Universitat Autònoma de Barcelona

Dipòsit digital de documents de la UAB

This is the **published version** of the bachelor thesis:

Fernández García, Alejandro; Sikora, Anna Bàrbara, dir. Generación automática de scripts SLURM para ejecuciones masivas. 2023. (Enginyeria Informàtica)

This version is available at https://ddd.uab.cat/record/280686

under the terms of the BY-NC-ND license

# Generación automática de scripts SLURM para ejecuciones masivas

Alejandro Fernández García

**Resumen–** HPC es una tecnología que utiliza clústeres de potentes procesadores trabajando en paralelo con el objetivo de procesar conjuntos de datos masivos (big data) y resolver problemas complejos a una velocidad extrema. Normalmente, las ejecuciones que mueven esta cantidad de datos, suelen estar paralelizadas y optimizadas para mejorar el tiempo, por lo que para estos casos HPC es la tecnología idónea. Esto es debido a que HPC es capaz de soportar una computación masiva en paralelo. Para ello suelen utilizarse Scripts que contienen la configuración del entorno (número de nodos, CPUs por tarea...), de la ejecución del experimento y opciones de compilación. Partiendo de esta premisa, en este trabajo final de grado, se ha creado una aplicación para generar, de forma automática, Scripts SLURM para ejecuciones masivas. Esta aplicación tiene como finalidad, facilitar la preparación y configuración de ejecuciones experimentales. Como las ejecuciones masivas de experimentos científicos requieren de una buena preparación de la ejecución, gracias al uso de este programa ha reducido el tiempo que necesita el investigador de una media hora u hora, más o menos, a solo unos pocos minutos. Esto permite agilizar las ejecuciones que se desean lanzar y optimizando el tiempo de trabajo del investigador.

**Palabras clave–** Jenkins, HPC, Cluster, Ejecuciones masivas, Slurm, Script

**Abstract–** HPC is a technology that utilizes clusters of powerful processors working in parallel with the objective of processing massive datasets (big data) and solving complex problems at high speeds. Typically, executions that involve moving this amount of data are parallelized and optimized to improve processing time, making HPC the ideal technology for such cases. This is because HPC is capable of supporting massive parallel computing. Scripts are often used for this purpose, which contain the configuration of the environment (number of nodes, CPUs per task, etc.), the execution of the experiment, and compilation options. Based on this premise, in this final undergraduate project, an application has been created to automatically generate SLURM scripts for massive executions. The purpose of this application is to facilitate the preparation and configuration of experimental executions. Since massive executions of scientific experiments require thorough preparation, the use of this program has reduced the time needed by researchers from an average of half an hour or an hour to just a few minutes. This allows for faster launching of executions and optimizes the researcher's working time.

**Keywords–** Jenkins, HPC, Cluster, Mass executions, Slurm, Script

## 1 Introducción - Contexto del trabajo

HPC[1] es una tecnología que utiliza clústeres de potentes procesadores trabajando en paralelo con el objetivo de procesar conjuntos de datos masivos (big data) y resolver problemas complejos a una velocidad extrema. Normalmente las ejecuciones que mueven esta cantidad de datos, suelen estar paralelizadas y optimizadas para mejorar el tiempo, por lo que para estos casos HPC es la tecnología idónea. Esto es debido a que HPC es ca-

• E-mail de contacto: alejandro.fernandezgar@uab.cat
• Mención realizada: Enginyeria de Computadors
• Trabajo tutorizado por: Ania Sikora (Department d'Arquitectura de Computadors i Sistemes Operatius )
• Curs 2022/23

paz de soportar una computación masiva en paralelo. Este trabajo, consiste en ejecutar varias tareas simultáneamente en varios procesadores o servidores informáticos. Por otro lado, la computación masiva en paralelo, es trabajo en paralelo (o computación paralela) que utiliza decenas de miles o millones, depende de las necesidades requeridas, de procesadores o núcleos de procesador.

Las aplicaciones HPC abarcan una amplia gama de áreas, desde simulaciones científicas hasta inteligencia artificial y aprendizaje automático. Durante décadas, se ha utilizado el poder de las supercomputadoras, que son ordenadores diseñados específicamente con millones de procesadores o núcleos de procesador, para abordar estos desafíos. Estas máquinas están optimizadas para realizar una gran cantidad de cálculos simultáneos a alta velocidad. Con el paso de los años han ido surgiendo nuevos paradigmas que aprovechan la tecnología HPC, como la memoria compartida y distribuida, así como las ejecuciones masivas de aplicaciones científicas. En el caso de la memoria compartida [2], los procesadores comparten un espacio de memoria común, que permite comunicación e intercambio de datos eficientes, haciendo que este paradigma sea útil cuando se necesita un grado de interacción alto entre procesadores. Por otro lado, la memoria distribuida [3] se diferencia de la compartida, es que los procesadores tienen su propia memoria y se comunican mediante mensajes. En este caso, la memoria compartida es útil para aplicaciones donde los datos se dividen en partes pequeñas y que pueden procesarse de manera independiente. Por otro lado, las aplicaciones científicas suelen implicar cálculos intensivos y requerir de una gran capacidad de procesamiento. Estas ejecuciones implican procesar grandes volúmenes de datos y realizar cálculos complejos en paralelo. Por este motivo HPC es tan importante, ya que proporciona la capacidad de procesamiento necesaria para realizar estos cálculos. En muchos casos, para poder llevar a cabo la ejecución del código, de estos experimentos masivos, es necesaria la ejecución con SLURM.

SLURM[4]es un sistema de gestión de clústeres tolerante a fallas y altamente escalable para clústeres Linux. Como gestor de trabajos en clústeres, SLURM cumple tres funciones clave. Para comenzar, asigna acceso exclusivo o no exclusivo a recursos (nodos) a los usuarios durante un tiempo determinado, para que estos puedan realizar su trabajo. En segundo lugar, proporciona un marco para iniciar, ejecutar y supervisar el trabajo dentro de los nodos asignados. Par finalizar, gestiona la contención de recursos dentro de la cola de trabajos pendientes.

Se utiliza SLURM debido a que, al ser ejecuciones masivas, estas pueden crear bloqueos en la máquina hasta que su trabajo finalice. Gracias al uso de esta herramienta, enviamos la ejecución a un nodo de la máquina, con una configuración definida, para que se ejecute y así poder seguir utilizando la máquina para otras tareas. Esta configuración es la que esta contenida dentro del Script que ejecutamos con SLURM. En este script, se define la configuración donde se incluyen diferentes parámetros como pueden ser, la cantidad de threads que se quieren utilizar, los archivos necesarios para la ejecución, la cantidad de kernels o kernel en concreto en el que se quiere trabajar, etc. También en el script, se debe configurar la compilación del código y la configuración de este.

En este trabajo final de grado, se busca crear una aplicación para generar, de forma automática, Scripts SLURM para ejecuciones masivas.

Esta aplicación tendrá como finalidad, facilitar la preparación y configuración de ejecuciones experimentales. Como las ejecuciones masivas de experimentos científicos requieren de una buena preparación de la ejecución, este programa reducirá dicho trabajo para el investigador.

La aplicación deberá tener en cuenta todas las posibles combinaciones de parámetros de entorno y de configuración que un experimento pueda tener. Para realizar el proyecto, se trabajará en un clúster HPC, con código y aplicaciones en C y un gestor de colas SLURM disponibles en la máquina de investigación del Departament d'Arquitectura de Computadors i Sistemes Operatius [5] de l'Escola d'Enginyeria de la UAB [6].

Este articulo esta organizado de la siguiente forma:

- Sección 2: Presenta los objetivos del trabajo.
- Sección 3: En ella se describe la metodología aplicada en este proyecto.
- Sección 4: En esta sección se explica todo el desarrollo realizado en el trabajo.
- Sección 5: En ella se pueden encontrar los resultados obtenidos de la realización de este tfg.
- Sección 6: En esta sección es donde se encuentran las conclusiones de este trabajo.

## 2 Objetivos

El principal objetivo de este trabajo final de grado consiste en generar scripts de forma automática. Esto gracias a la creación de un programa que recibe todos los parámetros y configuración necesaria para la ejecución del experimento. Una vez el programa a recibido estos datos, devuelve el script para poder ejecutar el trabajo en una cola SLURM. Después del análisis del problema y del objetivo principal, nació el siguiente objetivo secundario:

- Estudiar la posibilidad de que el programa sea capaz de guardar la información de ejecuciones anteriores para que no se repitan ejecuciones previamente realizadas.

A lo largo del proyecto se ha ido trabajando para cumplir con el objetivo principal y estudiando la viabilidad de nuevas propuestas de funcionalidades extra. Este trabajo continuo, ha dado lugar a la creación de tres nuevos objetivos, que se han considerado interesantes para el proyecto:

- Utilización de Jenkins [12] como método de llamada a la ejecución del programa. Se usa Jenkins como interfaz de usuario, para que el investigador pueda introducir, de una forma cómoda en un formulario, todos los datos que desea que esten en su script. Una vez introducidos, el usuario podrá ejecutar el programa directamente desde Jenkins y obtener su archivo al momento.
- Que el programa sea capaz de ejecutar el script, no solo de crearlo. Lo que se busca, es que ademas de crear el script, el mismo se ejecute en el cluster para así tener la ejecución del experimento al mismo tiempo. Es decir, el investigador introducirá los datos para crear

el script, y el programa a parte de crearlo, ejecutará el experimento.

– Posible automatización de la ejecución del programa en un intervalo de tiempo predefinido. Este objetivo esta enlazado con el anterior, y es que se pueda decidir si se quiere ejecutar todo el proceso (de creación de script y ejecución del mismo) automáticamente en un intervalo de tiempo definido, por ejemplo cada semana. De esta forma, si no se necesita cambiar la configuración, Jenkins de forma automática crea el script (con la configuración por defecto) y lo ejecuta.

Para resumir, el objetivo es una aplicación java, que se apoya en Jenkins para su ejecución. Gracias al uso de Jenkins, como gestor de la ejecución, se dispone de una serie de herramientas que complementan la aplicación para poder dotarla de características que servirán para ayudar a mejorar la aplicación en el futuro.

## 3 METODOLOGÍA

Al inicio del proyecto, se propuso un modelo de metodología Ágil [7] como herramienta para el desarrollo del trabajo. Dicha metodología, sirve para el desarrollo de proyectos que precisan de rapidez y flexibilidad. Cada proyecto se divide en pequeñas partes, llamadas Sprints, que deben de realizarse en periodos de pocas semanas. A su vez dichas Sprints pueden dividirse en tareas de corta duración para llevar un control más exhaustivo del trabajo que se está realizando.

Una vez estudiadas las posibilidades que ofrece este método, se decidió por implementarlo debido a su versatilidad en cuanto a la organización de trabajo se refiere. Gracias a haber dividido el programa en sprints, se han podido añadir, modificar e incluso eliminar tareas, con tal de reorganizar todo el flujo de trabajo, sin que el tiempo total se vea muy afectado, obteniendo así buenos resultados en el proyecto.

Paralelo a esta metodología, se creó un grupo de trabajo de varios estudiantes de grado, para poder compartir conocimientos y poder ofrecer ayuda entre los participantes. Esto, junto con la metodología escogida, ha resultado en un modelo de trabajo bastante fructífero y cómodo debido a que, gracias a las reuniones semanales, se puede llevar un control más exhaustivo del trabajo. Además, el poder compartir conocimientos con el resto de los compañeros, mejora la interpretación de los problemas que se van encontrando, gracias a los distintos puntos de vista, que cada uno aporta desde su especialidad.

## 4 DESARROLLO

Durante la realización de este TFG se ha trabajado sobre el código proporcionado por el estudiante de doctorado Suren Harutyunyan. Este trabajo, sirve para ayudar a crear los scripts y configuración para el proyecto de Suren, por lo que se tiene como referencia para la creación y las pruebas de la aplicación de generación de Scripts. El código de Suren, realiza el análisis de la máquina donde se ejecuta, pasando por todos los kernels deseados para obtener los perfonrmance counters. Con esto se crean datasets representativos, con performance counters de cada kernel e información sobre el número de threads. En los casos en los que un kernel se parece mucho a otro (comparando los performance counters) este ya no se incluye en el dataset y, por otro lado, si el dataset es nuevo se incluye.

Actualmente, el Script que contiene la configuración para poder realizar el análisis, se crea de forma manual y supone un gasto de tiempo considerable, teniendo en cuenta tanto el tamaño como la cantidad de datos y parámetros que se deben de configurar. Este Script se encarga de crear y eliminar los directorios donde se almacenan los datos extraídos. Además, también se encarga de ejecutar los análisis según la configuración deseada, que va desde la cantidad de threads o cores que se desean analizar, así como de los kernels que se desean analizar y la cantidad de veces que se deben realizar operaciones en ellos.

Ahora mismo, para ejecutar el programa, se debe lanzar el script dentro de una cola SLURM del clúster de investigación. Este clúster, es un clúster HPC, que dispone de dos colas de servicio. La primera, la cola global, está compuesta por 4 nodos:

– 1 nodo con arquitectura AMD Epyc 7551 de 2 sockets y 128 CPUs

– 2 nodos con arquitectura Intel Xeon E5-2620 de 2 sockets y 24 CPUs cada uno

– 1 nodo con arquitectura Intel Xeon E5-4620 de 4 sockets y 64 Cpus

Por otro lado, la segunda cola, la Intel queue (Llamada así por la arquitectura de sus nodos), dispone de 2 nodos con arquitectura Intel Xeon Gold 6338 de 2 sockets y 128 CPUs.

Cuando lanzamos el script en uno de los nodos del clúster, mediante SLURM, el programa empieza a recopilar todos los datos y los va guardando en los directorios creados para poder acceder a ellos, una vez acabada la ejecución, y proceder a su análisis. Debido a esto, el script hay que ir modificándolo según se quieran cambiar los datos recopilados y por lo tanto, el proyecto que se está desarrollando facilitará en gran medida este trabajo, además de proporcionar un script limpio y con la información necesaria.

### 4.1. Análisis del problema

En primer lugar, se encontró que el problema actual radica en la extensión del script para ejecutar el experimento. Por este motivo, mediante este trabajo, se busca reducir el tiempo que el investigador debe dedicar a preparar un script que contenga toda la configuración para ejecutar su experimento. Por este motivo se quiere crear una aplicación que se encargue de crear un script a partir de los datos específicos que se le proporcionen. De este modo el investigador solo deberá proporcionar los datos que desea utilizar y el programa se encargará de crear el archivo en unos pocos minutos.

Se ha analizado todo el script para determinar que datos, de los que genera la ejecución, son los que se almacenan para su posterior análisis, por parte del investigador. Dichos datos son los que se deben tener siempre en cuenta ya que son los que el investigador necesita para desarrollar su actividad. Una vez analizado el script, se ha llegado a la con-

clusión que se deben guardar datos tanto para los contadores como para las métricas.

Para los counter, que son los contadores hardware de los kernel, se guardaran los siguientes datos:

- Eventos. Es la cantidad de eventos que puede realizar el procesador. Eventos como número de instrucciones ejecutadas, fallos de caché, entre otros.
- Tiempo. Calcula el tiempo que se tarda en recopilar todos los datos, dentro de la ejecución del experimento.
- Region Label. Es el nombre que se le asigna a una región especifica, en este caso el grupo de kernels analizados.

En definitiva los contadores hardware [8] son componentes físicos de un procesador que se utilizan para realizar seguimiento y contar eventos específicos del rendimiento de un kernel.

Por otro lado, para las métricas, que son las métricas de datos de cada Thread de la máquina analizada, utilizadas para analizar el comportamiento y rendimiento de los threads analizados. De este grupo, se guardarán los siguientes datos:

- Eventos. Cantidad de eventos que puede realizar el thread analizado.
- Region Label. Es el nombre que se le asigna a una región especifica, en este caso el grupo de threads analizados.
- Data Size. Cantidad de datos dentro del thread.
- Threads. Numero de threads dentro de la máquina analizada.

Una vez determinados cuales son los datos mas importantes, se ha pasado a estudiar la posibilidad de optimizar los Scripts para hacerlos de una menor extensión. Se han realizado diferentes acciones dentro del script, como fusión de bucles y eliminación de algunos datos que podían ser irrelevantes para el resultado de la ejecución del script, y se han hecho pruebas con las distintas versiones que se han creado. Al terminar todas estas pruebas se comprobó, que si bien es cierto que el tiempo de creación del script mejora (ya que contiene menos datos), la ejecución del experimento devuelve datos que no son los esperados. Por este motivo lo que se ha decidido es extraer todas las partes de código que hay comentadas (ya que no son necesarias para la ejecución del trabajo) y dejar el resto de código funcional tal y como Suren estableció. Gracias ha esto se ha creado un script con menos información, ya que solo está la necesaria para la ejecución del script, y que genera el resultado esperado para el trabajo de Suren.

## 4.2. Diseño del programa

La idea principal de este trabajo final de grado es que un usuario pueda obtener un script de forma automática. Para ello se creará un programa que nos permita hacer esto. La idea es que el usuario pueda obtener un script personalizado a partir de los datos que este dispone. Por este motivo, el usuario le proporcionara los datos para la ejecución de su experimento al programa y este devolverá un archivo con el script listo para usar.

Para la creación del programa se han estudiado varias opciones. Primero surgió la idea de crear un programa en C al cual se le introducirían los datos a través de comandos, pero esta idea se descartó debido a la complejidad que supondría para el usuario usarlo. La segunda opción fue el mismo programa en C, pero esta vez introduciendo los datos a través de un fichero JSON. Esta opción gustó debido a la facilidad que tendría el usuario para crear su fichero JSON con los datos para despues enviárselo al programa. Para finalizar se propuso una tercera opción, que consiste en crear una aplicación que consta de dos partes. La primera es Jenkins el cual se encargara de la introducción de los datos y ejecución del código que crea el script. La segunda parte es el programa en java que crea el script. Después de analizar las tres propuestas, se ha tomado la decisión de implementar la tercera opción, en virtud de las capacidades que Jenkins proporciona, además de ser la alternativa más amigable para el usuario entre las tres.

Jenkins[9] es un servidor de automatización de código abierto y herramienta de integración continua (IC)[10], que se puede utilizar para automatizar todo tipo de tareas. Se puede instalar a través de paquetes de sistema nativos, como Docker[11], o se puede ejecutar de manera independiente en cualquier máquina con un entorno de ejecución Java. Sigue un modelo de disparador-acción, es decir, cuando se produce un cambio en el código fuente del software, Jenkins se activa y comienza a ejecutar unas acciones predefinidas. Pero su funcionalidad no queda ahí, también puede integrarse con otras herramientas de desarrollo, como sistemas de control de versiones, herramientas de prueba y gestión de proyectos. En definitiva, Jenkins es una herramienta de IC con la cual se puede automatizar la construcción, prueba y despliegue de software.

Para terminar de entender el funcionamiento de Jenkins se propone el siguiente flujo de integración continua (Figura flujo jenkins) como ejemplo[12]:

- Un desarrollador hace un commit de código en el repositorio.
- El servidor Jenkins realiza comprobaciones periódicas para detectar cambios en el repositorio.
- Al poco de realizarse el commit, Jenkins detecta los cambios y compila el código para proceder a preparar una build. Si la creación de esta builds es exitosa, se despliega en el servidor donde se quiere ejecutar el código. Por el contrario, si la creación falla, se envía una notificación de error.
- Al finalizar la prueba, Jenkins genera feedback y notifica a los desarrolladores la build y los resultados.

Gracias a las herramientas que ofrece Jenkins, se propone la siguiente solución al problema planteado en este TFG:

> La interfaz de Jenkins presenta una notable facilidad de uso, lo que permite al usuario introducir los datos necesarios para crear los scripts. Esta metodología agiliza y simplifica significativamente la introducción de parámetros, ya que estos son introducidos, mediante un formulario, al momento en el que el usuario ejecuta

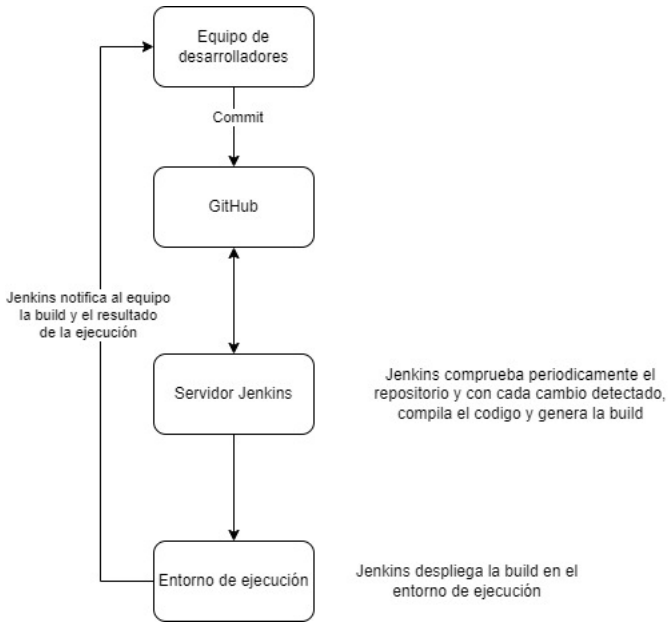


Fig. 1: Diagrama de flujo de Jenkins

el programa de creación de scripts, desde Jenkins. Por otro lado, en Jenkins es necesario configurar un pipeline que detalle de manera secuencial todas las ejecuciones e instrucciones requeridas, es decir, la descarga del programa de creación de scripts desde git, la compilación y la ejecución del mismo para obtener el archivo y el envió de la notificación al usuario.[5].

Para esta propuesta, el funcionamiento de Jenkins es muy sencillo. (Figura 2) :

- Jenkins se encontrará instalado en el clúster de investigación y se le asignará una URL y un puerto.
- Mediante la URL y puertos asignados, desde un ordenador externo al clúster, el investigador podrá acceder a la consola de Jenkins. En dicha consola, el investigador, podrá acceder al programa donde introducirá los datos que desea en su script y lo podrá ejecutar.
- Una vez el usuario a dado la orden de ejecutar, Jenkins comenzará a ejecutar, los pasos que se le han definido. Para empezar descargara el código, que crea el script, del repositorio git donde esta alojado. Una vez que el código esta descargado, pasa al siguiente paso, que es compilar y ejecutar (con los datos previamente introducidos por el usuario) el código descargado, para poder obtener el script con los datos deseados. Una vez que el código se ha ejecutado y ya esta el script creado, Jenkins notificará, mediante email, al usuario que la ejecución a terminado y ya tiene el archivo disponible. En caso que la ejecución haya fallado, el usuario también recibirá un correo pero, en este caso, informando de que la ejecución ha sido errónea.

Como resultado, se ha desarrollado una aplicación en la cual Jenkins asume la responsabilidad de ejecutar el programa de generación de scripts y configurarlo utilizando los datos suministrados. Por otro lado, el programa de generación de scripts, implementado en Java, se encarga de crear el archivo correspondiente basándose en los datos recibidos a través de Jenkins.

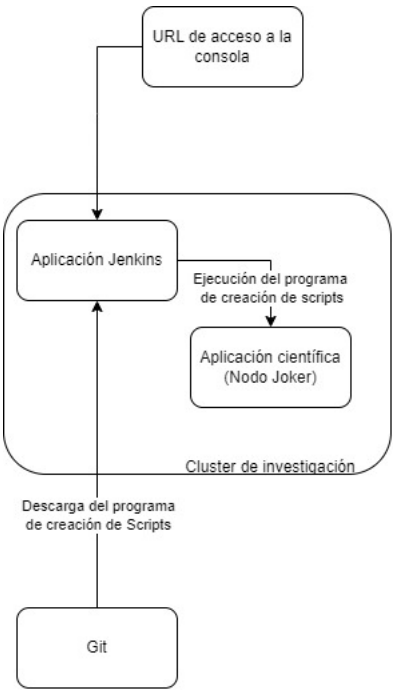


Fig. 2: Diagrama de flujo de la App

## 4.3. Prototipo de la aplicación

Para este prototipo se necesita un servidor Jenkins y el código Java encargado de crear el archivo. Para ello debemos instalar Jenkins en el servidor donde queremos trabajar, y crear el programa Java que genere el script.

Para comenzar con este proceso, lo primero que se hace, es crear el prototipo de la aplicación en una máquina virtual (MV), con el objetivo de asegurar su funcionamiento. Por lo tanto, Se ha creado una MV capaz de alojar el servidor jenkins. Para este propósito, se ha utilizado el programa VirtualBox [13] donde se ha creado una máquina con la distribución e interfaz gráfica de Ubuntu. Se ha decidido utilizar Ubuntu debido a que es una de las distribuciones más usadas, pero también serviría Debian o cualquier otra distribución. Además, se emplea una interfaz gráfica para configurar Jenkins, ya que es necesario acceder a la consola de la herramienta a través de un navegador web. Por último, se deben tener en cuenta unos requisitos mínimos, que Jenkins exige. Dichos requisitos son los siguientes [9]:

- 256 MB de RAM.
- 1 GB de disco.
- Java instalado.

Teniendo en cuenta estos requisitos, se pasa a crear la máquina con la siguiente configuración:

- 8,5 GB RAM. Aunque se necesitan solo 256 MB, se ha creado con esta cantidad debido a que es la configuración por defecto que VirtualBox propone. Por tanto, dado que no afecta al funcionamiento general, se ha optado por mantener esta configuración sin realizar modificaciones adicionales.
- 30 GB Disco.

```
curl -fsSL https://pkg.jenkins.io/debian-stable/jenkins.io-2023.key | sudo tee \
  /usr/share/keyrings/jenkins-keyring.asc > /dev/null
echo deb [signed-by=/usr/share/keyrings/jenkins-keyring.asc] \
  https://pkg.jenkins.io/debian-stable binary/ | sudo tee \
  /etc/apt/sources.list.d/jenkins.list > /dev/null
sudo apt-get update
sudo apt-get install jenkins
```

Fig. 3: Comandos instalación Jenkins

Una vez creada la máquina, se pasa a instalar Jenkins en el sistema. Para ello se siguen las instrucciones del manual [9] y se utilizan los siguientes comandos (Figura 3):

Con los dos primeros comandos (curl y echo) se obtienen los archivos necesarios para Jenkins y se crea el directorio dedicado en el sistema. Con las últimas dos instrucciones se actualizan los paquetes existentes y se instala Jenkins.

Una vez instalado Jenkins, se pasa a instalar la versión 11 de Java. Para ello se siguen las instrucciones proporcionadas por el manual [9] (Figura 4):

```
sudo apt update
sudo apt install openjdk-11-jre
```

Fig. 4: Comandos instalación Java

Terminada la instalación de Java ya se tiene todo listo para poder empezar a trabajar con Jenkins. Para ello lo primero que se debe hacer es activar el servicio, por lo tanto, se ejecuta la instrucción "*sudo systemctl enable Jenkins*" para activar Jenkins y acto seguido se ejecuta el comando "*sudo systemctl start Jenkins*" para arrancarlo.

Hecho esto, se procede a desbloquear Jenkins, debido a que este lo está al principio. Para ello se accede a la consola, a través de la URL http://localhost:8080 y aparece la siguiente ventana (Figura 5):

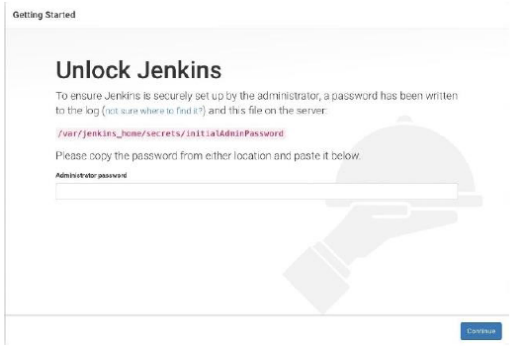


Fig. 5: Ventana desbloqueo de Jenkins

En esta ventana es donde debe introducirse la clave de desbloqueo. Para obtenerla se ejecuta el comando " *sudo cat /var/lib/jenkins/secrets/initialAdminPassword*" dicho comando mostrará por pantalla la contraseña que debe usarse, la cual tendrá un formato como el que puede verse en la siguiente imagen de ejemplo (Figura 6):

Al ingresar la contraseña, Jenkins se desbloqueará y se mostrará la ventana de selección de plugins, donde se instalarán automáticamente los plugins sugeridos.

Para finalizar se pasa a la página de creación de usuario, donde se creará el primer usuario admin, proporcionando

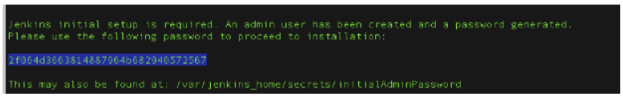


Fig. 6: Contraseña desbloqueo de Jenkins

el nombre de usuario, contraseña y mail asociado. Una vez hechos todos los pasos, ya podremos entrar a la consola de Jenkins para poder comenzar a trabajar (Figura 7):

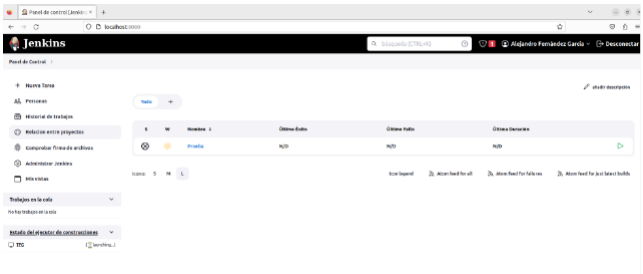


Fig. 7: Consola Jenkins desde la máquina Servidor

La elección de utilizar Jenkins se basó en su capacidad de acceso desde una máquina externa. Por lo tanto, es necesario configurar la máquina virtual para permitir el acceso desde un ordenador personal, en este caso, el mío. Para lograrlo, se debe cambiar la configuración del adaptador de red de la máquina virtual, pasando de un adaptador de red NAT [15] a un adaptador de red en puente [14]. Esta configuración permite que la máquina virtual se conecte a la red a través de la tarjeta de wifi del portátil que la aloja, lo que la hace visible desde la máquina remota (es decir, el mismo portátil, pero desde el sistema operativo nativo Windows).

Una vez realizado este cambio, ahora es posible acceder a la consola desde fuera del servidor. Mediante el uso del nombre de la máquina y el puerto correspondiente, se puede acceder a la consola desde otra máquina conectada a la misma red. En este caso, se utiliza .ªlejandro-virtualbox:8080"para acceder y mostrar la ventana de inicio de sesión de la consola (Figura 8):

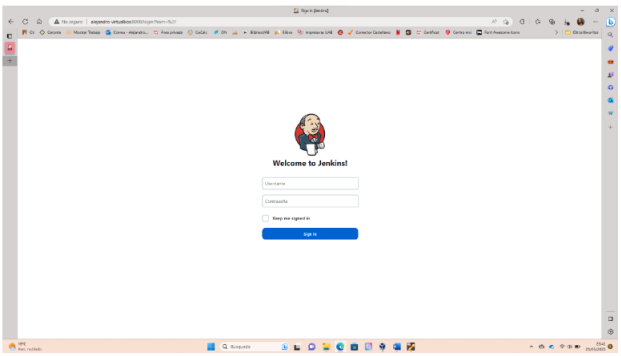


Fig. 8: Consola Jenkins desde máquina remota

Una vez esta Jenkins instalado, se crea la pipeline y se configura para que sea capaz de descargar el programa de creación de scripts y ejecutarlo con los datos que se han introducido.

Dentro de la interfaz de Jenkins, se encuentra la opción de crear una nueva tarea a través de la consola. Dentro de las distintas opciones disponibles (Figura 9), se selecciona la opción de pipeline.

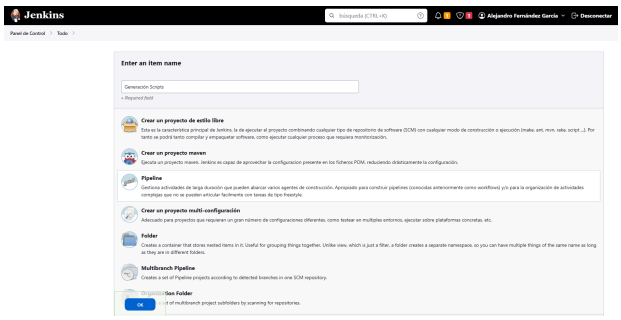


Fig. 9: Lista de tipos de tareas disponibles

Una vez creada la pipeline, se accede a la ventana de configuración específica para dicha pipeline (Figura 10). En esta ventana se realiza la configuración relacionada con la ejecución de la aplicación. Aquí se definen todos los parámetros de datos que el usuario debe completar, así como las acciones que Jenkins debe llevar a cabo.

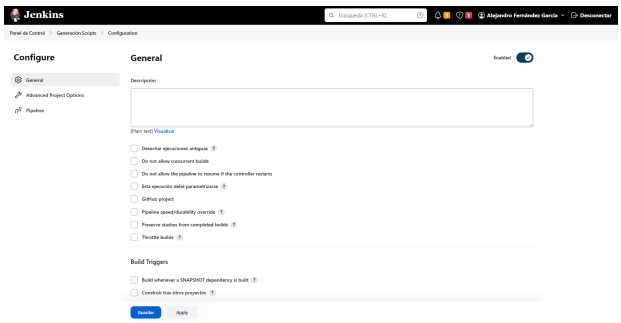


Fig. 10: Ventana de configuración de la pipeline

Para comenzar, se crean los parámetros que el usuario deberá completar con los datos deseados. En este caso, los parámetros son Threads, Size, Kernels y nTimes. Para crear estos parámetros, se debe marcar la casilla de verificación .Esta ejecución debe parametrizarse.en la configuración de la pipeline. Una vez activada, se puede seleccionar el tipo de parámetro, en este caso çadena”(string), y se definen el nombre y el valor por defecto. Una vez hecho esto, la ventana de configuración debe verse de la siguiente forma (Figura 11):

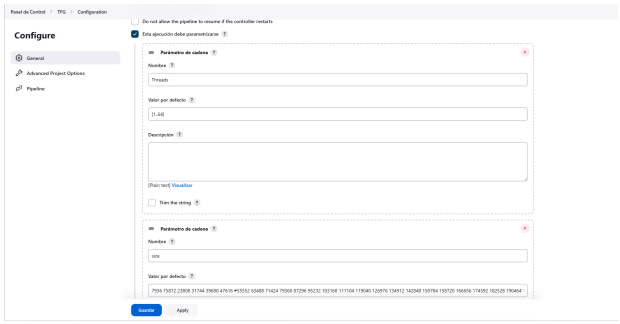


Fig. 11: Vista de la ventana de configuración de la pipeline con los parámetros creados.

Una vez establecidos los parámetros, se procede a definir las acciones que Jenkins debe realizar. Para ello, se crea un código JSON que contiene todos los pasos necesarios que Jenkins seguirá para ejecutar el programa de creación de scripts. Este código se compone de 3 etapas o acciones, que incluyen la descarga del código desde Git, la compilación y ejecución del mismo, y la notificación por correo electrónico sobre si el resultado es satisfactorio o no. En las siguientes imágenes se pueden visualizar las diferentes etapas:

```
stages {
    stage('Prepare program') {
        steps {
            echo 'Initiating prepare tests...'
            script {
                //URL de Git donde se encuentran los tests
                def gitUrl = 'https://github.com/AF1489848/TFG.git'
                def branch = 'main'

                //Descargamos las fuentes de los tests
                git branch: branch, url: gitUrl


            }
            echo 'End of prepare tests'
        }
    }
```

Fig. 12: Descarga del programa de GitHub

En el stage ‘Prepare program’ (Figura 12), se descarga el programa de generación de scripts, de un repositorio de GitHub para poder ejecutarlo en el siguiente stage.

```
stage('Execute'){
    steps {
        echo 'Initiating Execute...'
        script {
            // Ejecutamos los tests
            try {
                sh "mvn clean verify -DThreads=\"${Threads}\" -Dsize=\"${size}\" -DKernels=\"${Kernels}\"
                -DnTimes=\"${nTimes}\" " +
                "-DThreads2=\"${Threads2}\" -Dsize2=\"${size2}\" -DKernels2=\"${Kernels2}\" -DnTimes2=\"${nTime
                echo 'Script Creado Satisfactoriamente'
                currentBuild.result = 'SUCCESS'
            } catch (err) {
                echo 'Fallo en la creación del Script'
                currentBuild.result = 'FAILURE'
            }
        }
        echo 'End of Execute'
    }
```

Fig. 13: Ejecución del programa

En el stage ‘Execute’ (Figura 13) se ejecuta el programa de generación de scripts. Como se puede observar, es en este momento donde se añaden los datos que el usuario habrá introducido mediante el formulario. Es por ello que se utiliza la instrucción mvn junto con los parámetros.

```
stage('Notify'){
    steps {
        echo 'Initiating Notify...'
        script {
            def script = 'target/job_script.sub'
            archiveArtifacts artifacts: script

            //Send email
            def subject = "Creación de Script: ${currentBuild.result}"
            def body = "El resultado de la creación del Script es el siguiente: ${currentBuild.result}"

            emailext(
                    attachmentsPattern: script,
                    mimeType: 'text/html',
                    subject: subject,
                    body: body,
                    to: "alejandro.fernandezgar@autonoma.cat",
                    from: "no-reply@vjenkdev.tfg.com"
            )
        }
        echo 'End of Notify'
    }
```

Fig. 14: Envío de la notificación

Para terminar en el stage ‘Notify’ (Figura 14), se define el cuerpo y asunto del mail, además del destinatario, y se envía.

Una vez configurada la parte de Jenkins, se procede a la siguiente etapa de la aplicación, que consiste en crear el programa de generación de scripts. Este programa, escrito en Java, se encarga de crear los scripts utilizando una plantilla base y los datos proporcionados a través de Jenkins. El programa se divide en tres partes principales (Figura 15):

– La clase Main, que se encarga de ejecutar todo el proceso de creación del script.

– La clase ScriptSlurm, que se encarga de crear el script.

– La clase TestMain, que es un test que recibe la orden de ejecución desde Jenkins y, por lo tanto, ejecuta todo el programa a través de Main.

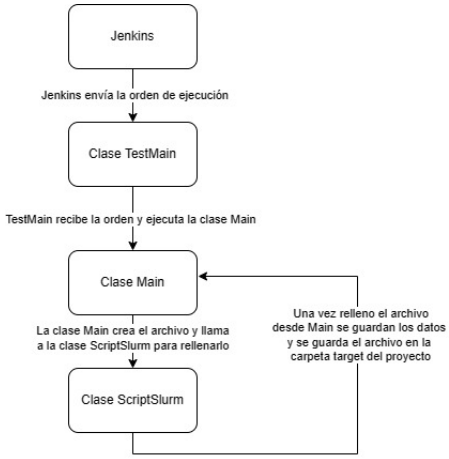


Fig. 15: Diagrama de flujo de la ejecución del programa de creación de scripts

Todo el código de estas tres clases se puede consultar en el Apéndice A.1

Una vez finalizada esta segunda parte, la aplicación para la creación automática de scripts está completa. Por un lado, Jenkins se encarga de la configuración del script (los datos que se desean) y de la ejecución del programa que lo genera. Por otro lado, el programa de creación de scripts utiliza los datos recibidos de Jenkins para generar el archivo.

Después de completar las pruebas y verificar que todo funciona correctamente, procedemos a realizar la instalación en el clúster Wilma a través del nodo de inicio aoclspv.uab.cat. Este nodo cuenta con 2 procesadores Intel(R) Xeon(R) CPU 5160 @ 3.00GHz, 2 cores por procesador y 16 GB de memoria. Además de este nodo, la máquina también incluye un servidor IBM x3650 con un procesador Intel Xeon 5150 2.66GHz, 2 cores, y otros 12 nodos Dell PowerEdge R415 con 2 procesadores AMD Opteron 4180, 6 cores por procesador y 8 GB de memoria cada uno.

Finalmente, se ha optado por instalar la aplicación en esta máquina debido a las condiciones particulares del clúster de investigación, que hacen que sea imposible abrir puertos para acceder a Jenkins desde el exterior.

Por lo tanto, se procede a instalar la aplicación siguiendo los mismos pasos que en la máquina virtual. Una vez completada la instalación, la aplicación está lista para ser utilizada en su totalidad.

## 5 Resultados

Una vez finalizado el desarrollo, se ha obtenido una aplicación que permite generar scripts funcionales de manera eficiente, evitando la necesidad de dedicar tiempo a realizar modificaciones manualmente entre cada prueba. Antes de la existencia de este proyecto, Suren tenía que acceder al archivo y modificar los parámetros uno por uno, buscándolos en el script. Esto implicaba una pérdida considerable de tiempo entre ejecuciones y la posibilidad de cometer errores al omitir alguna modificación.

La nueva aplicación resuelve este problema. Gracias a ella, es posible introducir rápidamente todos los datos necesarios de forma segura, sin tener que navegar a través del script. Al ingresar a la pipeline y ejecutar la aplicación, lo primero que se muestra es el siguiente formulario (ver Figura 16)

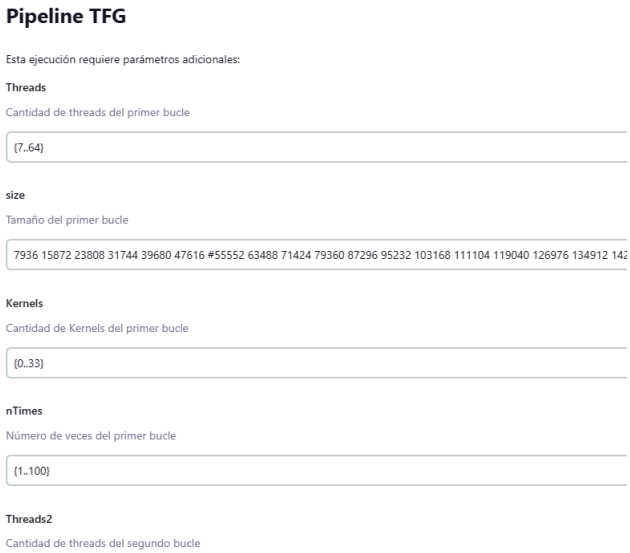


Fig. 16: Formulario para introducir los datos del script

Como se muestra en la Figura 16, se utiliza un formulario para ingresar todos los datos necesarios. Por defecto, aparecen los datos acordados con Suren, ya que son los más utilizados y no es necesario introducirlos repetidamente. Sin embargo, se pueden modificar y agregar otros datos según sea necesario antes de la ejecución.

Una vez iniciada la ejecución, Jenkins lleva a cabo todas las tareas definidas en la pipeline. Durante la ejecución, es posible monitorizar el progreso paso a paso que Jenkins está realizando. Una vez finalizada la ejecución, se puede acceder al archivo generado mediante la revisión de la ejecución correspondiente. Además, se envía una notificación por correo electrónico para informar sobre el resultado de la ejecución.

Gracias a esta aplicación, se ha logrado reducir significativamente el tiempo necesario para configurar un script. Lo que antes llevaba varios minutos, ahora se realiza en tan solo unos segundos.

## 6 Conclusiones

Durante el desarrollo de este trabajo, se han analizado diversas opciones para abordar la problemática planteada. Este proceso ha permitido encontrar una solución óptima que se adapta de manera efectiva a las necesidades planteadas. Mediante el uso de Jenkins, se ha logrado desarrollar una aplicación versátil que ofrece numerosas posibilidades, más allá de su aplicación específica en este proyecto.

El resultado obtenido es una aplicación funcional y amigable para el usuario, que ha logrado reducir significativamente el tiempo necesario para modificar la configuración de un script entre diferentes ejecuciones. Esta herramienta

ha demostrado su eficacia al agilizar el proceso y eliminar la necesidad de realizar modificaciones manuales laboriosas.

A pesar de las ventajas que ofrece esta aplicación, es importante destacar que también presenta ciertas limitaciones. Si se requiere realizar modificaciones sustanciales en las instrucciones que el script ejecuta, será necesario realizar cambios en la base del programa de creación de scripts. Esto es debido a que la aplicación se centra principalmente en la configuración de los datos utilizados, sin modificar directamente instrucciones del script.

Una posible área de mejora futura sería adaptar la aplicación para que también permita una modificación sencilla de las instrucciones del script. De esta manera, se brindaría una mayor flexibilidad al usuario y se ampliarían los usos de la aplicación.

Otra mejora que podría considerarse es la posibilidad de que la aplicación, además de crear el script, también lo ejecute. Jenkins ofrece soporte para ello, lo que permitiría realizar ambas tareas de configuración y ejecución del script de manera simultánea.

Asimismo, sería viable configurar la aplicación para que genere y ejecute el script de forma periódica, por ejemplo, semanalmente. Esta función sería especialmente útil cuando se requiere ejecutar el mismo experimento de forma regular.

En resumen, se ha logrado desarrollar una aplicación que cumple de manera satisfactoria con el objetivo planteado en este trabajo, que es la creación automática de un script con una configuración determinada. Esta herramienta ha demostrado su utilidad y efectividad en la optimización del tiempo y la simplificación del proceso de configuración, sentando las bases para posibles mejoras y aplicaciones futuras en este ámbito.

## AGRADECIMIENTOS

Me gustaría darle las gracias a mi tutora Ania Sikora, por toda la ayuda que me ha brindado. A mi familia y a mi pareja por el apoyo que he recibido de ellos durante todos estos meses. Por último a Suren por haber ayudado a que entendiera todo el script en el que se basa este trabajo de fin de grado.

## REFERENCIAS

[1] "¿Qué es HPC? Introducción a la computación de alto rendimiento — IBM," Ibm.com, 2023. https://www.ibm.com/es-es/topics/hpc (accessed Mar. 09, 2023).

[2] "IBM Documentation," Ibm.com, Mar. 2021. Available: https://www.ibm.com/docs/es/power8?topic=memory-shared. [Accessed: Jun. 24, 2023]

[3] "Definición de Memoria Compartida Distribuida — mcd-unerg," mcd-unerg, Jan. 18, 2009. Available: https://mcd-unerg.blogia.com/2009/011804-definici-n-de-memoria-compartida-distribuida.php. [Accessed: Jun. 24, 2023]

[4] "Slurm Workload Manager - Quick Start User Guide," Schedmd.com, 2021. Available: https://slurm.schedmd.com/quickstart.html. [Accessed: Mar. 09, 2023]

[5] A. de, "Departament d'Arquitectura de Computadors i Sistemes Operatius - UAB Barcelona," Uab.cat, 2021. [Online]. Available: https://www.uab.cat/departament/arquitectura-computadors-sistemes-operatius/ [Accessed: Mar. 10, 2023]

[6] A. de, "Escola d'Enginyeria - UAB Barcelona," Uab.cat, 2023. [Online]. Available: https://www.uab.cat/escola-enginyeria/ [Accessed: Mar. 10, 2023]

[7] BBVA, "Metodología 'Agile'. La revolución de las formas de trabajo," BBVA NOTICIAS, Nov. 20, 2018. [Online]. Available: https://www.bbva.com/es/metodologia-agile-la-revolucion-las-formas-trabajo/ [Accessed: Mar. 10, 2023]

[8] [1] DOMARS, "Acerca de los contadores de rendimiento del modo kernel - Windows drivers," Microsoft.com, Jun. 15, 2023. Available: https://learn.microsoft.com/es-es/windows-hardware/drivers/devtest/about-kernel-mode-performance-counters. [Accessed: Jun. 25, 2023]

[9] "Jenkins User Documentation," Jenkins User Documentation, 2023. https://www.jenkins.io/doc/ (accessed Apr. 19, 2023).

[10] Atlassian, "¿En qué consiste la integración continua? — Atlassian," Atlassian, 2023. Available: https://www.atlassian.com/es/continuous-delivery/continuous-integration. [Accessed: Jun. 25, 2023]

[11] "Contenedores de Docker — ¿Qué es Docker? — AWS," Amazon Web Services, Inc., 2023. Available: https://aws.amazon.com/es/docker/. [Accessed: Jun. 25, 2023]

[12] Sentrio, "Introducción a Jenkins: ¿qué es, para qué sirve y cómo funciona?," Sentrio, Sep. 16, 2021. https://sentrio.io/blog/que-es-jenkins/ (accessed Apr. 23, 2023).

[13] "Oracle VM VirtualBox," Virtualbox.org, 2023.https://www.virtualbox.org/ (accessed May 23, 2023).

[14] Oracle and/or its affiliates, "Chapter 6.3 Virtual Networking," www.virtualbox.org, 2023. `https://www.virtualbox.org/manual/ch06.html#network_bridged` (accessed May 28, 2023).

[15] Oracle and/or its affiliates, "Chapter 6. Virtual Networking," www.virtualbox.org, 2023. `https://www.virtualbox.org/manual/ch06.html#network_nat` (accessed May 28, 2023).

[16] A. Rodriguez, "Towards Automatic Generation of Performance Models for Dynamic Tuning using Machine Learning," Tdx.cat, 2021, doi: http://hdl.handle.net/10803/675104

# APÉNDICE

## A.1. Código del programa de creación de scripts

```
import org.junit.jupiter.api.Test;

AF1489848
public class TestMain {
    AF1489848
    @Test
    public void testMain(){
        Main.main();
    }
```

Fig. 17: Clase testMain

```
feralejandro +1
public class Main {
    1 usage   feralejandro +1
    public static void main() {
        System.out.println("Hello world!");


        String scriptPrefix = "job_script"; // Prefijo para los nombres de los scripts
        String outputDirectory = "./target"; // Directorio de salida para los scripts


        String scriptName = scriptPrefix + ".sub";
        String scriptPath = outputDirectory + "/" + scriptName;
        ScriptSlurm.ScriptGeneration(scriptPath);

    }
```

Fig. 18: Clase Main

```
import java.io.BufferedWriter;
import java.io.FileWriter;
import java.io.IOException;

1 usage   AF1489848
class ScriptSlurm {
    1 usage   AF1489848
    public static void ScriptGeneration(String scriptPath) {
        try (BufferedWriter writer = new BufferedWriter(new FileWriter(scriptPath))) {
            writer.write( str: "#!/bin/bash -l");
            writer.newLine();
            writer.newLine();
            writer.write( str: "#SBATCH --job-name=\"AC-red\"\n" +
                    "#SBATCH -w stanlee\n" +
                    "#SBATCH --exclusive\n" +
                    "#SBATCH --time=1200:00:00\n" +
                    "#SBATCH --nodes=1\n" +
                    "#SBATCH --ntasks=1\n" +
                    "#SBATCH --cpus-per-task=128\n" +
                    "#SBATCH --output=stanlee-red-OUT-%j.log\n" +
                    "#SBATCH --error=stanlee-red-ERR-%j.log");
            writer.newLine();
            writer.newLine();
            writer.write( str: "#module load gcc/7.3.0n\n" +
                    "module load gcc/9.2.0\n" +
                    "module load likwid/5.1.2\n" +
                    "module load openmpi/3.0.4\n" +
                    "module load conda");
            writer.newLine();
            writer.newLine();
            writer.write( str: "cd\n" +
                    "source $HOME/.bashrc\n" +
                    "source $HOME/tools/spack/share/spack/setup-env.sh\n" +
                    "#spack load python arch=linux-centos7-sandybridge\n" +
                    "spack load python arch=`spack arch`");
            writer.newLine();
            writer.newLine();
            writer.write( str: "#mkdir openmp");
            writer.newLine();
            writer.write( str: "cd /home/suren/legacy_tools/stream_modified/amd_stanlee");
```

Fig. 19: Extracto de la clase ScriptSlurm