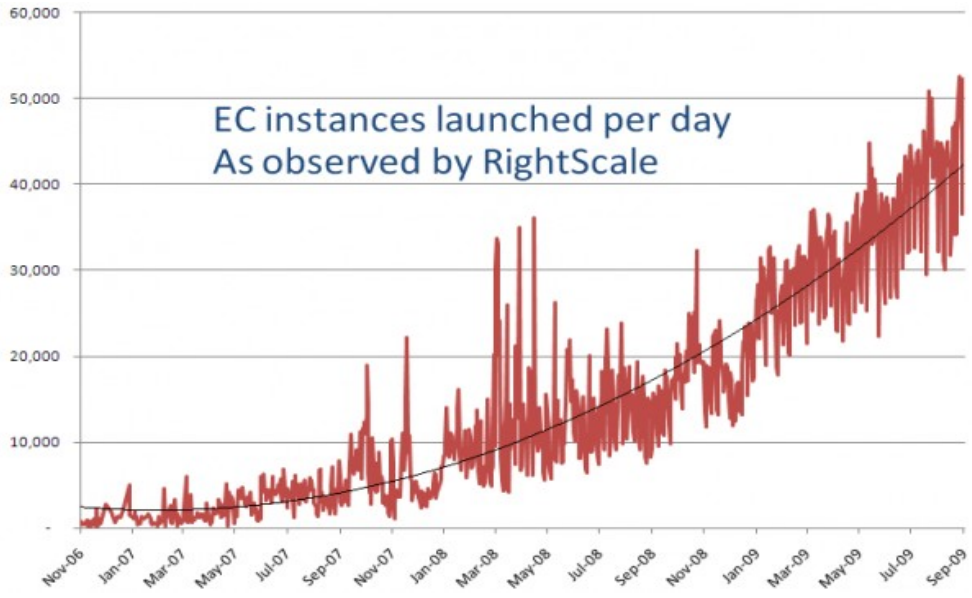


Imagen 2-3. Instancias ejecutadas por números de nodo en Amazon EC2 por mes

Universidades, empresas y gobiernos están investigando en cloud computing. Uno de estos proyectos es la IBM/Google Academic Cloud Computing Initiative (ACCI), multi-universidad y destinado a mejorar las técnicas del estudiante. Actualmente en investigación bioinformática se habla del movimiento *Bioinformatics in the cloud*, que pretende proveer servicios de análisis de grandes volúmenes de datos como servicio en sistemas distribuidos a través de una interfaz web. Los servicios bioinformáticos en la nube son todavía bastante inmaduros pero atienden al interés de investigaciones y publicaciones [3].

Por otro lado, Richard Stallman, creador de la *Free Software Foundation*, ha denunciado el peligro respecto a la pérdida del control de nuestros datos y como se cede nuestra privacidad a manos de

terceros. Denuncian la terminología “cloud computing”, en la lista de words to avoid[1] de la *Free software foundation*. Critican que “*Es un término demasiado vago y que se usa para un rango de actividades cuya única característica es que usan Internet para algo más allá de transmitir archivos*”.

2.3. Escalabilidad y NoSQL

2.3.1. Escalabilidad

La escalabilidad es la propiedad de un sistema para incrementar el rendimiento, bajo un incremento de carga, cuando se añaden nuevos recursos (típicamente hardware). La escalabilidad surge como respuesta a sistemas computacionales que necesitaban crecer para abarcar un incremento en la demanda de cálculo. En la imagen 2-4 se muestran los dos tipos de escalabilidad.

Escalabilidad vertical (scale up)

Consiste en añadir recursos al servidor de cómputo. Lo más usual es mejorar la capacidad de las CPUs y añadir memoria al ordenador.

Escalabilidad horizontal (scale out)

Añade más nodos al sistema. Este modo de escalar es el más usual. Hay distintos patrones asociados a este modelos como son el cliente-servidor, arquitectura en pizarra, tuple space y MapRededuce. Este modelo se enmarca en los sistemas distribuidos al poseer *n* nodos que abarcan un problema, necesitando incrementar *n*.

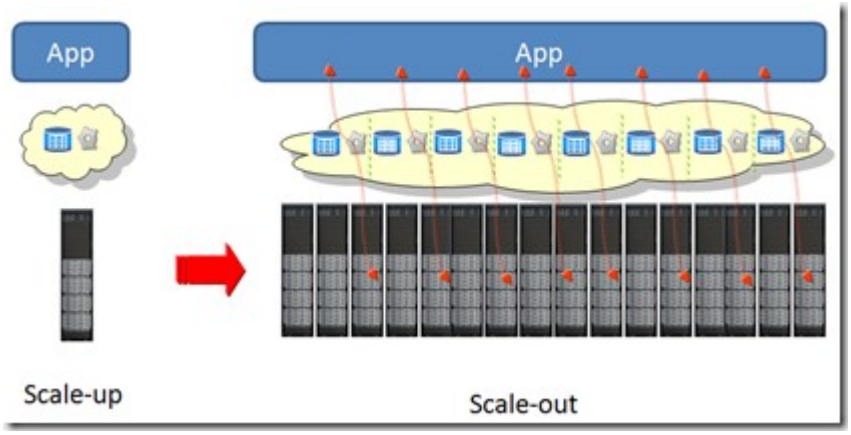


Imagen 2-4. Esquema de una arquitectura escalando vertical y horizontalmente

1 http://www.gnu.org/philosophy/words-to-avoid.html#CloudComputing

Hay varios motivos por los que la escalabilidad horizontal se elige antes que la vertical. La disponibilidad y la redundancia son necesarias ya que los fallos en el sistema son inevitables. Un gran sistema puede fallar debido a un único punto de error. El hardware tiende a variar y mejorar, por lo que si un sistema está preparado para escalar sólo verticalmente, estará más restringido el hardware al que puede acceder. Construir aplicaciones como una gran unidad hace difícil cambiar código individualmente sin afectar a todo el sistema. En este caso es mejor dividir la aplicación en conjuntos de servicios que se puedan mantener independientemente. La distribución geográfica que puede necesitar una aplicación, obliga a escalar en horizontal forzosamente.

Un proceso que se ejecuta sobre un sistema distribuido podría escalar si conectamos ordenadores a la red de cómputo y estos se unen al cálculo automáticamente. Es decir, podría ejecutar mayor cantidad de datos en el mismo tiempo. Supongamos un servidor que atiende peticiones online. Éste se haya saturado por el número de peticiones. Si el sistema está preparado para escalar verticalmente le añadiríamos las CPU necesarias. En el caso de que sea un sistema horizontalmente escalable, añadiríamos tantos nodos como se necesitaran para atender el cómputo generado por las solicitudes.

Por otro lado también se considera el software como escalable. Podemos decir que una aplicación de software es escalable, si al incrementar los procesadores donde se ejecuta, el rendimiento crece proporcionalmente. Por el contrario, decimos que una aplicación no es escalable si su rendimiento no escala o crece con el incremento de los procesadores.

2.3.2. Movimiento NoSQL

El movimiento NoSQL apareció por primera vez en 1998. La pregunta que lo motivó fue: *¿Es necesario tener una base de datos relacional cuando lo único que se quiere hacer es leer texto?*. Los defensores del *no* argumentaban que una base de datos relacional es útil cuando tenemos registros y hay que acceder a ellos, pero si hay que leer texto y modificarlo, entonces no es necesario. Actualmente es un tema candente en la comunidad informática y ya han aparecido bases de datos NoSQL como MongoDB, CouchDB o Hadoop Hbase.

Las bases de datos tienen un sistema centralizado. Esta arquitectura no les impide escalar, pero si que hace que sea una tarea más complicada que en otras arquitecturas distribuidas. Además han dado resultados pobres en servicios como indexación de documentos, atención de páginas web con alto tráfico o en media streaming. Las bases de datos NoSQL utilizan una arquitectura distribuida albergando los datos en servidores usualmente guardado en tablas hash.

La aparición del paradigma MapReduce empujó el movimiento NoSQL. Estas tecnologías incrementan la eficiencia, puesto que el texto es repartido entre todos los nodos, y el procesamiento se realiza directamente sobre los nodos que almacenan los datos. Por tanto su capacidad de escalar se incrementa. NoSQL guarda los datos en una tupla de <clave, valor> que se corresponde con los datos de entrada del paradigma MapRededuce.

2.4. MapReduce

2.4.1. Motivación de MapReduce

En el año 2004 ingenieros de Google presentaron un paper [2] en el que presentó un paradigma llamado MapReduce (imagen 2-5). Este paradigma enmarcado en los sistemas distribuidos daba una solución al cómputo paralelo para grandes volúmenes de datos. En sus sistemas se encontraban con tres elementos principales.

Data storage

Google presentó el sistema de ficheros empleado en sus servidores, conocido como Google File System (GFS) enfocado a mejorar la eficiencia en sus búsquedas. Éste sistema de ficheros se caracteriza, entre otros aspectos, por tener bloques de 64 MB para mejorar el acceso a grandes volúmenes de datos.

Data analysis

Los ingenieros en sistemas distribuidos de Google se dieron cuenta de que repetían implementaciones y de que el proceso de análisis de datos se podía automatizar más. De ahí surgió MapReduce. Una de las principales características de MapReduce es su simplicidad. El análisis de datos se basa en dos procesos: map y reduce. El primero se ocupa de mapear la entrada y el segundo de reducir la solución. Google utiliza MapReduce para el proceso de indexación que produce las estructuras de datos utilizadas por el servicio web de búsquedas. Toma como entrada los datos proveídos por su sistema de rastreo, albergado en su sistema de ficheros (GFS) y MapReduce ejecuta el proceso de indexación.

Coordination

El framework que implementaba el paradigma buscaba robustez y coordinación en sus sistemas distribuidos. Para ello consta de una serie de daemons que se ocupan de repartir datos y tareas y atender al estado de los procesos y nodos. Presentaban una arquitectura master/worker.

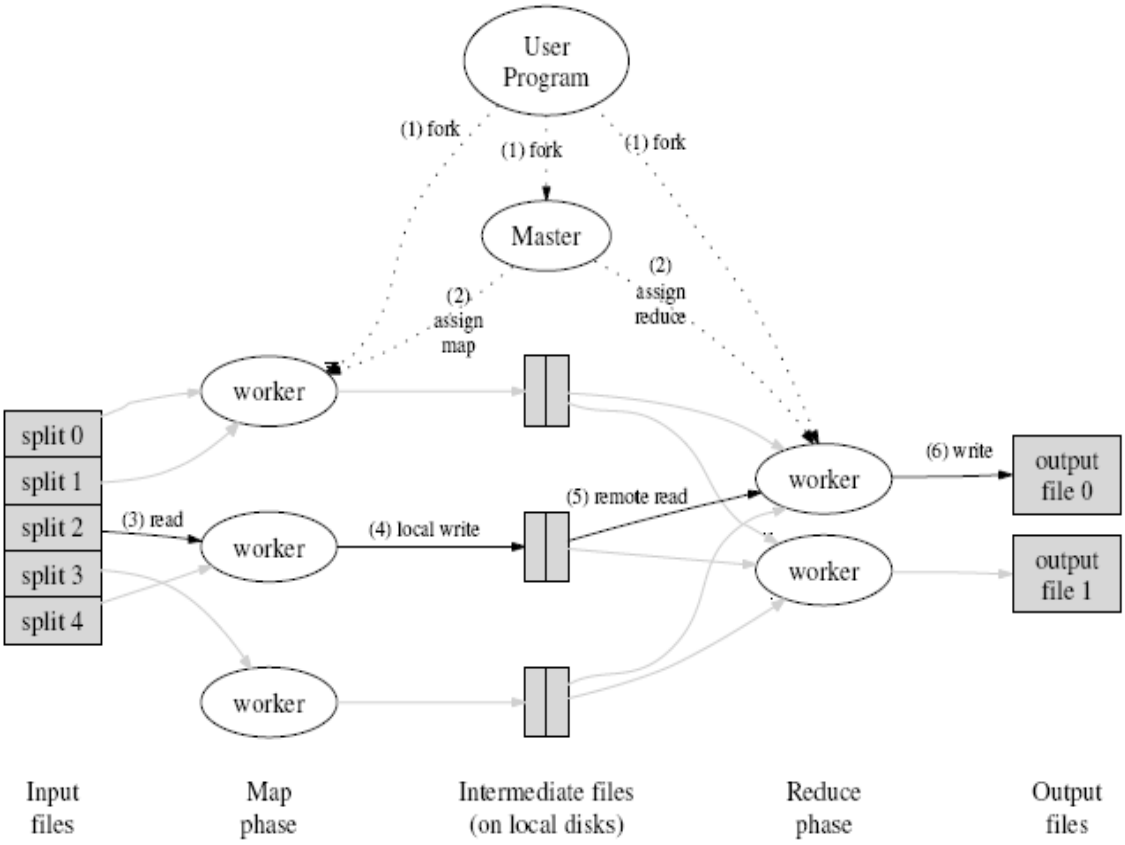


Imagen 2-5. Gráfico con el que Google presentó MapReduce

2.4.2. Funcionamiento de MapReduce

El paradigma MapReduce funciona separando en dos fases el proceso de ejecución: las fase map y la fase reduce. De ahí su nombre. Cada fase trabaja con miles de tuplas de <clave,valor> tanto como input como output, cuyos tipos de datos son elegidos por el programador. El programador sólo tendrá que implementar el código de dos funciones: map y reduce.

Datos en MapReduce

MapReduce se caracteriza por el análisis de grandes volúmenes de datos. Estos se alojan en un sistema de ficheros distribuido. La característica de los datos es su voluminosidad. Los sistemas de ficheros tienen características especiales, como el tamaño de los bloques, para poder aprovechar el tamaño y mejorar procesos de lectura y escritura en disco. Aunque está pensado para escritura única, lectura múltiple (Write Once, Read Many). Los datos se dividen en bloques y se reparten entre las CPU del cluster para su análisis. Finalmente los datos creados por el proceso MapReduce son escritos en archivos de salida en el sistema de ficheros distribuido.

Función map

La función map se encarga de leer el bloque de datos de entrada recibido como parejas de <clave,valor> y generar una serie de pares <clave,valor> (K, V) de salida de otro dominio distinto por cada ítem. El programador debe especificar en el objeto colector de la salida cual será el valor y la clave creada por la información leída. Así pues la función map recibe como parámetro claves y valores dando como resultado al final de la ejecución una lista de tuplas <clave,valor>.

```
Map(k1,v1) → list(k2,v2)
```

Función reduce

La fase reduce recibe como parámetro una lista de valores para cada clave. La lista ha sido creada en una fase intermedia nombrada sort y shuffle. Recorre la lista de valores y va tratándolos y escribiéndoles a medida que los procesa. La lista ha sido creada en una fase intermedia y transparente nombrada sort & shuffle. Entre map y educe hay un preproceso de los datos de ordenación para que así poder crear tuplas de los valores de cada clave, para la entrada de datos del proceso reduce. En la salida del proceso se especifica una lista de valores.

```
Reduce(k2, list (v2)) → list(k3,v3)
```

Distribución del cómputo

La única implementación de dos funciones es los que le da simplicidad al paradigma. En la distribución del cómputo y en la escalabilidad es donde radica la potencia. En el paradigma se presentan los wokers que ejecutan la fase map y la fase reduce. Un master es el encargado de coordinar las tareas y de asignar tareas y datos a los workers. Los workers reciben una configuración, el código y los datos. Son nodos repartidos en el sistema que se ocuparán del cálculo.

Flujo de los datos

En la imagen 2-6 se muestra el flujo de la ejecución de un caso concreto bajo el paradigma MapReduce. Inicialmente hay un archivo con un conjunto de claves. Map recibe en su entrada las claves y les asigna un valor. La fase de sort y shuffle auna las claves iguales y convierte en una lista sus valores. Reduce guarda aquellos valores para los que está programado y los escribe en el colector. Finalmente se escriben en el sistema de ficheros distribuidos.

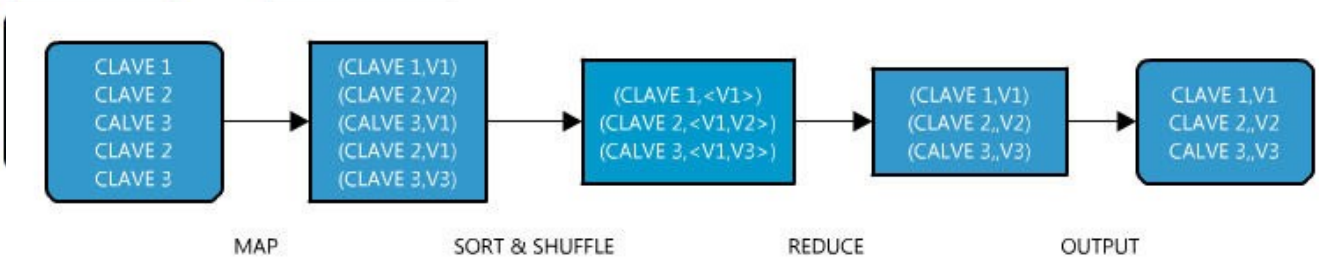


Imagen 2.6. Dataflow lógico de MapReduce

2.4.3. Ejemplo de MapReduce: contar palabras

Para ilustrar el funcionamiento del paradigma se utilizará un ejemplo simple: un contador de palabras. Junto a cada palabra guarda el valor respectivo a las veces que aparece en una serie de textos. La función map lee una línea. Para cada línea separa sus palabras y a cada palabra le asigna el valor 1. El valor 1 es debido a que se está leyendo esa palabra, que está apareciendo una vez. A continuación el código de la función:

```
funcion Map (int clave, string valor, colector colect)
      StringToken linea = separarPalabras(valor)
      mientras (linea.haySiguiente())
            colect.guardar(linea.siguiente(), 1)
```

Una vez ejecutado el map, al reduce le llegan claves con tuplas de valores correspondiente a cada palabra. Así, por ejemplo, la palabra "*mundo*" llegará como clave, y la tupla de valores será, si ha aparecido 3 veces, <1, 1, 1>. La función en este caso solo se ocupará de recorrer la tupla y de sumar los valores:

```
funcion Reduce (string clave, iterador<int>, colector colect)
      int suma = 0
      mientras (valor.haySiguiente())
            suma += valor.siguiente()
      colect.guardar(clave, suma)
```

El colector final se ocupa de guardar la palabra y el número de veces que aparece. Esta información se escribirá en el archivo de salida del proceso. En la imagen 2-7 se muestra la visión lógica de los datos. Primero el map recibe un texto y asigna a cada palabra el valor 1. Los procesos intermedios de MapReduce crean una tupla de valores por cada clave. Reduce suma los valores y la salida son las palabras junto al número de veces que aparecen.

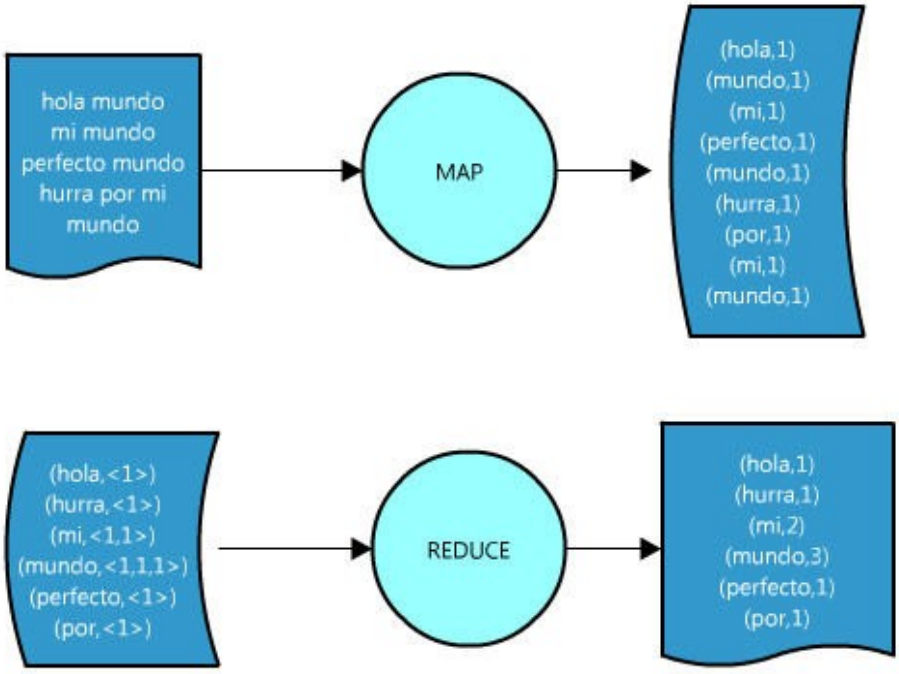


Imagen 2-7. Flujo de MapReduce ejecutando un contador de palabras

2.4.4. MapReduce como mejora al acceso de datos

El acceso a disco es de las operaciones más lentas de una máquina. Por lo que hay buscar tecnologías que permitan minimizar el acceso a los datos frente una petición del usuario. MapReduce lo consigue repartiendo piezas de texto entre los nodos haciendo una lectura secuencial localmente en cada nodo. La distribución de los datos mejora el tiempo de lectura de archivos enteros (que no de registros), y el paradigma de clave valor ha conseguido reducir considerablemente el acceso a datos. Más adelante se especificarán los detalles de esta tecnología.

La imagen 2-8 pertenece a un análisis comportamental de los datos en sistemas escalables [1]. Se muestra el tiempo de acceso a datos de Hadoop respecto a un DBMS y a la tecnología de bases de datos Vertica. La operación es un *grep*, por tanto una búsqueda. En el gráfico se muestran los segundos por número de

nodos en la carga de archivos al distribuirlos por los nodos. La imagen muestra como el tiempo se mantiene casi constante a partir de los diez nodos, mientras que el tiempo para DBMS crece exponencialmente y en Vertica crece linealmente. El acceso a datos es siempre el mejor en el caso de MapReduce. Se reduce el tiempo de consultas y no crece, sino que se mantiene más o menos constante. El gráfico refuerza la teoría del movimiento NoSQL que para analizar texto no es necesaria una base de datos relacional, repartiendo el peso entre nodos se muestra más eficiente.

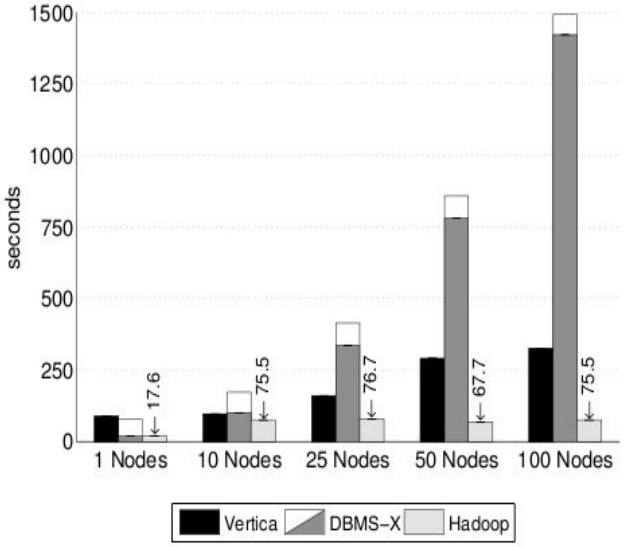


Imagen 2-8. Tiempos de carga en la operación grep con 535MB analizados por nodo

2.5. Hadoop

2.5.1. Introducción a Hadoop

Hadoop es una plataforma libre que implementa el paradigma MapReduce. Es un proyecto maduro, extendido, en continua mejora y crecimiento. La ASF da la siguiente definición de Hadoop [W3APA]: *El proyecto Apache Hadoop desarrolla software open-source para computación confiable, escalable y distribuida.*

Hadoop fue creado por Doug Cutting, en un principio como parte del buscador open source Apache Nutch. Era una solución a los problemas dados por la centralización de los datos. Finalmente Hadoop acabaría siendo un proyecto en sí en el 2007. Después de la publicación de las especificaciones del sistema de ficheros Google File System (GFS) crearon Nutch Distributed File System (NDFS) los cual les permitió crear una plataforma distribuida. Más tarde, en el 2008, Hadoop ya corría en un cluster de 10.000 cores en la compañía Yahoo!.

Un ejemplo de la potencia y capacidad es la competición que este software ganó con un cluster de Yahoo!. El problema a superar era la ordenación de un terabyte de datos repartido en 10 mil millones de archivos de 100 bytes los cuales debían estar completamente ordenados y escritos en disco. Lo consiguió con 910 nodos, batiendo el récord anterior que era de 297 segundos mientras que su tiempo fue de 209 segundos. Este caso ilustra la capacidad que tienen los sistemas escalables y el paradigma de programación MapReduce.

Actualmente Hadoop es utilizado por empresas como Facebook, Yahoo!, Microsoft, Last.fm o LinkedIn. Es considerado una tecnología accesible, por poder correr en clusters de miles de nodos; robusto, por estar preparado para ejecutarse con la asunción de fallos de hardware; escalable, porque escala horizontal y fácilmente añadiendo más nodos para procesar grandes volúmenes de datos. Y simple porque permite crear fácilmente código paralelizable.

2.5.2. Arquitectura de Hadoop

En la arquitectura de Hadoop existen una serie de daemons que se ocupan del funcionamiento del sistema. Unos atienden a la repartición de las unidades de trabajo entre los nodos del cluster. Gestionan el cómputo y sus estados. Por otro lados está el sistema de ficheros distribuidos Hadoop Distributed File System (HDFS). Los daemons que tratan con este sistema de ficheros gestionan los datos y los reparten entre los nodos.

Sistema de cómputo (Hadoop)

JobTracker (master):

Coordina todos los trabajos que se están ejecutando en el sistema planificando las tareas que se han de ejecutar en el TaskTracker. Localiza TaskTracker con disponibilidad y les asigna tareas. El TaskTracker le comunicará cuando falla y el JobTracker decide que hacer. Si ejecutar de nuevo todo el proceso o desde que punto.

TaskTracker (worker):

Ejecuta las tareas y le manda informes al JobTracker, el cual almacena un registro del conjunto de progresos de cada trabajo. Cada TaskTracker tiene un número de slots configurados, que serán sus unidades de trabajo. El TaskTracker envía mensajes *heartbeats* o latidos cada poco tiempo para asegurar que el TaskTracker sigue vivo.

Sistema de ficheros (HDFS)

NameNode:

Es el más vital de los daemons de Hadoop. Emplea una arquitectura de master/slave de HDFS, dirige al esclavo DataNode. Decide como cortar los archivos (normalmente en bloques de 64 Mb) y que nodos los almacenarán.

DataNode:

Daemon que alberga cada nodo slave. Lee y escribe bloques de HDFS al sistema de ficheros local. Cada DataNode se comunica con un NameNode. Alberga una serie de bloques de texto y permite al código leer estos bloques

Secondary NameNode:

Es un dameon asistente para monitorear el estado del cluster HDFS. Sincroniza los archivos periódicamente con el NameNode. Guarda el último punto de comprobación del directorio de archivos.

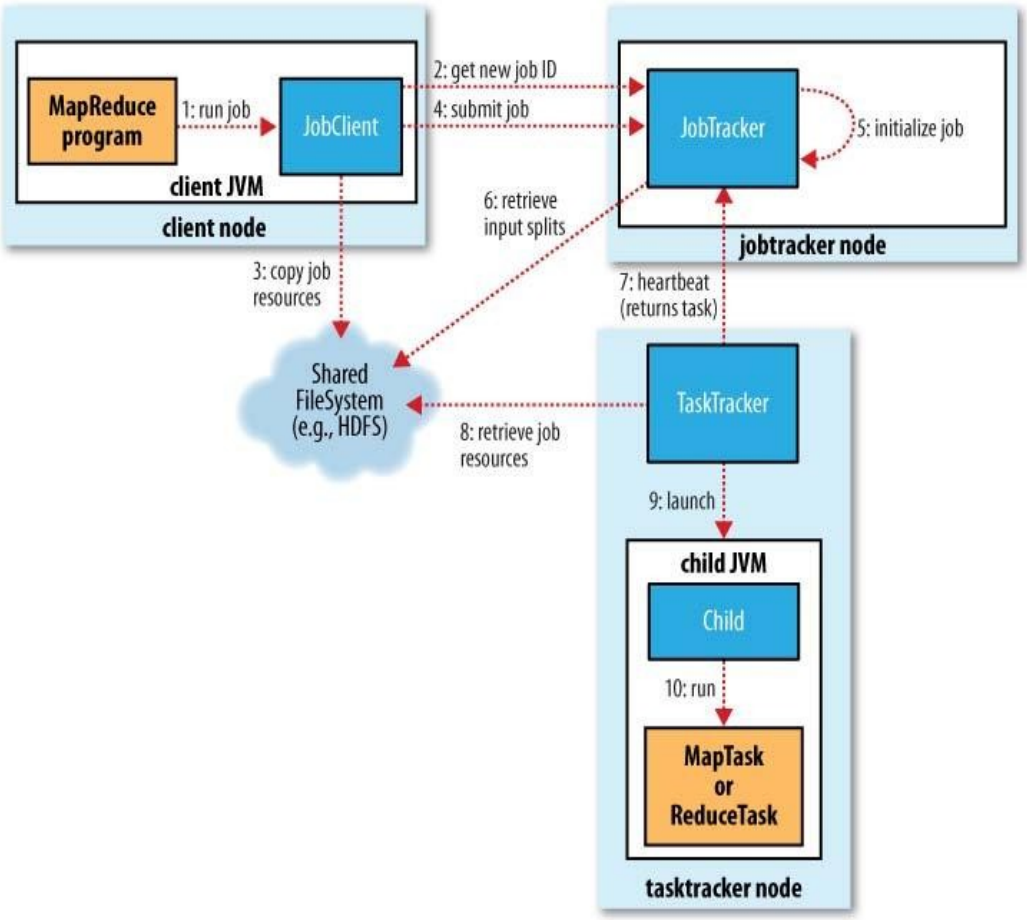


Imagen 2-9 Cómo Hadoop ejecuta un proceso MapReduce

En la imagen 2-9 se muestran los pasos que ejecuta MapReduce en Hadoop. Al iniciar un proceso en Hadoop JobTracker reparte los los datos a procesar en HDFS para que los TaskTracker los procesen secuencialmente. Cada TaskTracker ejecutará el proceso map o reduce, dependiendo de la fase de la ejecución. Al final se escribirá la salida de nuevo en HDFS.

En la figura se muestra cómo Hadoop ejecuta un proceso MapReduce. La figura describe 9 pasos. Al iniciarse una aplicación en Hadoop (1) JobClient recibe una nueva ID (2) y el NameNode copia tanto los datos del directorio de entrada, como las configuraciones del cluster a el sistema HDFS (3), de donde a su vez recogerán la configuración y los datos los otros nodos. El trabajo se crea (4) y se inicializa el JobTracker (5), responsable y coordinador de los trabajos del sistema. Los archivos del directorio son divididos para ser repartidos para los DataNode de cada nodo para su TaskTracker (6). El TaskTracker empieza a enviar “latidos” para informar de su funcionamiento al JobTracker (7). El DataNode obtiene los recursos necesarios para iniciarse (8) y TaskTracker lanza el proceso (9).

2.5.3. Ejemplo de funcionamiento de Hadoop: contador de palabras

Retomando como ejemplo el contador de palabras del punto 2.4.3. se explicará el funcionamiento de la aplicación. El proceso consiste en leer un texto de entrada y guardar junto a cada palabra el número de veces que aparece. La arquitectura será de un nodo master (JobTracker) y dos esclavos (TaskTrackers) como se muestra en a imagen 2-10.

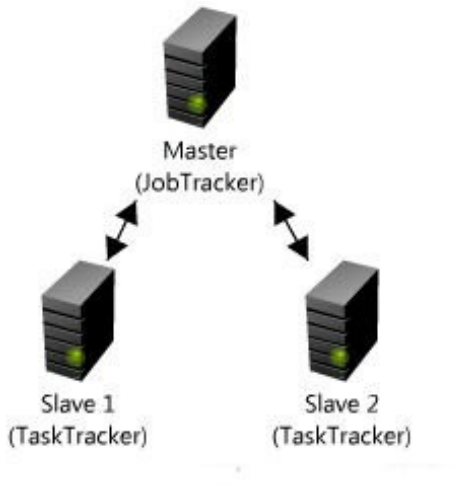


Imagen 2-10. Estructura con un master y dos esclavos

Inicialmente Hadoop se ha de instalar en cada uno de los nodos del cluster. A cada nodo se le ha de asignar en los archivos de configuración el rol que ejercerá en la ejecución del programa. Es decir, si será JobTracker o TaskTracker. En este caso los dos harán tanto la fase map como la fase reduce.

Al ejecutar el contador de palabras en MapReduce primero se inicializará el JobTracker y el NameNode. El NameNode cortará todos los archivos de palabras en bloques. Además escribirá en HDFS la configuración y el ejecutable que correrá en todos los nodos. NameNode divide los archivos de entrada y los reparte entre los dos DataNode que hay en el cluster. El TaskTracker inicia su ejecución. Primero ejecuta la fase map que asigna como clave cada palabra y como valor el número 1. Mientras ejecuta este proceso envía mensajes de estado al JobTracker para que este se cerciore que no ha caido el proceso y debe ser vuelto a ejecutar. Al acabar cada uno de los procesos map los archivos se van escribiendo en el DataNode. Como los dos ejecutan el proceso reduce, al acabar les llegan los datos en clave y lista de valores del NameNode. Cada uno de los dos TaskTrackers ejecuta el proceso reduce sumando la lista de valores para saber cuantas veces aparece cada palabra. La salida es una lista de tuplas con una palabra y el número de veces que aparece. Al finalizar los DataNode envían la información al NameNode del master y este finalmente escribe en HDFS. El JobTracker informa al usuario con las estadísticas del proceso finalizado.

2.5.4. Hadoop como sistema particular

La evolución de Hadoop ha sido notable en su breve período de vida. Muchos subproyectos han surgido, y otros subproyectos se han convertido en proyecto en si. Algunos de ellos como Hive se emplean cuando se quiere usar una base de datos NoSQL. Tres subproyectos son los principales en este proyecto. Hadoop MapReduce es un onjunto de componentes e interficies para sistemas de ficheros distribuidos y entrada/salida en general. HDFS es un sistema de ficheros distribuidos que corre en grandes clusters. Y Hadoop common es un framework para el proceso distribuido de conjuntos de datos en clusters. A parte hay otros subproyectos de Hadoop como: Pig, Chukwa, Hive, Hbase, ZooKeeper o Avro

Existen otros proyectos que implementan el paradigma MapReduce. Disco, desarrollado por Nokia Research listo para programar aplicaciones en Python. SkyNet es una framework open source escrito en Ruby con especial atención a Ruby on Rails. FileMap es un framework ligero para aplicar proceso de textos estilo UNIX. GreenPlum es una solución privativa con soporte para procesamiento paralelo y SQL. Apache Hadoop es el framework MapReduce más soportado, maduro, robusto y extendido. La solución con la que ha competido más en pruebas de análisis de datos es con la propia implementación de MapReduce de Google, pero esta es privada y de uso exclusivo de Google. En cuanto a desarrollo, madurez y también comunidad Hadoop evoluciona continuamente. Estos aspectos y su filosofía open source hacen a esta plataforma ser elegida por encima de otras.

2.6. Secuencias de ADN

2.6.1. Componentes y estructura de las secuencias de ADN

El ADN es una doble cadena en forma de hélice compuesta por nucleótidos. Los nucleótidos son moléculas orgánicas en las que se encuentra una base nitrogenada. Existen cuatro bases nitrogenadas, las cuales son representadas por su inicial en mayúsculas: adenina (A), citosina (C), guanina (G) y timina (T). La doble hélice es complementaria y antiparalela, es decir, cada nucleótido se empareja con su correspondiente. La adenina se une con la timina , y viceversa, y la citosina se corresponde con la guanina, tal y como se ve en la iamgen 2-11. Antiparalelas significa que son cadenas iguales con direcciones opuestas.

Una secuencia genética es una sucesión de letras (A, C, G o T) representando la estructura primaria del ADN. En su análisis se leen codones, que es un triplete de nucleótidos que codifica un aminoácido. Se codifican diferentes aminoácidos dependiendo del orden de los nucleótidos. Por ejemplo ACG codifica treonina, pero AGC codifica serina. Además distintos codones pueden codificar un mismo aminoácido, lo que es llamado código degenerado. Por ejemplo, la lisina se codifica por dos codones: AAA y AAG. También los codones son exclusivos, un triplete solo codifica un aminoácido.

La lectura de la secuencia de ADN se hace en un solo sentido (5' - 3'), desde el codón de iniciación hasta el codón de parada como se muestra en la figura. Esto hace que cuando analicemos una secuencia empecemos siempre leyendo, no desde un punto cualquiera, sino desde el principio.

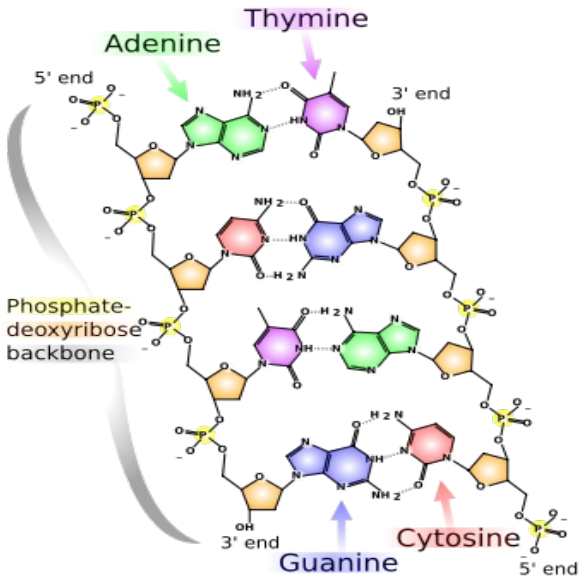


Imagen 2-11. Esquema de la estructura del ADN

2.6.2. Almacenamiento de secuencias de ácidos nucleicos y aminoácidos

En el almacenamiento de secuencias de ADN y otro tipo de secuencias (como aminoácidos) se estandarizaron formatos de archivo. Uno de los más utilizados es FASTA. El formato FASTA es de tipo texto. Representa secuencias de nucleótidos o de aminoácidos. También muestra los nombres de las secuencias y comentarios que aportan información de esta. FASTA contiene dos tipos de líneas.

Identificador de la secuencia

Es una línea con información antecedente de la línea descrita. Se distingue con el símbolo inicial '>' en la primera columna. Alberga el identificador de la secuencia. Además puede contener información extra separada por barras verticales '|'. A continuación se muestra un

```
> id | nombre del organismo | especie | familia | BD del archivo
```

Secuencia

La secuencia es una codificación de ácidos nucleicos o aminoácidos. Es el contenido descrito por el identificador. Se representa por letras o por guiones '-' en el caso de que se indique un salto en la secuencia. La secuencia finaliza al finalizar el archivo o haber otro identificador de secuencia.

Así por ejemplo, la cabecera del genoma de la bacteria Methanosarcina en formato FASTA es la siguiente:

```
>Methanosarcina acetivorans genome sequence
AGTATGAAACCTGAATATATTGATCTATACCTGCATGGGAATTGATCAAT
ATTTGTATGGGAATTGATCAATATTTGTATGGGAATTGATCTACATCTGA
CAATGAGTCGATCAATAATGTGAGTTGATCTATACCTGACTATGAATTAT
GATATGATCCTAGTCAAACCACAGGATCGTATATATTATTTCCTGAAAAG
```

Los archivos FASTA pueden ocupar del orden de decenas y cientos de megas. En el caso del genoma humano, distribuido en los 23 pares de cromosomas ocupan 3.200 MB. El par de cromosomas 1, más voluminoso, ocupa 270 MB, mientras que el par que ocupa menos, el 21, es de 45 MB. El análisis de muchos genomas requiere terabytes en disco. Por ejemplo OMA browser [W3OMA] alberga 1109 genomas para su análisis.

2.7. Alineamiento de secuencias

2.7.1. Concepto del alineamiento de secuencias

Un alineamiento de secuencias es una forma de representar y comparar secuencias o cadenas de ADN, ARN, o estructuras proteicas para resaltar sus zonas de similitud. Estas podrían indicar relaciones funcionales o evolutivas entre los genes o proteínas consultados. Las secuencias alineadas se escriben con letras (representando aminoácidos o nucleótidos) en filas de una matriz. Si es necesario en estas se insertan espacios para que las zonas con idéntica o similar estructura se alineen dando así una alineamiento.

Cuando se comparan los genomas de dos especies diferentes se obtienen genes equivalentes en ambas, con una secuencia algo diferente. El alineamiento de secuencias se realiza para determinar la homología. La homología de secuencia se refiere a la correspondencia entre las cadenas nucleotídicas de dos genes, que es precisamente la que permite reconocer que son homólogos.

Son varias las relaciones evolutivas entre genes, se definen como relaciones de pares y se presentan en dos tipos de homología. Unos son los genes ortólogos, aquellas secuencias homólogas que se han separado por un evento de especiación. En otras palabras, las secuencias ortólogas son las secuencias que se encuentran en diferentes especies y que son altamente similares debido a que se han originado en un ancestro común. Por otro lado, los genes parálogos se originan de una duplicación genética dentro de individuos de una misma especie. Los ortólogos son los genes más cercanos en dos genomas, ya que comenzaron su divergencia en la especiación, mientras que los parálogos lo hicieron anteriormente a la especiación, por duplicación.

2.7.2. Algoritmos de alineamiento de secuencias de ADN

La dificultad de la comparación de secuencias radica en que no son comparaciones alfanuméricas en la que se es mayor, menor o igual. Se ha de calcular el grado de homogeneidad. Son necesarios unos pasos, un algoritmo que permita compararlas. Con este fin se crearon varios algoritmos de alineamiento de parejas de secuencias. Los aquí explicados son los que se basan en el principio de la programación dinámica. Construyen una matriz de valores de dimensión dos. Con ella pueden reconstruir la solución. Y en la programación dinámica devuelven una solución óptima.

2.7.3. Algoritmo Needleman-Wunsch

El algoritmo Needleman-Wunsch fue propuesto en el año 1970 por Saul B. Needleman y Christan D. Wunsch [7]. Sirve para realizar alineamientos globales de secuencias. Los alineamientos globales buscan un alineamiento completo y no parcial del par de secuencias, es decir, conseguir un alineamiento desde la última posición hasta la primera de ambas secuencias. Para conseguir un alineamiento global se penaliza la primera fila y la primera columna a medida que se aleja de la posición en la fila 1 columna 1. A continuación se explica el algoritmo mediante un ejemplo la construcción de la solución en el algoritmo Needleman-Wunsch.

Inicialmente tenemos dos secuencias a comparar

```
Secuencia X: ACTTCG
Secuencia Y: GCCTTC
```

Estas dos secuencias forman una matriz. El tamaño de la matriz es el mismo que el de las secuencias a alinear +1. Esto es así para la construcción de la solución. La secuencia X en el eje x y la Y en el eje y. L Tanto la primera fila como la primera columna se inicializa al principio. Se rellena penalizando las coordenadas a medida que se alejan de la posición (1,1). En este caso se penaliza con un valor de -2.

		A	C	T	T	C	G
	0	-2	-4	-6	-8	-10	-12
G	-2						
C	-4						
C	-6						
T	-8						
T	-10						
C	-12						

Para construir la solución se pone un valor respecto si hay o no coincidencia. El algoritmo va recorriendo la matriz fila por fila de izquierda a derecha. El valor al que se suma o resta es al mayor que hay en la fila superior y en la misma columna, en la columna anterior y en la misma fila, y en la columna y fila anterior. Por tanto el que se encuentra en la fila 2, columna 2 que compara A y G cogería el máximo de 0, -2, -2. Para el match y el missmatch se definen valores parametrizados. En éste caso los valores son los siguiente:

· *Si hay coincidencia (match):* +2

· *Si no hay coincidencia (missmatch):* -1

· *Si hay varias coincidencias por letra repetida*: 0

		A	C	T	T	C	G
	0	-2	-4	-6	-8	-10	-12
G	-2	-1	-2	-3	-4	-5	-3
C	-4	-2	1	0	-1	1	0
C	-6	-3	1	0	-1	1	0
T	-8	-4	0	3	3	2	1
T	-10	-5	-1	3	5	4	3
C	-12	-6	1	2	4	7	6

Después de construir la matriz se construye la solución a partir de la posición última y se remonta hasta encontrar la posición en la fila 1, columna 1. Así se consigue entonces el alineamiento global.

		A	C	T	T	C	G
	0	-2	-4	-6	-8	-10	-12
G	-2	-1	-2	-3	-4	-5	-3
C	-4	-2	1	0	-1	1	0
C	-6	-3	1	0	-1	1	0
T	-8	-4	0	3	3	2	1
T	-10	-5	-1	3	5	4	3
C	-12	-6	1	2	4	7	6

Cada movimiento horizontal en la solución será interpretado como un salto o gap escrito como un guión en la secuencia X. Cada movimiento vertical como un gap en la secuencia Y. Por tanto nos quedan las siguientes soluciones.

```
A-C-TTCG
-GCCTTC-
```

2.7.4. Algoritmo Smith-Waterman

El algoritmo Smith-Waterman, realizado en 1981 por Mike Waterman y Temple Smith [5], sirve para realizar alineamientos locales de dos secuencias. Los alineamientos locales no pretenden encontrar cual es la mejor alineamiento entre cadenas enteras, sino entre subcadenas. En este caso la primera fila y columna se llenan de ceros. La matriz se llena con el mismo criterio con el algoritmo Needleman-Wunsch. Para reconstruir el valor se utiliza el valor máximo en la matriz, que es llamado *score*. A partir del score se construye la solución hasta llegar a un valor negativo o cero. Se muestra un ejemplo de la solución del algoritmo Smith-Waterman

		A	G	T	T	C	G
	0	0	0	0	0	0	0
T	0	-1	-1	2	2	-1	-1
C	0	-1	-2	1	1	4	3
C	0	-1	-2	0	0	4	3
T	0	-1	-2	2	2	3	3
T	0	-1	-2	2	4	3	2
C	0	-1	-2	1	3	6	5

Esta vez se recorre la solución desde el *score* y no se busca el alineamiento completo, sino parcial. Acaba de reconstruir la solución al encontrar un cero o valor negativo. El valor más grande es el score. En este caso el score es 6.

		A	G	T	T	C	G
	0	0	0	0	0	0	0
T	0	-1	-1	2	2	-1	-1
C	0	-1	-2	1	1	4	3
C	0	-1	-2	0	0	4	3
T	0	-1	-2	2	2	3	3
T	0	-1	-2	2	4	3	2
C	0	-1	-2	1	3	6	5

2.7.5. Complejidad en el alineamiento de secuencias

La programación dinámica provee la solución óptima, pero tiene como inconveniente la pesada construcción de la matriz. Siendo N la longitud de la primera secuencia proteica y M la longitud de la segunda, tenemos que el tamaño de la matriz será N x M por lo que la complejidad de cómputo crece de forma cuadrática por el número del operaciones a realizar.

En el apartado 2.6.1 se ha hablado acerca de los genes ortólogos. En el análisis realizado por el algoritmo OMA se comparan todos los genes 1000 genomas, y todos entre sí, lo cual tiene una complejidad factorial. Además este algoritmo tiene tres etapas añadidas que son formación, verificación y agrupamiento de parejas. El problema de los genes ortólogos es computacionalmente inabarcable por una sola computadora, por lo que se han de contemplar los sistemas distribuidos escalables para poder paralelizar la ejecución del software siendo ahí donde entra la plataforma Hadoop.

En la ejecución de un análisis en un cluster se esperará que el tiempo se divida entre el número de ordenadores respecto a uno solo. Así por ejemplo se espera que el análisis en un cluster de diez nodos, tarde una décima parte de lo que tardaría una sola computadora.

Capítulo 3

Viabilidad y planificación del proyecto

3.1. Posibilidades actuales del proyecto

Visto el estudio del capítulo 2, ahora podemos considerar hasta donde podemos alcanzar en este proyecto. Así pues las intenciones de este proyecto serán explotar la tecnología Hadoop en sistemas distribuidos y crear un software que se ejecute sobre esta plataforma para conseguir analizar cadenas de ADN. Hasta el día de la publicación de este proyecto se han creado implementaciones bioinformáticas sobre Hadoop como es el caso de CloudBurst o de MrsRF. El primero de read mapping de secuencias de ADN y el segundo de análisis de árboles filogenéticos de gran complejidad. Entonces considerando la situación actual es posible, pese a que no haya otro proyecto igual a este que facilite más el desarrollo. Pues los errores e implementaciones previas permiten prever posibles implementaciones o diseños erroneos. Uno de los problemas o limitaciones que puede presentar Hadoop es que es una plataforma paralela. Por tanto el problema ha de poder dividirse independientemente en subproblemas. Eso dificulta implementaciones en las que sea necesario comparar todos los datos con todos. Así pues problemas de clustering, comunes en la filogenética son más complejos de abarcar.

3.2. Objetivos generales del proyecto

- Planificación y viabilidad del proyecto.

- Estudio de los sistemas distribuidos.

- Estudio de MapReduce y Hadoop.

- Estudio y comparación de algoritmos de alineamiento.

- Diseño e implementación de Smith-Waterman en Hadoop.

- Pruebas del software.

- Memoria del proyecto

3.3. Requisitos funcionales

1. Apache Hadoop instalado

2. Sistema multi-nodo o pseudo-distribuido

3. Implementación del algoritmo Smith-Waterman

4. Configuración del cluster y de los roles del nodo

5. NameNode formateado con sistema de ficheros HDFS

6. Implementación de la configuración de trabajos en MapReduce

7. Comunicación entre nodos

8. Particionado del NameNode para brindar bloques a los DataNodes

9. Parametrización de secuencias base

10. Implemntación de clase propia con funcionalidades de Hadoop sobrecargadas

11. Comunicación por puerto SSH en localhost y en distribuido

12. Versión 1.6 de la máquina virtual de Java

3.4. Requisitos no funcionales

1. Usabilidad del software
2. Tolerancia a fallos
3. Control del flujo de los datos entre nodos
4. IDE de desarrollo con el plugin de Hadoop

3.5. Soporte hardware

Al tratarse de un sistema distribuido nos encontramos con múltiples unidades de hardware. Como en la imagen 3-1 de un cluster de Hadoop. Al tratarse de un sistema distribuido nos encontramos con múltiples unidades de hardware

Procesador

CPU Intel i5 2.26 Ghz

Memoria

8 GB ECC RAM

Almacenamiento

Disco SATA 1TB

Cable de red

Gigabit Ethernet

Router

3com 4 puertos.

Switch

Cisco 8 puertos.

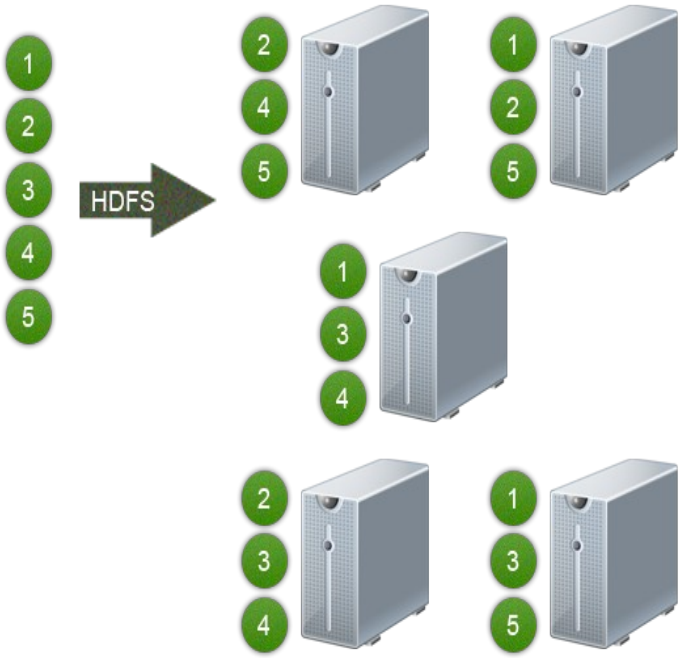


Imagen 3-1. Esquema de un cluster de Hadoop

3.6. Posibles riesgos del proyecto

1. Planificación temporal optimista: no se acaba en la fecha prevista.

2. Equipo de proyecto demasiado reducido: postergación del proyecto

3. No se realiza correctamente la fase de test: sistema con errores y falta de calidad.

4. Incumplimiento de alguna norma, reglamento o legislación.

5. Abandono del proyecto antes de la finalización

3.7. Catalogación de riesgos y plan de contingencia

Catalogación de riesgos

	Probabilidad	Impacto
R1	Alta	Crítico
R2	Alta	Crítico
R3	Alta	Crítico
R4	Baja	Crítico
R5	Mediana	Catastrófico

Plan de contingencia

	Solución a adoptar
R1	Replanteamiento de algunas actividades
R2	Negociar ayudas externas
R3	Diseñar test y crear una fase beta de pruebas
R4	Revisar la normativa y replantear los aspectos afectados
R5	No tiene solución

3.8. Alternativas

Construcción de un proyecto de inmersión en los sistemas distribuidos puramente teórico. Al ser la parte con más posible problemática el desarrollo de software una alternativa es un estudio bioinformático mas profundo, de la situación actual y estado del arte.

Contenido

- Descripción teórica y situación de los sistemas distribuidos
- El paradigma MapReduce: profundización en sus implementaciones y comparación
- Secuencias, evolución y clasificación filogenética
- Posibilidades futuras de la bioinformática
- Software paralelizable y Hadoop
- Conclusiones

3.9. Presupuesto y software libre

El proyecto será realizado con software libre. El software utilizado se encuentra bajo licencias EPL[2], Apache License v2.0[3], GPL v2.0[4], GPL v3.0[5] y LGPL v3[6].

Las aplicaciones a emplear son las siguientes:

- Apache Hadoop: servidor sobre sistemas distribuidos.
- OpenProj: editor de diagramas de Gantt y de tareas.
- GIT: gestor de versiones.
- Eclipse: IDE de Java.

2 Eclipse Public License: http://www.eclipse.org/legal/epl-v10.html

3 Apache License, Version 2.0: http://www.apache.org/licenses/LICENSE-2.0.html

4 General Public License v2.0: http://www.gnu.org/licenses/gpl-2.0.html

5 General Public License v3.0: http://www.gnu.org/licenses/gpl.html

6 Lesser General Public License v3.0: http://www.gnu.org/licenses/lgpl.html

- OpenOffice: suite ofimática libre.

- GNU/Linux: sistema operativo libre.

3.10. Beneficios e inconvenientes

Beneficios:

· Formación del creador

· Proyecto de gran interés para el autor.

· Creación de un proyecto de interés para el creador

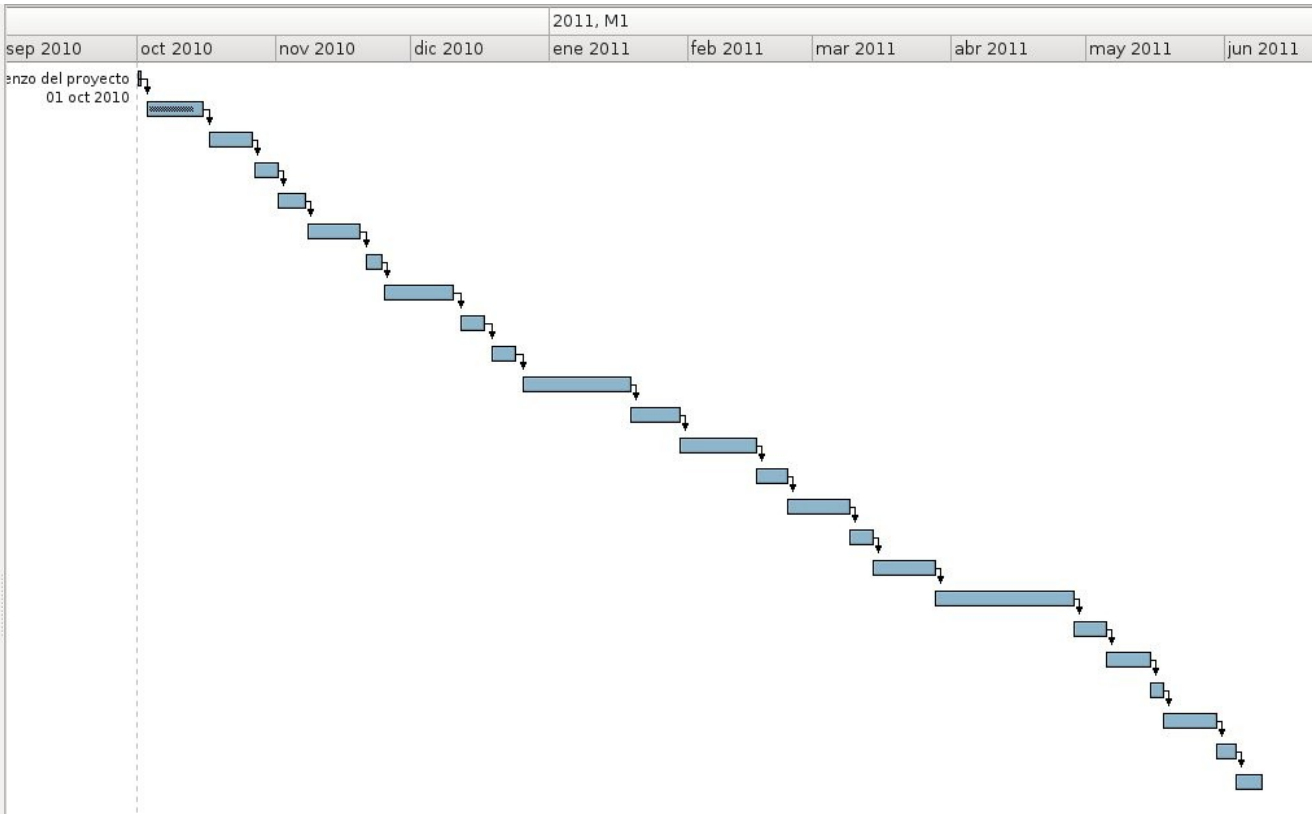
Inconvenientes:

· Inversión importante de tiempo

· Necesidad de un periodo de formación

3.11. Diagrama de Gantt

Diagrama de Gantt respectivo a la planificación del proyecto



3.12. Planificación del proyecto

3.12.1. Recursos

Recursos humanos y valoración de estos

Recursos humanos	Valoración
Jefe de proyecto	30 € /h
Analista	20 €/h
Programador	15 €/h
Técnico de pruebas	10 €/h

Requisitos mínimos del sistema

Software: Hadoop.

S.O.: GNU / Linux

CPU:

Capacidad: 100TB

3.12.2. Tareas

Nº	Descripción de la actividad	Duración	Recursos
	Inicio del proyecto: asignación y matriculación	**1h**	
	Planificación	**31h**	
	Estudio de viabilidad	30h	

	Aprobación del estudio de viabilidad	1h	
	Análisis de la aplicación	**16h**	
	Análisis de requisitos	1h	
	Análisis de datos	2h	
	Análisis algorítmico	10h	
	Documentación del análisis	3h	
	Estado del arte	**15h**	
	Documentación del estado de la materia	8h	
	Documentación del funcionamiento de la tecnología	7h	
	Diseño de la aplicación	**10h**	
	Diseño modular	3h	
	Diseño de las pruebas	3h	
	Documentación del diseño	3h	
	Aprobación del diseño	1h	
	Desarrollo de la aplicación	**68h**	
	Preparación de la IDE	5h	
	Diseño algorítmico	30h	
	Diagrama de clases	3h	
	Desarrollo algorítmico	30h	
	Test y pruebas	**20h**	
	Pruebas de unidad	5h	
	Pruebas de integración	5h	
	Test de rendimiento	5h	
	Documentación del desarrollo y test	4h	
	Aprobación del desarrollo de pruebas	1h	

	Implantación	**5h**	
	Instalación	1h	
	Pruebas reales	4h	
	Memoria del proyecto	**20h**	
	Defensa del proyecto	**5h**	

3.12.3. Planificación temporal

Calendario del proyecto: El proyecto se realizará desde octubre del 2010 hasta la convocatoria de presentación de proyectos en junio del 2011 a razón de unas 10 horas semanales.

Herramientas de planificación: GNU Planner.

Capítulo 4

Programación y diseño

4.1. Problema: volúmenes y análisis de datos genómicos

Actualmente el análisis de grandes volúmenes de datos necesarios para comparar una gran cantidad de cadenas de ADN no son abarcables por un computador. En todo caso por un conjunto de computadores. Siendo *n* y *m* longitudes de secuencia, la complejidad de los algoritmos de análisis de secuencias son de O(n·m). Al ser una complejidad cuadrática y volúmenes del orden de gigabytes el tiempo de cómputo se torna demasiado elevado.

4.2. Planteamiento: Hadoop y Smith-Waterman

Hay varias tecnologías que implementan el paradigma MapReduce. Hadoop es un proyecto libre, maduro y extendido entre multitud de empresas. Provee la eficiencia necesaria para poder realizar el análisis de bastos volúmenes de datos. Además provee un alto nivel de abstracción de los sistemas distribuidos.

Para elegir el algoritmo podríamos optar entre Needleman-Wunsch, que es un algoritmo basado en alineamientos globales, o bien Smith-Waterman, que siempre provee la mejor solución local. El algoritmo Smith-Waterman es el mejor para este caso. Las secuencias de ADN con cambios evolutivos suelen

codificar al final de la secuencia, por lo que el alineamiento no sería global sino local. El objetivo es entonces aplicar el algoritmo Smith-Waterman para comparar secuencias de ácidos nucleótidos en formato FASTA sobre la plataforma Hadoop que permitirá paralelizar y distribuir el cálculo

4.3. Diseño

4.3.1. Esquema de la paralelización de Smith-Waterman en Hadoop

La finalidad del programa es: dadas unas secuencias de ADN (input) obtener el mejor alineamiento (output) con una secuencia base. Para paralelizar el alineamiento de secuencias se introducirán todas las secuencias en el sistema de ficheros HDFS. Todos los nodos recibirán del NameNode un bloque con una serie de secuencias. Estas secuencias se alinearán con su respectiva secuencia base. El diseño dará o el mejor o les mejores alineamientos. En el archivo de texto de salida se escribirán estas secuencias homólogas.

4.3.2. Introducción al funcionamiento interno de Hadoop

Shuffle and sort

Todas las tareas reduce reciben un conjunto de datos ordenados por clave. Los mecanismos de Hadoop para ordenar los datos intermedios generados a la salida de las tareas del map son el shuffle y el sort. El shuffle y el sort se aplican una vez finalizada la función map. En la imagen 4-1 se muestra la evolución de los datos en un proceso en Hadoop. La fase de shuffle y sort se aplica entre el bloque de la función map y el de la función reduce de la imagen.

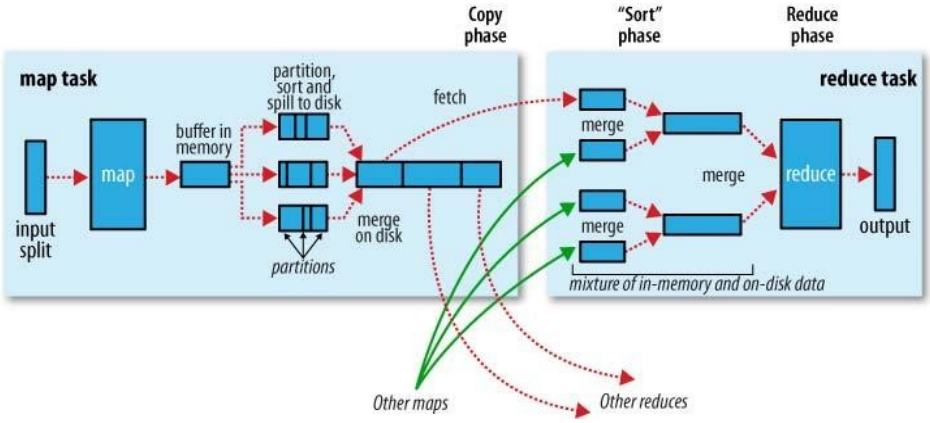


Imagen 4-1. Shuffle y sort en MaReduce

Funcionamiento del map

Cada tarea map escribe su salida en un buffer de memoria. Cuando el contenido del buffer rebasa un cierto umbral (por defecto del 80 %) un thread en background de Hadoop empezará la fase de spill o de escritura a disco. La fase de spill vacía los datos del buffer a disco para que la salida del map pueda seguir escribiéndose en el buffer. Si en algún momento de la ejecución del map el buffer se llena por completo, entonces el map se detendrá hasta que el proceso de spill finalice. Cada vez que el buffer de memoria rebasa el umbral del spill, se crea un nuevo archivo para el spill. Así que después de que el map haya escrito su último registro pueden haber varios archivos spill. El buffer por defecto es de 100 MB. Hadoop permite modificar en los archivos de configuración varios parámetros de ejecución. El tamaño del buffer puede regularse mediante la propiedad *io.sort.mb*.

Antes de que se escriba en disco se realiza un particionado de los datos (o partitioning). El thread primero divide los datos en un número de particiones correspondiente al número de reducers a los que serán enviados. Una vez definida cada partición de los datos se realiza una ordenación en memoria de todas las tuplas <clave,valor> que hay en la partición por clave.

Funcionamiento del reduce

Los maps finalizan en diferentes tiempos por lo que se empiezan a copiar los datos de salida de las tareas del map tal y como van acabando. Esto es conocido como la *fase de copy* de la tarea reduce. La tarea reduce tiene ejecutandose una serie de threads para copiar la salida del map. Busca las salidas del map en paralelo. Por defecto son cinco threads pero este número puede ser modificado la propiedad *mapred.reduce.parallel.copies*. Las salidas de los maps se copian en la memoria de los tasktrackers de los reduce si son suficientemente pequeños y sino en disco.

Mientras las copias se acumulan en memoria un thread en background se encarga de la fase de merge o shuffle. En esta fase se combinan los datos que han llegado de todos los map creando archivos agregados más grandes. Cuando todas las salidas del map han sido copiadas, la tarea del reduce cambia a la fase de merge. Esta combina todos los datos recibidos manteniendo el orden de las claves. Después se invoca la función reduce por cada clave de los datos ordenados de entrada. La salida de esta fase se escribe directamente a la salida del sistema de archivos HDFS.

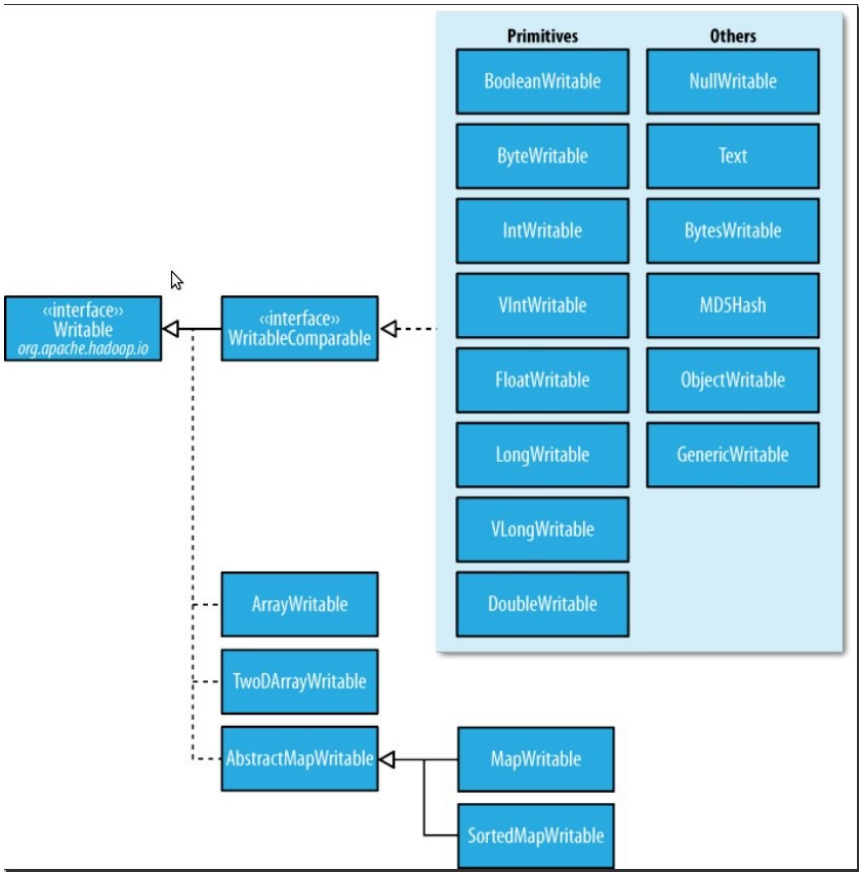


Imagen 4-2. Jerarquía de las Writable Classes

Las clases por defecto proporcionadas por Hadoop (imagen 4-2) atienden a distintos tipos de variables y proveen una serie de funciones. Estas implementan getters, setters, funciones hash, de comparación, ordenación, etc. Si queremos podemos usarlas o bien crear una nueva clase que tenga las mismas funciones que estas. Entre las funciones está la de ordenación de claves llamada *RawComparator*. Esta función es la encargada de ordenar los datos cuando se ordena el buffer de datos intermedios al finalizar la fase map. En concreto, se ocupa de comprobar si entre dos objetos uno es mayor, menor o igual. Por lo tanto, será llamada por Hadoop para cada pareja de tuplas <clave,valor> que ha de ordenarse al finalizar la fase map y empezar el shuffle.

4.3.3. Diseño 1

En el primer diseño como secuencia de entrada hay cómo mínimo una secuencia de referencia y una de consulta. El map tan solo se ocupa de leer los bloques de texto de entrada. El alineamiento de secuencias se da en el proceso de ordenación. En éste todas las secuencias se procesan por pares, al igual que los alineamientos. Aprovechar este proceso propio de Hadoop permite reutilizar operaciones para introducir el diseño. El reduce escribe en disco los mejores alineamientos.

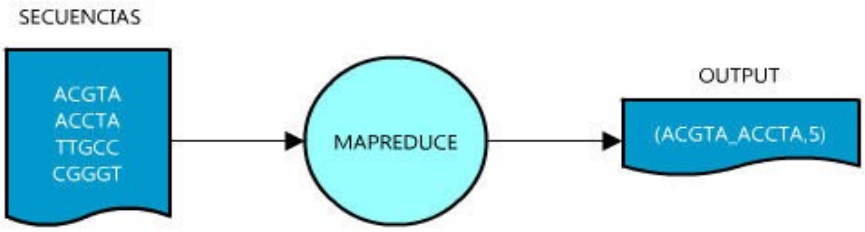


Imagen 4-3. Visión lógica del primer diseño.

El proceso de ordenación interno de las claves en Hadoop se hace con aquellos pares de <clave,valor> recogidos por el output collector del map. Cada tipo de datos implementa una función llamada RawComparator, que devuelve -1, 0 o 1 dependiendo de si el valor comparado es menos, igual o mayor que el valor con el que se compara. Esto se hace para cada map. El proceso de ordenación de claves acabará agrupando en tuplas los valores de las claves iguales antes de llegar al proceso reduce. MapReduce garantiza que el input de cada reduce está ordenado por clave. En la imagen 4-3 se muestra la entrada y la salida de los datos. La salida es el mejor alineamiento entre las secuencias de entrada con su respectivo score. El programa lee una lista de secuencias. Las secuencias se alinearan entre ellas, de donde se calcularán los scores. Posteriormente calcula el mejor alineamiento. Al finalizar el proceso escribe el par de secuencias que han dado el mejor alineamiento y el score de éste.

Se quiere sobrecargar la función de ordenación por defecto. Para esto, se ha creado una llamada DnaWritable. Para poder utilizarla tan solo se ha de especificar en los parámetros del map y del reduce y en la función main, en la configuración del trabajo. La función de comparación es llamada transparentemente. Para modificar la función de comparación por defecto, ya que al comparar secuencias genómicas no sirve la comparación exacta entre dos cadenas alfanuméricas. Se deben tener en cuenta los casos de mismatch y de añadir gaps al comparar dos cadenas de ADN. La configuración es la siguiente:

```
conf.setOutputKeyClass(DnaComparator.class);

conf.setOutputValueGroupingComparator(DnaComparator.Comparator.class);
```

Dentro de la clase DnaWritable se creó una clase para comprobar la similaridad entre dos secuencias. Esta clase implementa la función de Smith-Waterman y considera que dos cadenas son iguales si tienen un porcentaje alto de similitud. Para el primer diseño se implementó una función de comparación propia. Como se ha explicado previamente, se llamaba automáticamente a esta función en el proceso de ordenación de datos intermedios en los buffers de salida de las tareas map. En el primer diseño, la clave era la secuencia de DNA y el valor el score.

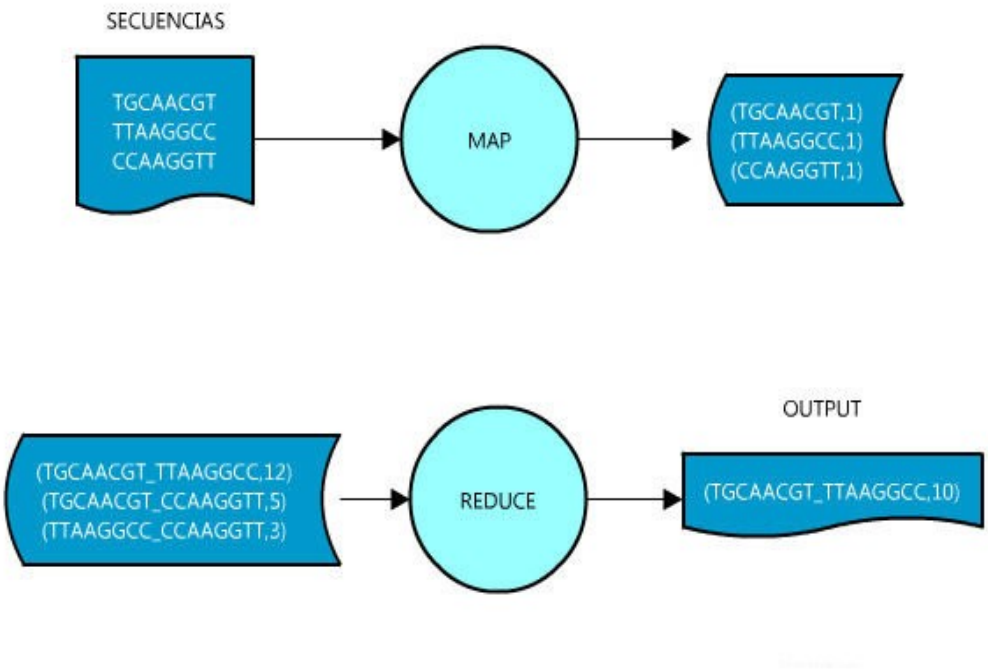


Imagen 4-4. Visión del proceso del primer diseño con map y reduce

En la imagen 4-4 se muestra el primer diseño con las entradas y salidas de datos de las funciones map y reduce. Como entrada cada map recibe toda una lista de secuencias. El map genera todas las parejas posibles entre las secuencias de entrada del map como a entrada del reduce. Al reduce le llegan los pares de cadena comparados y el score entre ellas. En el reduce se guardan los que tienen mayor *score* y se escriben en el fichero de salida.

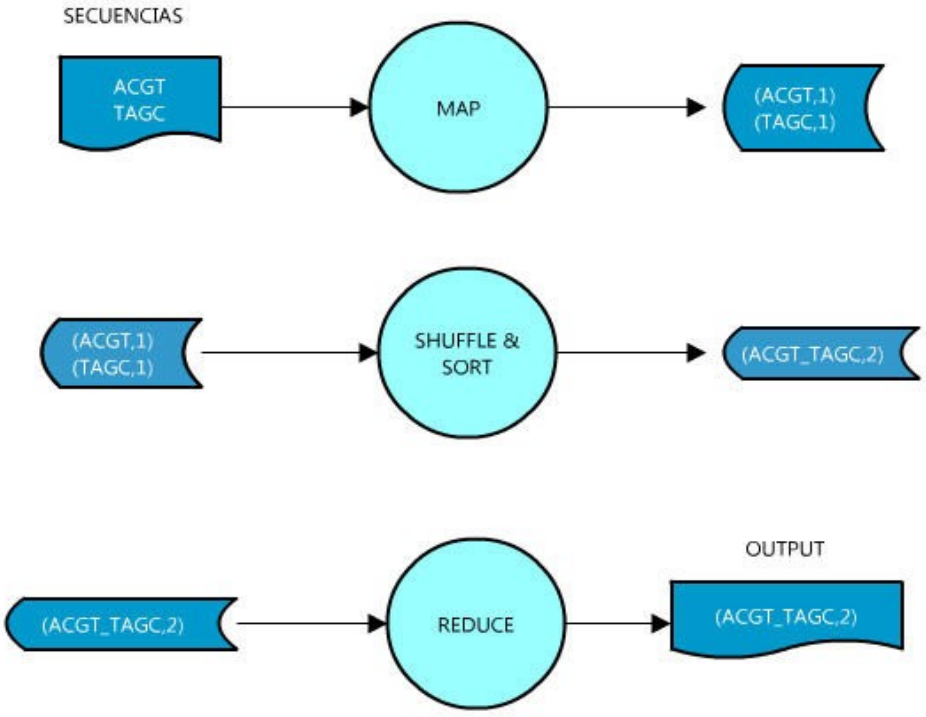


Imagen 4-5. Visión lógica del primer diseño incluyendo shuffle y sort

En la imagen 4-5 se muestra un ejemplo del primer diseño. Tenemos un archivo FASTA que contiene las secuencias ACGT y TAGC. El map lee las secuencias. La clave es la secuencia y el valor un valor inicial marcado en 1. En el proceso de shuffle y sort se encuentra cada pareja de secuencias y se alinea. El nuevo score es registrado y la pareja conforma la clave. El reduce recibe el par de claves con su score y los guarda para que sean escritos en la partición HDFS. El pseudocódigo de las tres funciones principales es el siguiente:

```
Función map (string linea)
      output.collect(linea, 1)
Función igualación (linea1, linea2)
      score = SmithWaterman(linea1, linea2)
      output.collect(linea1 + "_" + linea2, score)
Funcion reduce (clave, valores[ ])
      max = 0
      por valor en valores [ ]
              si valor > max
                  max = valor
      output.collect(calve, max)
```

4.3.4. Problemas del diseño 1

En el primero diseño en el proceso de ordenación no se comparaban todos con todos. Eso es así porque Hadoop realiza la ordenación comparando si una variables, respecto a otra, es mayor, igual o menor. Por ejemplo, teniendo tres secuencias: AGTC, TAGC y ACGT. Si llegan en este orden, entonces primero tendremos un único elemento en la lista:

```
AGTC
```

Al llegar TAGC se colocará segundo en la lista:

```
AGTC
TAGC
```

Pero al ordenar ACGT solo se comparará con AGCT, verá que éste es mayor y se colocará primero, por lo que nunca se comparará con TAGC

```
ACGT

AGTC

 TAGC
```

Esto hace que el diseño no se pueda llevar a cabo porque el algoritmo de ordenación de claves intermedias de Hadoop no compara a todos con todos.

Otro problema es escribir en la fase de shuffle y sort. Como el score se produce en esa fase, entonces ahí debía ser guardado. Cómo solo hace la ordenación, entonces no permite la escritura y modificación de la pareja <clave, valor>.

4.3.5. Diseño 2

En el segundo diseño se planteó ejecutar el algoritmo Smith-Waterman en la función map. La lógica era comparar una secuencia con una serie de secuencias de referencia. El algoritmo Smith-Waterman se ejecuta en el map y el reduce recoge el mejor alineamiento.

Como entrada se recibe una secuencia que se comparará con otras n secuencias. En el map, a parte de leer todas las secuencias de entrada, se ejecuta el algoritmo Smith-Waterman. La clave de salida es la pareja que ha formado el mejor alineamiento. En este caso hay dos reduce. En el primer reduce se descartan aquellas secuencias que no tienen más de un 90% de similitud. En el segundo reduce se guarda la mejor. En la imagen 4-6 se muestra cual sería la entrada y la salida del programa.

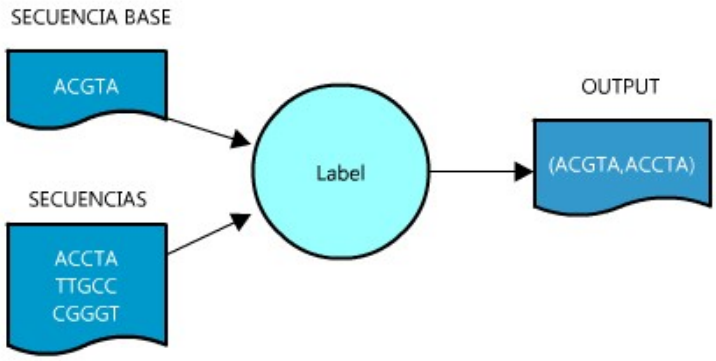


Imagen 4-6 Proceso del diseño 2

Los datos que recibe el map son una secuencia de referencia y *n* secuencias de consulta. La salida del map es el score de alineamiento de la secuencia de consulta con respecto a cada secuencia de referencia. En la imagen 4-6 se muestra el score respecto a la secuencia ACGTA. El reduce recibe estos pares de <clave,valor> con <secuencia_referencia, score> ordenados alfabéticamente. El reduce genera como salida para el siguiente reduce aquellos alineamientos que tengan una homogeneidad ortóloga. Superior al 90% de similitud. El siguiente reduce recibe como tuplas de <clave,valor> la secuencia base y la secuencia comparada junto a su score. Escribe finalmente en un archivo que guarda las secuencias del mejor alineamiento respecto a nuestra consulta.

Observando el proceso solo como entrada y salida tenemos una secuencia base que comparamos con una serie de secuencias. La ejecución de MapReduce devolverá la secuencia de la referencia que obtiene un mejor alineamiento con la secuencia base y el score obtenido de este alineamiento.

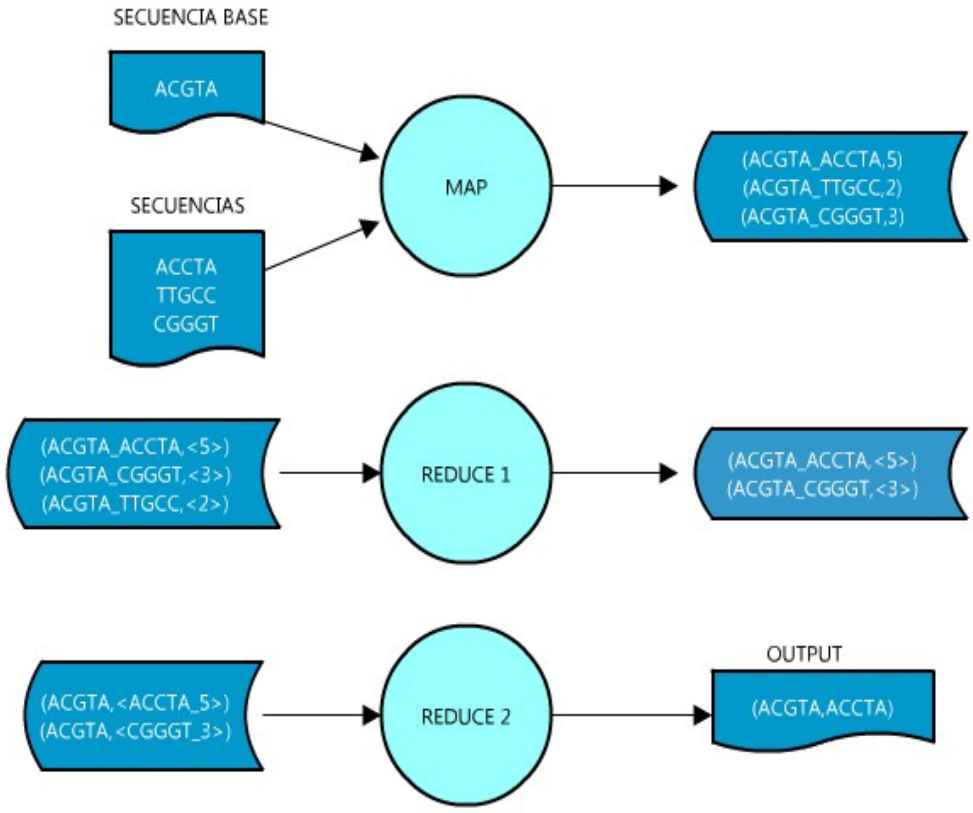


Imagen 4-7. Entrada y salida del diseño 2

Primero Hadoop reparte los bloques de texto entre los distintos TaskTracker y la secuencia base. Cada TaskTracker ejecuta un map, donde se llama al algoritmo Smith-Waterman que devuelve un valor entero: el score. En el output collector se especifica la secuencia de entrada y base como valor y el score de la comparación con la secuencia base como clave. El output collector recoge todos los valores, en la fase sort se ordenan, y en el shuffle se juntan en tuplas aquellos que tienen claves iguales. En el primer proceso reduce, se comparan los valores de las secuencias y se guardan las que rebasan el umbral del 90%. Los datos vuelven a hacer la fase de shuffle y sort. El NameNode devuelve los datos al siguiente reduce, que al recibir una lista de valores para una sola clave podrá guardar el mejor alineamiento final. A continuación se muestra el pseudocódigo del algoritmo.

```
function Map:
     int score = 0;
     base = leerBase();
      mientras linea.siguiente()
            score = SmithWaterman(base, linea)
            output.collect(linea +”_”+ base,, score)

function Reduce1:
     float umbral = 0.9;
      mientras valor.siguiente()
            si (superarUmbral(valor, umbral)
                        output.collect(clave1, clave2 +“_”+ valor)

function Reduce12
     int max = 0;
     string seq = “”;
      mientras valor.siguiente()
            si (score > max)
                  seq = valor1;
     output.collect(clave, seq)
```

4.4. Pruebas

4.4.1. Instalación de Hadoop en un nodo

Hadoop está desarrollado en Java y requiere como mínimo la versión 1.6 de JVM. Para acceder y gestionar los nodos Hadoop utiliza el protocolo ssh. Cada nodo tiene generada una clave. Los esclavos listarán entre su lista de claves autorizadas la del master. Hadoop se conecta por Ipv6 y debemos deshabilitar esta opción si no vamos a conectarnos a una red de esta versión del protocolo IP.

Apache provee mirrors para descargar su software. La última versión estable es siempre la más recomendable, ya que soluciona bugs y provee mejoras. Hadoop no requiere de un proceso de integración en el sistema como otros proyectos software. Lo podemos ejecutar desde su carpeta casi directamente. En la subcarpeta del directorio raiz del proyeto Hadoop se encuentra la carpeta *conf*. Todas las configuraciones se especifican en los archivos XML contenidos en esta. En el shell script hadoop-env.sh se especifica la ruta a la máquina virtual de Java con versión 1.6.

Antes de iniciar Hadoop hay que dar formato al NameNode. Al darle formato se borran todos los datos contenido. La primera vez que se formatea permite poder escribir posteriormente en HDFS. En la carpeta *bin* se hayan todos los binarios que permite arrancar, para e interaccionar con el sistema.

4.4.2. Configuración de Hadoop en modo pseudo-distribuido

Al configurar Hadoop para que se comporte como un sistema distribuido se han de especificar la cantidad de funciones map y reduce. Pese a que se ejecute en un nodo lanza los mismos daemons que si estuviese en un cluster. Cada TaskTracker ejecuta su función con el bloque de datos enviado por el NameNode. En la ejecución del programa Hadoop envía mensajes de estado del cluster por trabajo. Los logs muestran el porcentaje ejecutado de cada map o reduce. Estos se escriben junto a cada instante en que se realiza. De este modo podemos cronometrar los tiempos de cada map y calcular el tiempo de ejecución en un cluster.

El número de procesos map está relacionado con el número de archivos que hay en el directorio de entrada del proceso. Así pues una ejecución de Hadoop con 8 archivos en el input path creará 8 funciones map. De otra manera se puede parametrizar el número de maps con la llamada a la siguiente función:

```
    JobConf.setNumMapTasks(int num)
```

4.4.3. Pruebas de instalación: contador de palabras

Para hacer las pruebas de Hadoop se retoma el ejemplo del apartado 2.4.3. El contador de palabras. Para cada palabra se calculará el número de veces que aparece en los archivos de entrada. Hadoop escribirá un archivo de salida con la lista de palabras que aparecen junto al número de veces que están en el texto.

El entorno de programación es la IDE Eclipse con el plugin desarrollado por Hadoop. Este plugin permite conectarse al sistema de ficheros HDFS y ejecutar tareas desde Eclipse. Tan solo se han de rellenar los parámetros de la configuración. Estos son los puertos de los daemons y carpetas de HDFS y la raíz de Hadoop y el código para ejecutar el software.

Para el contador de palabras se han extraído libros del proyecto Gutenberg. La shell de Hadoop provee funciones para poder interactuar con HDFS. Los libros se copian al sistema de ficheros del usuario a la partición de HDFS mediante el siguiente comando:

```
hadoop dfs -copyFromLocal <localsrc> <destination>
```

Algo importante en Hadoop es que todas las carpetas de destino no pueden existir al lanzar la aplicación. Esto es así porque se considera que los resultados de una ejecución de Hadoop no se deben mezclar con otra ya existente. Si la carpeta de salida existe debe ser borrada.

Ahora se puede ejecutar Hadoop con la clase WordCount desde Eclipse, utilizando el plugin MapReduce. En la figura 4-8 se muestra Eclipse al finalizar la ejecución. En la terminal se muestran las estadísticas del proceso e información interna útil para ver los datos manipulados.

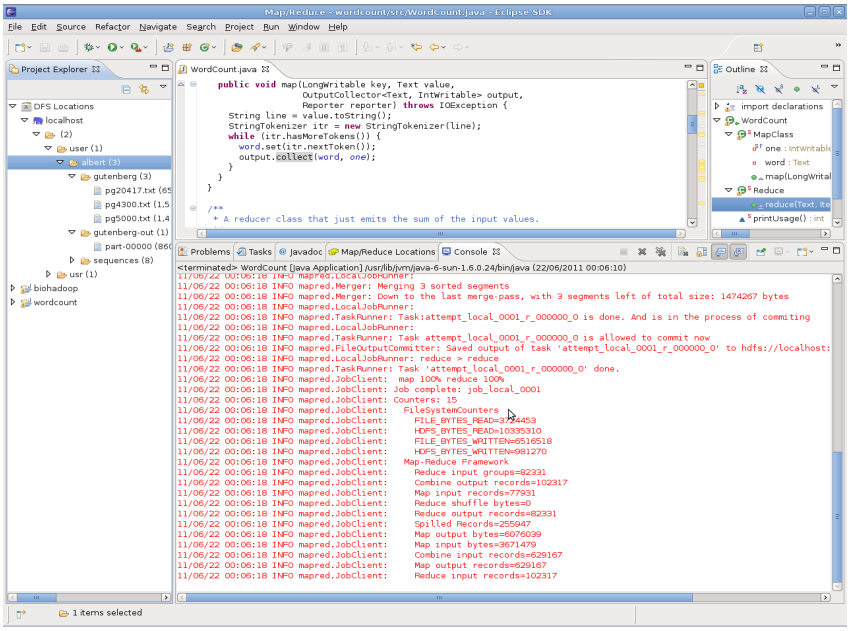


Imagen 4-8. Aspecto del contador de palabras en Hadoop

Hadoop provee información y estadísticas a través del navegador accediendo al puerto 50030 en localhost. Los resultados son de todo el sistema. Proyectos finalizados, ejecutándose, fallados y estados actuales. Es útil y usado para seguir el estado del sistema y su evolución incluso remotamente a través de un navegador. En la imagen 4-9 se muestra esta interfaz al inicio de la aplicación.

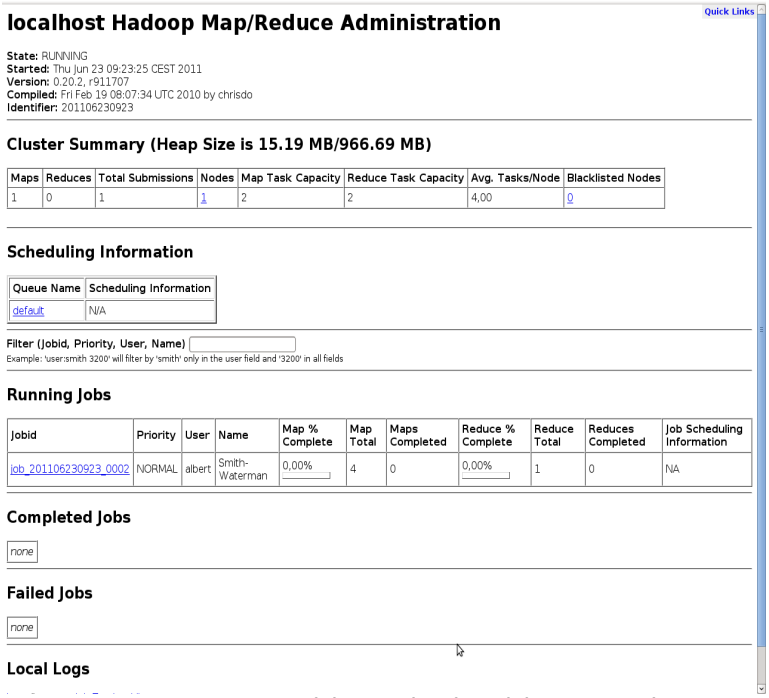


Imagen 4-9. Aspecto del contador de palabras en Hadoop

4.4.4. Pruebas de validación: 50 MB

Para comprobar la correcta implementación del Smith-Waterman se probaron tres casos base. No match (< 50% similitud), semi-match (50-90% similitud), match (>90% similitud) para ver si funcionaba correctamente. En los tres casos dio resultados positivos. Además el algoritmo ha sido probado múltiples veces en las siguientes pruebas de la implementación en Hadoop con volúmenes mayores.

En el proceso MapReduce tiene de entrada una serie de archivos que contienen secuencias genéticas. Estas serán comparadas con una secuencia base. Para cada una de las secuencias con una similitud superior al 50% entonces se escribirá en la salida la secuencia junto a el score del alineamiento.

Para las pruebas en Hadoop se ha utilizado un volumen de 50 MB de datos genómicos. En la imagen 4-10 se muestra un gráfico con los tiempos de ejecución de media por proceso y nodos. Los procesos map en un nodo se han ejecutado de media en 169 segundos. El proceso reduce en 4 segundos. Estos resultados ilustran cuan mayor es la necesidad de repartir el cómputo en procesos map. En las consiguiente imágenes se divide el tiempo por el número de nodos. Por tanto el tiempo es inversamente proporcional al número de nodos.

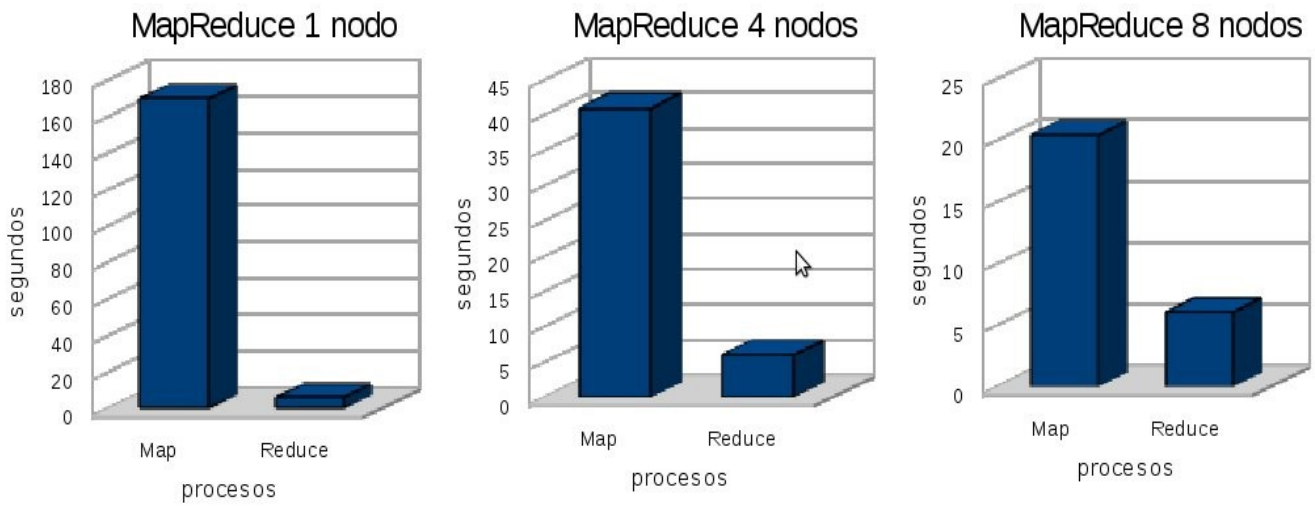


Imagen 4-10. Tiempo en segundos por proceso y nodos (50 MB)

Por tanto al distribuir el cómputo baja el tiempo de ejecución del map y se acerca más al tiempo del reduce. Un caso de comparación interesante es alargar las secuencias de referencia para el algoritmo Smith-Waterman.. La prueba siguiente cambia la longitud de las secuencias de Hadoop de 80 a 120, pero sigue manteniendo un volumen de 50 MB. La longitud de las secuencias es mayor pero el número de secuencias es menor.

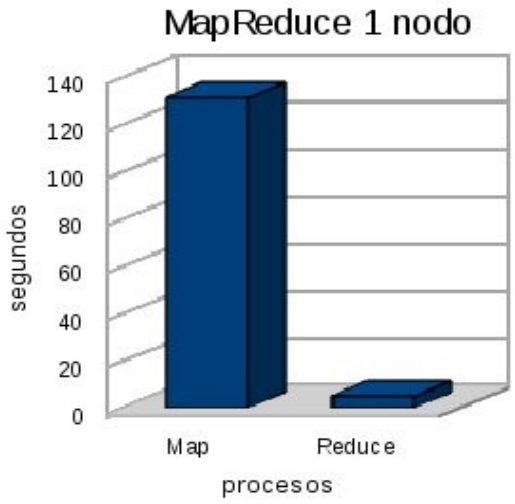


Imagen 4-11. Tiempo en segundos por proceso en ocho nodos (50 MB, l=120)

El resultado de la imagen 4-11 es interesante, porque pese a aumentar la longitud de las secuencias el tiempo de ejecución a disminuido. Alargar la secuencia depende de las propiedades del nodo y de su memoria. En pruebas con longitudes grandes se recibía el error *Java heap space* por falta de memoria.

4.4.5. Pruebas de validación: 500 MB

Para poner a prueba con más datos se ha decuplicado el volumen de la entrada de datos y se ha probado con 500 MB. La longitud de las secuencias sigue siendo la misma, de 80 nucleótidos. Lo único que incrementa es el número de secuencias. Para las pruebas se ha probado con 1 y 10 maps. En la imagen 4-12 se muestran los resultados de los tiempo. Con 10 nodos el tiempo reduce linealmente hasta poder asemejar el tiempo del map al del reduce. Los resultados son similares a las pruebas de 50MB. Pero se han más que decuplicado. Así al incremento del volumen de datos crece más el tiempo que los datos.

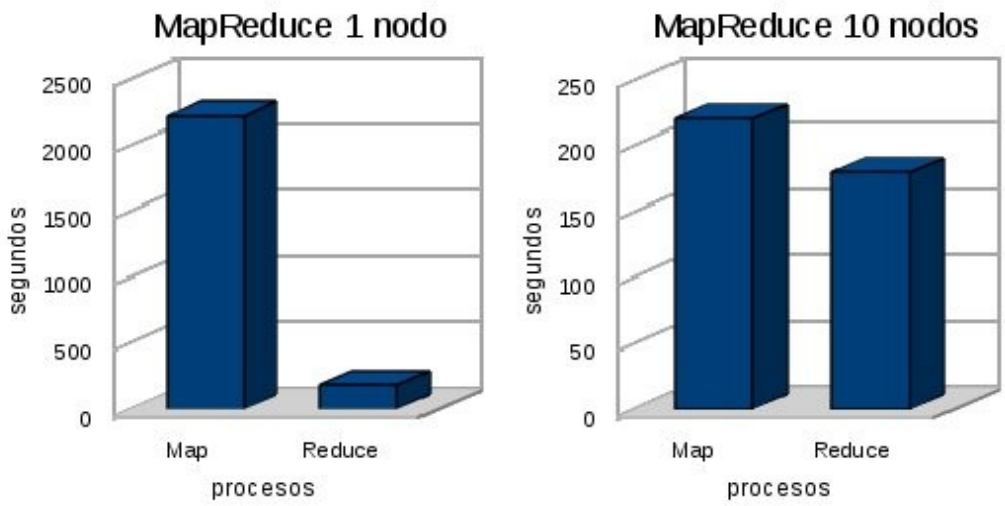


Imagen 4-12. Tiempo en segundos por proceso en un nodo (500 MB)

El software ha superado las pruebas dando las salidas esperadas. Se puede afirmar que el software es funcional. Las pruebas de validación han servido para enfrentarse a problemas más cercanos a la realidad puesto que los datos genómicos suelen ser del orden de cientos de megabytes o gigabytes.

Capítulo 5

Conclusiones

5.1. Objetivo: alineamiento paralelo de secuencias

El alineamiento paralelo de secuencias se ha logrado. En el estudio de viabilidad había una serie de objetivos generales a conseguir. Los objetivos generales han sido alcanzados por lo que se puede afirmar que el proyecto ha sido realizado.

Objetivo	Estado
Planificación y viabilidad del proyecto.	*Alcanzado*
Estudio de los sistemas distribuidos.	*Alcanzado*
Estudio de MapReduce y Hadoop.	*Alcanzado*
Estudio y comparación de algoritmos de alineamiento.	*Alcanzado*
Diseño e implementación de Smith-Waterman en Hadoop.	*Alcanzado*
Pruebas del software.	*Alcanzado*
Memoria del proyecto.	*Alcanzado*

5.2. Ampliaciones futuras

5.2.1. Distancias evolutivas y árboles filogenéticos

Las modificaciones entre especies marcan las distancias evolutivas. A partir de estas se puede saber cuan lejos está evolutivamente una especie de otra. Una manera de concluirlo es a partir de los resultados del score del algoritmo Smith-Waterman y la longitud de las secuencias. Las distancias evolutivas son la base de la filogenética, que es una ciencia que organiza las especies según proximidades evolutivas. La clasificación en árboles filogenéticos se construyen teniendo en cuenta

Una vez alcanzado este se puede integrar el algoritmo MrsRF [6]. Éste es una implementación abierta en el paradigma MapReduce, que analiza grandes conjuntos de árboles evolutivos. La implementación ha sido optimizada y ha sido probada hasta con 33.000 árboles filogenéticos. Esta ampliación consistiría entonces en considerar el presente proyecto como la primera parte de una cadena de procesos.

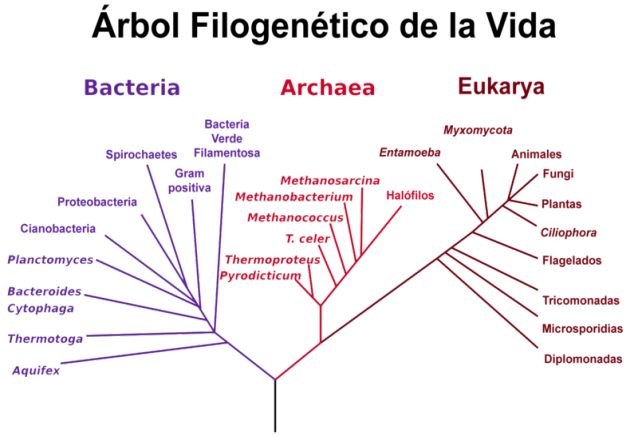


Imagen 5-1 Árbol filogenético para genes rRNA

5.2.2. Alineamiento múltiple de secuencias (MSA)

El alineamiento múltiple de secuencias es un alineamiento de tres o más secuencias biológicas. En este caso se podrían alinear un conjunto de secuencias base. El MSA es utilizado para ver relaciones, genéticas, evaluar conservación de dominios proteicos, alineamiento con testigo evolutivo, etc. El problema tiene similitudes con el del presente proyecto. Se emplea programación dinámica para resolver el MSA, pero para *n* secuencias se ha demostrado que es un problema NP-completo. La ampliación, en este caso, sería una modificación del proyecto original.

Apéndice 1

Definiciones y acrónimos

Bioinformàtica

ADN: Ácido desoxirribonucleico

ARN: Ácido ribonucleico

Alineamiento de ADN: proceso de búsqueda de homología entre secuencias locales de ADN

Aminoácido: Conjunto de tres nucleótidos.

Bioinformática: Aplicación de la computación a la gestión y análisis de datos biológicos.

Codón: Conjunto de tres nucleótidos que codifica un aminoácido

Filogenética: clasificación científica de las especies basada en las relaciones de proximidad evolutiva.

Gen: Un gen es una secuencia de nucleótidos en la molécula de ADN que contiene la información para la síntesis de una proteína

Genoma: Totalidad de la información genética que posee un organismo en particular

Homología: Relación que existe entre dos partes orgánicas diferentes cuando sus determinantes genéticos tienen el mismo origen evolutivo.

Nucleótido: Compuesto orgánico constituido por una base nitrogenada, un azúcar y ácido fosfórico.

Proteína: Biopolímero formado por una o varias cadenas de aminoácidos

Score: Resultado del alineamiento en Smith-Waterman, expresión de la similarity

Similarity: Similitud determinada por el algoritmo Smith-Waterman

Informática

ASF: Apache Software Foundation

GFS: Google File System

Hadoop: Framework libre que implementa el paradigma MapReduce

HDFS: Hadoop Distributed File System

MapReduce: Paradigma creado por Google de computación paralela para grandes volúmenes de datos para grupos de ordenadores.

Nodo: Punto de intersección o unión de varios elementos que confluyen en el mismo lugar

Sistema distribuido: Colección de computadoras separadas entre si físicamente, pero conectadas por una red distribuida.

WAN: Wide Area Network

Recursos

Bibliografía

[WHI09] Tom White (2009) *Hadoop; the definitive Guide*. O'Reilly.

[TAN97] Andrew S. Tanenbaum y Albert S. Woodhull (1997) *Sistemas operativos.* Prentice Hall

[TAN08] Andrew S. Tanenbaum y Maarten Van Steen (2008) *Sistemas distribuidos.* Prentice Hall

[VEN09] Jason Vener (2009) *Pro Hadoop - Build Scalable, Distributed Applications in the Cloud.* Apress

[LAM11] Chuck Lam (2011) *Hadoop in action*. Manning

[NEI04] Neil C. Jones and Pavel A. Pevzner (2004) *An Introduction to Bioinformatics Algorithms*

Artículos

[1] Anrew Pavlo, Erik Paulson, Alexander Rasin, Daniel J. Abadi, David J. DeWitt, Samuel Madden, Michael Stonebraker (2009) *A Comparison of Approaches to Large-Scale Data Analysis* .

[2] Jeffrey Dean and Sanjay Ghemawat (2004) *MapReduce: Simplifed Data Processing on Large Clusters*

[3] Xiaohoung Qiu, Jaliya Ekanayake, Scott Beason1, Thilina Gunarathne, Geoffrey Fox (2009) *Cloud Technologies for Bioinformatics Applications.*

[4] Alexander CJ Roth, Gaston H Gonnet and Christophe Dessimoz (2008) *Algorithm of OMA for*

large-scale orthology inference

[5] T.F. Smith and M. S . Waterman (1981) *Identification of Common Molecular Subsequences*

[6] Suzanne J Matthews and Tiffani L Williams (2010) *MrsRF: an efficient MapReduce algorithm for analyzing large collections of evolutionary trees*

[7] Saul B. Needleman and Christian D. Wunsch. (1970) *A general method applicable to the search for similarities in the amino acid sequence of two proteins*

Recursos web

[W3CLO] Hadoop Installation – Documetation – Cloudera Wiki

https://docs.cloudera.com/display/DOC/Hadoop+Installation

[W3HAD] Apache Hadoop Official website

http://hadoop.apache.org/

[W3WIK] Hadoop Wiki

http://wiki.apache.org/hadoop/

[W3DIS] Cloudera's distribution for Hadoop

http://www.cloudera.com/hadoop/

[W3MOL] Análisis moléculares de ADN - Wikipedia

http://es.wikipedia.org/wiki/An%C3%A1lisis_moleculares_de_ADN

[W3MRW] MapReduce – Wikipedia

http://en.wikipedia.org/wiki/MapReduce

[W3IPP] Introduction to Parallel Programing and MapReduce – Google Code University

http://code.google.com/edu/parallel/mapreduce-tutorial.html

[W3DPA] Dynamic programing and sequence alignment

http://www.ibm.com/developerworks/java/library/j-seqalign/index.html

[W3HOW] Homología (Biología) - Wikipedia

http://es.wikipedia.org/wiki/Homolog%C3%ADa_%28biolog%C3%ADa%29

[W3OLB] Online lectures on bioinformatics

http://www.bioinfo.org.cn/lectures/index.htm

[W3SIN] Running Hadoop on Ubuntu Linux (Single-node cluster)

http://www.michael-noll.com/tutorials/running-hadoop-on-ubuntu-linux-single-node-cluster/

[W3MUL] Running Hadoop on Ubuntu Linux (Multi-node cluster)

http://www.michael-noll.com/tutorials/running-hadoop-on-ubuntu-linux-multi-node-cluster/

[W3YDN] Hadoop tutorial - YDN

http://developer.yahoo.com/hadoop/tutorial/index.html

[W3OMA] OMA Browser

http://www.omabrowser.org/

Firma el creador del proyecto "Análisis bioinformáticos sobre la plataforma Hadoop". Albert Gutiérrez Millà.