Table of Contents

Y.A.M.A.

Yet Another MR Administrator

Trabajo practico de Sistemas Operativos de la U.T.N. F.R.B.A. Segundo cuatrimestre 2017.

Introducción y Arquitectura

El objetivo de este TP será poder crear una plataforma que permite **paralelizar la carga de trabajo** en varias máquinas físicas, logrando así reducir los tiempos de procesamiento de datos.

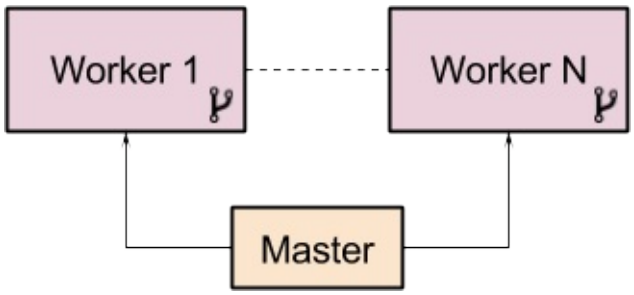
YAMA es una plataforma de **procesamiento distribuido de datos que maximiza el rendimiento ejecutando tareas de manera concurrente en diversos nodos**. Dichos nodos contienen los bloques que componen los archivos a procesar organizados bajo una estructura de filesystem distribuido.

Patrón Master/Worker

YAMA utiliza una arquitectura Master/Worker, que nos permitirá distribuir la carga del sistema en varios procesadores (Workers) utilizando un coordinador (Master).

Existirá una instancia del proceso Master en el sistema para cada tarea (Job) de procesamiento que se requiera. Dicho proceso Master se conectará con los Workers quienes serán los encargados de *forkearse (multiplicarse)* para realizar tareas específicas.

Durante el desarrollo notará que algunos componentes de esta arquitectura resultan una caja negra a la hora de desarrollar la solución, por lo que resulta importante respetar la especificación de manera correcta.

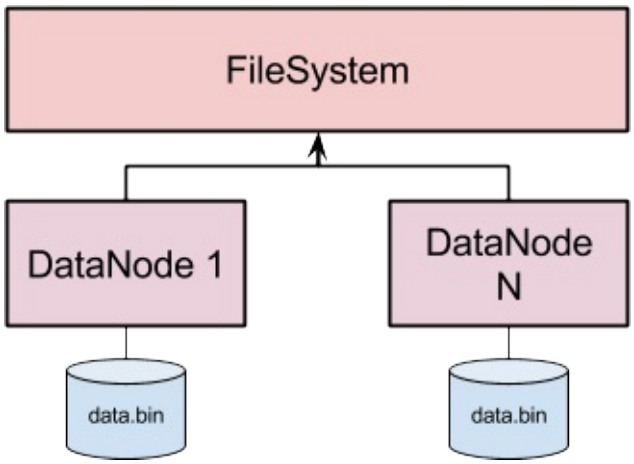


File System Distribuido

El manejo de los datos que consumirá cada Worker también será responsabilidad del sistema. Para ello utilizaremos una arquitectura de File System Distribuido, cuyo objetivo principal es poder distribuir y replicar la información en varios Data Nodes.

Un factor interesante de este tipo de arquitecturas es la posibilidad de garantizar la Alta Disponibilidad. Esto significa que, si uno de los Data Nodes dejase de estar disponible, sería posible acceder a los datos de todos los archivos del File System, ya que los bloques

que este contenía se encuentran replicados en otros Data Nodes del sistema.

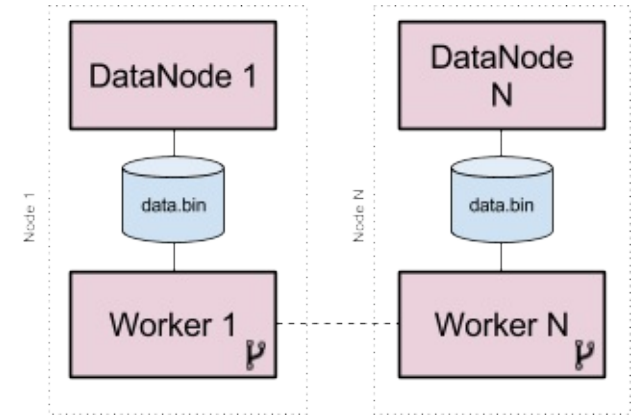


El proceso **FileSystem, que contará con una única instancia en el sistema** será el coordinador principal de las operaciones de lectoescritura y el encargado de gestionar el acceso a los archivos almacenados. Tendrá contacto permanente con todas las instancias de los procesos DataNode y conocerá el estado de todos sus bloques de datos.

Por su parte, los procesos **DataNode** serán los encargados de persistir los bloques de datos del FileSystem en su archivo de datos (data.bin). Una vez almacenados los datos de un archivo en el archivo de datos, **estos serán considerados inmutables**, es decir, que no podrán ser modificados.

Nodo

Para aprovechar al máximo los recursos, cada proceso Worker correrá en una computadora física distinta. Considerando que **enviar un paquete vía red es sustancialmente más lento que consumirlo de memoria principal o de disco** se procurará que el procesamiento de los datos se realice en la computadora física (Nodo) donde se encuentran los mismos.

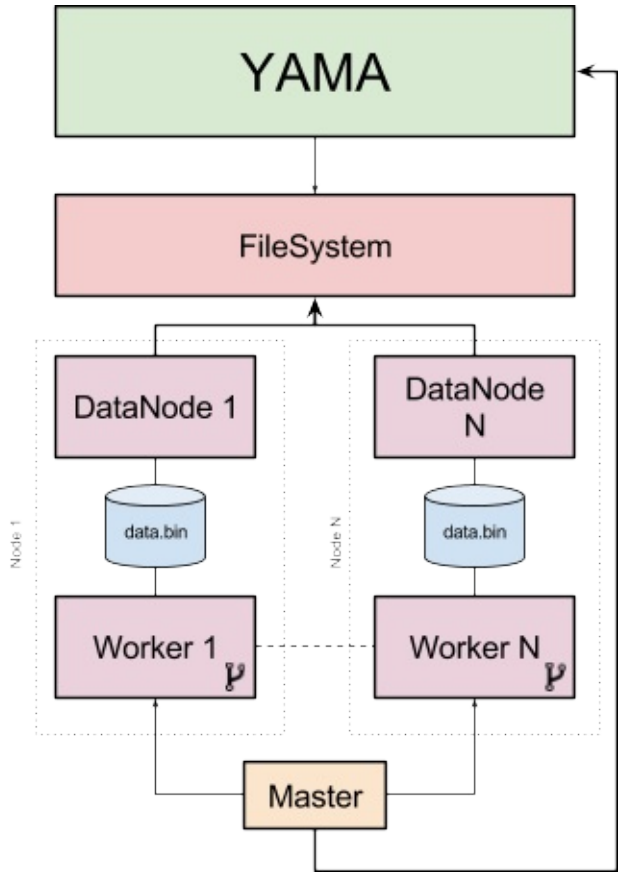


Definiremos entonces al **Nodo** como **la estructura abstracta que agrupa un DataNode, con su archivo de datos y un Worker**. Cada Nodo representará una computadora física distinta, desde la que se realizarán tanto operaciones de almacenamiento de datos (DataNode) y procesamiento (Worker).

Es importante notar que **los Worker consumen datos directamente del archivo de datos del DataNode**. Dicho procedimiento se realiza con la intención de agilizar el acceso a los datos en cada Nodo. Este trabajo práctico intencionalmente rompe el encapsulamiento del FileSystem con el propósito de maximizar la performance.

Arquitectura del Sistema

Para poder balancear la carga del sistema, garantizando la sinergia entre el almacenamiento de datos y las tareas de procesamientos, utilizaremos un proceso denominado **YAMA**. Habrá una única instancia de este proceso en el sistema y será el encargado de conectarse con el FileSystem para analizar las solicitudes de los procesos Master, pudiendo así planificarlas. Teniendo en cuenta estos datos, la arquitectura final del TP será la siguiente:



Deployment y Testing del Trabajo Práctico

Al tratarse de una plataforma distribuida, los procesos involucrados podrán ser ejecutados en diversas computadoras. La cantidad de computadoras involucradas y la distribución de los diversos procesos en estas será definida en cada uno de los tests de la evaluación y **es posible cambiar la misma en el momento de la evaluación**. Es responsabilidad del grupo automatizar el despliegue de los diversos procesos con sus correspondientes archivos de configuración para cada uno de los diversos tests a evaluar.

La disposición del trabajo práctico consistirá en **cargar una serie de archivos en el proceso Filesystem y ejecutar una serie de tareas (Jobs) sobre dichos archivos mediante los procesos Masters**. Dependiendo del resultado de dicha ejecución se evaluará el correcto funcionamiento de la plataforma.

Ejemplo de ejecución de un Job

1. Se le brindará al grupo un archivo de datos, un programa de Transformación de datos y un programa de Reducción de datos.
2. El grupo iniciará el Filesystem, los correspondientes Nodos (DataNode y Worker) y por último el proceso YAMA.
3. El grupo deberá cargar el archivo de datos en el Filesystem para luego iniciar un proceso Master que ejecute los programas de Transformación y Reducción sobre el archivo de datos.
4. Se revisará el archivo resultante de la ejecución del Job para validar el correcto funcionamiento.

Todo esto estará descrito en el documento de pruebas que se publicará cercano a la fecha de Entrega Final. Archivos y programas de ejemplo se pueden encontrar en el repositorio de la cátedra.

Existe también la posibilidad de brindarle al grupo los archivos de datos ya distribuidos en el FileSystem, de forma tal que tan solo iniciando el mismo, se pueda recuperar el estado del sistema.

Finalmente, recordar la existencia de las Normas del Trabajo Práctico donde se especifican todos los lineamientos de cómo se desarrollará la materia durante el cuatrimestre.

Proceso Master

Un proceso Master permitirá al usuario **ejecutar una tarea (Job)** sobre un sólo archivo de datos almacenado en el Filesystem. Dicha tarea nos permitirá procesar los datos de un archivo del FileSystem mediante la aplicación de dos operaciones: Transformador y Reductor.

Estas operaciones resultan de caja negra para el alumno y serán provistas por la cátedra como archivos externos (binario o script)

El resultado de la aplicación de estos programas será guardado en formato de archivo en el FileSystem. Para ello, se deberá indicar la ruta de almacenamiento del archivo resultante.

Ejemplo: `./Master /scripts/transformador.sh /scripts/reductor.rb yamafs:/datos.csv yamafs:/analisis/resultado.json`

Se observa que el comando inicia un proceso Master para ejecutar el script transformador.sh y el script reductor.rb sobre el archivo datos.csv en yamafs. El resultado deberá ser almacenado en /analisis/resultado.json también en yamafs.

Cabe destacar **que utilizaremos el prefijo "yamafs:" para referirnos a los archivos de nuestro FileSystem**, evitando de esta forma las ambigüedades posibles entre los archivos locales y los que se encuentran cargados en nuestro sistema. Por ejemplo: `yamafs:/datos.csv` .

A través de la plataforma YAMA, el programa Transformador analizará los datos de los **bloques** (ver FileSystem) de un archivo y generará como resultado un archivo temporal. Luego el programa Reductor, tomará esos archivos temporales y los **unificará** en un único archivo final.

La característica principal de estos programas es que múltiples instancias de estos se ejecutarán de forma paralela en todo el sistema. Para ello, **el proceso Master se encargará de distribuir a los distintos Workers el programa el Transformador y el programa Reductor e indicarles a cada uno qué operación deben hacer sobre qué bloque u archivo**.

Etapas del proceso Master

Al iniciar, leerá su archivo de configuración, se conectará al Proceso YAMA, le indicará el archivo sobre el que desea operar y quedará a la espera de indicaciones de YAMA. Para resolver dicha solicitud YAMA ejecutará tres etapas una a continuación de la otra:

- Etapa de Transformación
- Etapa de Reducción Local
- Etapa de Reducción Global

Debido a que la responsabilidad de la ejecución de las etapas está dividida entre el proceso YAMA y el proceso Master es conveniente, para una mejor comprensión del enunciado, leer la responsabilidad de cada proceso en cada etapa de manera conjunta.

Etapa de Transformación

En esta etapa el proceso Master se conectará a los Workers en los nodos correspondientes y le indicará cuáles son los bloques sobre los que se deberá aplicar el programa de Transformación y dónde guardar sus resultados:

En esta etapa el proceso YAMA le indicará:

- A qué procesos Worker deberá conectarse con su IP y Puerto
- Sobre qué bloque de cada Worker debe aplicar el programa de Transformación.
- El nombre de archivo temporal donde deberá almacenar el resultado del script de Transformación.

El proceso Master deberá entonces:

1. Iniciar un hilo por cada etapa de Transformación indicada por el proceso YAMA.
2. Cada hilo se conectará al correspondiente Worker, le enviará el programa de Transformación y le indicará el bloque sobre el cuál quiere ejecutar el programa, la cantidad de bytes ocupados en dicho bloque y el nombre del archivo temporal donde guardará el resultado. Quedará a la espera de la confirmación por parte del proceso Worker.
3. Notificara del éxito o fracaso de la operación al proceso YAMA.

Ejemplo de una respuesta a una solicitud de Transformación de YAMA

Nodo	IP y Puerto del Worker	Bloque	Bytes Ocupados	Archivo Temporal
Nodo 1	192.168.1.10:5000	38	10180	`/tmp/Master1-temp38`
Nodo 1	192.168.1.10:5000	39	1048576	`/tmp/Master1-temp39`
Nodo 2	192.168.1.11:5555	44	1048576	`/tmp/Master1-temp44`
Nodo 2	192.168.1.11:5555	39	1048576	`/tmp/Master1-temp39`
Nodo 2	192.168.1.11:5555	46	1048576	`/tmp/Master1-temp46`

Note que en el ejemplo se ejecutarán dos transformaciones en el Worker del Nodo 1 y tres en el Worker del Nodo 2 por ende habrá 5 archivos temporales en 2 computadoras

Cuando en un Nodo se ejecutaron todas las operaciones de Transformación, inmediatamente pasará a la Etapa de Reducción Local.

Etapa de Reducción Local

En esta etapa el proceso Master deberá indicarle a los Workers correspondientes **cuáles son los archivos temporales sobre los cuales deberá aplicarse el programa de Reducción**, y dónde guardar sus resultados.

En esta etapa el proceso YAMA le indicará:

- Nombre de los archivos temporales almacenados en cada uno de los Nodos que deberá procesar con el programa de Reducción
- Nombre de archivo temporal de reducción local por cada Nodo.

Ejemplo de una respuesta a una solicitud de reducción local de YAMA

Nodo	IP y Puerto del Worker	Archivo Temporales de Transformación	Archivo Temporal de Reducción Local
Nodo 1	192.168.1.10:5000	`/tmp/Master1-temp38`	`/tmp/Master1-Worker1`
Nodo 1	192.168.1.10:5000	`/tmp/Master1-temp39`	`/tmp/Master1-Worker1`
Nodo 2	192.168.1.11:5555	`/tmp/Master1-temp44`	`/tmp/Master1-Worker2`
Nodo 2	192.168.1.11:5555	`/tmp/Master1-temp39`	`/tmp/Master1-Worker2`
Nodo 2	192.168.1.11:5555	`/tmp/Master1-temp46`	`/tmp/Master1-Worker2`

Note que en el ejemplo se ejecutarán dos reducciones locales. Una en el Worker del Nodo 1 y otra en el Worker del Nodo 2 con los archivos temporales correspondientes al paso anterior.

El proceso Master deberá:

1. Iniciar un hilo por cada Nodo y conectarse al Worker.
2. Enviar el programa de Reducción, la lista de archivos temporales del Nodo y el nombre del archivo temporal resultante.
3. Quedará a la espera de la confirmación por parte del proceso Worker.
4. Notificara del éxito o fracaso de cada etapa al proceso YAMA.

De esta forma, el proceso Master **esperará a que todos los Workers terminen su etapa de Reducción Local**. Cuando esto suceda, **comenzará con la Etapa de Reducción Global**.

Etapa de Reducción Global

En esta etapa el proceso Master deberá indicarle al Worker encargado por YAMA que debe realizar la Reducción Global sobre todos los archivos temporales de la etapa de Reducción Local, y dónde guardar almacenar el resultado final.

Resulta importante destacar que, si bien un solo Worker será el encargado de realizar la Reducción Global, este deberá conectarse con sus homónimos para realizar dicha operación. Se detalla este procedimiento en el apartado correspondiente al Worker.

El proceso YAMA le indicará:

- Lista de Nodos con la IP y Puerto del Worker.
- El nombre de archivo temporal de Reducción de cada Worker.

- Un Worker designado como encargado.
- La ruta donde guardar el archivo resultante de la Reducción Global.

Ejemplo de una respuesta a una solicitud de Reducción Global de YAMA

Nodo	IP y Puerto del Worker	Archivo Temporal de Reducción Local	Archivo de Reducción Global	Encargado
Nodo 1	192.168.1.10:5000	`/tmp/Master1-Worker1`		
Nodo 2	192.168.1.11:5555	`/tmp/Master1-Worker2`	`/tmp/Master1-final`	SI

Note que en el ejemplo vemos los archivos temporales de Reducción Local del ejemplo del paso anterior. De manera arbitraria YAMA designó al Worker2 como encargado.

El proceso Master deberá:

1. Conectarse al Worker encargado.
2. Enviar la rutina de Reducción, la lista de procesos Worker con sus IPs y Puertos y los nombres de los archivos temporales de Reducción Local.
3. Quedará a la espera de la confirmación por parte del proceso Worker.
4. Notificara del éxito o fracaso de la operación al proceso YAMA.

Al finalizar la ejecución de esta Etapa, el proceso Master le comunicará al proceso YAMA y este le indicará que realice el **Almacenado Final**.

Almacenado final

Para finalizar la tarea se almacenará el archivo resultante del Job en el Filesystem distribuído para ser accedido por el usuario o por otro Job con el nombre y la ruta especificadas por el usuario al ejecutar el proceso Master.

El proceso YAMA le indicará:

- La IP y Puerto del Worker al cual conectarse
- El nombre del archivo resultado de la Reducción Global.

Ejemplo de una respuesta a una solicitud de almacenado final de YAMA

Nodo	IP y Puerto del Worker	Archivo de Reducción Global
Nodo 2	192.168.1.11:5555	`/tmp/Master1-final`

El proceso Master entonces le solicitará al Worker encargado que se conecte al Filesystem y le envié el contenido del archivo de Reducción Global y el nombre y ruta que este deberá tener.

Concluida esta operación se da por terminado el Job y el proceso Master terminará. Si la operación de escritura del archivo en el FileSystem fallara, el Worker deberá informar el error al proceso Master, quien a su vez deberá notificarlo a YAMA.

Replanificación

Puede suceder que un proceso Worker finalice su ejecución de forma abrupta o con errores. Si este fuera el caso, deberá notificarse a YAMA y este será encargado de re-planificar sólamente las tareas de transformación que hayan fallado en un nodo que contenga una copia de los datos.

Cabe aclarar que YAMA puede definir que **no existe posibilidad de replanificar las etapas que fallaron, y dar por finalizado el Job**. Como por ejemplo, en casos donde no se posean nodos con copias de los datos, o se encuentre en una etapa de Reducción o Almacenamiento Final.

Sobre el proceso Master

Es esperable que varias instancias del proceso Master se encuentren ejecutando de manera simultánea en el sistema. Tanto si el proceso Master llegase a tener un error de ejecución o la ejecución concluyera de manera satisfactoria **no se requiere que se eliminen los archivos temporales producidos por las diversas etapas**.

Métricas

El proceso Master mostrará luego de concluir su ejecuciòn, las siguientes métricas:

1. Tiempo total de Ejecución del Job.
2. Tiempo promedio de ejecución de cada etapa principal del Job (Transformaciòn, Reducción y Reducción Local). Can3. tidad máxima de tareas de Transformación y Reducción Local ejecutadas de forma paralela.
3. Cantidad total de tareas realizadas en cada etapa principal del Job.

4. Cantidad de fallos obtenidos en la realización de un Job.

Archivo de Configuración y Logs

El proceso Master deberá poseer un archivo de configuración ubicado en una ubicación conocida donde se deberán especificar, al menos, los siguientes parámetros:

```
YAMA_IP=
YAMA_PUERTO=
```

Queda a decisión del grupo el agregado de más parámetros al mismo. Además, el proceso Master deberá registrar toda su actividad mediante un archivo de log, mostrando los mismos por pantalla y finalizando mostrando sus métricas.

Proceso YAMA

Este proceso es el encargado de resolver la solicitud de un Master administrando y balanceando la carga de trabajo[1] de los diversos procesadores de datos (Worker) conociendo y controlando el estado de todos los Masters en el sistema y de cada una de las operaciones que estos realizaron y deben realizar.

El proceso YAMA **es agnóstico de qué operación realiza el proceso Master sobre los datos** sin embargo **es quién conoce dónde y cuándo ejecutar las operaciones** para obtener el resultado deseado.

Para resolver la solicitud de un Master deberá planificar las siguientes operaciones e indicarle al proceso Master la tarea a realizar:

- **Etapa de Transformación**: Debe planificar la ejecución del script de Transformación sobre todos los bloques del archivo guardando los resultados en un archivo temporal de nombre único en la computadora.
- **Etapa de Reducción Local**: Debe planificar la ejecución del script de Reducción sobre los archivos temporales del paso anterior que estén en el mismo Nodo.
- **Etapa de Reducción Global**: Debe planificar la ejecución del script de Reducción sobre los archivos temporales resultantes del paso anterior entre Nodos.
- **Etapa de Almacenamiento Final**: Deberá indicarle al Master el archivo temporal resultante a almacenar en el FileSystem.

Tabla de Estados

YAMA mantendrá una estructura interna denominada **tabla de estados** con las solicitudes que le envió al Master y su estado:

Job	Master	Nodo	Bloque	Etapa	Archivo Temporal	Estado
1	1	Nodo 1	8	Transformación	`/tmp/j1n1b8`	En proceso
1	1	Nodo 3	2	Transformación	`/tmp/j1n3b2`	En proceso
1	1	Nodo 2	9	Transformación	`/tmp/j1n2b9`	En proceso
...	...	...	...	...	...	...
25	3	Nodo 5	2	Reducción Local	`/tmp/j1n2b2`	Error
W	X	Nodo Y	Z	Reducción Global	`/tmp/jxnybz`	Finalizado OK

Los nombres temporales de los archivos, así como los identificadores de cada Job, deberán ser generados automáticamente por el proceso YAMA evitando que los mismos se repitan.

Al recibir de parte del Master notificaciones de que las operaciones concluyeron actualizará los correspondientes estados. Así mismo, YAMA obtendrá el proceso FileSystem toda la información referente a los Nodos sobre los cuales se encuentran los bloques a utilizar, y la utilizará para poder generar la tabla.

Es necesario aclarar, que las entradas de Jobs no se deberán eliminar una vez finalizada su ejecución (sea correcta o no), a fines de evaluar los resultados del Sistema.

Etapa de Transformación

Ante la solicitud de un Master de procesar un archivo, le solicitará al Filesystem información del mismo y su composición.

Recibirá del Filesystem la lista de bloques que lo componen, con la correspondiente ubicación de sus dos copias y espacio ocupado en el bloque e iniciará la etapa de Transformación.

Como vimos anteriormente, en esta etapa debe enviarle al Master la indicación de que ejecute **el programa de Transformación en todos los bloques que componen el archivo**.

Al contar con dos copias por cada bloque de datos **optará en función de la cantidad de operaciones que esté ejecutando ese Worker** actualmente considerando todos los Masters en ejecución, intentando minimizar la carga del Nodo.

Balanceo de Carga

El proceso YAMA deberá efectuar la distribución de las tareas en los Nodos. Dicha distribución responderá al algoritmo de balanceo de carga configurado, siendo sus posibles valores **Clock** o **Weighted Clock**(W-Clock).

A diferencia del algoritmo Clock simple, el algoritmo W-Clock va a efectuar una valoración sobre cada Nodo en función a la cantidad de tareas que se encuentre realizando y haya realizado; eligiendo el que menos tareas se encuentre realizando y, en caso de empate, el que menos cantidad de tareas haya realizado. Es responsabilidad del grupo conocer los posibles límites que pueda conllevar la particular implementación del mismo realizada por el grupo. Una ampliación sobre el tema es tratada en el Anexo II: Algoritmos de Planificación.

Ejemplo Archivo `misdatos.csv`

Bloque	Copia 0	Copia 1	Bytes ocupados
0	Nodo 1 - Bloque 8	Nodo 2 - Bloque 11	10180
1	Nodo 2 - Bloque 12	Nodo 3 - Bloque 2	10201
2	Nodo 1 - Bloque 7	Nodo 2 - Bloque 9	10109

Como se ve en este ejemplo, podríamos solicitar tres operaciones de Transformación al Nodo 2 dejando **ociosos** a los Nodos 1 y 3 o bien solicitar una operación al Nodo 1, otra al Nodo 2 y otra al Nodo 3 y así **maximizar el paralelismo**. Un ejemplo completo de una respuesta de esta etapa se ve en la sección Ejemplo de una solicitud de Transformación.

Cuando las tareas de Transformación vayan concluyendo podrá continuar con la etapa de Reducción Local.

Fallo de una tarea de Transformación

Podría pasar que una etapa falle, dado que por ejemplo se pierda la conexión con un Nodo. En ese caso YAMA deberá re-planificar dicha operación en la otra copia.

Etapa de Reducción Local

Al recibir los mensajes del proceso Master notificando que una tarea de Transformación fue concluida deberá revisar en la tabla de estados ha concluído todas las Transformaciones para un mismo nodo, en ese caso puede iniciar las tareas de Reducción Local.

Es importante registrar esta información en la tabla de estados agregando los campos o columnas que considere pertinente.

El propósito de la Reducción Local es minimizar la transferencia de datos por red entre nodos por lo que la Reducción Local debe ejecutarse aún si hay un solo archivo temporal en ese nodo.

Nuevamente un nombre de archivo temporal para el nodo deberá generarse, intentando que el mismo no se repita. Puede ver un ejemplo de la solicitud de Reducción Local en este ejemplo. Si se generase una falla en la tarea, se deberá registrar la misma y abortar el Job.

Etapa de Reducción Global

Al recibir la última notificación de una tarea de Reducción Local deberá ejecutar la etapa de Reducción Global involucrando los archivos pre-agregados de todos los nodos. Para la misma, se seleccionará un Nodo que actuará como Encargado de realizar la misma. Dicho Nodo será seleccionado por ser el que menos carga de trabajo poseía al momento de empezar la etapa (Least Loaded Node).

Es importante registrar esta información en la tabla de estados agregando los campos o columnas que considere pertinente. La Reducción es el último paso, el cual una vez finalizado debe dejar un archivo temporal con el resultado final. Si se generase una falla en la tarea, se deberá registrar la misma y abortar el Job. Puede ver un ejemplo de la solicitud de Reducción en este ejemplo.

Archivos de Configuración y Logs

El proceso YAMA deberá poseer un archivo de configuración ubicado en una ubicación conocida donde se deberán especificar, al menos, los siguientes parámetros:

```
FS_IP=
FS_PUERTO=
RETARDO_PLANIFICACION=
ALGORITMO_BALANCEO=
DISP_BASE=
```

Queda a decisión del grupo el agregado de más parámetros al mismo. Es menester aclarar que el retardo de la planificación es un elemento meramente para fines de pruebas, dado que en situaciones normales se lo seteará con el valor de 0ms. Para su implementación, se recomienda que el grupo investigue sobre la función `usleep()` .

Además, YAMA deberá registrar toda su actividad mediante un archivo de log, mostrando por pantalla solamente las actualizaciones de la tabla de estados.

Recarga de la Configuración

YAMA será capaz de volver a cargar la configuración provista ante un posible cambio del retardo de planificación, Disponibilidad Base y el algoritmo de balanceo. Eso se realizará enviando a YAMA la señal `SIGUSR1`.

Es menester aclarar que el cambio del algoritmo de balaceo será realizado cuando sea necesaria realizar una nueva planificación o una re-planificación por parte de YAMA. Lo mismo aplica para el retardo.

Será responsabilidad del grupo investigar sobre la forma de enviar señales a procesos mediante la consola bash o el programa `htop`.

> [1]. El objetivo ideal es lograr que todos los Workers tengan la misma carga de trabajo ↩

Proceso FileSystem

Este proceso será el encargado de organizar los bloques de datos distribuidos en los procesos DataNode de cada Nodo, asociarlos a los correspondientes nombres de archivos para así exponer el sistema de archivos a los usuarios.

Al iniciar, leerá su archivo de configuración, si lo tuviese, y quedará a la espera de las conexiones de los procesos DataNode. Una vez que se hayan conectado los DataNode requeridos, se deberá formatear el FileSystem, pasando el mismo a un **estado estable**[2], en el cual permitirá que se conecten Workers o YAMA, a quien responderá solicitudes sobre los archivos. Mientras que el FileSystem no adquiera un estado estable, el mismo no deberá permitir la conexión de YAMA u otro proceso que no sean DataNode. Además, una vez formateado el FileSystem, no se permitirán nuevos Nodos en el sistema.

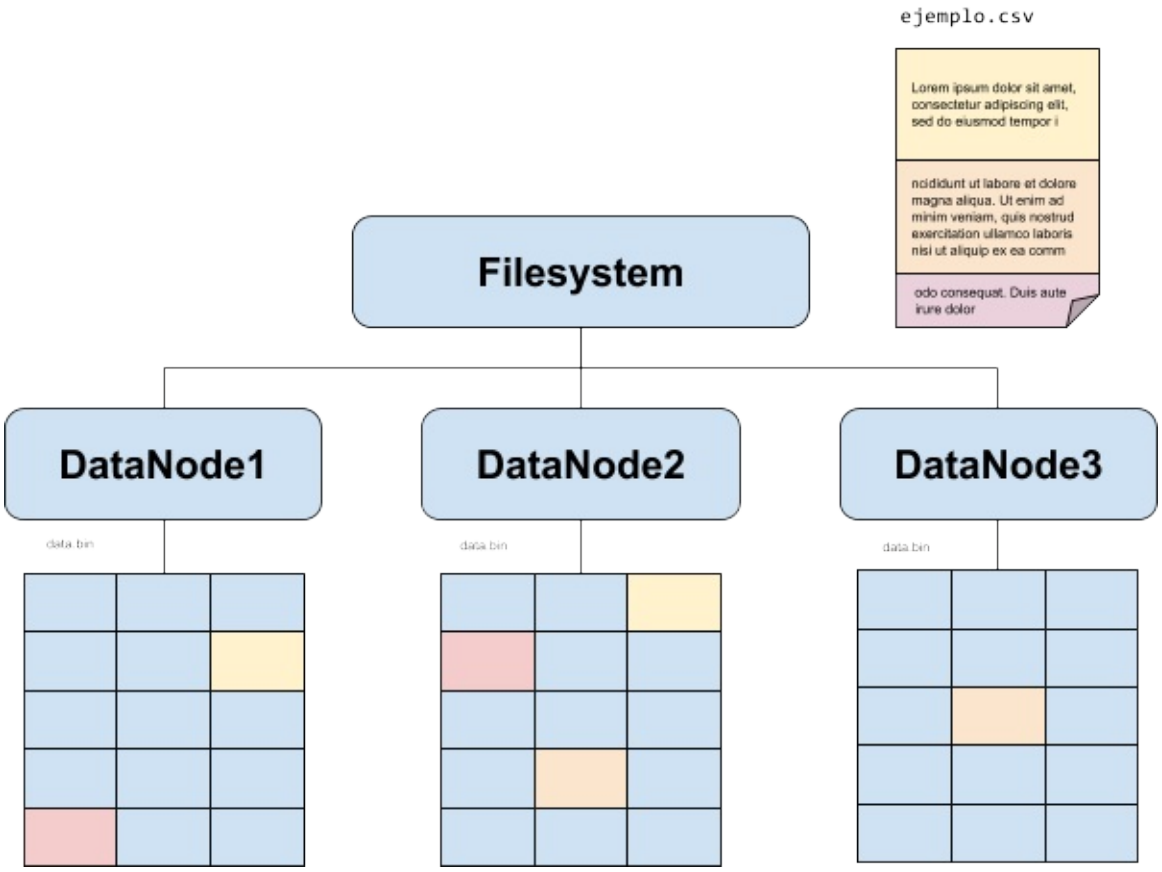
Estructura del FileSystem

El diseño de las estructuras 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.

Adicionalmente tiene la particularidad de que almacena una copia de cada bloque de datos en otro nodo para así lograr la redundancia en caso de que un Nodo salga de línea abruptamente.

La asignación de cada copia de un bloque en un nodo se realizará de forma **contigua**. Es decir, se asignará el espacio dentro del Nodo al primer espacio libre, siguiendo los lineamientos del algoritmo First Fit.

A continuación se mostrará un ejemplo de asignación de los espacios para los bloques de un archivo de ejemplo. Como aclaración, se deberá asumir que la imagen es solamente de carácter ilustrativo.



Como se observa en el diagrama, el archivo ejemplo.csv, compuesto por 3 bloques, está distribuido en los Nodos 1, 2 y 3 teniendo dos copias de cada bloque. Puede ver las estructuras administrativas para este archivo `ejemplo.csv` *en: Esquema de filesystem de ejemplo.*

Bloques

El tamaño de bloque es fijo de **1MB** y con la particularidad de que este filesystem permite diferenciar entre archivos de texto o binarios.

En el caso de los archivos de texto, dado que los mismos pueden no necesariamente terminan alineados a 1 MB, para evitar que queden registros partidos la operación de escritura debe asegurarse antes de escribir cada nuevo renglón (`\n`)[3] que el espacio en el bloque sea suficiente. En caso de que no lo sea, anotará la posición del último byte con contenido válido en la estructura administrativa del archivo y continuará en el próximo bloque. Cabe aclarar, que incluso si un bloque no estuviera completamente ocupado, solamente podrá ser ocupado por un único archivo a la vez.

En el caso de los archivos binarios, todo el bloque debe utilizarse completo.

El proceso FileSystem debe en todo momento conocer la cantidad de bloques libres y utilizados de cada Nodo y por consiguiente **el espacio libre total**[4]. 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 esté más de una vez en el mismo nodo.

Esquema de filesystem de ejemplo

- `ejemplo.csv`
  - Tamaño: 3.145.592 bytes
  - Tipo: Delimitado
  - Directorio Padre: 5
  - Estado: Disponible
  - Bloques:

Bloque	Copia 1	Copia 2	Fin Bloque
0	Nodo 1 - Bloque 5	Nodo 2 - Bloque 2	1048576
1	Nodo 2 - Bloque 10	Nodo 3 - Bloque 7	1048500
2	Nodo 1 - Bloque 12	Nodo 2 - Bloque 3	1048516

- `foto.jpg`
  - Tamaño: 1.472.461 bytes
  - Tipo: Binario
  - Directorio Padre: 1
  - Estado: Disponible
  - Bloques:

Bloque	Copia 1	Copia 2	Fin Bloque
0	Nodo1 - Bloque 43	Nodo2 - Bloque 18	1048576
1	Nodo3 - Bloque 56	Nodo4 - Bloque 66	1048576
2	Nodo4 - Bloque 91	Nodo2 - Bloque 75	1048576
...	..	..	

Proceso de Inicialización

Al iniciar el proceso Filesystem y los diversos procesos DataNode el sistema debe poder restaurarse de un estado anterior, si el mismo existiese. Para ello, el proceso FileSystem persiste una suerte de archivos que mantienen las estructuras almacenadas en disco. Si

existe un estado anterior (es decir, existiesen todos los archivos mencionados), el proceso FileSystem levantará todas las estructuras y quedará a la espera de los procesos DataNode.

Durante el establecimiento de la conexión, el FileSystem comparará el id del Nodo al que pertenece el DataNode, y si el mismo corresponde a un Nodo de un estado anterior, el mismo será agregado como válido, asumiendo que los datos de los bloques son correctos.

Mientras que el FileSystem no posea al menos 1 copia de cada uno de los archivos almacenados en los DataNodes conectados, se denomina que **el proceso FileSystem está en un estado no-estable**. Una vez adquiridas al menos 1 copia de cada archivo, se considerará que **el proceso FileSystem ha pasado a un estado estable** y permitirá nuevamente conexiones de los demás procesos. Además, aunque el proceso FileSystem pase a un *estado estable*, permitirá que los nodos que todavía no se conectaron, vuelvan a hacerlo.

Como aclaración final, si el proceso FileSystem es ejecutado con el flag `--clean` se procederá a ignorar e eliminar el estado anterior.

Ejemplo de ejecución de un proceso FileSystem ignorando el estado anterior: `./yamafs --clean`

Interfaz del proceso Filesystem

Contará con una interfaz detallada contra los Nodos, la cual no puede ser modificada ni ampliada pero la definición del protocolo será definida por el grupo.

Almacenar archivo

Recibirá una ruta completa, el nombre del archivo, el tipo (texto o binario) y los datos correspondientes. Responderá con un mensaje confirmando el resultado de la operación. Es responsabilidad del proceso Filesystem:

1. “Cortar” el archivo en registros completos hasta 1MB en el caso de los archivos de texto o en bloques de 1 MB en el caso de los binarios.
2. Designar el DataNode y el Bloque que va a almacenar cada copia[5] utilizando sus estructuras administrativas y balanceando el contenido de manera homogénea entre los diversos DataNodes disponibles.
3. Enviar el contenido del bloque a los DataNodes designados.

Leer archivo

Recibirá una ruta completa con el nombre del archivo. Responderá con el contenido completo del mismo. Es responsabilidad del proceso Filesystem:

1. Solicitar a los diferentes DataNodes los bloques correspondientes al archivo considerando que las solicitudes deberán distribuirse **de forma equitativa** en los diferentes DataNodes según cómo estén distribuidas las copias de los bloques.

Consola del Proceso Filesystem

Este proceso tendrá una consola desde la cual se podrán ejecutar comandos de administración de archivos. Se recomienda la utilización de la biblioteca readline[6] para la captura de teclas especiales.

A continuación se hará una breve descripción de los comandos a utilizar mediante una sintaxis cómoda pensado por la cátedra. La sintaxis de los mismos **no es de carácter obligatorio**, pero se recomienda su uso dada la similaridad con los comandos usuales. Como aclaración, algunos comandos pueden llegar a ser reutilizables para diferentes tareas. Debido a esto, cada una de ellas tendrán asignadas un flag (no combinables entre sí) para identificar de qué comportamiento se trata.

1. `format` - Formatear el Filesystem.
2. `rm [path_archivo]` ó `rm -d [path_directorio]` ó `rm -b [path_archivo] [nro_bloque] [nro_copia]` - Eliminar un Archivo/Directorio/Bloque. Si un directorio a eliminar no se encuentra vacío, la operación debe fallar. Además, si el bloque a eliminar fuera la última copia del mismo, se deberá abortar la operación informando lo sucedido.
3. `rename [path_original] [nombre_final]` - Renombra un Archivo o Directorio
4. `mv [path_original] [path_final]` - Mueve un Archivo o Directorio
5. `cat [path_archivo]` - Muestra el contenido del archivo como texto plano.
6. `mkdir [path_dir]` - Crea un directorio. Si el directorio ya existe, el comando deberá informarlo.
7. `cpfrom [path_archivo_origen] [directorio_yamafs]` - Copiar un archivo local al yamafs, siguiendo los lineamientos en la operaciòn Almacenar Archivo, de la Interfaz del FileSystem.
8. `cpto [path_archivo_yamafs] [directorio_filesystem]` - Copiar un archivo local desde el yamafs
9. `cpblock [path_archivo] [nro_bloque] [id_nodo]` - Crea una copia de un bloque de un archivo en el nodo dado.
10. `md5 [path_archivo_yamafs]` - Solicitar el MD5 de un archivo en yamafs
11. `ls [path_directorio]` - Lista los archivos de un directorio
12. `info [path_archivo]` - Muestra toda la información del archivo, incluyendo tamaño, bloques, ubicación de los bloques, etc.

Estructuras del proceso filesystem

Las distintas estructuras administrativas del proceso Filesystem se persisten en archivos individuales respetando el siguiente formato

Tabla de Directorios

El Filesystem soporta directorios hasta un total de 100 directorios. Todos los archivos y directorios deben tener un directorio padre, el cual puede ser otro directorio o el directorio raíz (padre=0).

Index	Directorio	Padre
0	root	-1
1	user	0
2	jose	1
3	juan	1
4	temporal	0
5	datos	3
6	fotos	3
99	...	...

Como pueden ver este esquema permite el siguiente diagrama:

```
raiz (/)
|--- /user
|     |--- /user/juan
|     |    |--- /user/juan/datos
|     |    |--- /user/juan/fotos
|     |--- /user/jose
|--- /temporal
```

La tabla de directorios debe ser almacenada en un archivo en el filesystem local (no yamafs) denominado metadata/directorios.dat como un array de 100 posiciones de `struct t_directory` .

```
struct t_directory {
  int index,
  char nombre[255],
  int padre
}
```

Tabla de archivos

La información de cada archivo dentro de yamafs será almacenada en archivo texto compatible dentro de un directorio cuyo nombre corresponde al índice de la tabla de directorios. Siguiendo con el ejemplo anterior todos los archivos de `/user/juan/datos serán almacenados en metadata/archivos/5/` .

```
metadata/archivos/5/ejemplo.csv
TAMANIO=12356334
TIPO=TEXTO
BLOQUE0COPIA0=[Nodo1, 33]
BLOQUE0COPIA1=[Nodo2, 11]
BLOQUE0BYTES=1048500
BLOQUE1COPIA0=[Nodo1, 34]
BLOQUE1COPIA1=[Nodo2, 12]
BLOQUE0BYTES=1048532
...
```

Tabla de Nodos

Iniciar el filesystem, el mismo deberá registrar los DataNodes que se vayan conectando y terminan componiendo dicho sistema de archivos. Además, deberá registrar su cantidad de bloques total y libres. Esto se persistirá en el `archivo metadata/nodos.bin` .

```
TAMANIO=300
LIBRE=171
NODOS=[Nodo1, Nodo2, Nodo3]
Nodo1Total=50
Nodo1Libre=16
Nodo2Total=100
Nodo2Libre=80
Nodo3Total=150
Nodo3Libre=75
```

Bitmap de Bloques por Nodo

Por cada DataNode del Filesystem se deberá almacenar un bitmap indicando el estado (libre/ocupado) de cada bloque. Esta información se deberá persistir en `metadata/bitmaps/[nombre-de-nodo].dat` . Ejemplo: `metadata/bitmaps/nodo2.dat`

Archivo de Logs

El proceso FileSystem deberá registrar toda su actividad mediante un archivo de log, sin mostrar por pantalla dichos logs, por la existencia de una consola. Sin embargo, sí se deberán mostrar por pantalla los resultados de las tareas realizadas mediante la consola.

> [2]. Ver apartado sobre Proceso de Inicialización ↩

> [3]. El valor `\n` indica 'New Line' o 'Nueva línea'. Equivale al ingreso de un `[Enter]` en el teclado. ↩

> [4]. El alumno debe investigar opciones para almacenar esta información de manera eficiente ↩

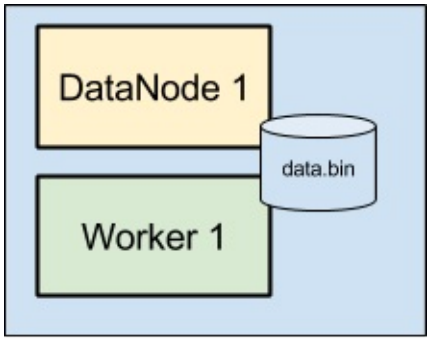
> [5]. No está permitido almacenar las dos copias en el mismo DataNode ↩

> [6]. Se recomienda la lectura previa del tutorial Cómo realizar una consola interactiva ↩

Componente Nodo

El nodo en este trabajo práctico es un concepto abstracto compuesto por los procesos Worker y DataNode ejecutando en la misma computadora. Ambos comparten archivo de configuración y solo un Nodo puede ser ejecutado por computadora.

- DataNode
- Worker
- Archivo de Datos (data.bin)



Nodo 1

Como muestra el diagrama el archivo de datos es accedido por ambos procesos de manera concurrente. No hay inconvenientes de sincronización dado que el acceso del Worker es únicamente para lectura de bloques previamente escritos.

Ambos procesos son agnósticos del contenido de data.bin, únicamente cumplen con el objetivo de almacenar/leer bloques (en el caso del DataNode) y procesar bloques en el caso del Worker.

Ejemplo de archivo de configuración del Nodo

```
IP_FILESYSTEM=192.168.1.254
PUERTO_FILESYSTEM=5040
NOMBRE_NODO=NODO1
PUERTO_WORKER=5050
RUTA_DATABIN=/home/utnso/data.bin
```

DataNode

Al iniciar este proceso, leerá el archivo de configuración del nodo, abrirá el archivo data.bin y deberá conectarse al proceso Filesystem, identificarse y quedar a la espera de solicitudes por parte del proceso Filesystem respetando la interfaz descrita a continuación:

Interfaz con el FileSystem

El FileSystem podrá 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

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

Archivo de Log

El proceso DataNode deberá contar con un archivo de configuración en el cual se logearán todas las operaciones realizadas. Las mismas deberán ser mostradas por pantalla.

Worker

El Worker es un proceso encargado de aplicar operaciones sobre bloques de archivos o sobre archivos temporales. Las operaciones son recibidas por parte del Master quien se conecta al Worker para ejecutar una tarea comandada por YAMA.

Al iniciar leerá el archivo de configuración del nodo y quedará a la espera de conexiones por parte de procesos Master. Al recibir dicha conexión Worker creará un proceso hijo (`fork()`) para atender la solicitud el cual terminará al concluir la operación. Es menester aclarar que no se deberán crear hilos para atender las conexiones.

Recibirá del proceso Master, independientemente de en qué etapa se encuentre, un código a ejecutar, un origen de datos y un destino.

En el caso de una etapa de Transformación el origen debe ser una porción del archivo data.bin (un bloque de datos) y el destino un archivo temporal, en el caso de una operación de Reducción el origen y el destino son archivos temporales del filesystem local del Nodo.

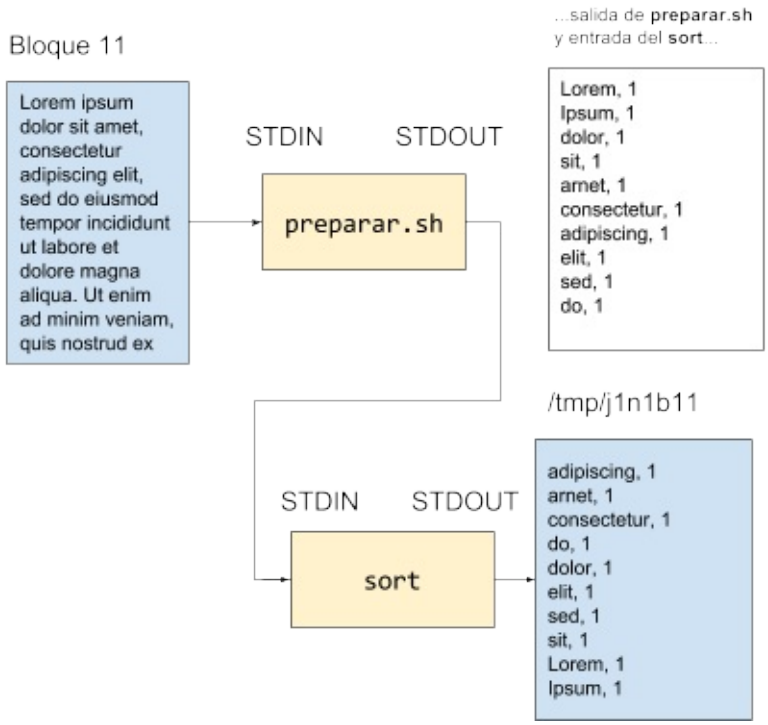
El archivo de procesamiento es un programa o un script que debe ejecutarse localmente en el nodo, recibiendo por entrada estándar (`STDIN`) el origen y almacenando el resultado expuesto por salida estándar (`STDOUT`) en el correspondiente destino[7].

El Worker es completamente agnóstico de lo que cada programa realiza sobre los datos. Su función es pasarle por entrada estándar el contenido indicado por el Master y almacenar la salida estándar en el archivo temporal indicado por el Master.

Existen particularidades del procesamiento dependiendo en qué etapa nos encontremos:

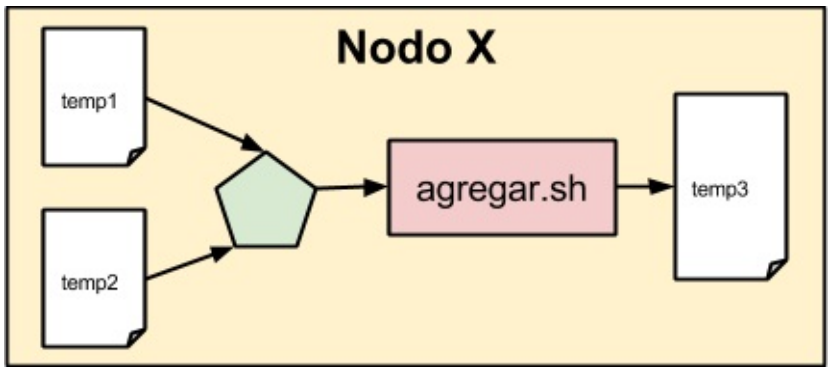
Etapa de Transformación

En esta etapa existe una particularidad en la ejecución de la rutina de Transformación donde, para poder lograr un eficiente procesamiento distribuído, **el resultado debe ser almacenado de manera ordenada (** `sort` **)**.



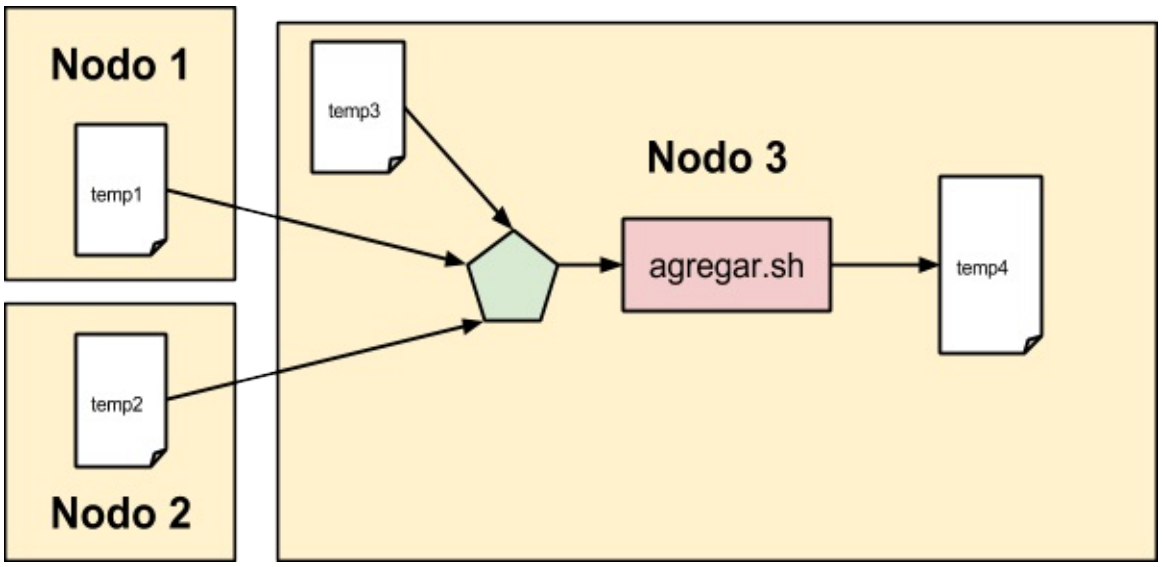
Etapa de Reducción Local

En esta etapa se asume que los datos recibidos están ordenados de manera alfabética ya que el paso anterior lo hizo antes de almacenarlos. Para esta etapa los diversos archivos temporales deben ser apareados antes de ser enviados al programa de Reducción.



Etapa de Reducción Global

Para esta etapa se asume que los archivos nuevamente están ordenados alfabéticamente por lo que en este caso el apareo debe ocurrir con los archivos entre nodos antes de ser enviados al programa de Reducción



Para esta etapa, los Nodos quedarán a la espera de una conexión de Master (si fuesen elegidos Encargados por YAMA) o una conexión de otro Nodo.

Archivo de Log

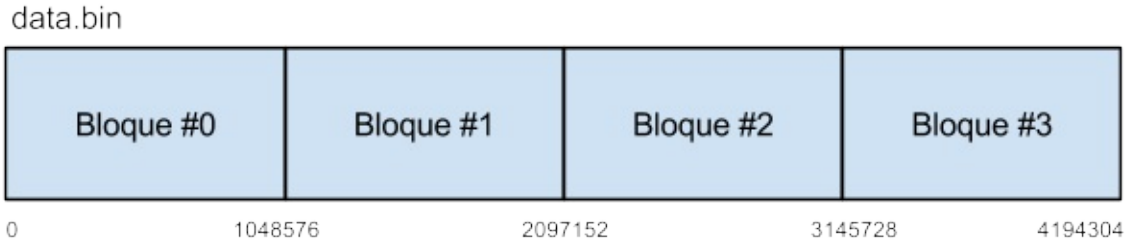
El proceso Worker deberá contar con un archivo de configuración en el cual se logearán todas las operaciones realizadas. Las mismas deberán ser mostradas por pantalla.

Archivo de Datos (data.bin)

El Archivo de Datos del Nodo será un archivo almacenado localmente de tamaño pre-solicitado. 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 20MB, antes de iniciar los procesos del Nodo por primera vez, se deberá generar un archivo local de 20MB llamado data.bin. Luego el primer MB de este archivo corresponden al bloque 0 del nodo, el segundo MB al bloque 1 y sucesivamente. Tanto el proceso Worker como el DataNode deben obtener el tamaño total del archivo de sus propiedades (ver syscall `stat()`)).

En general para acceder al bloque `X` , se acceder a este archivo en la posición `X * 1 MB` .



> [7]. Lectura recomendada: https://github.com/nicozar95/ejemplo_correr-script_so ↩

Descripción de las entregas

Hito 1: Checkpoint 1

- **Fecha**: 16 de Septiembre
- **Timepo Estimado**: 2 semanas
- **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
  - Definir el Protocolo de Comunicación
  - Implementar la consola del FileSystem sin funcionalidades
  - Desarrollar una comunicación simple entre los procesos que permita propagar un archivo abierto por Master.
  - Familiarizarse con el desarrollo de procesos servidor multihilo (proceso Master) y server con forks(Worker)
- Lectura recomendada:
  - http://faq.utnso.com/arrancar
  - Beej Guide to Network Programming
  - Linux POSIX Threads
  - SO UTN FRBA Commons Libraries
  - Sistemas Operativos, Silberschatz, Galvin - Capítulo 4: Hilos

Hito 2: Avance del Grupo[8]

- **Fecha**: 7 de Octubre
- **Timepo Estimado**: 3 semanas
- **Objetivos**:
  - Implementación de la base del Protocolo de Comunicación
  - Comprender y aplicar `mmap()`
  - Desarrollar las Estructuras del FileSystem (Tablas de Directorios, Archivos, Nodos, Bitmap)
  - YAMA debe poder planificar tareas mediante el algoritmo Clock
  - Master y YAMA pueden efectuar todo un recorrido (sin comunicación contra el Worker ni FS) respetando el RR
  - El FileSystem debe permitir la copa de un archivo al yamafs y almacenarlo partiéndolo en bloques.

- Lectura recomendada:
  - Sistemas Operativos, Silberschatz, Galvin - Capítulo 4: Hilos
  - Sistemas Operativos, Silberschatz, Galvin - Capítulo 10 y 11: Sistemas de Archivos
  - Sistemas Operativos, Silberschatz, Galvin - Capítulo 17: Sistemas de Archivos Distribuidos

Hito 3: Checkpoint Presencial en el Laboratorio

- **Fecha**: 21 de Octubre
- **Tiempo Estimado**: 2 semanas
- **Objetivos**:
  - Los procesos Master y YAMA manejan errores durante la ejecución de un Job.
  - El proceso YAMA puede replanificar una etapa de planificación y admite planificación con W-Clock.
  - El proceso YAMA debe poder conectarse al FileSystem y levantar la metadata sobre un archivo sobre el cual ejecutar un Job.
  - La Consola del FileSystem es capaz de reproducir las operaciones de copia de archivos contra el FileSystem local y la generación del md5 de un archivo almacenado.
  - Los procesos Worker deben poder recibir archivos por socket y darles permisos de ejecución.
- Lectura recomendada:
  - Ejemplos de implementación de redirección de stdin/stdout mediante pipes
  - Cómo realizar una consola interactiva

Hito 4: Entrega Final

- **Fecha**: 25 de Noviembre
- **Objetivos**:
  - El proceso Master calcula las métricas de un Job
  - La Consola del FileSystem es capaz de efectuar operaciones sobre Nodos y Bloques.
  - El proceso YAMA maneja correctamente la señal SIGUSR1 y recarga su configuración.
  - Es posible realizar una inicialización del FileSystem con datos viejos.
  - Todos los componentes del TP ejecutan los requerimientos pedidos de forma integral, bajo escenarios de stress.
- Lectura recomendada:
  - Guías de Debugging del Blog utnso.com

    - MarioBash: Tutorial para aprender a usar la consola

Otras entregas:

- Segunda entrega: 2 de Diciembre
- Tercera entrega: 16 de Diciembre

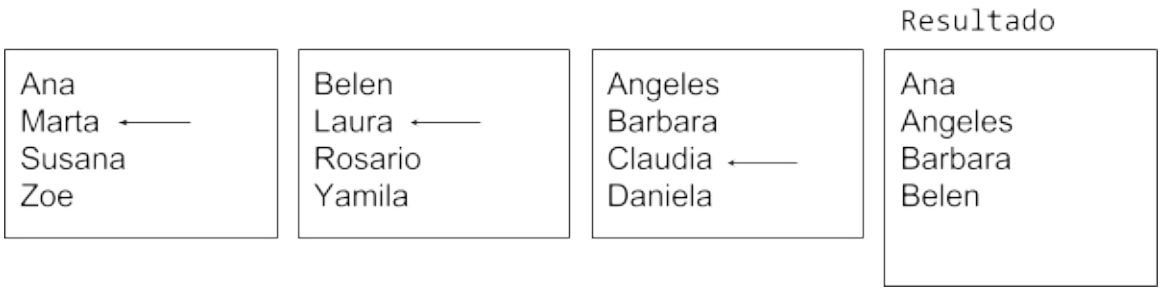
[8] No es requerido entregar nada, es a título informativo

Anexo I - Apareo de Archivos

Apareo es el nombre que se le da a una rutina que dado N archivos ordenados genera uno nuevo con el resultado ordenado de la unión de los mismos.

En términos de procesamiento, realizar el concatenado de los N archivos y luego ordenarlo resulta muy costoso, por lo que teniendo en cuenta que los archivos de origen ya están ordenados se realiza comparando registro por registro.

Como puede observar en el ejemplo en una primera etapa se comparan los primeros registros de cada archivo (Ana, Belen y Angeles). Dado que Ana es el menor alfabéticamente este es trasladado al archivo de resultado. Luego leemos el segundo nombre del primer archivo (Marta) y comparamos nuevamente.



Anexo II - Algoritmos de Planificación

Historia Previa

A raíz de varias consultas sobre el funcionamiento del funcionamiento del algoritmo Round Robin y Weighted Round Robin; la cátedra decidió realizar una estandarización de una implementación de los mismos bajo un nombre diferente, a fin de evitar confusiones con los vistos en la teoría de Planificación de Procesos: Clock y W-Clock. Para ello, primero hemos de realizar una pequeña introducción sobre las particularidades de estos algoritmos aplicados al Trabajo Práctico.

(Pre?) Planificación

Existen diferencias entre el algoritmo que se ve en la teoría y la implementación necesaria para este Trabajo Práctico. La principal diferencia radica en que lo denominado *planificación* es, en realidad, una **pre-planificación**.

En un sistema convencional nuestro sistema puede *quitar un recurso a un proceso y asignárselo a otro*, o bien esperar a que un *proceso desaloje* un recurso para planificarlo. Esto significa que, durante las etapas de planificación, nuestro sistema **reasigna recursos**.

Dicho concepto, **no existe en el trabajo práctico**, ya que no sólo, los mismos Nodos son capaces de procesar múltiples tareas de Jobs al simultáneo, sino que además, las tareas de reducción deben ejecutarse en los mismos Nodos que las tareas de transformación. Esto contamina el concepto anterior y evitando que la ejecución pueda ser cortada y resumida en otro Nodo.

Detalle de los Algoritmos

Ambos algoritmos poseerán un modo de ejecución similar, diferenciándose en la forma de obtención de la función de Availability (o disponibilidad) de los Workers, que determinará cuántas operación-bloque a realizar.

Preconceptos

En esta pre-planificación, los recursos sobre los cuales trabajar en por YAMA serán los Workers, siendo la unidad de pre-planificación la tarea a efectuarse sobre el bloque de un archivo en particular.

El algoritmo se compone de tres partes:

- Una **función de Availability** o Disponibilidad
- Un puntero a la instrucción base, denominado **Clock**, de funcionamiento similar al del algoritmo de Memoria visto en la teoría que lleva el mismo nombre.
- Un conjunto de **Reglas** a seguir.

Obtención de la función de Availability

Como se dijo anteriormente, dado que el funcionamiento del Clock y el set de Reglas son idénticos tanto para el algoritmo Clock como para W-Clock, *la diferenciación radica en su función de disponibilidad*. Para poder definir la misma, **es necesario conocer el valor del conocido Disponibilidad Base**, que deberá estar detallado en el archivo de configuración de YAMA, y podrá cambiar en tiempo de ejecución, siendo tomado en cuenta para la próxima pre-planificación.

La función de Availability de un Worker, definida como A(w) responderá a lo siguiente:

A(w) = Base + PWL(w)

Siendo PWL(w)=0 bajo el algoritmo *Clock* y, bajo el algoritmo *W-Clock*, será la disponibilidad del Worker en función de su carga de trabajo actual. En este segundo caso, responderá a la fórmula:

PWL(w) = WLmax - WL(w)

Siendo WLmax: la mayor carga de trabajo existente entre todos Workers *WL(w)*: la carga de trabajo actual del Worker.

Nota: La cargas de trabajo deberán corresponder a un *número entero sin signar de 32bits*.

Reglas de Funcionamiento

1. Inicio:
    - Se calcularán los valores de disponibilidades para cada Worker
    - Se posicionará el Clock en el Worker de mayor disponibilidad, *desempatando por el primer worker que tenga menor cantidad de tareas realizadas históricamente*.
2. Se deberá evaluar si el Bloque a asignar se encuentra en el Worker apuntado por el Clock y el mismo tenga disponibilidad mayor a 0.
    - Si se encuentra, se deberá reducir en 1 el valor de disponibilidad y avanzar el Clock al siguiente Worker. Si dicho clock fuera a ser asignado a un Worker cuyo nivel de disponibilidad fuera 0, *se deberá restaurar la disponibilidad al valor de la disponibilidad base y luego, avanzar el clock al siguiente Worker, repitiendo el paso 2*.

    - En el caso de que no se encuentre, se deberá utilizar el siguiente Worker que posea una disponibilidad mayor a 0. *Para este caso, no se deberá modificar el Worker apuntado por el Clock*.
    - Si se volviera a caer en el Worker apuntado por el clock por no existir disponibilidad en los Workers (es decir, da una vuelta completa a los Workers), *se deberá sumar al valor de disponibilidad de todos los workers el valor de la* ***Disponibilidad Base*** *configurado al inicio de la pre-planificación*.
3. Si existe otro Bloque sobre el cual realizar una tarea, efectuar el paso (2) nuevamente con el nuevo bloque.

Una vez finalizado el paso (3), se habrá obtenido la planificación a entregar al Master. Es necesario recalcar que la creación de los planes de ejecución se realizará de forma secuencial, obteniendo cada Job su propia planificación de tareas.

Actualización de la Carga de los Workers

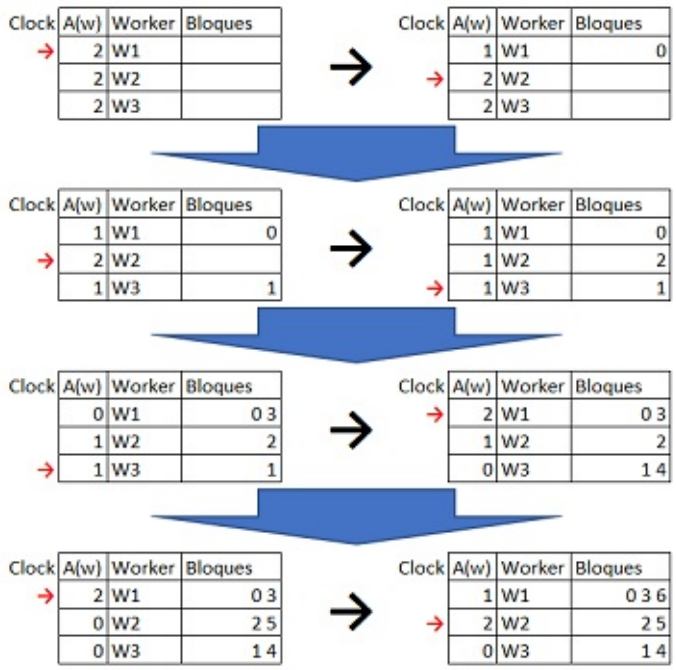
Tras haber finalizado la creación del plan de ejecución, es necesario que YAMA actualice la carga de trabajo Worker (WL(w)), importantes para la ejecución del algoritmo W-Clock, siendo una unidad por cada Bloque sobre el cual ejecutar una transformación + reducción local.

Como caso particular durante la Reducción Global, a la hora de elegir un Worker Encargado se utilizará el que menor WL(w) tuviera al efectuar dicha etapa. Además, se deberá sumar una cantidad igual a la mitad (redondeando hacia arriba) de la cantidad de archivos temporales a reducir, a la carga de trabajo Worker Encargado, a modo de denotar el diferente peso de las tareas a realizar entre los Workers en dicha etapa.

Finalmente, luego de que se finalice un Job, se deberá reducir la WL(w) la cantidad de unidades que se sumaron, a fin de reducir la carga de trabajo del Worker.

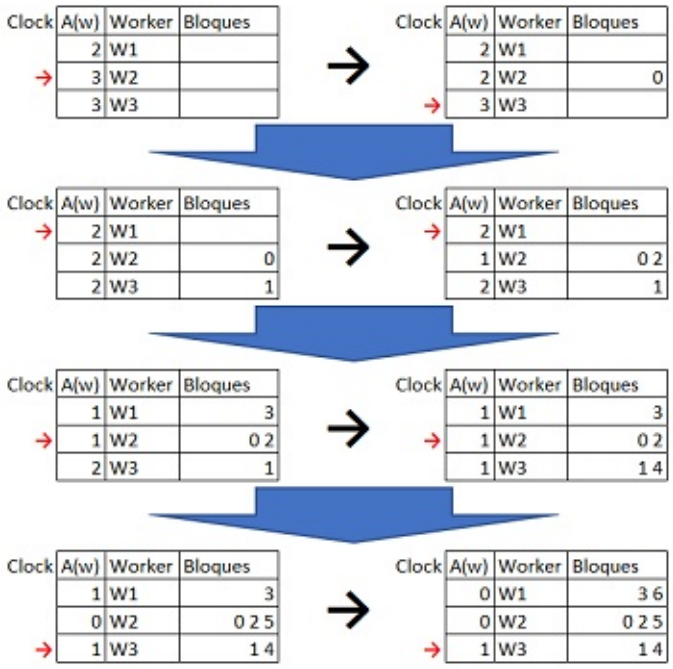
Ejemplos

Supongamos un Archivo compuesto por 6 bloques, con la siguiente distribución de los Bloques: DataNode1: 0 1 3 4 6 DataNode2: 0 2 3 5 6 DataNode3: 1 2 4 5 y un valor de Disponibilidad Base de 2, veamos cómo se comportan los algoritmos. Bajo la primera ejecución, tanto el Clock como el W-Clock se comportarán de forma idéntica. Veamos cómo resultaría:



Como vemos, terminaría generando una carga de trabajo en los Workers de 3 para el primero y 2 para los restantes. Si lanzamos un nuevo Job que actuara sobre el mismo archivo, mediante el algoritmo Clock obtendremos el mismo resultado, generando una carga de trabajo para los Workers de 6 y 4 para los restantes y así sucesivamente.

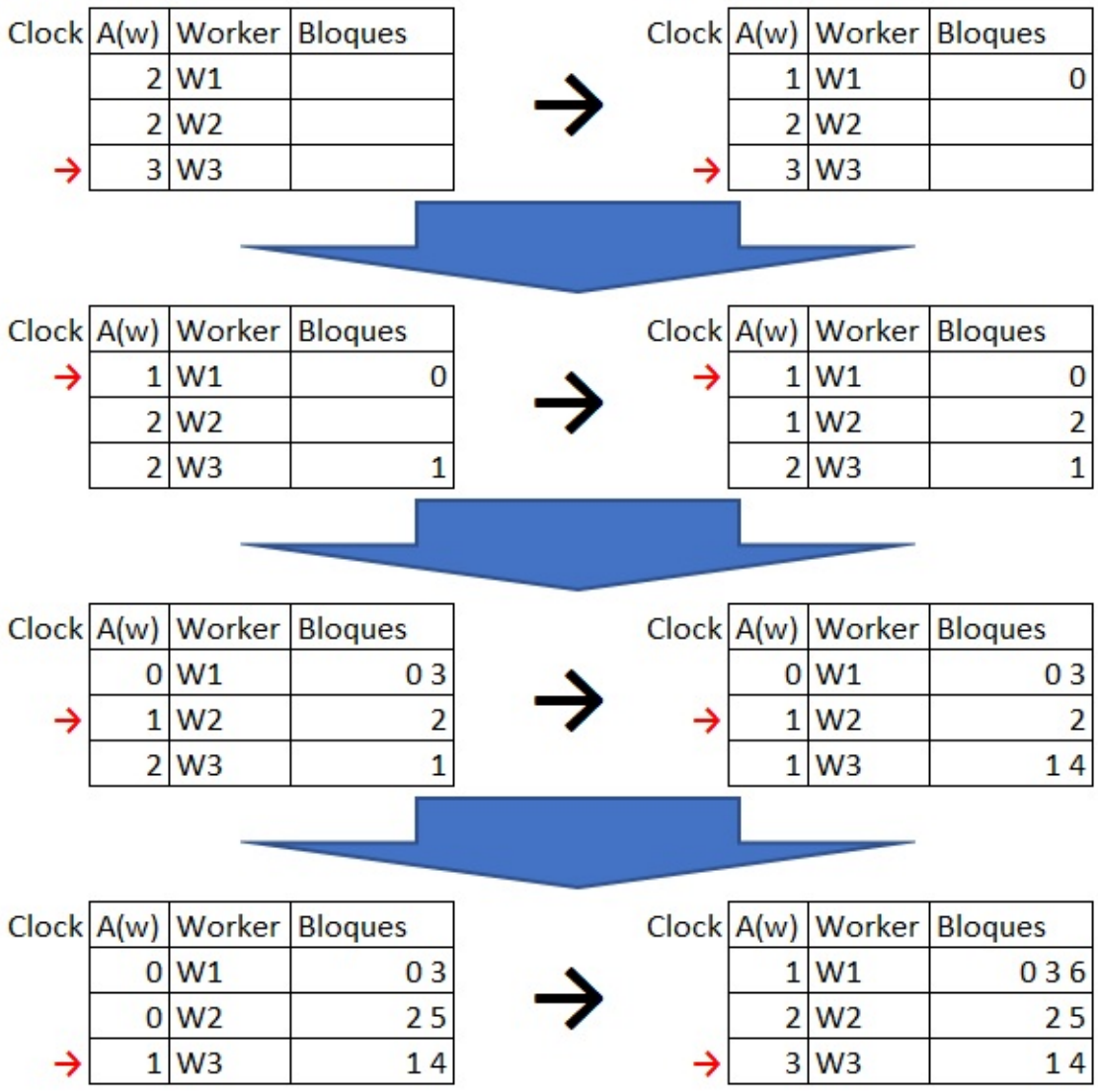
Sin embargo, si utilizamos el algoritmo W-Clock que toma en cuenta la carga de trabajo, obtendremos lo siguiente:



Primero que nada, al cambiar los valores de Disponibilidad [A(w)], el Clock arrancaría apuntando al primer Worker de mayor carga de trabajo. Además, el funcionamiento estará atado al los diferentes valores de disponibilidad, a diferencia del algoritmo Clock.

El resultado de la ejecución del W-Clock resultaría en una carga de trabajo final de 5 para los 2 primeros Workers y 4 para el tercero.

Algo a notar, es que en contraste con el algoritmo Clock, si ejecutamos un tercer Job sobre, por ejemplo, los mismos archivos mientras se estuvieran ejecutando los otros 2, obtendrìamos algo como lo siguiente:



Asimismo, el hecho que genere una carga de trabajo final de: Worker 1: 8 Worker 2: 7 Worker 3: 6

contra 9 para el primer Worker y 6 para los restantes; demuestra que el algoritmo W-Clock de pre-planificación realiza una mejor distribución de la carga de trabajo que su contraparte que no contempla la carga de trabajo de los Workers.