**Universidad Complutense de Madrid**

Facultad de Informática


# Aceleración Hardware en entornos de Ciberseguridad

Trabajo de Fin de Grado

Directores:

**Alberto Antonio del Barrio García**

**Guillermo Botella Juan**

Autores:

**Jorge Burgaleta Lobejón**

**Luis Muñoz María**

**Diego Maestre Vidal**

Curso: 2018/2019

# Agradecimientos

En primer lugar, nos gustaría agradecer a nuestros dos directores, Alberto del Barrio y Guillermo Botella, por todo el apoyo y esfuerzo que nos han ofrecido desde el inicio del proyecto, ya que, gracias a la confianza que nos dieron desde el inicio para poder realizar el trabajo con ellos, hemos logrado superar momentos difíciles donde nos han dirigido hacía la mejor dirección. En segundo lugar, queremos agradecer a todas aquellas personas, tanto familia como amigos, que nos han ayudado a nivel personal a superar esta difícil etapa.

Por último, agradecemos a la Facultad de Informática de la Universidad Complutense de Madrid y a todo su personal, por los esfuerzos que han puesto a lo largo de todos estos años para que nuestro aprendizaje se realice de la mejor manera posible, ofreciéndonos grandes facilidades a la hora de poder realizar nuestras tareas diarias dentro de la facultad. Además, queremos agradecer al Departamento de Arquitectura de Computadores y Automática (DACYA) que nos ha prestado las placas de desarrollo con las que hemos realizado este trabajo.

# Resumen

En este trabajo presentamos la construcción con Systems on a chip (SoCs) de un clúster de bajo consumo de energía y coste, que ejecuta un algoritmo de aprendizaje sobre el dataset introducido.

Actualmente existen diferentes tipos de clúster, en nuestro caso estará formado por SoCs de bajo consumo y coste. Las placas utilizadas son DE1-SOC de Altera, en este tipo de SoC se pueden introducir diferentes sistemas operativos. Para este proyecto hemos decidido usar la imagen que tiene la versión más reciente del kernel de Linux que es la 4.5.

Para nuestro trabajo vamos a ejecutar un algoritmo de aprendizaje automático, en concreto SVM sobre un dataset llamado KDDCUP`99 el cual contiene un análisis del tráfico en distintas redes en las que se han llevado ataques y otras en las que es un tráfico normal. Para esto nos ayudaremos de la librería libsvm la cual es una implementación en distintos lenguajes de una SVM, en nuestro caso en concreto usaremos la versión en C++ para realizar el entrenamiento de SVM en el procesador ARM de las placas DE1-SOC.

Finalmente, hemos procedido a hacer una comparativa de los datos de tiempo empleado en la ejecución y tasa de acierto que hemos logrado desde la utilización de una Workstation de altas prestaciones hasta la utilización del clúster con 6 nodos, donde hemos podido comprobar que la capacidad de acierto es mayor en la ejecución de la Workstation, pero sin embargo la ganancia de tiempo que logramos con el clúster de 4 ó 5 nodos hace que sean mejores opciones.



**Palabras Claves:** FPGA, Clúster, C, Workstation, Nodo, SVM, SoC

# Abstract

In this work we present the construction with Systems on a chip (SoCs) of a cluster of low energy consumption and cost, which executes a learning algorithm on the introduced dataset.

Currently there are different types of clusters, in our case it will be formed by SoCs of low consumption and cost. The boards used are Altera DE1-SOC, in this type of SoC different operating systems can be introduced. For this project we have decided to use the image that has the most recent version of the Linux kernel that is 4.5.

For our work we are going to execute an automatic learning algorithm, specifically SVM on a dataset called KDDCUP`99 which contains an analysis of the traffic in different networks in which attacks have been carried out and others in which it is a normal traffic. For this we will use the libsvm library which is an implementation in different languages of an SVM, in our case in particular we will use the C ++ version to perform SVM training in the ARM processor of the DE1-SOC boards.

Finally, we have proceeded to make a comparison of the time data used in the execution and success rate that we have achieved from the use of a high performance Workstation to the use of the cluster with 6 nodes, where we have been able to verify that the capacity of success is greater in the execution of the Workstation, but nevertheless the gain of time that we achieved with the cluster of 4 or 5 nodes makes them better options.



**Keywords**: FPGA, Cluster, C, Workstation, Node, SVM, SoC

# Índice General

# Índice de tablas

# Índice de gráficos

# Índice de imágenes

# Capítulo 1: Introducción

En esta sociedad informatizada nos enfrentamos a innumerables retos diarios para securizar nuestros sistemas. Para ello es necesario seguir investigando, y desarrollando nuevos mecanismos de defensa con la tecnología que tenemos en la actualidad.
Por ello hemos decidido continuar la experimentación de un proyecto creado por Mariano Hernández García [1], basando nuestra idea en su clúster que empleaba DE1-SoC para llevar a cabo una herramienta de seguridad de bajo coste.
La realización de este trabajo ha requerido revisar diferentes datasets con tráfico no legítimo y a su vez, las técnicas más utilizadas por ciberdelincuentes.

## 1.1 Objetivos

La solución común para evitar la fuga de datos en cualquier red son los Firewalls y en redes extensas se utilizan en forma de clúster, pero el problema que tiene es que muy costoso.

Nuestro objetivo principal es la división de la carga de trabajo de un SVM a las distintas SoCs (DE1-SOC) [2] y ser capaces de identificar si se ha llevado a cabo un ataque o no, para ello usaremos el dataset KDDCUP '99 el cual divide el tráfico en tráfico normal, o ataque y a su vez categoriza dichos ataques en DDos, probing, U2R y R2L.

El trabajo se ha dividido de las siguientes formas:

- El nodo maestro se encargará de identificar cada uno de los nodos del clúster y el número total. Para ello tendremos un archivo con cada una de las IPs.

- Dividiremos el archivo de entrenamiento en cada una de las placas, para optimizar el rendimiento.

- Una vez tengamos el entrenamiento, decidiremos si se ha producido un ataque o no.

## 1.2 Organización del proyecto

La memoria la dividimos en 6 capítulos en el que explicamos la investigación y desarrollo llevado a cabo:

- Capítulo 1: Se plantean los objetivos y la organización del trabajo, así como los conceptos básicos para comprenderlo.

- Capítulo 2: En el estado del arte hemos revisado las tendencias de ataques y diferentes medidas de protección.

- Capítulo 3: En este capítulo se lleva a cabo, una explicación detallada del dataset y algoritmos utilizados.

- Capítulo 4: Se presentan la experimentación con sus resultados y la comparativa de los mismos.

- Capítulo 5: El quinto capítulo consta de una lluvia de ideas para un trabajo futuro y sobre el esfuerzo realizado

- Capítulo 6: Anexo donde se puede encontrar los scripts utilizados durante el proyecto.

## 1.3 Motivación

Una de las motivaciones que debemos destacar es el uso de la tecnología por parte de los usuarios que habitualmente tienen unos bajos conocimientos en estas áreas, para este tipo de usuario necesitamos facilitarles unos sistemas que se encarguen de detectar los ataques que están sufriendo y vulnerabilidades que tienen en sus equipos.

Otra motivación importante por la que decidimos adentrarnos a realizar un trabajo de fin de grado sobre ciberseguridad fue debido a la complejidad y diversos avances en todos los ámbitos de la informática. Este avance hacia sistemas mucho más complejos produce que se creen nuevas vulnerabilidades que todavía no han sido descubiertas y por las cuales estos sistemas pueden ser víctimas de ataques informáticos produciendo pérdidas muy importantes de información que conlleva unos perjuicios demasiado altos para los afectados por este tipo de delitos.

Debido a este tipo de avances dentro del campo de la informática provoca que se generen una gran cantidad de información que debe ser procesada por nuestros sistemas informáticos necesitando tener una capacidad de cómputo muy elevada. Anteriormente para conseguir alcanzar la capacidad de cómputo necesaria bastaba con aumentar el número de procesadores, memorias más veloces, uso de algoritmos más eficientes… Actualmente hemos llegado a un punto en el que es necesario ir un paso más adelante debido a que el consumo energético de estos sistemas comienza a ser superior a la capacidad de procesamiento de los datos provocando que no sea eficiente. Para solucionar este tipo de problemas donde queremos que el consumo energético sea el menor posible se ha comenzado a usar la agrupación de sistemas informáticos, a este tipo de agrupación la definimos como clustering. Estos clústeres emplean una programación distribuida cuya finalidad es realizar las tareas en el mínimo tiempo, por ello nos parece muy importante poder generar un sistema que trabaje sobre un clúster para la detección de diferentes ataques, ya que, tendrá la capacidad de detección más rápida en relación con un único equipo con la misma tarea y mismas especificaciones técnicas.

# Capítulo 2: Estado del arte

## 2.1- Ataques DDoS

En la actualidad la utilización de ataques DDoS [3] está siendo muy utilizada por los delincuentes en la red ya que han sufrido un avance técnico que han conseguido que sean muy peligrosos.

La historia de Internet es bastante reciente pero junto a internet siempre han estado los activistas que la han empleado como medio de protesta, uno de los tipos de protesta más utilizados por los activistas ha sido la utilización de ataques de denegación de servicio o DDoS. Los primeros ataques DDoS surgieron a finales de los años 80 en donde cabe destacar el gusano Morris que no era capaz de detectar si un ordenador estaba infectado por el mismo, entonces provocaba que se autocopiase en la misma máquina produciendo un auto ataque DDoS. Otro ataque que cabe destacar es el producido por un grupo activista italiano en 1995 que atacaba al gobierno francés por su política nuclear. Ya en el año 2000 se produjo uno de los ataques más importantes por parte de un niño de 15 años llamado "Mafia Boy" que hizo caer con un ataque DDoS a Amazon, Ebay, Dell e incluso a Yahoo que en ese momento era mayor motor de búsqueda de internet en este caso lo único que quería el atacante era mostrar su poderío de ciberdelincuente a todo el mundo sin ningún interés económico ni político. A partir de este momento los ataques se empezaron a hacer mucho más normales en los que podríamos encontrar Code Red, SQL Slammer, un ataque a Estonia con la primera botnet, opositores rusos realizan un DDoS por la tensión política en Rusia.

En los primeros ataques DDoS, era necesario estar frente al ordenador durante todo el proceso del ataque, además como en esa época las conexiones tenían un coste elevado los ataques tenían una duración media de 1 hora. Un momento que produjo un avance en este tipo de ataques fue con la creación del grupo Electronic Disturbance Theater (EDT) que era un grupo de activista que se distinguía de los surgidos anteriormente porque empleaban herramientas que habían sido desarrolladas por ellos mismos, una herramienta que desarrollaron fue FloodNet que se encargaba de dirigir el tráfico de los usuarios hacia un blanco fijado por EDT, para ello solo tenían que pulsar en un menú desplegable el botón de

“ataque” y FloodNet se encargaba de atacar al servidor, por ellos cualquier persona que quisiera apoyar la “protesta” podría unirse de una manera sencilla. Finalmente apareció el grupo ANONYMOUS que empezó a popularizar el uso de Botnets voluntarios que es un elevado número de sistemas unidos que producen que el atacante tenga un gran poder para realizar sus actividades delictivas.

De esta manera llegamos a lo que conocemos actualmente como un ataque de denegación de servicios o DDoS, como hemos visto los ataques DDoS se han convertido en un tipo de ataque técnicamente simple a nivel tecnológico, ya que, el proceso es transmitir una gran cantidad de información desde varios puntos de conexión hacía un mismo punto de destino. Un servidor, ante la llegada de tanta información, provoca una saturación en el equipo y deja de prestar los servicios, por eso su nombre de Denegación, pues consigue que el servidor no tenga capacidad de responder tantas peticiones al mismo tiempo. La técnica más empleada consiste en la utilización de una red de Bots. Además, con el auge de dispositivos IoT, los cuales, no suelen tener demasiada seguridad y por tanto pueden ser fácilmente usados para la realización de estos ataques por los delincuentes informáticos.

Podemos encontrar tres tipos diferentes de ataques:

- Sobrecarga del ancho de banda

Este ataque se dirige directamente a la red y a sus dispositivos, porque si es capaz el ataque de solicitar toda la cantidad de datos que puede procesar el router, este no puede atender el resto de las peticiones de otros equipos.

- Saturación de recursos del sistema

En este tipo de ataque si se emplean los recursos del sistema, ya que el número de peticiones que puede establecer un servidor web son limitadas, entonces si estas peticiones son ocupadas por peticiones sin sentido o inválidas terminará bloqueando el resto de las peticiones válidas de otros usuarios.

- Uso de fallos de seguridad y software

Si existe algún tipo de vulnerabilidad en el sistema operativo y es descubierta por ciberdelincuentes, puede ser explotada para conseguir errores en el funcionamiento normal del software o fallos en el sistema, un ejemplo sería, cuando los paquetes IP se envían fragmentados y transmiten información errónea, algún sistema operativo puede que sea engañado para que genere paquetes IP mayores de 64KB, produciendo como consecuencia volver a montar el paquete generando un buffer overflow.

Para intentar contrarrestar este tipo de ataques, se ha diseñado una serie de medidas que pueden bloquear el ataque o al menos minimizar sus consecuencias. Por ello, existe una serie de listas negras de IPs, que permiten identificar aquellas direcciones son críticas y el descarte de sus paquetes, esta medida puede ser utilizada a través del cortafuegos. Otra medida que podemos emplear es indicar los límites en la cantidad de datos que son procesados, así bloqueamos todo tipo de paquetes que no sean normales.

Por último, podemos destacar las SYN Cookies que en este caso solo comprometen la capacidad del equipo, pero no la memoria y un ataque SYN Flood no tendría éxito. Esta medida lo que realiza es que los paquetes con el flag SYN activado, no se almacenan en el servidor, sino que se envían al cliente como una cookie encriptada.

## 2.2- Botnets

En la actualidad las amenazas más comunes son DDoS, phishing, envío de spam, minería de Bitcoin…En la mayoría de todos estos tipos de amenazas se encuentra el uso de Botnets, aunque la tendencia habitual es relacionar los botnets directamente con los ataques DDOS pero la realidad es que los botnets pueden ser usados en la mayor parte de los diferentes tipos de ataques informáticos existentes. Los botnets poseen una gran cantidad de características que provocan que sean muy eficaces, entre sus características cabe destacar el anonimato, resiliencia, adaptación y contagio. Otra característica que produce que algunos botnets tengan una gran eficacia, es que existen iniciativas de código abierto provocando que un gran número de expertos colaboren en el desarrollo del código para lograr una mayor perfección, por ejemplo, la SDBot y Agobot, de esta manera se consigue que existan mutaciones de Bots en función de la finalidad para la que vaya a ser empleada.

Con todo esto, es importante que los Botnets no sean asociados a un ataque, sino que los botnets son una “herramienta” que podrá ser empleada en cualquier tipo de ataque informático.

La palabra Botnet surge de la unión de dos palabras “BOT” y “NET”, estas dos palabras se refieren a robot y red informática entonces podemos describir que los botnets son una red de robots (ejército de máquinas infectadas). Mediante la introducción de software podemos infectar una máquina y obtener su control, lo llamaremos “zombie”, cuando conseguimos que el software haya sido introducido en muchas máquinas entonces disponemos de un ejército que pueden realizar acciones coordinadas para atacar al objetivo que ha sido elegido por el dueño del software consiguiendo el anonimato y éxito por parte del “amo” del ejército de zombies.

En definitiva, un atacante puede llegar a tener el control de miles de equipos para realizar sus actividades coordinadas delictivas, para que el resto de las máquinas realicen las actividades, existe una estructura jerárquica donde los encargados de dar las órdenes a las máquinas infectadas(bots) y también reciben la información proveniente de estos bots se llaman servidores de mando y control.

## 2.3- Machine Learning

En la actualidad disponemos de una gran cantidad de datos que provienen de diferentes fuentes, este gran volumen de datos produce que el ser humano no sea capaz de analizarlos por sí mismo teniéndose que apoyar en el uso de las nuevas técnicas que están constituidas por diferentes algoritmos capaces de aprender según se le van introduciendo los datos provocando grandes beneficios, tales como el uso de grandes cantidades de datos ya estén estructurados como no estructurados para ser usados en la toma de decisiones, al conseguir que los procesos de modelaje y autoaprendizaje tengan una capacitación de automatización consiguen que una vez han sido implementados su mantenimiento se reduce de manera que disminuyen los costes y tiempos de modelización posteriores teniendo siempre un modelo con una gran capacidad predictiva.

Para conseguir que el aprendizaje automático logre la función de detectar de manera automática los patrones de datos y poder realizar una predicción de datos futuros o

encargarse de la toma de decisiones en momentos de indecisiones se deben tener en cuenta los diferentes componentes que constituyen este tipo de aprendizaje. Los principales componentes que podemos destacar son los siguientes: el agente (el que aprende o toma decisiones), el entorno (todo con lo que interactúa el agente) y acciones (lo que el agente puede hacer).

Para que el aprendizaje automático tenga una eficacia elevada necesitamos tener en cuenta los siguientes aspectos:

- Fuentes de Información
- Algoritmos para emplear
- Capacidad de Autoaprendizaje
- Sistemas de programación y visualización

En la actualidad el aprendizaje automático tiene una gran capacidad de uso en diferentes áreas como puede ser en la medicina para detección de enfermedades, motores de búsqueda para la recomendación, videojuegos para dar mayor autonomía a los bots, en finanzas para la inversión en mercados bursátiles, en la educación para analizar fortalezas de los estudiantes, reconocimiento de voz e imágenes...

Según la información que empleamos para el aprendizaje podemos encontrar dos técnicas de modelización que son:

- Aprendizaje supervisado: En este aprendizaje se proporciona un conjunto de datos que tiene las entradas y las salidas deseadas, entonces el algoritmo se encarga de la búsqueda de un método para determinar cómo lograr obtener esas entradas y salidas. Este tipo de aprendizaje se emplea en problemas de regresión como predicciones meteorológicas.
- Aprendizaje no supervisado: Este tipo de aprendizaje no dispone de variables output, por lo tanto, en el aprendizaje no supervisado se realiza una búsqueda de patrones o relaciones en los datos útiles y entendibles. Las Redes neuronales son un ejemplo de este modo de aprendizaje.

### 2.3.1-SVM

Las máquinas de vectores soporte [4] SVM, del inglés Support Vector Machines fueron desarrolladas en los años 90 por Vladimir Vapnik y su equipo de colaboradores. En un principio fueron creadas para la resolución de problemas lineales, aunque en este momento se están usando para problemas de clasificación, agrupamiento y regresión.

Este método de aprendizaje consta de una primera fase donde se introducen unos ejemplos de entrenamiento (muestra) estos ejemplos ya resueltos están en forma de pares [problema, solución]. En estos ejemplos tenemos soluciones positivas y negativas para poder generar un modelo predictivo lo suficientemente eficaz. Después existe una segunda fase que se encarga de la resolución de problemas generando una respuesta (Salida) al problema introducido (Entrada).

Las SVM se categorizan como clasificadores lineales, donde buscan un hiperplano que sea capaz de dividir de forma eficiente los puntos, según la clase a la que pertenezcan. Este algoritmo intenta que el hiperplano esté lo más separado posible de los puntos más cercanos a él, por ello, también se les conoce como clasificadores de margen máximo. En algunos casos es muy difícil crear hiperplano N-dimensional, ya que en la mayoría de los casos del mundo real no se pueden dividir dos dimensiones ideales. Para todos estos casos que tienen gran complejidad empleamos las funciones Kernel que aumentan la capacidad computacional de las actuales máquinas de aprendizaje lineal creando un espacio de mayor dimensionalidad.

### 2.3.2- Redes Neuronales

Las redes neuronales comenzaron a principios de los años 40 gracias a Donald Hebb que creó lo que actualmente conocemos como aprendizaje de Hebb que es un tipo de aprendizaje no supervisado que se basa en la plasticidad neuronal. Las redes neuronales intentan imitar el cerebro humano para la resolución de problemas.

Las redes neuronales son un sistema de computación compuesto por un gran número de elementos simples, elementos de procesos muy interconectados, los cuales procesan información por medio de su estado dinámico como respuesta a entradas externas. Es decir, están formadas por diferentes unidades (neuronas) que se encuentran interconectadas a

través de unos enlaces por donde la información va atravesando la red neuronal para obtener unos valores de salida. Este sistema se encarga de aprender por sí solo según se va introduciendo la información, donde cada neurona actualiza sus valores de peso para disminuir el valor de la función de pérdida.

Las redes neuronales tienen un gran número de ventajas debido a la similitud que tienen con el cerebro humano porque tienen la capacidad de aprender con la experiencia. Entre sus ventajas podemos destacar:

- Aprendizaje adaptativo.
- Auto – organización.
- Tolerancia a fallos.
- Operación en tiempo real.
- Fácil inserción en la tecnología.

### 2.3.3- Regresión Lineal

La regresión es una forma estadística de establecer una relación entre una variable dependiente y un conjunto de variables independientes.

El algoritmo de regresión lineal es el empleado principalmente en Machine Learning. El algoritmo se basa en la creación de un modelo para lograr conseguir una relación de dependencia entre la variable independiente(x) y la variable dependiente(y) que es el "resultado".

Esta relación crea una línea arbitraria que es calculada por la distancia de la recta a los puntos de datos (x, y), en cada iteración esta recta se va recalculando para obtener un mayor ajuste a los datos (x, y) logrando que la línea tenga un menor error.

## 2.4- DE1-SoC

Para este proyecto hemos utilizado 6 placas del modelo DE1-SOC, que nos han sido cedidas por el departamento de Arquitectura de Computadores al que pertenecen nuestros directores de proyecto.

Este tipo de placas combinan un procesador ARM con RAM, conexión de red (normalmente ethernet), como si de un dispositivo electrónico compacto se tratara y esto lo combina con un array programable (una FPGA como tal). Estos dispositivos nos permiten realizar muchas más cosas que con FPGAs más simples y el coste tampoco es elevado, ya que una placa de este tipo la podemos encontrar sobre los 200$ y en el caso de que sea para algún tipo de investigación este coste se reduce bastante. En el caso concreto de nuestras placas, están desarrolladas por Terasic, la DE1-SOC con procesador ARM cortex A9 de doble núcleo, 1GB de RAM DDR3, varios conectores USB e incluso salida de vídeo VGA y audio stereo, además incorpora gran cantidad de conectores de distinto tipo como los GPIO, y pulsadores, interruptores y switchs y LEDS para distintas funciones. Esta modelo de placa está a la venta por 175$ para fines académicos y 249$ en caso contrario, aunque podemos encontrar muchas más opciones tanto a un menor precio como a un precio más elevado.

A continuación, podemos observar la distribución de todos los componentes del modelo DE1-SOC usado en este proyecto, así como la disposición de todos sus conectores, leds…

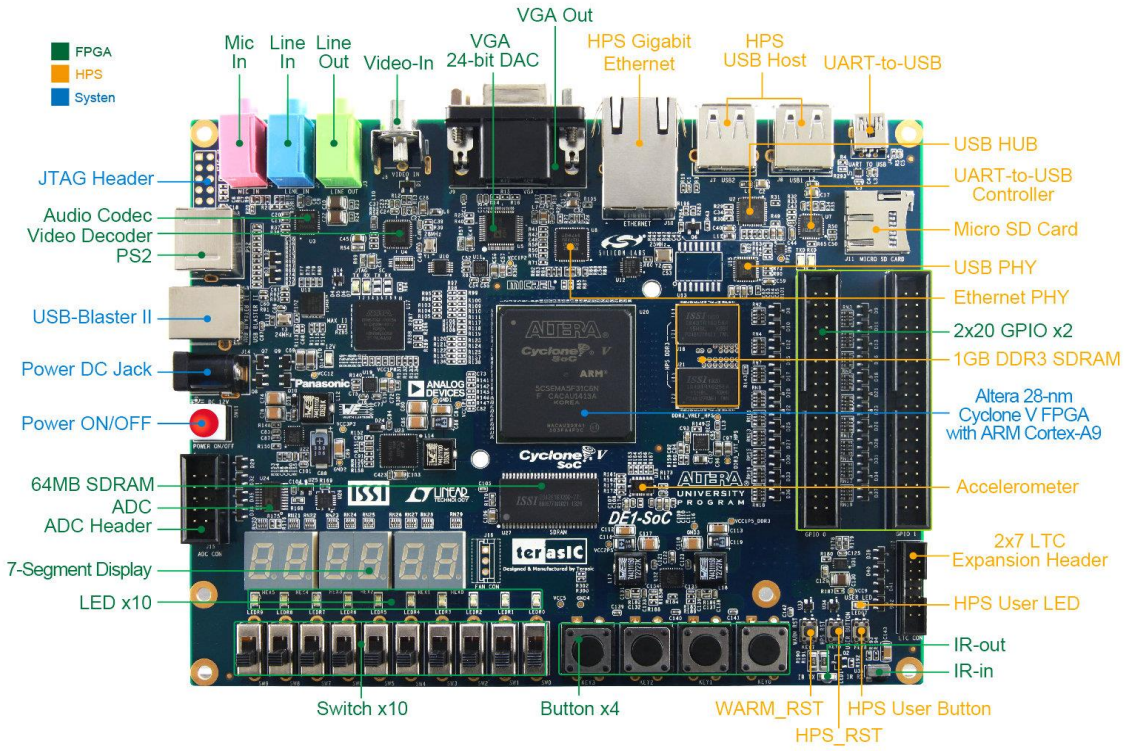


*Imagen 1: Detalles placa DE1-SOC*

La gran cantidad de conectores hace que sea fácilmente ampliable y esto nos permita realizar con ella gran cantidad de proyectos de distinto tipo ya que permite la incorporación de accesorios como pantallas táctiles, módulos WiFi o Bluetooth, cámaras…

En nuestro caso solo utilizaremos el conector Ethernet, y el UART-to-USB, además del conector de alimentación Power DC y el lector de la tarjeta de memoria.

### 2.4.1- ARM Cortex-A9

Las placas DE1-SOC, se componen de la FPGA propiamente dicha que es una Cyclone V y además incluye un procesador ARM Cortex-A9 que es donde nosotros mandaremos la ejecución del entrenamiento de SVM.

Este ARM Cortex-A9 se trata de un procesador de 32 bits cuya licencia tiene ARM Holdings e implementa la arquitectura ARMv7-A, es Dual-core y tiene una frecuencia máxima de 800 MHz. Además, dispone de 32 KB de caché de nivel L1 y 512 KB de caché de nivel L2.

Este procesador ha sido ampliamente usado por distintos fabricantes no solo para el desarrollo de FPGAs sino que fabricantes como Apple lo ha usado para el desarrollo de su chip A5 el cual usaban varios de sus dispositivos como el Apple TV de primera generación, el IPad mini de primera generación o el iPhone 4. Otros fabricantes como Sony lo utilizaron en su PlayStation Vita y Samsung en su chip Exynos 4, el cual era utilizado por sus algunos de sus terminales móviles en el año 2012.

## 2.5- Clústeres

Un clúster se puede definir como una agrupación o conjunto de computadoras independientes que se encuentran unidas entre ellas, habitualmente por una red de gran velocidad en las que todas las computadoras tienen una serie de componentes hardware comunes.

A este conjunto de elementos le llamamos clúster, pero cuando nos referimos a cada elemento independiente le denominamos nodo. Los nodos pueden ser FPGA, estaciones de trabajo, sistemas multiprocesadores etc.

Existen dos tipos de nodos:

- Nodos dedicados, que solo pueden realizar tareas del clúster
- Nodos no dedicados que contienen los periféricos, por tanto, se usan para otras funciones a parte de las tareas del clúster.

Para unir los nodos de un clúster podemos usar un tipo de red básica como puede ser Ethernet debido a su bajo coste, pero también podríamos usar tecnologías de alta velocidad que conllevan un coste mayor como puede ser InfiniBand, Myrinet, SCI, Gigabit Ethernet…

Según las características que dispongan los clústeres los podemos clasificar:

-High-Availability ("Alta Disponibilidad"): Este tipo de clúster está formado por varias máquinas que se monitorizan entre ellas y mantienen una serie de servicios compartidos donde la función es que mantengan la máxima disponibilidad del servicio que están ofreciendo, es decir, intentar que el servicio nunca tenga interrupciones incluso si se necesita reducir el rendimiento por ejemplo una base de datos.

-HPC ("Alto Rendimiento): Estos clústeres necesitan tener gran cantidad de memoria y/o gran capacidad computacional, ya que, este tipo de clúster busca tener el mayor rendimiento del servicio que está ofreciendo incluso si esto puede comprometer la disponibilidad del sistema. Estos clústeres se emplean cuando el proceso necesita una gran capacidad de cálculo.

-Load-balancing ("Balanceo de Carga"): Este tipo de clúster tiene la capacidad de realizar un balanceo de la carga de trabajo entre los diferentes nodos, de forma que permite mantener un sistema sin caídas en el servicio. Por tanto, este tipo de clúster también son High-Availability pero con la función extra de ser capaz de realizar el balanceo de la carga y habitualmente con mayor número de nodos.

Los clústeres tienen una gran serie de ventajas que podemos enumerar:

1. Alta disponibilidad
2. Balanceo de Carga
3. Escalabilidad
4. Resistencia a ataques DDoS

Pero como en todos los sistemas también tienen una serie de desventajas que debemos conocer y valorar antes de proceder a su implementación:

1. Alto coste
2. Complejidad
3. Tiempo de implementación

# Capítulo 3: Propuesta

## 3.1. Descripción del dataset

Actualmente encontramos multitud de dataset en la red de diversos tipos de datos y estudios. Si concretamos en datasets relacionados con tráfico en redes, la cantidad de datasets disponibles se reduce y muchos de ellos son privados o hay que cumplir ciertos requisitos para acceder a ellos.

Nosotros nos hemos decantado por el dataset **KDDCUP '99** [6] el cual es uno de los más famosos en lo relacionado con análisis de redes y su clasificación en ataques. Además es público y se compone de 125973 líneas de datos, esto es una de las características principales que nos han llevado a decantarnos por este dataset, ya que durante la ejecución del algoritmo de optimización mínima secuencial "SMO", el dataset será dividido en fragmentos según el número de placas con el que estemos trabajando y el resto de datasets que encontramos se componían de una cantidad mucho menor de datos, lo que provocaría que al fragmentar el dataset para la ejecución obtuviéramos fragmentos demasiado pequeños con los que no conseguiríamos unos buenos resultados en la predicción.

Otra de las características es que es multiclase y los ataques están separados en cuatro categorías, de manera que cada línea de datos puede tener alguna de estas cinco etiquetas:

- **Normal**: el tráfico es normal sin ningún tipo de ataque.
- **DDoS:** saturar el equipo atacado
- **Probing:** obtener información del equipo atacado
- **U2R:** acceso no autorizado a privilegios root
- **R2L:** obtener acceso a alguna de las cuentas del equipo atacado.

En nuestro caso nos centraremos en la clasificación binaria por lo que esta característica no la usaremos, pero puede ser de utilidad para en el futuro desarrollar el proyecto con una clasificación multiclase.

Debido a que realizaremos una clasificación binomial hemos convertido los datos a dos clases:

- **Normal (0):** el tráfico es normal sin ningún tipo de ataque.
- **Anómalo (1):** en el tráfico se ha detectado algún tipo de ataque sin especificar el tipo de ataque

Además, hemos encontrado que este dataset ha sido usado en diversas competiciones y proyectos de machine learning lo que hace posible que se puedan comparar los resultados con el resto de los proyectos que han usado este dataset, aunque no hemos encontrado de manera pública los resultados de estos proyectos.

- Composición del dataset:

Cada línea de datos del dataset se compone de 43 valores, de ellos, 42 son valores y uno la etiqueta. De estos 42 valores hemos eliminado tres, debido a que son campos de texto. Estos campos son:

- Protocolo: nos aporta información sobre el tipo de protocolo usado en la conexión (tcp, udp, icmp… )
- Servicio: nos da información sobre el servicio de red de destino usado (ftp, http, bgp...)
- Flag: estado de la conexión, nos indica si la conexión está activa, hay algún error, o si ha sido rechazada.

Hemos eliminado estos campos debido a que no nos aportan demasiada información si lo que vamos a hacer es clasificación binaria, si realizáramos una clasificación multiclase sí que nos podrían aportar más valor estos campos. Al eliminarlos nos quedamos con 39 campos más la etiqueta, algunos de los campos más destacados son los siguientes, aunque en la web del dataset[1] podemos encontrar la descripción detallada del dataset.

- Src_bytes/Dst_bytes: número de bytes transferidos desde el origen al destino y desde el destino al origen
- Wrong_fragment: número total de fragmentos erróneos en la conexión
- Urgent: número de paquetes con el bit “urgent” activado.
- Hot: el número de indicadores “hot” de la conexión como accesos a directorios del sistema, crear programas o ejecutarlo


1- http://kdd.ics.uci.edu/databases/kddcup99/kddcup99.html

- Num_failed_logins: contador del número de logins fallidos
- Logged_in status: indica si el inicio de sesión se ha completado o no.
- Root_shell: número de acciones realizadas como root
- Num_shells: número de accesos a la terminal
- Diversos campos que nos aportan contadores y porcentajes sobre las estadísticas de la conexión, como por ejemplo, paquetes con un determinado flag activado, conexiones al mismo servicio, conexiones con la misma IP destino…

En el repositorio del proyecto[1] podemos encontrar tanto el dataset original como nuestra versión adaptada.

## 3.2. Libsvm

Para el proyecto hemos decidido usar una librería libre, llamada **libsvm** [7]**.** Está desarrollada en la Universidad Nacional de Taiwan, y en ella encontramos una implementación de *Support Vector Machines* y está disponible en distintos lenguajes, algunos creados por los mismos desarrolladores de la librería y otras versiones creadas por la comunidad. Encontramos desde la versión en C++ que usaremos nosotros hasta una versión en CUDA, pasando por versiones en PHP, Node.js, MatLab, R o Python.

La librería ha sido ganadora de distintos premios en competiciones de machine learning y tiene un gran soporte y comunidad. Actualmente la última versión estable es la 3.23 que fue lanzada el 15 de Julio de 2018 y puede descargarse tanto de su repositorio de GitHub como de su página web oficial[2].

Una vez descargado el código encontraremos algunas versiones como la de Java, Matlab o C++, además de algunas utilidades en Python y MatLab para la preparación de los datasets o alguna ejecución de prueba. También se facilita la compilación del código ya que se incluye los archivos MakeFile tanto para la compilación en Linux como para Windows.


1- https://github.com/jorgburg/tfg-network-detection-acc.git
2- https://www.csie.ntu.edu.tw/~cjlin/libsvm/

## 3.3. Preparación del dataset

Libsvm utiliza un formato de datos en concreto, en el que la primera columna es la etiqueta de cada línea de entrenamiento, y a continuación se escriben los datos de cada columna, con la particularidad de que se incluye el índice de la columna antes del dato, por lo que si la columna 2 el dato es 45,34 el archivo generado contendrá-> 2:45,34

Otra particularidad es que los valores que son iguales a 0 en el dataset no hay que incluirlos, lo que reduce el tamaño final del archivo de datos. Para crear estos archivos partiendo del dataset original usaremos el Excel y Matlab ya que facilita el trabajo con gran cantidad de números. Para ello lo primero pasamos el dataset completo a un Excel en el que quitaremos las columnas que contengan texto, entre ellas encontramos tipo de protocolo...

Después creamos un csv (archivo delimitado por comas), y para poder realizar el proceso completo usaremos tres archivos de datos, uno para el entrenamiento, uno para testear la validez y error del entrenamiento y el último para hacer una prueba "real" del resultado del experimento. Esto se conoce en machine learning como entrenamiento, test y validación. Para ello creamos un csv para cada uno de los tres archivos. El dataset completo contiene aproximadamente 125000 filas de datos, por lo que lo dividiremos usando un 75% 25%, para ello el 75% del dataset lo usaremos para el entrenamiento y el 25% restante para el test y la validación. Esto hará que el archivo de entrenamiento contenga aproximadamente 93 mil filas de datos y las del archivo de test y validación unas 16 mil líneas cada uno.

El archivo csv para el entrenamiento lo crearemos sin problema, para el archivo de test y validación mantenemos las etiquetas originales del dataset, ya que la función de predicción omitirá estos datos. Una vez que ya tenemos el csv nos vamos a Matlab y nos situamos en donde tengamos los archivos Matlab de libsvm pues contiene los archivos necesarios para formatear correctamente los csv. Posteriormente usamos las siguientes líneas de código:

```
Matlab> SPECTF = csvread('SPECTF.train');
Matlab> labels = SPECTF(:, 1);
Matlab> features = SPECTF(:, 2:end);
Matlab> features_sparse = sparse(features);
Matlab> libsvmwrite('SPECTFlibsvm.train', labels, features_sparse);
```

Con esto lo que hacemos es leer los datos del csv, separar las etiquetas por un lado y los valores por otro, a la matriz de los valores le aplicamos la función sparse que elimina de la matriz todos los valores iguales a 0 y una vez hecho esto usamos la función libsvmwrite para volver a guardar las etiquetas y los valores (ya con los 0 eliminados) usando la estructura necesaria para ejecutar libsvm.

Este proceso lo tenemos que repetir con los tres archivos de datos y de esta forma ya podremos empezar a trabajar con el algoritmo.

## 3.3 Creación de un clúster de computación científica basado en FPGAs de bajo coste y consumo de Mariano Hernández García

En todos los trabajos o proyectos siempre hay una sección con trabajo futuro que acaba quedándose en el olvido, y creemos que es importante continuar con proyectos de innovación para aplicar conocimientos y mejoras.
Por ello, decidimos basarnos en el TFM de Mariano Hernández Garcia [1] estudiante del Master en Ingeniería Informática en la Universidad Complutense, publicado en Julio de 2017 en los eprints de la UCM, recibiendo una excelente calificación.
Este proyecto, plantea un clúster de 4 placas DE1-SoC que nosotros recreamos en un primer momento, modificamos y lo ampliamos, llegando a usar 6 placas DE1-SoC. El Sistema Operativo utilizado fue Debian, pero que corriera nuestro algoritmo tuvimos que instalar Ubuntu [8] en cada una de las placas, otra diferencia con este proyecto es sobre donde se ha ejecutado el algoritmo mientras que en el proyecto de Mariano Hernández se ejecutaba sobre la FPGA Cyclone V, nosotros lo hemos ejecutado en el procesador ARM del que dispone la DE1-SoC.

Estructura del clúster de Mariano Hernández, imagen 2:

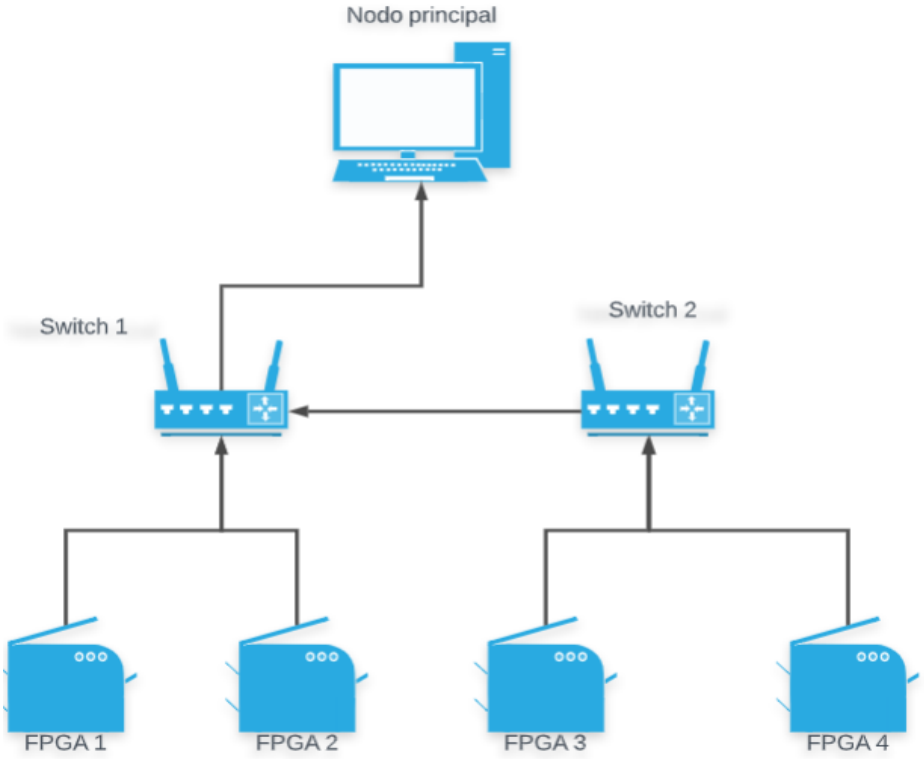


*Imagen 2: Estructura del clúster de Mariano Hernández*

Modificación del clúster para adaptarlo a 6 DE1-SoCs, imagen 3:

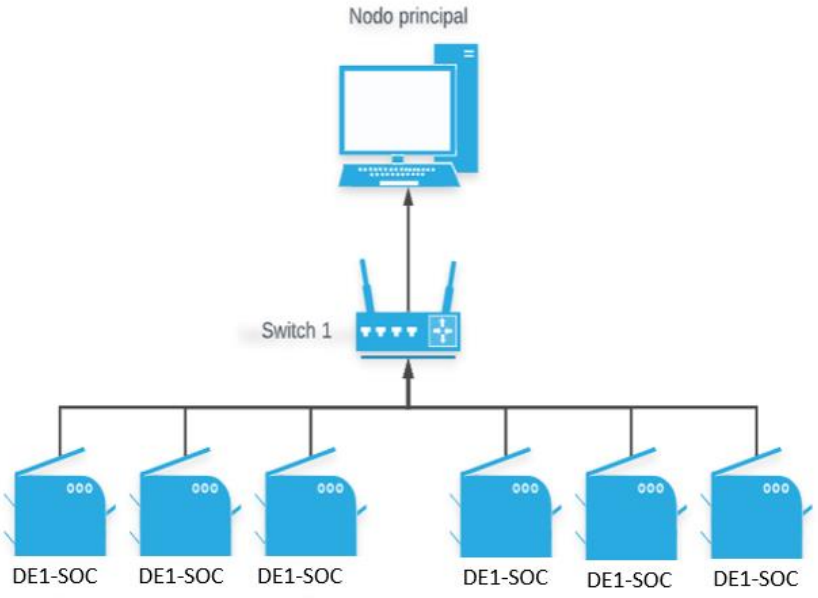


*Imagen 3: Estructura de nuestro proyecto*

Al trabajar con un conjunto de datos de mayor tamaño, decidimos ampliarlo para tener mayor nivel de cómputo.

La experimentación estaba basada en detectores de bordes de imágenes, y para ello utilizó los filtros Sobel y Laplace, así como las técnicas de emborronamiento gaussiano y print error diffusion y el algoritmo de flujo óptico.

La idea común al aplicar los algoritmos era, dividir la imagen en partes para ser procesadas por cada FPGA, a continuación, mostramos un ejemplo en la imagen 4, con el filtro de Laplace:

*Imagen 4: Imagen completa preparada para su división*

Una vez dividida la imagen, se ejecutaba el algoritmo desde las FPGAs con el resultado que se puede observar en la imagen 5:

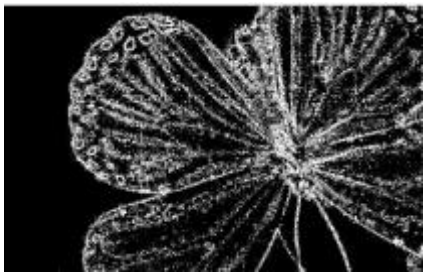

*Imagen 5: Resultado obtenido por una de las placas*

Las partes de la imagen procesadas por las FPGAs se enviaban al nodo principal (ordenador) que se encargaba de juntar las piezas para obtener el resultado final como podemos ver en la imagen 6:

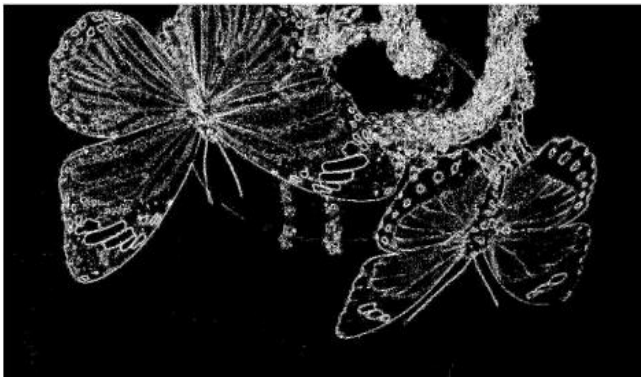

*Imagen 6: Resultado general tras juntar las imágenes*

La idea del clúster se puede aplicar a diversos algoritmos, y en nuestro caso, al tener perfiles de ciberseguridad decidimos modificarlo para la división de datasets y detección de ataques.

# Capítulo 4: Experimentación y resultados

## 4.1. Preparación de las SoCs

Actualmente podemos encontrar multitud de sistemas operativos para este tipo de dispositivos, en nuestro caso nos dirigimos a la página web de Altera [9] donde encontramos 6 versiones distintas para su descarga creadas concretamente para nuestra placa DE1-SOC. Para este proyecto y como recomiendan desde Altera, seleccionamos la imagen con la versión más reciente del kernel de Linux, que en este caso es la 4.5. Esta imagen en concreto incorpora un escritorio gráfico que nos puede facilitar el trabajo si así lo deseamos, aunque esto aumenta la carga de procesamiento de la placa.

En nuestro caso hemos trabajado mediante ssh, en primer lugar usando putty para crear una conexión mediante los puertos COM, conectando la placa al USB de nuestra máquina, esta conexión la usamos para configurar la ip de las placas cuando no está disponible el DHCP, una vez que la placa tiene configurada la dirección IP podemos conectarnos a la placa desde nuestra máquina usando el comando ***ssh root@[IP_SoC]*** y se nos abrirá una terminal remota para trabajar con la placa, este proceso podemos hacerlo más sencillo compartiendo nuestras claves públicas [10], de manera que al hacer esta conexión remota no necesitaremos introducir ninguna contraseña (este proceso será indispensable en la ejecución del algoritmo para que el script en nuestra máquina pueda ejecutar comandos remotos en la placa sin introducir la contraseña de usuario).

Lo primero será descargarnos la imagen desde la web de Altera [9] antes mencionada y una vez descargada procedemos a grabar la imagen en una tarjeta microSD [11]. En la página nos indican que para las versiones con escritorio gráfico es necesaria una tarjeta de mínimo 8GB de capacidad. En nuestro caso, usamos tarjetas de 32GB, aunque no es necesaria tanta capacidad. Lo que sí es importante es la velocidad de transferencia de dicha tarjeta, normalmente este tipo de sistemas de almacenamiento son usados en cámaras o dispositivos compactos en los que se trabaja con archivos de poco tamaño, en este caso los

estamos usando en una placa de manera que vamos a tener un acceso continuo a los datos y estos datos en muchos casos van a ser de un gran tamaño, de manera que la velocidad a la que funcione esta tarjeta puede afectar de manera considerable al funcionamiento de la placa. Lo mejor es usar una tarjeta de clase 10, todas las tarjetas de esta clase deberían asegurar una velocidad mínima de transferencia de 10MB/segundo, aunque hay modelos que teóricamente alcanzan los 90MB/segundo, por lo que es recomendable utilizar una tarjeta de gran calidad.

Para grabar esta imagen en la tarjeta podemos usar varios métodos:

- Si utilizamos Windows encontraremos multitud de utilidades para quemar la imagen en la tarjeta, en concreto podemos usar el programa Win32DiskImager, aunque debemos prestar atención ya que algunas de estas herramientas no son capaces de trabajar con el sistema de archivos de Linux (Ext4) y tendremos problemas a la hora de iniciar la placa.
- Si utilizamos Linux también encontramos varias utilidades, aunque lo más sencillo es usar el comando ***sudo dd if=[origen] of=[destino]*** [12], el campo origen lo sustituiremos por la ruta donde tengamos guardada la imagen descargada y el campo destino lo sustituiremos por el dispositivo donde queremos grabar la imagen, en nuestro caso la queremos grabar sobre la microSD [9] completa, no sobre ninguna partición, ya que la imagen ya tiene creadas todas las particiones necesarias (u-boot y preloader) y que son específicas para cada versión de FPGA. Podemos usar el comando ***fdisk -l*** para saber que nombre tiene asignada la microSD y será algo similar a /dev/sda. Una vez ejecutemos el comando esperaremos a que termine el proceso y ya tendremos la tarjeta lista.

Una vez tengamos preparada la tarjeta microSD debemos configurar la placa para el tipo de sistema que vamos a usar. Para ello la placa tiene un switch en la parte trasera llamado MSEL[4:0].

Para el tipo de sistema que vamos a usar, los 6 switch deben estar configurados a 0. Una vez tengamos la placa configurada y con la microSD introducida podemos conectar el cable de alimentación y al pulsar el botón de Power ON/OFF, situado a continuación del conector de alimentación, la placa deberá iniciarse, si la placa se inicia correctamente el display de 7

segmentos situado en una esquina de la placa debe quedarse fijo con todos los segmentos de todos los dígitos encendidos.

Una vez esté la placa arrancada, nos conectaremos a ella de la forma que deseemos, conectándola por USB a través del puerto COM o conectar un monitor por VGA.

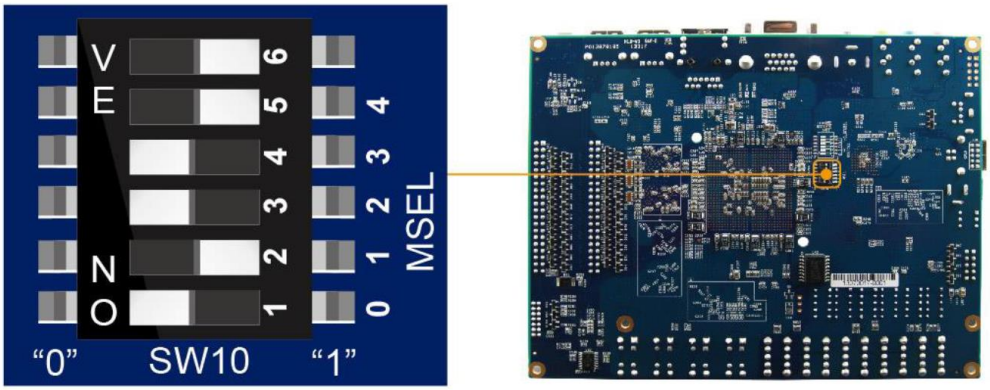


*Imagen 7: Configuración switches*

Para este proyecto hemos utilizado la librería libsvm [7] la cual es una implementación de SVM en diferentes lenguajes (MatLab, Java, C…), en este caso utilizaremos la versión en C. Para ejecutarlo en la placa tenemos dos opciones:

- La primera consiste en instalar ***gcc*** [13] en una de las placas, copiar a esta placa el código fuente y compilarlo directamente desde la placa y una vez compilado exportar el ejecutable en el resto de las placas.
- Por otro lado, podemos utilizar la compilación cruzada, mediante la cual compilamos el código desde nuestra máquina para ejecutarlo en el ARM sin tener que compilar el código directamente en la placa.

Si optamos por la primera opción simplemente instalaremos el paquete y en una terminal nos situaremos sobre el directorio del código fuente, una vez aquí ejecutaremos el comando ***make*** ya que el código ya incluye un archivo ***Makefile*** para la generación de los ejecutables, una vez termine el proceso, ya tendremos los archivos ***svm-train svm-predict*** y ***svm-scale*** listos para ejecutar en dispositivos arm32. Aunque en las placas solo necesitaremos el svm-train, ya que la predicción la ejecutaremos en la máquina master. Por esta razón tendremos que compilar también el código en nuestra máquina máster, para ello ejecutamos el mismo

comando ***make,*** habiendo instalado previamente los paquetes necesarios para compilar en C/C++.

Si elegimos compilar el código fuente mediante compilación cruzada, no necesitamos realizar la compilación en la placa, para ello en nuestra máquina máster instalaremos los paquetes: ***gcc-arm-linux-gnueabi*** [14] y ***binutils-arm-linux-gnueabi*** y en el Makefile de nuestro código fuente cambiaremos el compilador gcc por ***arm-linux-gnueabi-gcc,*** de esta manera el código se compilará como si lo estuviéramos compilando directamente en la placa, creando un archivo ejecutable válido y compatible con la arquitectura arm32.

Ya solo tendremos que copiar a la memoria de cada placa el archivo svm-train que hemos compilado desde nuestra máquina, y volver a compilar el código fuente usando el Makefile original para crear el ejecutable del svm-predict que usaremos en nuestra máquina máster.

Es importante tener en cuenta los permisos de los ejecutables que creemos, ya que si los copiamos de una placa a otra nos dará problemas de permisos y no podremos ejecutarlos aún estando logueados como root. Para solucionar esto ejecutamos el comando ***chmod u+x svm-train*** de manera que otorgamos al creador del archivo permisos de ejecución sobre el archivo.

Una vez tenemos la placa preparada debemos crear el archivo state en el que la placa irá actualizando el estado de la ejecución y será el que la máquina máster vaya leyendo para saber cuándo ha terminado la ejecución de cada placa. Para ello podemos usar el comando ***touch state,*** aunque debemos asegurarnos que estamos situados en la ruta /root, que es la que usaremos para la ejecución. Por último, copiaremos el script bash creado para la ejecución de cada placa, este script será el que ejecutaremos mediante una shell remota desde el script que ejecutamos en la máquina master [ANEXO1]. Cuando tengamos copiado el archivo ya tendremos la placa lista para la ejecución, en este punto podemos crear una imagen actual de la tarjeta microSD en este punto de manera que podremos exportar fácilmente todos los archivos y SO creado. para ello usamos de nuevo el comando ***dd if=[nombre_microSD] of=[nombre_archivo.img]*** [12], en este caso lo ejecutamos al revés, como entrada ponemos la tarjeta y como destino el nombre de la imagen que queremos crear, de esta manera podremos grabar esta imagen en el resto de las tarjetas para las

demás placas y habremos replicado el estado final de la primera placa en el resto de las placas.

## 4.2. Ejecución del algoritmo

Para ejecutar el algoritmo crearemos dos scripts bash, uno que ejecutaremos desde la máquina base y otro que estará almacenado en la memoria de cada placa y que será la que ejecute el entrenamiento en el procesador ARM de cada placa DE1-SOC.

El script en la máquina estará soportado por otro archivo .config donde escribiremos la IP de cada nodo. Lo primero será leer este archivo para identificar cada uno de los nodos y el número total de nodos, una vez tengamos el número de nodos, dividiremos el archivo de entrenamiento entre el número de nodos que haya conectados. De esta manera obtendremos tantos archivos data_XX.train como nodos tengamos disponibles. Tras esto copiaremos cada archivo en el nodo que corresponda y empezaremos el entrenamiento en las placas.

Lo primero será escribir el estado running en el archivo state de cada placa y con el comando ***ssh ./launc_svm_train data_XX.train*** se lanzará el svm-train y comenzará el entrenamiento.

Mientras se ejecuta en cada placa, la máquina leerá el archivo state de las placas y el entrenamiento termina cuando la placa escribe finish en este archivo. En ese momento se habrá generado el archivo ***data_XX.model*** y cada placa copiará dicho archivo .model a la máquina máster.

Tras la finalización del algoritmo de entrenamiento en las placas, tendremos en la máquina máster tantos archivos .model como número de nodos hayamos usado al ejecutar el entrenamiento. Cada archivo .model ha sido generado usando una porción distinta del dataset.

A la hora de realizar la predicción podríamos usar distintos métodos: el primero sería hacer una predicción sobre el dataset de test y el modelo con el que obtengamos mayor porcentaje de acierto usarlo para la validación, otra opción sería que para cada modelo hagamos el test y usando un votador para cada línea de test o validación seleccionemos un

resultado según la predicción que más veces haya salido entre todos los modelos. Con esta segunda opción deberíamos obtener unos mejores resultados, pero esto se vería afectado con un mayor tiempo en la ejecución de la predicción.

En la sección de resultados encontraremos por un lado los porcentajes obtenidos usados el primer método: haciendo la predicción con cada modelo generado con una parte del dataset, pero este método no es demasiado útil ya que a medida que aumentamos el número de nodos reducimos el tamaño de cada fragmento del dataset, lo que conlleva a que disminuya la tasa de aciertos.

Para contrarrestar esto, hemos creado un pequeño script bash que hace las funciones de un votador, es un script que, sobre el fichero de test o validación, realiza la predicción usando el modelo de cada uno de los nodos con los que hayamos ejecutado el entrenamiento. De esta manera se genera un archivo que lo hemos llamado ***out_X.predict*** (siendo la X el número de nodo). Cada uno de estos archivos .predict es una sucesión de 0 y 1 según el algoritmo determina si es un ataque o no dependiendo del modelo generado.

Una vez tenemos todos los archivos .predict el script suma cada línea de los archivos y genera un array con la suma de cada línea. Como en cada línea solo puede haber un 0 o un 1, el resultado del array será la suma de 1 que hay por cada línea entre todos los archivos .predict.

Por último, crearemos el resultado final del votador, el cual dará como resultado lo que haya predicho la mayoría de los modelos. Para ello dividimos el número de nodos entre dos y si el número de cada posición del array es mayor que la mitad de los nodos la predicción final será 1, en caso contrario será 0.

A continuación, mostramos las tablas 1 y 2 que resumen el comportamiento del votador en algunas situaciones:

| Funcionamiento del votador con 4 nodos | | | | |
|---|---|---|---|---|
| **Predicción 1** | **Predicción 2** | **Predicción 3** | **Predicción 4** | **Predicción Final** |
| 0 | 1 | 1 | 1 | 1 |
| 0 | 0 | 1 | 1 | 0 |
| 0 | 0 | 0 | 1 | 0 |

*Tabla 1: Resultados votador con 4 nodos*

| Funcionamiento del votador con 5 nodos | | | | | |
|---|---|---|---|---|---|
| **Predicción 1** | **Predicción 2** | **Predicción 3** | **Predicción 4** | **Predicción 5** | **Predicción Final** |
| 0 | 1 | 1 | 1 | 1 | 1 |
| 0 | 0 | 1 | 1 | 1 | 1 |
| 0 | 0 | 0 | 1 | 1 | 0 |

*Tabla 2: Resultados votador con 5 nodos*

El funcionamiento es igual independientemente del número de nodos, en caso de que haya el mismo número de 0 que de 1 en la suma de las predicciones el resultado final será 0.

Los resultados del votador se podrían afinar mucho más aplicando una política de prioridades en la que la predicción que mayor porcentaje de acierto haya obtenido se aplique una mayor prioridad en caso de igualdad de resultados.

## 4.3. Resultados obtenidos

A continuación, mostramos los resultados obtenidos realizando la ejecución con distinto número de placas desde dos placas hasta seis, incluyendo una ejecución desde nuestra máquina con Linux. En cada escenario de ejecución hemos utilizado el mismo archivo de entrenamiento y el mismo archivo de test, por lo que todos los resultados se obtienen sobre los mismos datos. Además, hemos utilizado en todas las ejecuciones el mismo switch para conectar las placas a nuestra máquina y los mismos cables de red, en este caso cables Ethernet.

Todos los tiempos mostrados en los resultados están en segundos.

En cada sección de resultados incluimos por un lado una tabla con las estadísticas relativas a la ejecución del entrenamiento:

- Tiempo empleado en la copia del fragmento del dataset: tiempo que tarda la máquina en copiar cada fragmento del dataset al respectivo nodo.
- Número de iteraciones: es el número de iteraciones que cada nodo realiza durante el entrenamiento de SVM
- Tiempo de ejecución del entrenamiento: tiempo transcurrido en cada placa desde que se empieza la ejecución del svm-train hasta que obtenemos el modelo.
- Tiempo de copia del modelo al nodo máster: tiempo que tarda cada placa en copiar al nodo máster el archivo .model generado en el entrenamiento.
- Tiempo total: tiempo total transcurrido durante la ejecución, desde que se llama al script bash en nuestra máquina máster y realizando todo el proceso hasta que obtenemos todos los modelos, uno por cada placa.

Por otro lado, podemos ver otra tabla con los resultados obtenidos, en esta tabla observamos una fila por cada nodo conectado y por cada nodo observamos:

- Número de aciertos sobre el número total de líneas del fichero de test que en nuestro caso son 25000.
- Tasa de aciertos: mostrado en porcentaje.

En esta última tabla encontramos una fila que contiene los resultados del votador utilizando los modelos de todos los nodos usados para la ejecución.

### 4.3.1: Ejecución con 2 nodos

En esta ejecución podemos observar con las tablas 3 y 4, que el tiempo total es muy alto debido a que el Nodo 1 tiene un tiempo de ejecución de entrenamiento bastante alto. Podemos observar que la tasa de acierto en cada uno de los nodos es muy similar, ya que tienen una cantidad muy alta de datos para el entrenamiento. Cuando usamos el votador se reduce entre un 1% y 2% comparando las predicciones de cada uno de los nodos.

| Nodo | Tiempo de ejecución del entrenamiento | Tiempo de copia del modelo del master | Tiempo total |
|---|---|---|---|
| **Nodo 1** | 4348,436 s | 1,024 s | 4354,43 s |
| **Nodo 2** | 3916,763 s | 0,948 s | |

*Tabla 3: Tiempos con 2 nodos*

| Nodo | N.º aciertos/N.º total | Tasa de aciertos |
|---|---|---|
| **Nodo 1** | 24315/25000 | 97,26% |
| **Nodo 2** | 24306/25000 | 97,224% |
| **Votador** | 23844/25000 | 95,376% |

*Tabla 4: Resultados con 2 nodos*

### 4.3.2: Ejecución con 3 nodos

Cuando realizamos la ejecución con 3 nodos observamos en las tablas 5 y 6 que el tiempo total de ejecución se reduce de gran manera. En este caso podemos observar que la tasa de acierto entre los tres nodos es casi similar y empleando el votador logramos una mayor tasa de acierto.

| Nodo | Tiempo de ejecución del entrenamiento | Tiempo de copia del modelo del master | Tiempo total |
|---|---|---|---|
| **Nodo 1** | 1935,75 s | 0,991 s | 1942,32 s |
| **Nodo 2** | 1651,305 s | 0,917 s | |
| **Nodo 3** | 1744,291 s | 0,908 s | |

*Tabla 5: Tiempos con 3 nodos*

| Nodo | N.º aciertos/N.º total | Tasa de aciertos |
|---|---|---|
| **Nodo 1** | 24118/25000 | 96,472% |
| **Nodo 2** | 24118/25000 | 96,472% |
| **Nodo 3** | 24090/25000 | 96,36% |
| **Votador** | 24129/25000 | 96,516% |

*Tabla 6: Resultados con 3 nodos*

### 4.3.3. Ejecución con 4 nodos

La ejecución de 4 nodos volvemos a lograr mejorar el tiempo total pero podemos observar en las tablas 7 y 8 que hay una diferencia bastante alta entre el tiempo del Nodo 3 que es el primero en acabar y el resto de Nodos. En esta ejecución podemos observar que la tasa de acierto puede llegar a variar hasta en un 1% entre los diferentes nodos. En este caso el uso del votador provoca que se reduzca la tasa de acierto en relación a la media de todos los nodos.

| Nodo | Tiempo de ejecución del entrenamiento | Tiempo de copia del modelo al master | Tiempo total |
|---|---|---|---|
| **Nodo 1** | 1015,342 s | 0,917 s | 1060,596 s |
| **Nodo 2** | 1024,104 s | 0,904 s | |
| **Nodo 3** | 909,667 s | 0,873 s | |
| **Nodo 4** | 1056,054 s | 0,919 s | |

*Tabla 7: Tiempos con 4 nodos*

| Nodo | N.º aciertos/N.º total | Tasa de aciertos |
|---|---|---|
| **Nodo 1** | 23962/25000 | 95,848% |
| **Nodo 2** | 24004/25000 | 96,016% |
| **Nodo 3** | 23996/25000 | 95,984% |
| **Nodo 4** | 23923/25000 | 95,692% |
| **Votador** | 23759/25000 | 95,036% |

*Tabla 8: Resultados con 4 nodos*

### 4.3.4. Ejecución con 5 nodos

En esta ejecución podemos volver a ver en las tablas 9 y 10, que se produce una reducción en el tiempo total de ejecución. Con relación a la tasa de acierto de esta ejecución observamos que el porcentaje de acierto entre todos los nodos es muy similar. Ahora el votador ha obtenido una tasa de acierto inferior a todos los nodos

| Nodo | Tiempo de ejecución del entrenamiento | Tiempo de copia del modelo al master | Tiempo total |
|---|---|---|---|
| **Nodo 1** | 608,961 s | 0,912 s | |
| **Nodo 2** | 538,6 s | 0,896 s | |
| **Nodo 3** | 605,762 s | 0,860 s | 616,961 s |
| **Nodo 4** | 583,833 s | 0,868 s | |
| **Nodo 5** | 601,788 s | 0,856 s | |

*Tabla 9: Tiempos con 5 nodos*

| Nodo | Nº aciertos/Nº total | Tasa de aciertos |
|---|---|---|
| **Nodo 1** | 23831/25000 | 95,324% |
| **Nodo 2** | 23844/25000 | 95,376% |
| **Nodo 3** | 23865/25000 | 95,46% |
| **Nodo 4** | 23847/25000 | 95,388% |
| **Nodo 5** | 23794/25000 | 95,176% |
| **Votador** | 23770/25000 | 95,08% |

*Tabla 10: Resultados con 5 nodos*

### 4.3.5. Ejecución con 6 nodos

En la última ejecución que hemos realizado hemos logrado un tiempo total de ejecución levemente inferior al anterior de 5 nodos. En este caso los diferentes nodos tienen todos una tasa de acierto muy similar pero cuando empleamos el votador su tasa de acierto disminuye. Esto se puede observar en las tablas 11 y 12.

| Nodo | Tiempo de ejecución del entrenamiento | Tiempo de copia del modelo al master | Tiempo total |
|---|---|---|---|
| **Nodo 1** | 491,199 s | 0,906 s | 500,114 s |
| **Nodo 2** | 402,875 s | 0,848 s | |
| **Nodo 3** | 417,776 s | 0,856 s | |
| **Nodo 4** | 401,818 s | 0,841 s | |
| **Nodo 5** | 407,952 s | 0,849 s | |
| **Nodo 6** | 382,386 s | 0,836 s | |

*Tabla 11: Tiempos con 6 nodos*

| Nodo | Nº aciertos/Nº total | Tasa de aciertos |
|---|---|---|
| **Nodo 1** | 23743/25000 | 94,972% |
| **Nodo 2** | 23761/25000 | 95,044% |
| **Nodo 3** | 23779/25000 | 95,116% |
| **Nodo 4** | 23758/2500 | 95,032% |
| **Nodo 5** | 23751/25000 | 95,004% |
| **Nodo 6** | 23697/25000 | 94,788% |
| **Votador** | 23575/25000 | 94.3% |

*Tabla 12: Resultados con 6 nodos*

### 4.3.6. Ejecución en el nodo máster

Para esta ejecución hemos utilizado una máquina cuyas características son las siguientes:

- CPU: AMD Ryzen 5 1600 a 3.2GHz
- RAM: 8GB de memoria DDR4 a 2400MHz de velocidad.
- Tarjeta gráfica: NVIDIA GTX 1050 con 4GB de memoria gráfica dedicada.

| Nodo | Tiempo de ejecución del entrenamiento | Tiempo de copia del modelo al master | Tiempo total |
|---|---|---|---|
| **Nodo Máster** | 1449,394 s | X | 1449,394 s |

*Tabla 13: Tiempo del nodo master*

| Nodo | N.º aciertos/N.º total | Tasa de aciertos |
|---|---|---|
| **Nodo Máster** | 24584/25000 | 98,336% |

*Tabla 14: Resultados nodo master*

En la ejecución en el nodo master, tabla 13, no tenemos ni tiempo de copia del fragmento ni copia del archivo modelo, al igual que la predicción final es la que conseguimos con el predict, no tiene sentido utilizar el script del votador.

## 4.4 Comparación de resultados

Las diferencias son mínimas en cuanto a los resultados finales a la hora de añadir un nuevo nodo.

En cambio, el tiempo total sí que varía al dividir la carga de trabajo. Se ha de tener en cuenta que hemos trabajado con un máximo de 6 nodos, en caso de ampliación, los tiempos de ejecución y la precisión de la predicción se verían afectados notablemente.

Vamos a comparar los tiempos totales de cada ejecución según el número de nodos y el nodo con mayor tiempo de ejecución, ambos en segundos.

Las diferencias se observan claramente, con una progresión descendente desde la primera prueba hasta la última.

En el gráfico 1, vamos a mostrar la tasa de aciertos obtenida, con el mayor y el menor porcentaje por cada nodo.

Podemos observar que cuantos más nodos tienes, menos precisión se obtiene. Al dividir la carga de trabajo, el entrenamiento se vuelve menos preciso ya que a la hora de entrenar, tiene menos datos.

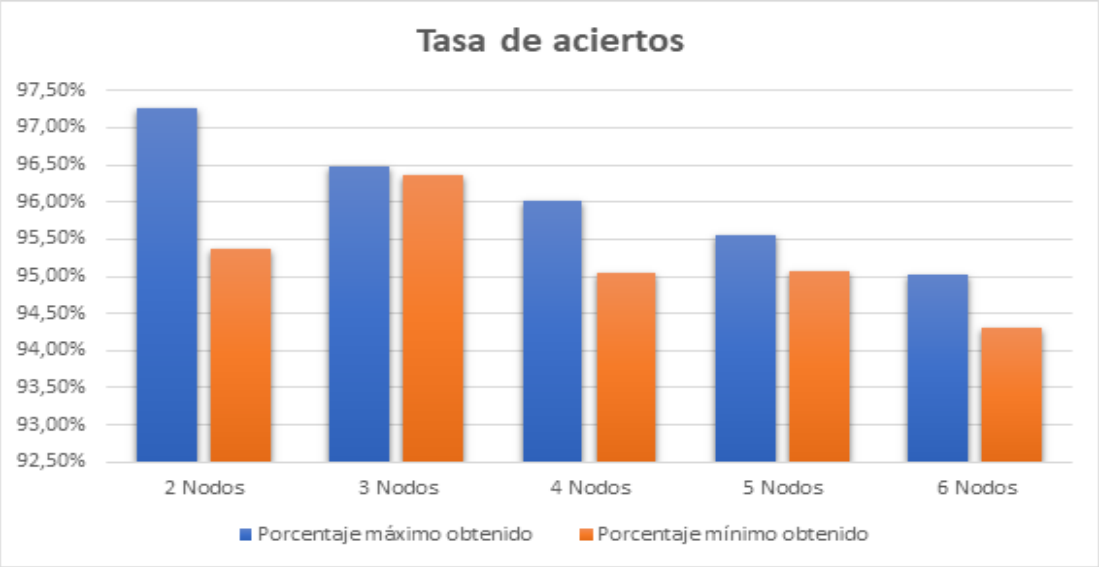


*Gráfico 1: Comparativa de la tasa de aciertos*

De igual forma, los resultados obtenidos son satisfactorios, pudiendo concluir que nuestra experimentación es precisa y fiable, ya que los comportamientos son lo esperado, cuantos más nodos se utilicen, menor será el tiempo ya que cada uno procesa menos datos, pero la tasa de acierto se reducirá ligeramente.

Si observamos los anteriores gráficos de barras podemos observar como el tiempo de ejecución no se comporta de forma lineal a medida que aumentamos el número de nodos, por lo que a la vista de los resultados obtenidos podemos pensar que aumentar el número

de nodos por encima de seis no obtendremos unos resultados atractivos, ya que la reducción en el tiempo de ejecución no será muy notable a costa de perder en el porcentaje de aciertos. Esto lo podemos ver ya que la reducción del tiempo empleado en el caso de los cinco y seis nodos es muy inferior a la disminución conseguida en las ejecuciones con menos nodos.

El porcentaje de aciertos sí se comporta de una forma bastante lineal.Si que observamos que con 3 nodos la tasa obtenida se sale ligeramente de esta tendencia lineal y se sitúa en una tasa prácticamente igual que la tasa con 2 nodos.

A continuación, podemos observar otro gráfico similar al anterior en el que también vemos el tiempo empleado en la ejecución frente al porcentaje de acierto obtenido, pero en este caso ordenamos la tasa de acierto de manera ascendente, de esta forma podemos apreciar mejor los distintos resultados en función del objetivo que se busque: maximizar la tasa de aciertos o reducir el tiempo de ejecución.

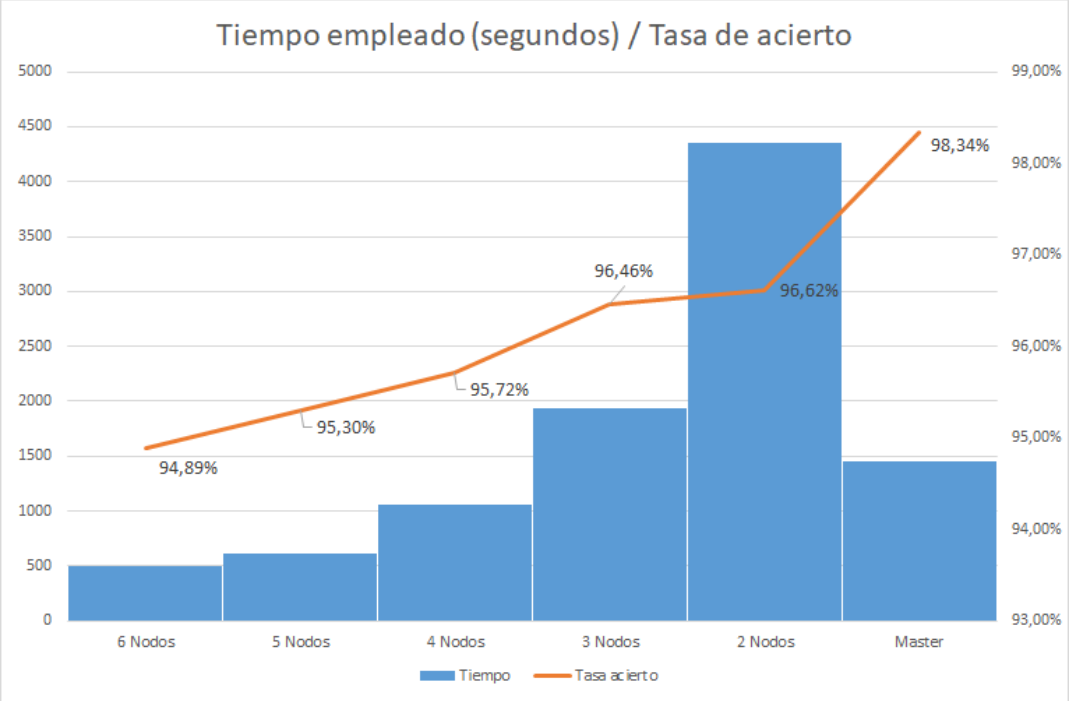


*Gráfico 2: Resultados y comparativa del tiempo empleado y la tasa de aciertos*

Si analizamos esta última gráfica 3 podemos ver claramente que el porcentaje obtenido aumenta a medida que disminuimos el número de nodos, destacando especialmente el

porcentaje obtenido por la ejecución con el nodo master. En esta gráfica podemos observar mejor que el incremento de porcentaje en el paso de seis nodos a dos nodos es prácticamente lineal, solamente el resultado de los tres nodos difiere un poco de esta tendencia y en este caso se obtiene un porcentaje muy cercano al porcentaje obtenido con dos nodos pese a haber reducido el tiempo en más de la mitad.

De esta forma, realizar la ejecución con tres placas parece más rentable que la ejecución con dos, ya que solamente perdemos el 0,16% de tasa de acierto, pero reducimos el tiempo de ejecución en un 55,4% pasando de 4354,43 segundos en el caso de dos nodos a 1942,32 segundos en el caso de los tres nodos. A partir de ahí vemos como entre los cuatro y seis nodos el porcentaje se mantiene prácticamente lineal pero el tiempo cada vez presenta una reducción menor. Además, sería bueno comprobar los resultados aumentando el número de nodos y ver si el porcentaje de aciertos se sigue manteniendo prácticamente lineal y en caso contrario detectar en qué punto comienza a reducirse en mayor medida este porcentaje.

Además de lo comentado de que los tres nodos frente a los dos nodos, no podemos tener muchas más conclusiones, ya que las disminuciones en los porcentajes no son lo suficientemente significativas y el uso de uno u otro número de nodos dependerá de las necesidades que se tengan, de esta manera si se va a realizar un reentrenamiento para mejorar el aprendizaje de forma más habitual puede ser interesante una reducción del tiempo de ejecución frente a la tasa de acierto, mientras que si el reentrenamiento no va a ser habitual se puede valorar ese aumento en el tiempo de ejecución si esto supone un aumento en la tasa de aciertos.

Además, también hemos detectado que en algunas ocasiones la red llegaba a un punto de saturación, haciéndose esto más notable a medida que aumentamos el número de nodos. Por ejemplo, al transferir los fragmentos de dataset a cada una de las placas este tiempo de transferencia oscilaba entre 1.1 segundos y 1.6 segundos, dependiendo del tamaño de cada fragmento que depende del número de nodos. Pero en ocasiones este tiempo en alguna de las placas sufría un retraso, llegando a tardar alrededor de 3-4 segundos. En esto además influye que no hemos utilizado ni cables ni dispositivos de red de alta gama, esto supone que el factor red y los tiempos de transferencias en la red llegue un punto que no podamos reducirlos pese a estar enviando y recibiendo archivos de menor tamaño a medida que

aumentamos el número de nodos. Podemos intentar mitigar esta sobrecarga aumentando la categoría de los cables Ethernet o usando un switch de mejor calidad o incluso utilizando una tarjeta de red de gama más alta en el nodo máster. Sin embargo, aunque a medida que aumentemos el número de nodos esta mejora de la infraestructura de red será mucho más necesaria, de manera que aquí encontramos otro motivo para no utilizar un clúster de tamaño mucho mayor al que hemos utilizado en las pruebas.

Otro detalle a tener en cuenta a la hora de realizar la comparación es el tema de los costes, para ello tenemos el nodo master cuyo coste es de aproximadamente unos 700$ con los componentes que tiene, mientras que en el caso de las SoCs utilizadas tienen un coste de 180$ si sumamos el precio de la tarjeta microSD más el precio de la propia placa.

Calculando el coste de las placas para realizar el experimento con 6 nodos nos encontramos con que el coste total serian 1080$, a esto tendríamos que sumar el coste del switch usado que ronda los 20$ un dispositivo de estas características, aunque en caso de implementarlo en un entorno real contamos con que la red ya dispondría de este tipo de dispositivos por lo que no contaremos con el coste del switch para hacer la comparativa.

Por otro lado, el coste con cuatro placas sería de 720$ y con cinco nodos serían 900$. Si comparamos estos costes con los del nodo master, vemos que son muy similares en el caso de cuatro nodos y en el caso de los cinco nodos supone un aumento del coste de aproximadamente un 20%, aunque en nuestra opinión este aumento del coste merece la pena viendo que el tiempo se reduce a la tercera parte.

# Capítulo 5: Conclusiones y trabajo futuro

## 5.1 Conclusiones

Este trabajo se esfuerza en presentar la construcción de un clúster formado por SoCs que tienen un bajo coste. Sobre este sistema hemos ejecutado un algoritmo de aprendizaje sobre un dataset que hemos seleccionado.

Las placas que hemos empleado para el proyecto son DE1-SOC de Altera donde hemos introducido la imagen Linux 4.5 que es la que posee la versión más reciente del kernel de Linux.

El dataset sobre el que el algoritmo de aprendizaje de SVM va a trabajar es el llamado KDDCUP`99 que posee el análisis de tráfico de datos de diferentes redes sobre las que se estaban produciendo ataques informáticos, además de otras redes en las que existía un tráfico normal.

La librería que hemos empleado es Libsvm que está implementada en diferentes lenguajes de programación, en concreto nosotros hemos decidido usar la versión de C++ para completar el entrenamiento de SVM sobre el ARM del procesador de nuestras placas.

Finalmente, hemos realizado pruebas desde la utilización de una única Workstation de altas prestaciones hasta la última prueba en la que hemos obtenidos datos con un clúster de 6 nodos, y donde se ha podido observar que el mayor equilibrio de tiempo/acierto se produce cuando lo realizamos con el clúster formado por 4 o 5 nodos.

## 5.2 Conclusions

This work strives to present the construction of a cluster formed by SoCs that have low energy consumption, in addition to the low cost of such plates. On this system we have executed a learning algorithm on a dataset that we have selected.

The boards that we have used for the project are DE1-SOC from Altera where we have introduced the Linux 4.5 image, which is the one with the latest version of the Linux kernel.

The dataset on which the SVM learning algorithm is going to work is the so-called KDDCUP`99 that has the analysis of data traffic of different networks on which computer attacks were occurring, in addition to other networks in which there was a traffic normal.

The library we have used is Libsvm which is implemented in different programming languages, specifically we have decided to use the C ++ version to complete the SVM training on the processor ARM of our boards.

Finally, we have carried out tests from the use of a single high-performance Workstation to the last test in which we have obtained data with a cluster of 6 nodes, and where it has been observed that the greatest balance of time / success occurs when we do it with the cluster formed by 4 or 5 nodes.

## 5.3 Trabajo futuro

En este proyecto hemos realizado una aproximación a una herramienta de detección de ataques en redes de bajo coste y consumo utilizando el aprendizaje automático.

Como trabajo futuro hemos llegado a varias conclusiones, en primer lugar, se podría profundizar más en un algoritmo multiclase de manera que el algoritmo sea capaz de predecir el tipo de ataque que se ha llevado a cabo, no solamente determinar si ha habido un ataque. Para esto se podría utilizar gran parte de lo utilizado en este proyecto, ya que todo el SO montado en las placas y los compiladores utilizados servirían para desarrollarlo, incluso la librería Libsvm dispone de una posibilidad de realizar un entrenamiento multiclase del SVM.

Por otra parte, sería interesante realizar el proyecto usando OpenCL al igual que se hacía en el TFM comentado en el Capítulo 3, de manera que se aproveche la FPGA Cyclone V de la placa usada y no solo el procesador ARM como hemos hecho en este proyecto. Esto debería reducir el tiempo de ejecución debido a la gran velocidad de las FPGA en la realización de cálculos aritméticos.

También podría desarrollarse otro algoritmo de aprendizaje automático como una implementación de redes neuronales y probar la ejecución en las FPGA para comparar los resultados con los que hemos obtenido en este proyecto.

Siguiendo en la línea de nuestro proyecto otra opción sería desarrollar un IDS en tiempo real de manera que una máquina que esté conectada continuamente a la red se encargue de analizar en tiempo real todo el tráfico en dicha red. Para esto, la máquina además de estar conectada a la red se conectaría a las FPGA y cada cierto tiempo se repetiría el entrenamiento para mejorar el modelo entrenado con nuevos datos recogidos y usando el último modelo disponible, la máquina máster analice todo el tráfico de la red, pudiendo avisar cuando la predicción muestre un posible ataque y se puedan tomar a tiempo las medidas necesarias.

# Referencias


[1] Hernández, M. (2017). *Creación de un clúster de computación científica basado en FPGAs de bajo coste y consumo* (Grado). Universidad Complutense de Madrid., Madrid, España.

[2] Cyclone V SoC Development Kit and SoC Embedded Design Suite. (2019). Retrieved 26 June 2019, from https://www.intel.com/content/www/us/en/programmable/products/boards_and_kits/dev-kits/altera/kit-cyclone-v-soc.html

[3] Oleg Kupreev, Ekaterina Badovskaya, Alexander Gutnikov, *Ataques DDoS en el primer trimestre de 2019,* 2019. [En línea]. Available: https://securelist.lat/ddos-report-q1-2019/88828/

[4] SVM clasificación y explicación. [En línea]. Available: https://scikitlearn.org/stable/modules/svm.html

[5] FPGA (Field-Programmable Gate Array) Definition. (2019). Retrieved 8 January 2019, from https://techterms.com/definition/fpga

[6] KDD Cup 1999Dataset, 1999. [En línea]. Available: http://kdd.ics.uci.edu/databases/kddcup99/kddcup99.html

[7] LIBSVM -- A Library for Support Vector Machines. (2019). Retrieved 4 March 2019, from https://www.csie.ntu.edu.tw/~cjlin/libsvm/

[8] About the Ubuntu project | Ubuntu. (2019). Retrieved 17 July 2019, from https://ubuntu.com/about

[9] Terasic - SoC Platform - Cyclone. (2019). Retrieved 17 June 2019, from http://www.terasic.com.tw/cgi-bin/page/archive.pl?Language=English&CategoryNo=205&No=

[10] How To Set up SSH Keys on a Linux / Unix System - nixCraft. (2019). Retrieved 12 February 2019, from https://www.cyberciti.biz/faq/how-to-set-up-ssh-keys-on-linux-unix/

[11] Card, C. (2019). Creating and Updating SD Card. Retrieved 17 September 2019, from https://rocketboards.org/foswiki/view/Documentation/GSRD141SdCard

[12] Kerrisk, M. (2010). *The Linux Programming Interface*. 6th ed. San Francisco: Riley Hoffman.

[13] GCC, the GNU Compiler Collection- GNU Project - Free Software Foundation (FSF). (2019). Retrieved 19 August 2019, from https://gcc.gnu.org/

[14] Tanzilli, S. (2019). Install the ARM cross compiler toolchain on your Linux Ubuntu PC. Retrieved 8 June 2019, from https://www.acmesystems.it/arm9_toolchain

# Organización del Proyecto

- **Jorge Burgaleta Lobejón**:

Jorge Burgaleta se encargó de gran parte de la preparación de las placas y el correcto funcionamiento de este algoritmo. Primero se hizo una búsqueda de los distintos dataset disponibles y se selecciona el que mejor encajaba en este proyecto. Una vez elegido el dataset se buscan distintas opciones para realizar el proyecto, ya disponía de algunos conocimientos de Machine Learning y se realizan algunas pruebas con pequeños algoritmos de este tipo sobre todo en MatLab.

Tras analizar distintos algoritmos se llega a la conclusión de que el uso de la librería libsvm es la mejor opción pues tiene detrás un alto nivel de desarrollo por lo que los resultados que obtendremos serán mejores que si desarrolláramos un algoritmo de este tipo desde cero.

Posteriormente, ya con la librería libsvm se prepara el dataset elegido para que libsvm pueda leerlo y se realiza una primera ejecución del entrenamiento sobre la versión en MatLab de libsvm, usando una máquina con Debian 8 y convirtiendo el dataset al formato necesario usando también MatLab. Para esta primera ejecución se usan fragmentos del dataset muchos más pequeños de los que usamos en la ejecución con las DE1-Soc y el nodo máster.

Tras esto se empieza a preparar las placas para la ejecución de la versión en C++ de libsvm, para ello no nos sirvió el sistema operativo que se desarrolló en el TFM en el cual nos basamos ya que la versión del kernel de Linux era demasiado antigua y una vez compilada el código de libsvm no se consigue que la placa ejecute el algoritmo. Tras esto se decide usar un sistema operativo nuevo, buscando varias opciones y decidiendo la versión de Ubuntu proporcionada por Terasic con una versión más actualizada del kernel de Linux.

Paralelamente a esta instalación del nuevo sistema operativo en las placas, realiza la ejecución del entrenamiento en la máquina master con Debian para poder comparar después los resultados.

Una vez se tenía el sistema operativo funcionando en una placa, se comienza con el desarrollo de los scripts bash para automatizar el proceso de entrenamiento y división del dataset y una vez estaba terminado se ejecuta el entrenamiento en solo una placa, deteniendo la ejecución debido al alto tiempo de ejecución.

Una vez estaba hecha la versión inicial de los scripts, se crea una imagen del estado actual de la DE1-Soc que estaba funcionando y posteriormente se graba esta imagen en las tarjetas de otra placa, en el resto de las placas esta imagen será grabada posteriormente, pues nos repartimos las placas para poder trabajar paralelamente.

Tras esto se van perfeccionando los scripts para además ofrecer cierta información por consola del estado de la ejecución en las DE1-Soc como el tiempo transcurrido en cada etapa de la ejecución, los nombres de las placas que se está usando y se van realizando las distintas ejecuciones. Tras esto desarrollo el script para el votador, utilizando instrucciones bash para la lectura desde los archivos generados en la predicción sumando la predicción de cada modelo generado y obteniendo la opción más predicha.

Una vez se tenían los resultados de todas las ejecuciones, ejecuta el script del votador con los resultados obtenidos en cada una de las pruebas y genera las tablas con todos los porcentajes y tiempos obtenidos tanto de las placas como del nodo master, incluyendo el resultado del votador, para que otro compañero aplicara estilos y formatos a dichas placas.

Por último, también colabora en la realización de la memoria del proyecto realizando el apartado 2.4 del estado del arte, los apartados 3.1, 3.2 y 3.3 consistentes en explicación y preparación del dataset y de la librería libsvm, y por último parte del punto 4.4 el cual consta de la comparativa de los resultados obtenidos en las distintas ejecuciones.

- **Luis Muñoz María:**

Luis Muñoz ha colaborado en las distintas partes de las que se compone este proyecto. Del Trabajo Fin de Grado se han obtenido los resultados que todos los componentes del grupo esperábamos desde el principio. Al inicio del proyecto todos los participantes se reunieron para aportar compromiso a la hora de trabajar y participar activamente de forma eficaz para lograr los resultados propuestos.

Inicialmente, se indica que se ha prestado especial labor de investigación para poder realizar el Trabajo Fin de Grado. Esta tarea de investigación se ha basado principalmente en dos partes, por un lado, en la ciberseguridad que se encarga del uso fraudulento de la red y los equipos. Por otro lado, del System On a Chip que integra varios elementos que poseen los sistemas informáticos.

En primer lugar, se ha encargado de comenzar con la réplica del montaje del clúster realizado por Mariano Hernández en su trabajo de TFM. Sobre dicha replica se basó nuestra idea inicial, atrayéndonos la idea de usar un clúster para un entorno de ciberseguridad. Durante esta fase de réplica del clúster, se tuvo que hacer una investigación de cómo realmente trabajan estos sistemas. Todo ello, con la finalidad de que posteriormente se pudiese centrar nuestro objetivo del uso de ciberseguridad sobre los clústeres.

En segundo lugar, se comenzó con la búsqueda del dataset con el que podríamos trabajar. Tras una larga búsqueda y análisis en conjunto, se escogió el dataset que más se adaptaba a lo que necesitábamos. Nuestro compañero Jorge, que ya tenía conocimientos en aprendizaje automático nos estuvo dando unas ideas de cómo funcionaba. Seguidamente, tras profundizar en los nuevos conocimientos adquiridos se comenzó a realizar pruebas en Matlab con diferentes algoritmos. Finalmente, se decidió que la mejor librería que se adaptaba a nuestro entorno era la Libsvm. Dicha librería es de código abierto y tiene una gran comunidad de desarrolladores, generándose unos resultados mucho mayores a los que podríamos conseguir nosotros mismos creando un algoritmo desde el inicio.

En tercer lugar, se introdujo el sistema operativo Debian en las diferentes tarjetas MicroSD que usan las placas DE1-SoC. Dicho sistema operativo se empleaba en el Trabajo Fin de Máster de Mariano Hernández. A continuación, se realizó la fase de compilación sin

detectar errores. No obstante, en la siguiente fase no fue posible la ejecución del algoritmo SVM, ya que la versión del Kernel que poseía el Debian era demasiado antigua. Sin embargo, se siguieron realizando pruebas en este proceso e intentando buscar alternativas con la finalidad de tener éxito. Finalmente, se escogió la versión de Ubuntu debido a que tiene la parte del Kernel de Linux más actualizada.

En cuarto lugar, una vez que se tuvo todo funcionando correctamente en una placa se volvió a grabar la nueva imagen en el resto de MicroSDs. Tras este proceso, se prosiguió a la realización de las ejecuciones con diferentes números de nodos obteniéndose datos desde el uso de una Workstation hasta la utilización del clúster de 6 nodos. Seguidamente, se ejecutó el votador sobre los diferentes datos obtenidos en la fase anterior. El resultado de los datos obtenidos en las dos fases anteriores (fase ejecución y fase de utilización del votador) fueron introducidos con estilo y formato en tablas con la finalidad de añadirlas en la memoria del Trabajo de Fin de Grado.

En último lugar, se comenzó con la memoria del proyecto definiéndose su estructura, índice y contenido, consiguiéndose así, una secuencia lógica. Posteriormente, se realiza el resumen y su respectiva traducción, así como los agradecimientos a todas aquellas personas que nos han ayudado a crecer académicamente y personalmente para afrontar el proyecto. Además, se ha realizado parte del capítulo de estado del arte destacando los puntos de ataques DDoS, Botnets, Machine Learning, SVM, Redes neuronales y Clústeres. Además, se colabora activamente en el capítulo de experimentación y, concretamente, en los apartados de ejecución del algoritmo y resultados obtenidos. Para finalizar, se indica que se realiza el análisis y redacción de los apartados de conclusiones y referencias.

- **Diego Maestre Vidal**

Quiero agradecer personalmente a mis compañeros el esfuerzo realizado, y aunque haya sido largo y complicado, hemos conseguido llevar a cabo el trabajo satisfactoriamente.

Voy a detallar las funciones que he llevado a cabo. Lo primero consistió en el estudio y comprensión del Trabajo de Fin de Master de Mariano Hernández [1], necesario para la creación de nuestro clúster, y utilización para dividir una imagen y repartir los datos en las distintas SoCs, se nos ocurrió la idea de utilizarlo para la división del Dataset.

Una vez tenía la idea de cómo funcionaba, procedimos a la replicación del clúster, en principio con 2 FPGAs y posteriormente, tras comprobar que funcionaba correctamente la experimentación, ampliamos a las 4 FPGAs que utilizaba.

Respecto a la elección del Dataset, se propuso en primera instancia el CTU-13, generado en la universidad CTU de la República Checa, pero Jorge, nos convenció de utilizar el actual KDDCUP ’99 ya que tenía el tamaño idóneo para poder fragmentarlo y enviarlo a las FPGAs además de un feedback amplio.

Se llevó a cabo la experimentación con OpenCL pero no nos fue posible, y por ello se instaló Ubuntu en cada una de las placas y se realizó la experimentación en C++. Llevé a cabo la instalación de Ubuntu en cada una de las placas y la configuración de los switches para poder correr el kernel, y comprobé su correcto funcionamiento.

Tras comprobar que funcionaba correctamente con una placa, se llevó a cabo la construcción de los scripts.

El laboratorio que en principio estaba compuesto por 4 FPGAs lo ampliamos a 6, para obtener mejores conclusiones gracias al Departamento de Arquitectura de Computadores y Automática (DACYA) que nos las facilitó.

Fuimos probando con distintos números de placas, empezando por 2 y acabando por 6, y guardando los resultados obtenidos tanto de los tiempos como de las tasas de aciertos en imágenes para posteriormente incluirlos en las tablas observadas en el apartado 4.3.

Una vez se obtuvo todos los resultados, llevé a acabo la realización algunas de las gráficas de comparación de resultados en el apartado 4.4.

Por último, en cuanto a la memoria, participó en los apartados 1.1, 1.2, 1.3, donde se introduce el proyecto a realizar y la portada. También se establece la comparativa entre los 2 proyectos y las diferenciaciones en el clúster en el apartado 3.3. Se construye las tablas con los resultados de la experimentación en el apartado 4.3 y realiza la parte 4.4.

Hay partes comunes que las hicimos entre los 3, porque necesitábamos trabajar de forma presencial, ya que el laboratorio se encontraba montado en mi casa y era tedioso tener que montarlo y desmontarlo continuamente.

# Capítulo 6: Anexos

## 6.1 Anexo 1

```
#!/bin/bash
CONFIG="de1_soc-cluster.conf"
RUNPATH="/root/"
 if [ $# != 1 ]; then
  echo "Uso: ./run_train_svm.sh [dataset file]"
  exit -1
fi
function get_time(){
  echo `date +%s%N`
}
 OUTPATH="/root/"
 # Lee la configuracion del cluster
echo -n "Reading cluster config..."
IFS=$'\n' read -d '' -r -a NODES < $CONFIG
 NUM_NODES=${#NODES[@]}
 echo "(num nodes: $NUM_NODES)"
 for ((i=0; i<$NUM_NODES; i++))
do
  echo " |- Node $i: ${NODES[i]}"
done
echo " |- done"
# Divide el argumento (dataset de entrada) en NUM_NODOS trozos
echo "Spliting $1 in $NUM_NODES part(s)..."
time0=$(get_time)
split -d -a 1  --additional-suffix=.train -n l/$NUM_NODES $1 data_
echo " |- done"
time1=$(get_time)
# Copia el dataset a cada nodo
echo "Copying $1 and executing..."
for ((i=0; i<$NUM_NODES; i++))
do
  echo -n " |- ${NODES[i]}:$RUNPATH ..."
  ssh root@${NODES[i]} 'echo "running" > state' &&
  ini=$(get_time) &&
  scp -q $1 data_$i.train root@${NODES[i]}:$RUNPATH &&
  end=$(get_time) &&
  let SCPTIME[i]=$end-$ini &&
  echo -n " |- Time copying to node $i (sec): " &&
  echo "${SCPTIME[i]} * 0.000000001" | bc -l &&
  ssh root@${NODES[i]} " ./launch_svm_train.sh data_$i.train" &
  echo "done"
done
echo " |- done"
# Espera a que se envien los resultados
echo "Waiting for termination..."
for ((i=0; i<$NUM_NODES;))
do
  state=$(ssh -T root@${NODES[i]} "cat state")
  if [ $state = "finish" ]; then
    echo " |- ${NODES[i]} finished"
    ssh -tt root@${NODES[i]} 'echo "idle" > state' &
    i=$((i + 1))
  fi
done
echo " |- done"
time3=$(get_time)

let tiempo_total=$time3-$time0
let tiempo_dividir=$time1-$time0
let tiempo_ejecutar=$time3-$time1

echo "  # Execution time (s):"
```

## 6.2 Anexo 2

```
#!/bin/bash

function get_time(){
  echo `date +%s%N`
}
EXECUTABLEPATH="/root"
MASTER="192.168.0.100"

mkdir -p $EXECUTABLEPATH/out

#state=$(cat state)
#if [ $state = "idle" ] || [ $state = "finish" ];
#then
  echo "running" > state
  time0=$(get_time)
  $EXECUTABLEPATH/svm-train $1
  time1=$(get_time)
  scp $1".model" burga@$MASTER:"/home/burga/tfg-d1soc-opencl-neuralnet/"$1".model"
  time2=$(get_time)
 #rm -rf /tmp/*

  let execTime=$time1-$time0
  let scpTime=$time2-$time1

  echo "$execTime" > $1".log"
  echo "$scpTime" >> $1".log"

#fi

echo "finish" > state
```

## 6.3 Anexo 3

```
#!/bin/bash
if [ $# != 1 ]; then
  echo "Uso: ./script_votador.sh [num_nodos]"
  exit -1
fi
num_lineas_predict=25000
NUM_NODES=$1
MITAD=$(($NUM_NODES/2))
declare -a nums_total
printf "%s\n" "${nums_total[@]}" > file0.txt
echo "Ejecutando votador for $NUM_NODES nodes"
for ((i=0; i<$NUM_NODES; i++))
do
  declare -a nums_$i
done
for ((i=0; i<$NUM_NODES; i++))
do
  mapfile nums_$i < out_$i.predict
        for ((j=0; j<$num_lineas_predict; j++))
        do
                nums_total[$j]=$((nums_total[$j]+nums_$i[$j]))
        done
done
for ((i=0;i<$num_lineas_predict; i++))
do
        if ((nums_total[$i] > $MITAD)); then
                nums_total[$i]=$((1))
else
                nums_total[$i]=$((0))
        fi
done
printf "%s\n" "${nums_total[@]}" > file.txt
num_aciertos=0
declare -a test_labels
mapfile test_labels < datos_test.labels
for((j=0;j<$num_lineas_predict; j++))
do
        if ((nums_total[$j] == test_labels[$j]));then
                num_aciertos=$((num_aciertos+1))
        fi
done
porcentaje=$(echo "scale=5; $num_aciertos/$num_lineas_predict" | bc)
porcentaje=$(echo "scale=5; $porcentaje*100" | bc)
echo "Aciertos: $num_aciertos/$num_lineas_predict"
echo "Porcentaje: $porcentaje"
```