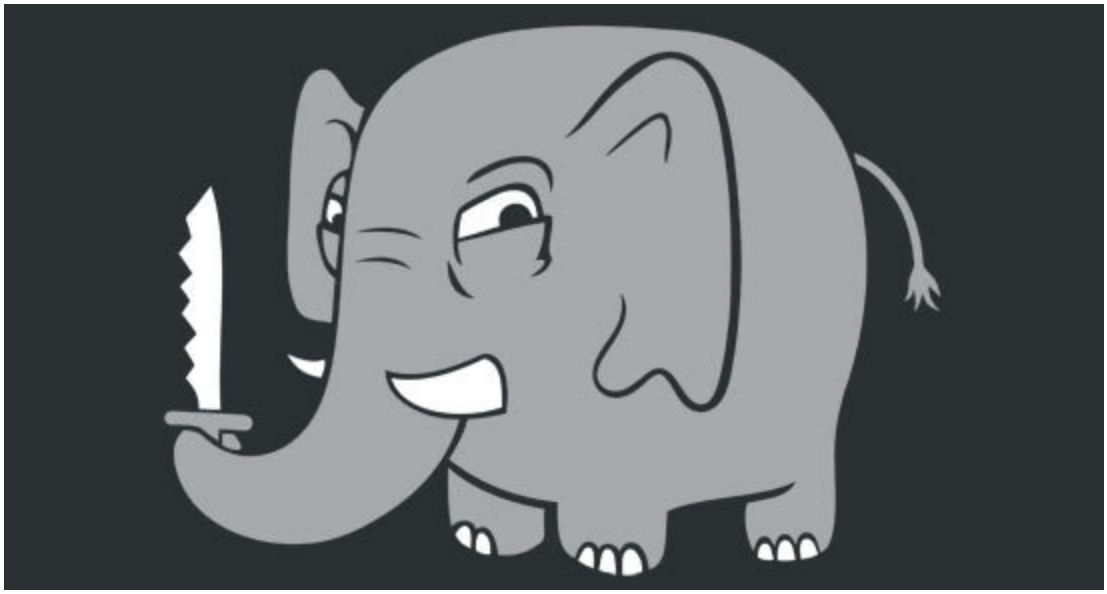
UTN.BA
UNIVERSIDAD TECNOLÓGICA NACIONAL
FACULTAD REGIONAL BUENOS AIRES

Ingeniería en Sistemas de Información

ReduceMapFast

Los grandes artistas copian, los genios roban - *Pablo Picasso*



Cátedra de Sistemas Operativos

Trabajo práctico Cuatrimestral

- 1C2015 -
Versión [1.0]

Recuperatorios:
Normas del Trabajo Práctico

Introducción

ReduceMapFast, en adelante RMF, es una plataforma de multiprocesamiento paralelo inspirada en los conceptos de Google MapReduce, emulando algunas características del funcionamiento del proyecto Hadoop.

El propósito del sistema **es permitir al usuario ejecutar operaciones de análisis sobre grandes volúmenes de datos** dividiendo el trabajo en sub-tareas y distribuyéndolas para ser ejecutadas de manera paralela en los diversos nodos del cluster de procesamiento.

Para una mayor comprensión utilizaremos el siguiente ejemplo:

Imaginemos que tenemos el requerimiento de contar la cantidad de palabras escritas en un libro impreso de 100 hojas.

Podríamos hacerlo nosotros solos con mucha paciencia, o bien podríamos darle 10 hojas a 10 personas y sumar sus resultados parciales; también podríamos dar una hoja por persona a 100 personas y luego sumar los 100 resultados parciales.

Cada uno de estos escenario demanda menor tiempo de procesamiento (contar palabras) pero requiere de mayor disponibilidad de recursos (personas) e incrementa el tiempo de planificación y organización (llevar el control de quién tiene cada hoja y luego recopilar y sumar los resultados parciales).

Claramente debe haber un equilibro entre el nivel de distribución y el tamaño de la tarea para así optimizar el rendimiento. Una incorrecta planificación (por ejemplo, darle de contar una palabra a cada persona) no sólo no aumenta la performance, sino que la puede disminuir drásticamente.

En este trabajo práctico el alumno desarrollará una plataforma que permitirá, dado un set de datos genérico (el libro en el ejemplo), aplicarle una rutina de mapping (separarlo por palabras) y luego aplicar una rutina de reducción (suma) sobre los resultados parciales distribuidos en diversos nodos de un mismo cluster.

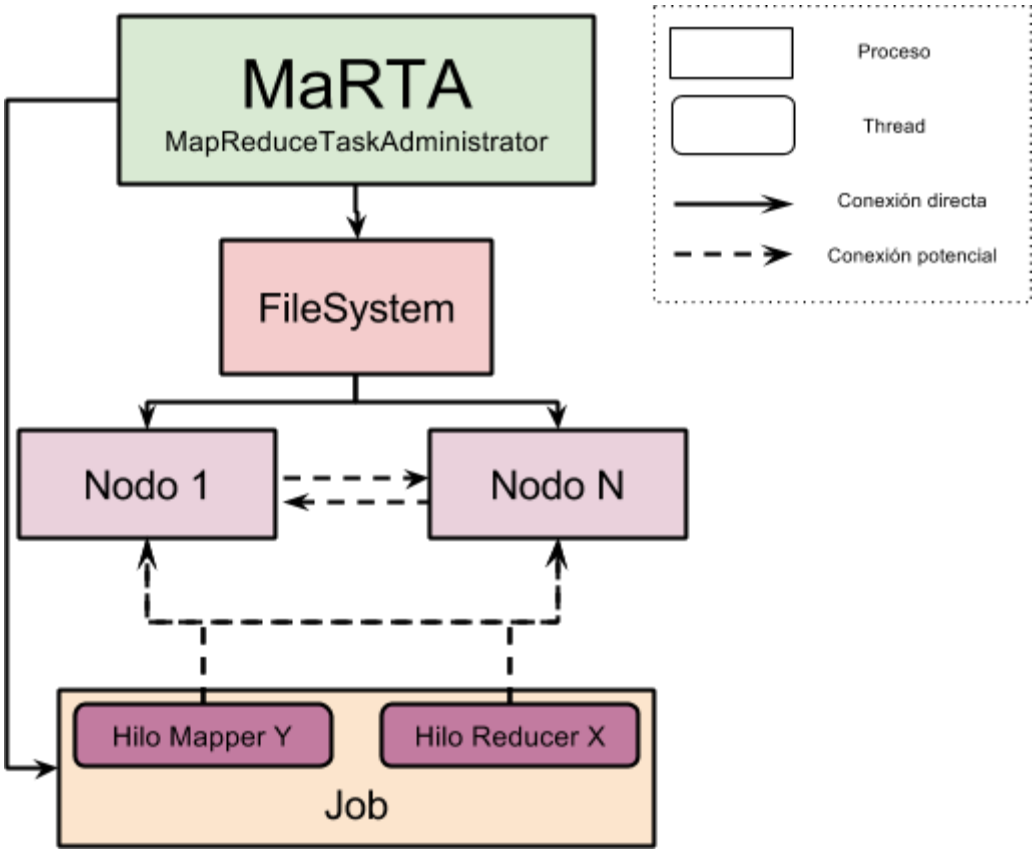
Ahora, extendamos el ejemplo anterior a **contar las palabras en un conjunto de libros**.

Entenderá que no demanda el mismo esfuerzo un libro escrito en Arial 10 que un libro con ilustraciones para niños. Dado que el resultado final (la cantidad de palabras) solo se puede obtener al concluir todas las tareas en las que dividió el trabajo, es necesario para maximizar la utilización de los recursos tener una distribución balanceada de tareas haciéndolas lo más equitativas posibles.

Comprenderá el valor de dividir los set de datos en bloques de igual tamaño y así lograr que todas las tareas demanden el mismo esfuerzo y poder así lograr una planificación determinística.

Arquitectura de ReduceMapFast

ReduceMapFast constará de los siguientes componentes:



Proceso Map-Reduce Tasks Administrator (MaRTA): Habrá una única instancia de este proceso en el sistema y será el encargado de administrar la carga de trabajo de todo el sistema. Le indicará a los diversos procesos Job cuándo iniciar hilos Mapper e hilos Reducer y qué deberá hacer cada uno. Tendrá contacto permanente con el Filesystem, quien le notificará información respecto a los archivos almacenados.

Proceso FileSystem: Este proceso también tendrá una única instancia en el sistema y será el encargado de gestionar y facilitar a los distintos procesos del sistema el acceso a los archivos almacenados. Tendrá contacto permanente con todas las instancias de los procesos Nodo y conocerá los bloques de datos que cada uno de ellos almacena.

Proceso Job: Las instancias de este proceso serán creadas por el usuario para ejecutar un trabajo sobre uno o varios archivos del Filesystem. Será el encargado de crear y conocer el estado de todos los hilos mapper y reducer que el proceso MaRTA le indique.

Proceso Nodo: Las instancias de este proceso se encargarán de almacenar los bloques de datos del FileSystem, y de realizar rutinas de Map y Reduce según lo requerido por los Jobs.

Proceso Job

El objetivo de este proceso es ejecutar un Job de ReduceMapFast sobre uno o varios archivos almacenados en el FileSystem. Para ello creará y administrará instancias de los hilos Mapper y los hilos Reducer, según sea indicado por el Proceso MaRTA.

Al iniciar, leerá su correspondiente archivo de configuración, se conectará al Proceso MaRTA, le indicará los archivos sobre los que desea ejecutar su Job y quedará a la espera de indicaciones de MaRTA, al que notificará los resultados de sus operaciones.

Es esperable que varias instancias del proceso Job se encuentren ejecutando de manera simultánea en el sistema y cada uno de ellos ejecutando instancias de los hilos mapper e hilos reducer.

Job de ReduceMapFast

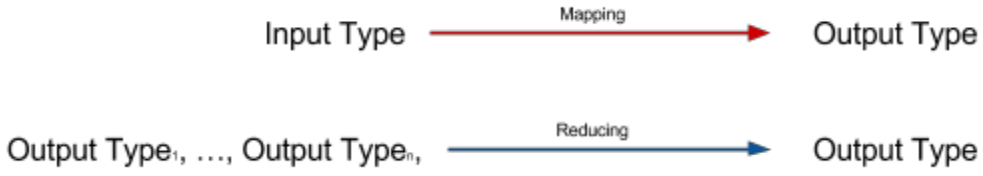
Una Job de ReduceMapFast se compone por dos rutinas, uno que realiza operaciones de Map y otro que realiza las operaciones de Reduce.

La rutina de *map* tomará los datos que sean ingresados por el Standard Input *(STDIN)* y los convertirá en un formato deseable *(por ejemplo, tomar ciertos campos de un archivo delimitado por comas)* y los expondrá por el Standard Output *(STDOUT)*. De forma análoga, la rutina de *reduce* tomará datos por STDIN, generados por el conjunto de rutinas de map o por otras rutinas de reduce, y aplicará las operaciones correspondientes para finalmente devolver un resultado parcial o final por STDOUT.

Es importante resaltar que estas operaciones *pueden y deben correr en paralelo* y de manera independiente entre sí.

Además, las rutinas de Reduce son *netamente recursivas*. Esto implica que los resultados que obtengamos de una rutina de reducing *pueden ser input nuevamente de otra rutina de reduce.*

Analizando el tipado de ambas rutinas obtenemos:



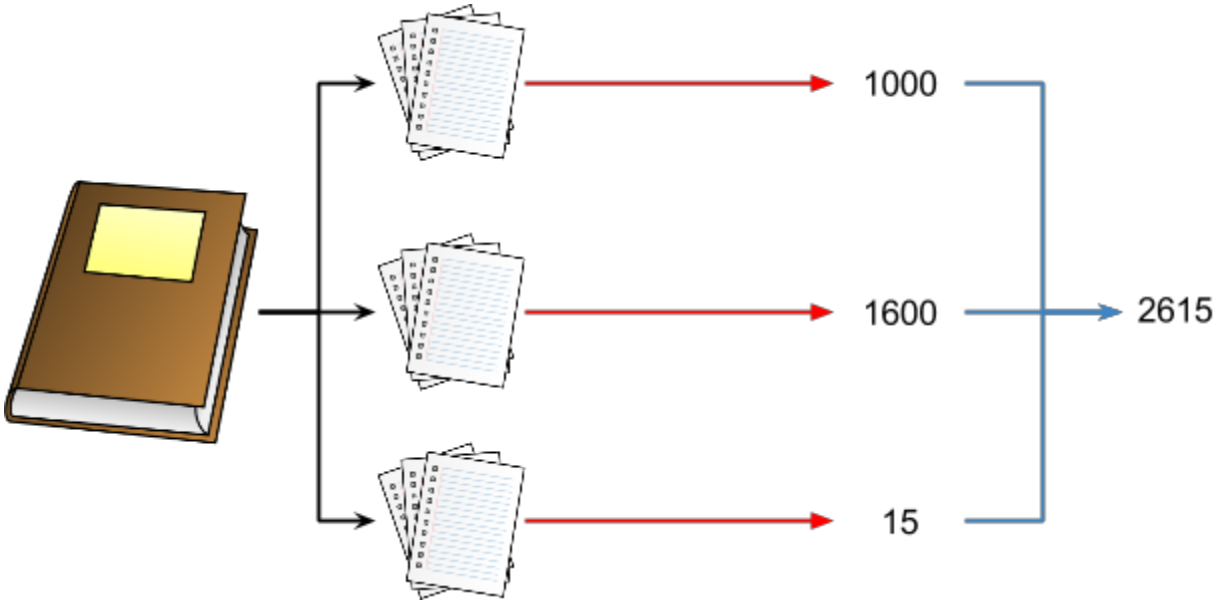
Es importante resaltar que la operación de reduce es idempotente[1], lo que implica que *al reducir un valor previamente reducido, obtendremos el mismo valor.* Por ejemplo, sabemos que si nuestro objetivo es contar la cantidad de palabras que tiene un libro, y por reducciones parciales llegamos a un valor (digamos, 100), si volvemos a utilizar la función de reducción sobre este valor, nuestro resultado será, otra vez, 100.

Como vemos en el gráfico a continuación, el Mapping transforma el input en un set comprensible y manejable por la rutina de Reduce.

Las rutinas de Reduce toman esos datos para generar el resultado final, o pueden también generar un resultado parcial que puede volver a ingresar a la rutina de Reduce cuantas veces sea necesario hasta obtener el resultado final.

Para el ejemplo de la introducción de contar la cantidad de palabras que tiene un texto, las rutinas de mapping contarán la cantidad de palabras de cada libro y las de reduce sumarán los valores de cada uno de esos outputs. De esta forma, transformamos el Input (un texto) en un dato manejable (palabras contadas) y luego los unificamos (sumando la cantidad de palabras).

[1] La **idempotencia** es la propiedad para realizar una acción determinada varias veces y aun así conseguir el mismo resultado que se obtendría si se realizase una sola vez.



Estas rutinas, además, serán *polimórficas,* por lo que podrán ser desarrolladas en cualquier lenguaje de desarrollo que respete la siguiente interfaz:

- La rutina recibirá datos por la entrada estándar (STDIN).
- La rutina devolverá datos por la salida estándar (STDOUT).
- La rutina finalizará con un valor de return code cero en caso de éxito y distinto de cero en caso de error.
- La rutina de mapping generará registros parciales, delimitados por un \n[2].
- La rutina de reduce recibirá los datos ordenados.

Tipo de Job

Uno de los parámetros de un job al iniciar debe ser si el job soporta o no combiner. Esto se lo deberá notificar a MaRTA dado que de este dato depende la planificación de las rutinas.

Hilo Mapper

MaRTA notificará al proceso Job cada vez que deba aplicar la rutina de mapping sobre un bloque almacenado en un Nodo. Para cada una de estas solicitudes el proceso Job creará una instancia de hilo Mapper.

[2] El valor \n indica 'New Line' o 'Nueva línea'. Equivale al ingreso de un [Enter] en el teclado.

Cada hilo Mapper tiene como objetivo conectarse al proceso Nodo correspondiente e indicarle que aplique la rutina de Mapping sobre un bloque de datos, y que almacene el resultado de manera ordenada (sort) en el FileSystem Temporal del Nodo.

Luego quedará a la espera de la respuesta por parte del proceso Nodo y notificará al proceso Job el éxito o fracaso de la operación.

El proceso Job notificará al MaRTA el resultado de esa operación. En caso de error, el MaRTA puede re-planificar la operación en otro Nodo que contenga una copia de ese bloque.

Hilo Reducer

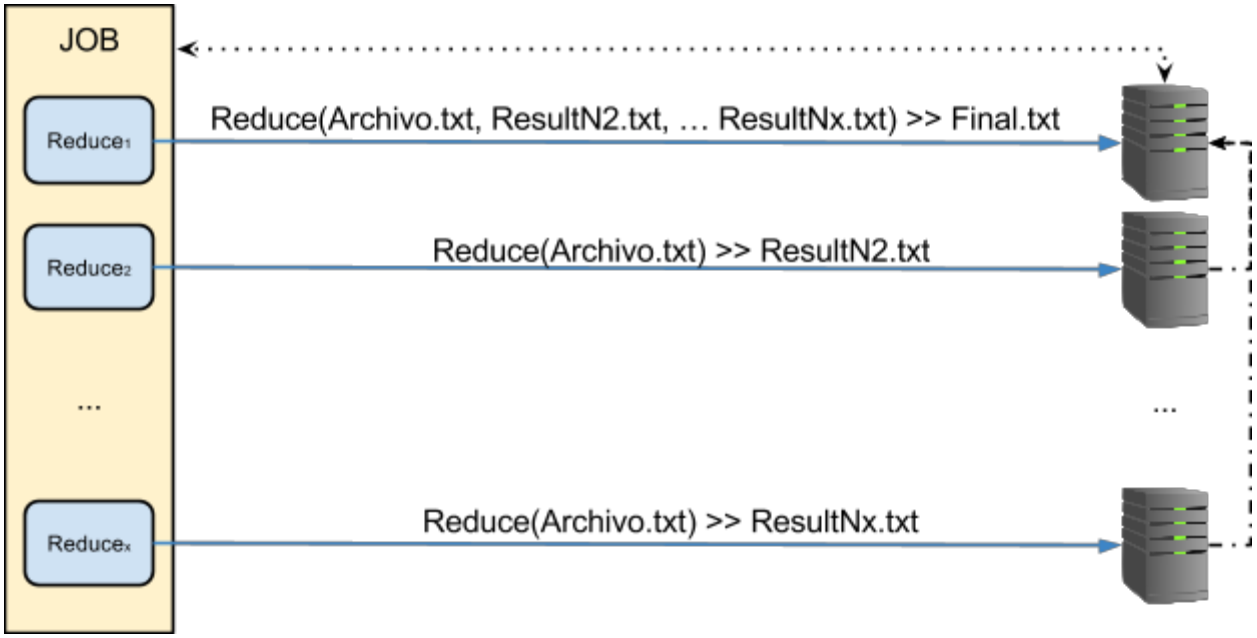
Al ir concluyendo las operaciones de mapping, el MaRTA comenzará a solicitar operaciones de reducción de los resultados obtenidos. Ante dicha solicitud el proceso Job creará una nueva instancia de Hilo Reduce.

Este hilo le indicará al proceso Nodo correspondiente que aplique la *rutina de reduce* entre varios archivos existentes en el espacio de almacenamiento Temporal, de los cuales **uno siempre deberá ser local al Nodo**, mientras que los demás podrían existir en otros Nodos.

Al ser lanzado, este hilo se conectará al Nodo correspondiente, le enviará los nombres y los Nodos de los archivos a reducir, el código de la *rutina de reduce*, y el nombre del archivo donde almacenará el resultado.

Dado que una rutina de reduce puede recibir input de varios archivos (incluso almacenados en otros Nodos por red), quedará a criterio del grupo el diseño de un algoritmo que permita aparear múltiples archivos de manera simultánea, eficiente y que respete una distribución de carga del sistema.

Por ejemplo, se podría decidir un algoritmo que sitúe un Nodo arbitrario, encargado del apareo de los distintos resultados parciales de todos los otros Nodos. Supongamos que contamos con una cantidad *N* de Nodos, podríamos distribuir la carga de tal forma que *todos los nodos obtengan los resultados parciales locales*, y **solo uno** sea el encargado de *unificar dichas reducciones:*



Ante la finalización de cada rutina de Reduce, se debe informar al proceso Job y este le notificará al proceso MaRTA.

El fracaso de una operación de reduce puede involucrar que se deban ejecutar nuevamente operaciones de Mapping. Estas situaciones están descritas dentro de la sección del Proceso MaRTA.

Parámetros de configuración

Campo	Valor de ejemplo	Descripción
IP_MARTA	x.y.z.w	Dirección IP de la computadora que está ejecutando el proceso MaRTA
PUERTO_MARTA	5000	Puerto donde el proceso MaRTA está escuchando nuevas conexiones.
MAPPER	map.sh	Programa rutina de mapping
REDUCE	reduce.sh	Programa rutina de reduce
COMBINER	SI/NO	El Job soporta que se lo planifique en modo combiner

ARCHIVOS	/user/juan/datos/temperatura-2012.txt/user/juan/datos/temperatura-2013.txt	Lista de archivos sobre los que se va a aplicar el job
RESULTADO	/user/juan/datos/resultado.txt	Nombre del archivo donde se va a almacenar el resultado

Archivo de Log

El proceso deberá mostrar al menos los siguientes datos, sea en pantalla, en un archivo de log o en ambos.

Descripción	En pantalla	En log
Conexión y desconexión al proceso MaRTA con IP y puerto	X	X
Creación de hilos, motivo y parámetros recibidos	X	X
Finalización de hilos, motivo y resultado	X	X
Cabeceras de mensajes enviados y recibidos de cada hilo/proceso		X

Proceso FileSystem

Este proceso será el encargado de organizar los bloques de datos distribuidos en los procesos Nodos, asociarlos a los correspondientes nombres de archivos para así exponer el FileSystem MDFS.

Al iniciar, leerá su archivo de configuración y quedará a la espera de las conexiones de los procesos Nodo. Una vez que se hayan conectado los Nodos requeridos para un estado estable permitirá que se conecte el Proceso MaRTA, a quien responderá solicitudes sobre los archivos.

MDFS FileSystem

MDFS es un FileSystem diseñado para soportar grandes volúmenes de datos ofreciendo un elevado nivel de performance y redundancia en equipamiento de baja gama.

Su diseño está orientado a flujos de datos WORM (write once, read many), en otras palabras para situaciones donde los datos son leídos frecuentemente pero ocasionalmente (o nunca) son modificados.

Tamaño de bloques

Dado que las operaciones de mapping se aplican sobre un bloque por vez es necesario que el tamaño de estos sea el apropiado. Un bloque muy grande reduciría la cantidad de mappers que se pueden ejecutar de manera simultánea y un bloque chico haría que las operaciones fueran muchas e insignificantes. Es por eso que se elige un tamaño de bloque de 20MB.

Dado que los archivos a cargar no necesariamente terminan alineados a los 20 MB para evitar que queden registros partidos los procesos de escritura y de lectura de datos de bloques deben asegurarse que no haya datos pasados el último '\n'. Por ende, el sistema debe asegurarse de que no quede otro '\n' en posiciones siguientes al último registro válido.

Redundancia

Al igual que un sistema de archivos convencional MDFS asocia un nombre de archivo con los bloques que lo componen. Particularmente MDFS triplica cada bloque de datos de un archivo en diferentes Nodos logrando así evitar que ante la eventual caída de un Nodo los archivos se pierdan o corrompan.

El proceso FileSystem debe en todo momento conocer la cantidad de bloques libres y utilizados de cada Nodo y por consecuente el espacio libre total.[3]

Al recibir una operación de escritura deberá balancear la asignación de los bloques entre todos los Nodos del sistema junto con evitar que el mismo bloque de datos de un archivo más de una vez en el mismo nodo.

Concurrencia

MDFS deberá soportar múltiples conexiones y operaciones de lectura y escritura en archivos diferentes de manera concurrente. Un archivo estará disponible sólo cuando haya sido escrito completamente. Esto evita que se ejecuten operaciones sobre archivos que se están escribiendo.

[3] El alumno debe investigar opciones para almacenar esta información de manera eficiente

Conexiones y desconexiones de Nodos

MDFS debe soportar el ingreso y egreso de nuevos nodos al sistema.

Al iniciar, el Proceso FileSystem esperará a que la cantidad mínima de nodos definidos en su archivo de configuración estén disponibles. Luego cambiará su estado a **Operativo** y empezará a atender solicitudes.

Luego podrán ingresar y salir del sistema Nodos para lo cual el Proceso Filesystem deberá tomar las siguientes acciones:

- Al recibir un Nodo nuevo, deberá asumirlo como vacío, registrarlo como espacio disponible del filesystem y priorizarlo al momento de tener que almacenar nuevos bloques intentando que la cantidad de bloques usados sea equitativa con el resto.
- Al sufrir la desconexión de un Nodo, deberá revisar la lista de archivos y marcar como desactivado aquellos archivos que tengan bloques que no puedan ser servidos. Ver esquema de ejemplo.
- Al recibir un Nodo que se había desconectado, deberá validar si su presencia habilita algún archivo desactivado y marcarlo como habilitado.

Esquema de ejemplo

temperatura-2013.txt
Tamaño: 1.338.943.001 bytes
Directorio Padre: 5
Estado: **Disponible**
Bloques:

Bloque	Copia 1	Copia 2	Copia 3
0	NodoA - Bloque 33	NodoC - Bloque 18	NodoD - Bloque 99
1	NodoG - Bloque 34	NodoQ - Bloque 65	NodoB - Bloque 13
2	NodoA - Bloque 90	NodoD - Bloque 75	NodoC - Bloque 42
...	...	...	...

temperatura-2012.txt
Tamaño: 1.472.461.155 bytes

Directorio Padre: 5
Estado: **Disponible**
Bloques:

Bloque	Copia 1	Copia 2	Copia 3
0	NodoC - Bloque 43	NodoG - Bloque 18	NodoB - Bloque 91
1	**NodoF** - Bloque 56	**NodoD** - Bloque 66	**NodoA** - Bloque 87
2	NodoD - Bloque 91	NodoQ - Bloque 75	NodoF - Bloque 65
...	..	..	..

Como puede observarse de este diagrama para leer el bloque 1 del archivo temperatura-2012.txt basta con que al menos uno de los nodos NodoF, NodoD o NodoA esté funcionando.
Supongamos que los tres nodos son desconectados, entonces el bloque 1 no podrá ser servido. Esta situación es condición suficiente para marcar el estado como No Disponible hasta que alguno de los tres vuelva a estar disponible.

Directorios

MDFS soporta directorios hasta un total de 1024 directorios. Todos los archivos y directorios deben tener un directorio padre, el cual puede ser otro directorio o el directorio raíz (padre=0).

Ya que nuestra estructura de directorios necesita poder ser recuperada ante una eventual caída, esta debe ser persistida bajo algún medio.

Index	**Directorio**	**Padre**
1	user	0
2	jose	1
3	juan	1
4	temporal	0
5	datos	2
6	fotos	2

Como pueden ver este esquema permite el siguiente diagrama

raiz (/)
- /user
  - /user/juan
    - /user/juan/datos
      - temperatura-2012.txt
    - temperatura-2013.txt
    - /user/juan/fotos
  - /user/jose
- /temporal

Nota: Debido a que la naturaleza de la información almacenada responde al modelo una clave (nombre de archivo) con muchos valores (lista de bloques), el alumno podrá optar por utilizar sistema de almacenamiento pre-existente como puede ser MongoDB o BerkeleyDB.

Consola

Este proceso tendrá una consola desde la cual se podrán ejecutar comandos de administración de archivos.

- Formatear el MDFS
- Eliminar/Renombrar/Mover archivos
- Crear/Eliminar/Renombrar/Mover directorios
- Copiar un archivo local al MDFS
- Copiar un archivo del MDFS al filesystem local
- Solicitar el MD5 de un archivo en MDFS
- Ver/Borrar/Copiar los bloques que componen un archivo.
- Agregar un nodo de datos
- Eliminar un nodo de datos

Parámetros de configuración

Campo	Valor de ejemplo	Descripción
PUERTO_LISTEN	3000	Puerto en donde el FileSystem va a escuchar solicitudes de nuevas conexiones

LISTA_NODOS	[NodoA,NodoB, NodoC,Nodo4]	Lista de nodos mínima que deben estar conectados para que el FileSystem pase a estado Operativo

Archivo de Log

El proceso deberá mostrar al menos los siguientes datos, sea en pantalla, en un archivo de log o en ambos.

Descripción	En pantalla	En log
Conexión y desconexión de procesos	X	X
Solicitudes de lectura y escritura de archivos	X	X
Lista de bloques de un archivo solicitado		X
Estado de los nodos	X	X
Espacio total del sistema y sus cambios	X	X

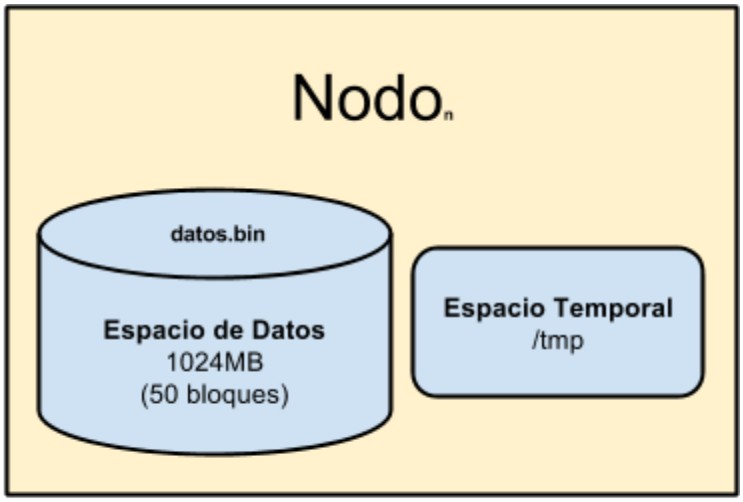
Proceso Nodo

El proceso Nodo es uno de los más complejos del trabajo práctico ya que contará con múltiples interfaces y responsabilidades.

Contará con dos espacios de almacenamiento, el **Espacio de Datos**, un archivo de tamaño fijo destinado a almacenar de manera contigua el contenido de los bloques de los archivos del FileSystem MDFS y el **Espacio Temporal**, un directorio local donde se almacenarán archivos temporales de los resultados parciales de las rutinas[4].

Al iniciar recibirá una conexión por parte del proceso FileSystem y quedará a la espera de conexiones y solicitudes de los siguientes procesos/hilos:

[4] Se recomienda la utilización de una carpeta temporal en /tmp del FileSystem de Linux. Tener en cuenta que dos Nodos pueden correr en una misma PC y chequear eventualidades.



Interfaz con otros Nodos y el FileSystem

El FileSystem u otro proceso Nodo podrán solicitar las siguientes operaciones:

- *getBloque(numero)*: Devolverá el contenido del bloque solicitado almacenado en el Espacio de Datos.
- *setBloque(numero, [datos])*: Grabará los datos enviados en el bloque solicitado del Espacio de Datos
- *getFileContent(nombre)*: Devolverá el contenido del archivo de Espacio Temporal solicitado.

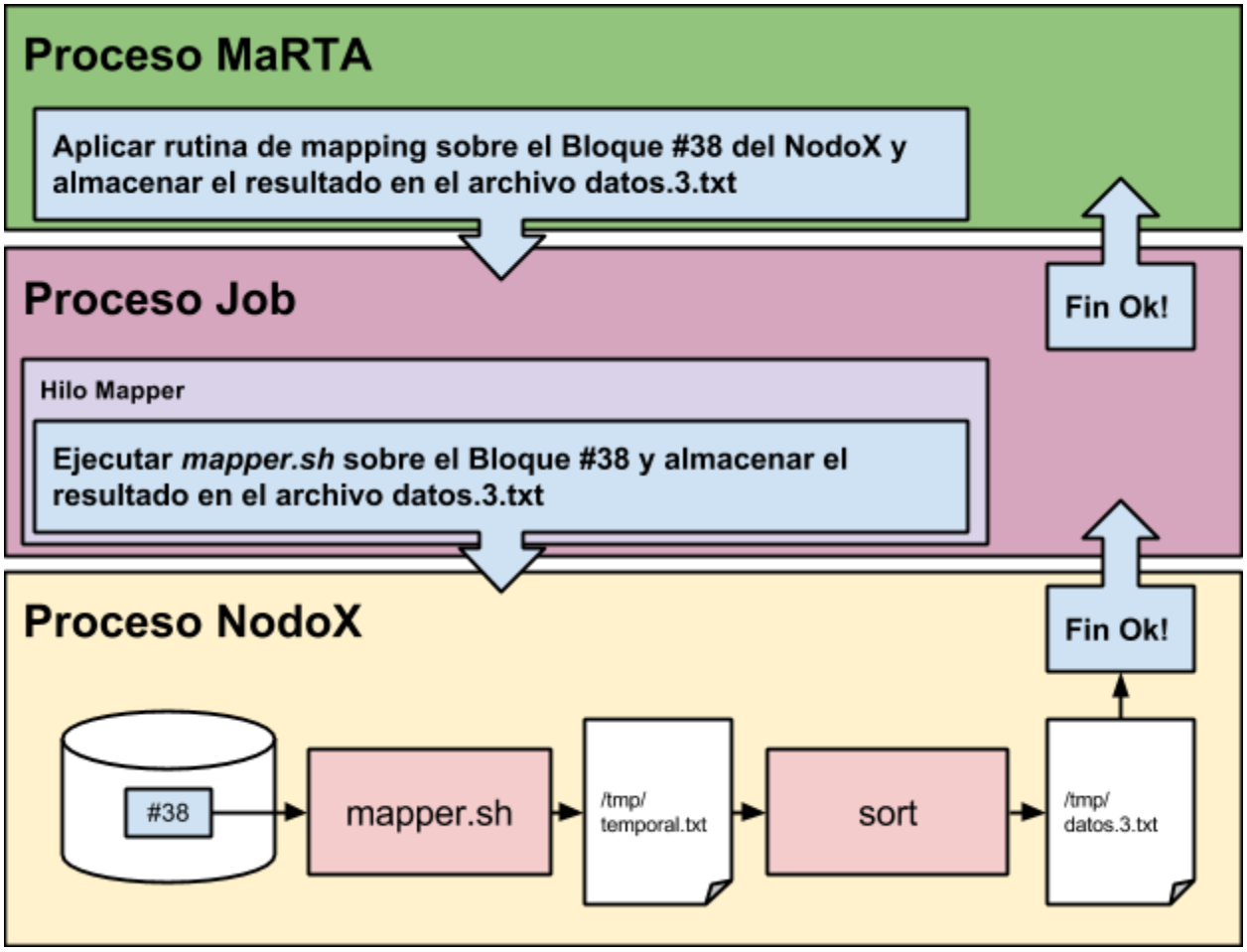
Interfaz con los hilos Mapper

Hilo Mapper de un Job podrá enviar una solicitud de *mapping* que consta de:

- Un programa ejecutable
- Un número de bloque del espacio de Bloque de Datos
- Un nombre de archivo del espacio Temporal

Deberá ejecutar el programa enviando el contenido del bloque como entrada estándar (STDIN) y almacenando la salida del programa (STDOUT) en un archivo temporal en el Espacio Temporal.
Al concluir el programa deberá ordenar el resultado por clave (sort[5]) y almacenar el resultado en el archivo del Espacio Temporal recibido como parámetro.

[5] No es necesario que el alumno desarrolle la rutina de ordenamiento. Puede invocarse el programa ***sort*** de Bash

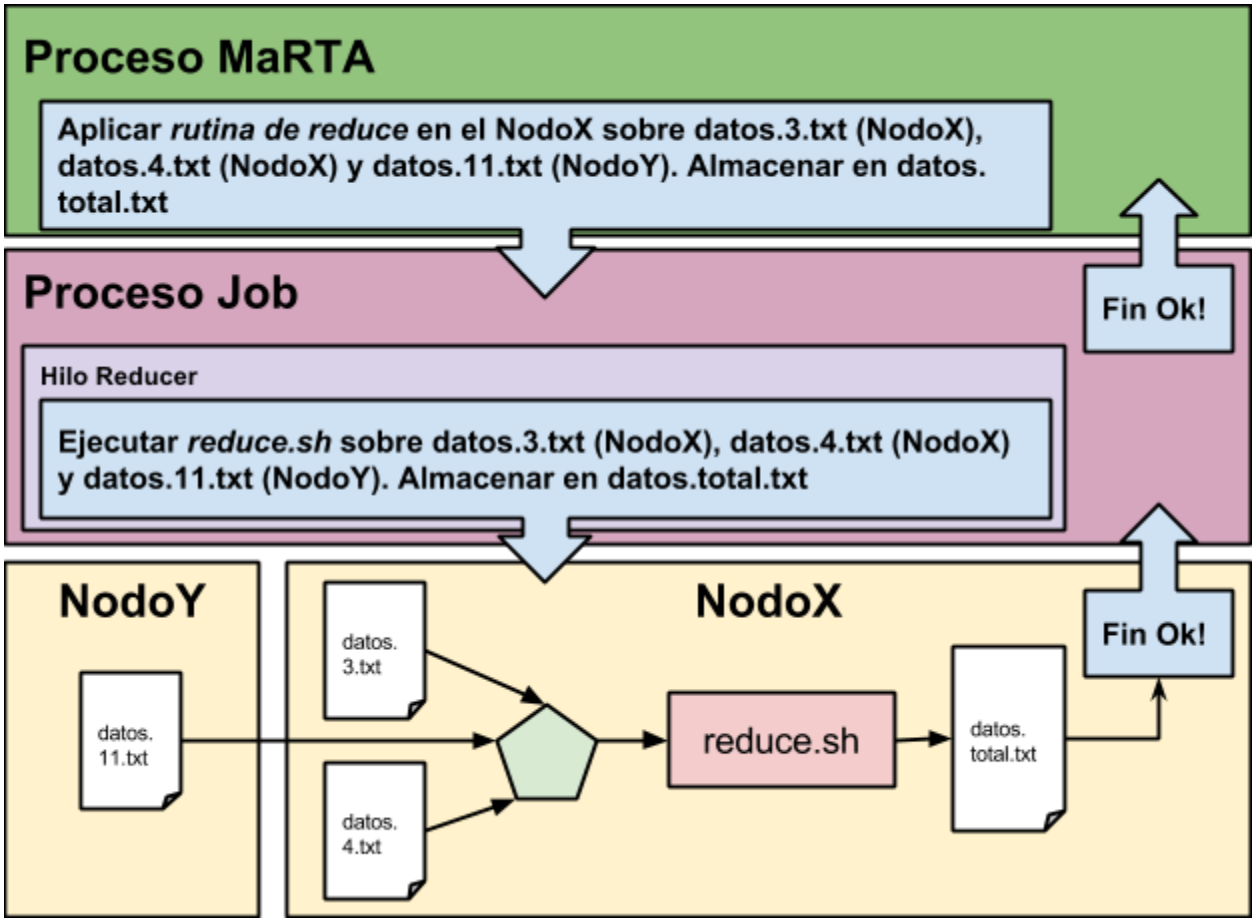


Interfaz con los hilos Reducer

Un Hilo Reducer podrá enviar una solicitud de reduce que consta de:

- Un programa ejecutable
- Una lista de archivos compuesta por Nodo y Nombre de Archivo, de los cuales al menos uno deberá estar en el Nodo local
- Un nombre de archivo del espacio temporal

Deberá aparear el contenido de todos los archivos de la lista de manera simultánea, los cuales por definición están ordenados por clave y cada registro termina en '\n' (enter), enviando el contenido al programa por entrada estándar (STDIN) y almacenando la salida del mismo (STDOUT) en el archivo del espacio Temporal recibido como parámetro.



Estado del Nodo

Un nodo que se conecta al sistema puede ser un nodo sin datos (nuevo) o un Nodo que ya perteneció al FileSystem y contiene bloques de datos que corresponden a archivos de MDFS. Esta situación debe ser parametrizable por archivo de configuración o por argumento del programa para poder permitir que un nodo regrese al sistema.

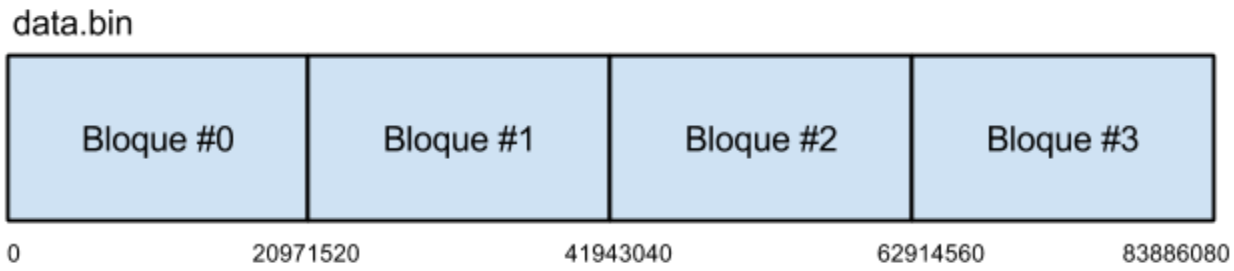
Espacio de Datos

El Espacio de Datos del proceso Nodo será un archivo almacenado localmente de tamaño pre-solicitado, cuyo nombre estará definido en el archivo de configuración. El contenido del mismo será exclusivamente los bloques de datos y no podrá contener estructuras administrativas.

Es decir para que un Nodo disponga de un espacio de datos de 1GB, antes de iniciar el proceso Nodo por primera vez, se deberá generar un archivo local de 1GB y ese nombre

deberá ser escrito en el archivo de configuración. Luego los primeros 20 MB de este archivo corresponden al bloque 0 del nodo, los segundos 20 MB al bloque 1 y sucesivamente.

En general para acceder al bloque X, se acceder a este archivo en la posición X * 20MB



Para simplificar el acceso a los datos de forma aleatoria el alumno debe investigar e implementar la llamada al sistema mmap()

Parámetros de configuración

Campo	Valor de ejemplo	Descripción
PUERTO_FS	3000	Puerto donde el FileSystem está escuchando solicitudes de nuevas conexiones
IP_FS	x.y.z.w	IP de la computadora donde está ejecutándose el FileSystem
ARCHIVO_BIN	data.bin	Nombre del archivo que contiene los bloques de datos
DIR_TEMP	/tmp	Directorio donde se van a almacenar los archivos temporales
NODO_NUEVO	SI/NO	Un nodo nuevo es aquel que nunca formó parte del cluster. Este valor debe estar en SI la primera vez que un nodo ingresa al sistema y luego cambiar a NO.
IP_NODO	a.b.c.d.	IP de la computadora donde está ejecutándose este proceso nodo. (puede obtenerse del sistema)
PUERTO_NODO	6000	Puerto en el cual este proceso Nodo espera recibir conexiones nuevas.

Archivo de Log

El proceso deberá mostrar al menos los siguientes datos, sea en pantalla, en un archivo de log o en ambos.

Descripción	En pantalla	En log
Conexión y desconexión de procesos	X	X
Solicitudes de mapping o reduce y su estado	X	X
Solicitudes de lectura/escritura de bloques y archivos temporales		X

Proceso MapReduceTasksAdministrator (MaRTA)

El proceso MapReduceTasksAdministrator (MaRTA) es el encargado de ordenar y priorizar el trabajo del sistema intentando maximizar la utilización del cluster.

Al recibir un Job nuevo, solicitará al FileSystem los bloques que componen dicho/s archivo/s a procesar. Buscará la combinación que maximice la distribución de las tareas en los nodos e irá indicando al Job cada Hilo Mappers que deberá iniciar y qué Nodo-Bloque debe procesar hasta que la rutina de Mapping haya sido aplicada en todo el set de datos.

Luego planificará los Hilos Reducers de la siguiente manera:

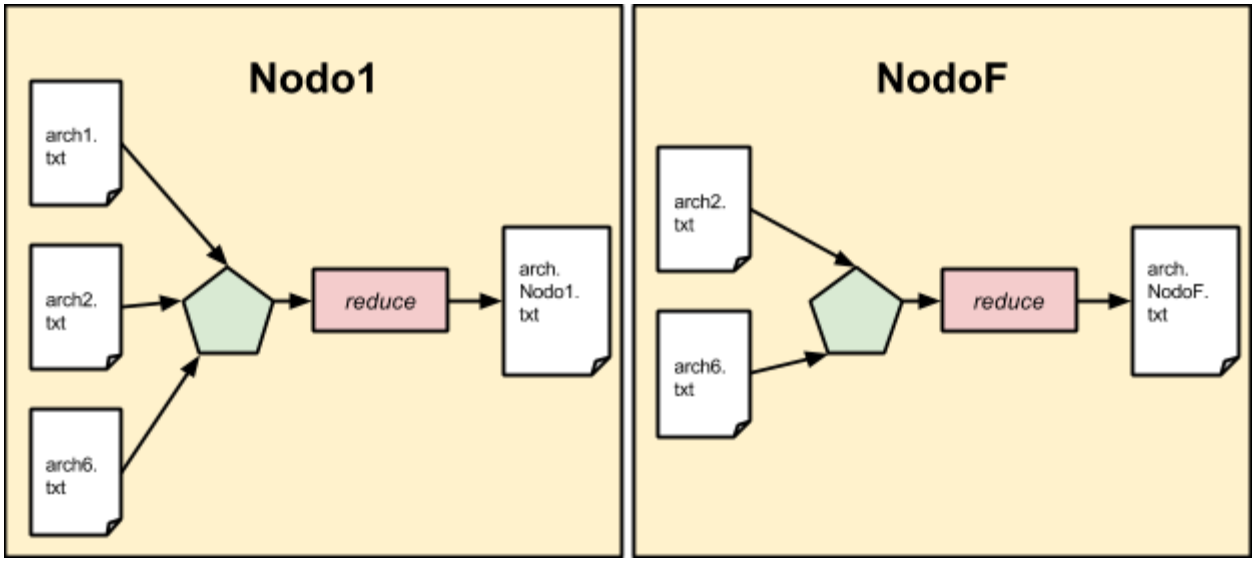
Planificación de rutinas de Reduce

En función de si el Job reportó soportar combiner o no la planificación de las rutinas de reduce cambia su comportamiento.

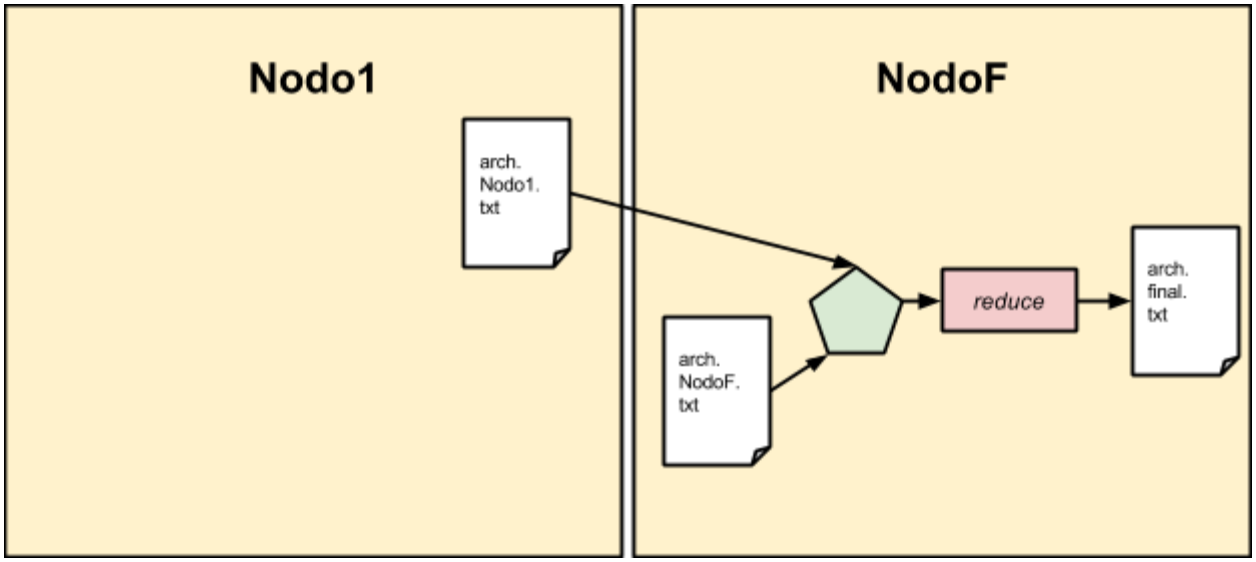
Con soporte de Combiner

Si el job soporta *combiner*, aplicará la rutina de reduce primero entre todos los archivos temporales del mismo nodo y luego los unirá.

Paso 1



Paso 2



Sin soporte de Combiner

Si el job no soporta *combiner*, aplicará la rutina de reduce entre todos los archivos (locales y remotos) manera simultánea.

En el ejemplo del combiner en el paso 1 reduciría Nodo1-arch1.txt, Nodo1-arch2.txt, Nodo1-arch6.txt, NodoF-arch6.txt, NodoF-arch2.txt en arch.final.txt. Las operaciones las debería ejecutar en Nodo1 dado que es el que más archivos locales tiene.

Fin de la operación

Al concluir la rutina de Reduce le solicitará al FileSystem que copie el archivo de resultado al MDFS y le notificará al Job que la operación fue concluida.

Planificación

MaRTA conoce en todo momento la carga total de trabajo en el sistema y en cada uno de los nodos dado que es quien asigna tareas. Su objetivo es maximizar la utilización del sistema logrando una distribución lo más equitativa posible de las tareas.

En todo momento debe informar por pantalla:

- El estado de cada uno de los nodos
- Las tareas que está ejecutando cada nodo
- La cantidad de tareas pendientes de mapping y reduce de cada Job

Re-planificación

Nodos podrían abandonar el sistema durante o luego de que rutinas de mapping o reduce hayan sido ejecutadas.
MaRTA deberá re-planificar y ejecutar dichas operaciones en otros nodos capaces de procesarlas.
Nodos existentes podrían ingresar al sistema permitiendo que MaRTA ejecute una mejor distribución de las tareas.

Descripción de las entregas

Para permitir una mejor distribución de las tareas y orientar al alumno en el proceso de desarrollo de su trabajo, se definieron una serie de puntos de control y fechas que el alumno podrá utilizar para comparar su grado de avance respecto del esperado.

Checkpoint 1

Fecha: 9 de Mayo

Objetivos:

- ★ Familiarizarse con Linux y su consola, el entorno de desarrollo y el repositorio
- ★ Aplicar las Commons Libraries, principalmente las funciones para listas, archivos de conf y logs
- ★ Comprender y aplicar mmap()
- ★ Desarrollar una función que permita redireccionar STDIN y STDOUT a un script
- ★ Modelar las estructuras necesarias para gestionar el FileSystem
- ★ Desarrollar un modelo de consola del FileSystem
- ★ Familiarizarse con el desarrollo de procesos servidor multihilo

Lectura recomendada:

http://faq.utn.so/arrancar
Beej Guide to Network Programming - link
Linux POSIX Threads - link
SisopUTNFRBA Commons Libraries - link

Checkpoint 2

Fecha: 23 de Mayo

Objetivos:

- ★ Realizar un cliente simple que permita enviar mensajes estructurados a los servidores
- ★ Utilizar semáforos para sincronizar hilos que acceden a una lista compartida
- ★ Modelar las estructuras de MaRTA
- ★ Modelar la interacción del proceso Nodo y el FileSystem así como el proceso Job y MaRTA
- ★ Crear programas que implementen estas interacciones

Lectura recomendada:

Sistemas Operativos, Silberschatz, Galvin - Capítulo 4: Hilos
Sistemas Operativos, Silberschatz, Galvin - Capítulo 5: Planificación del Procesador

Checkpoint 3 - Presencial

Fecha: 13 de Junio

Objetivos:

- ★ Tener un Proceso FileSystem funcional
- ★ Modelar la interacción entre el Job y el Nodo y MaRTA y el FileSystem

Checkpoint 4

Fecha: 4 de Julio

Objetivos:

- ★ Concluir los procesos y sus interacciones
- ★ Desarrollar la planificación completa de un Job en el sistema
- ★ Validar los tests provistos
- ★ Realizar pruebas de stress de varios jobs funcionando simultáneamente

Entrega Final

Fecha: 11 de Julio

Objetivos:

- ★ Probar el trabajo práctico en el laboratorio
- ★ Empaquetar el desarrollo para un fácil deployment
- ★ Subir a GIT la última versión estable

Recuperatorios:

- Primer recuperatorio: 18 de Julio
- Segundo recuperatorio: 1ero de Agosto

Normas del Trabajo Práctico

El trabajo práctico de este cuatrimestre se rige por las Normas del Trabajo Práctico (NTP), detalladas en el siguiente link: http://faq.utn.so/ntp