JtR Cluster es un sistema distribuido y escalable que optimiza el cracking de contraseñas aprovechando integralmente recursos CPU/GPU heterogéneos utilizando JohnTheRipper. Su arquitectura centralizada gestiona la asignación dinámica de "hash jobs" a través de nodos distribuidos y permite monitorizar el sistema a través de una interfaz web.

1.- Descripción general

Este proyecto permite el reparto de tareas de cracking entre múltiples nodos de procesamiento, aprovechando tanto CPUs como GPUs, e integrando módulos de servidor, cliente, base de datos y una interfaz web para la gestión y monitorización de trabajos.

2.- Estructura del repositorio

- **BD.tgz**: Script SQL para la creación y despliegue de la base de datos (`miweb_db.sql`).
- **Cliente.tgz**: Código fuente, binarios y scripts de los nodos de procesamiento.
- **Server.tgz**: Código fuente y ejecutable del servidor principal de coordinación.
- **Web.tgz**: Archivos de la interfaz web, scripts PHP y logs de gestión.
- **Web-en.tgz**: Archivos de la interfaz web, scripts PHP y logs de gestión. Versión en Inglés.
- **john.tgz**: Binarios, configuraciones y recursos personalizados de John the Ripper.

3. Instalación

A continuación, se detallan los pasos de instalación y configuración tanto para el nodo servidor como para los nodos de procesamiento (clientes), partiendo de los archivos y scripts incluidos en el repositorio.

NODO SERVIDOR

1. Sistema Operativo y herramientas básicas

- Ubuntu Server 24.04 (o similar), instalado desde cero.
- Instalar utilidades y dependencias:

```
# apt update
# apt install build-essential libmysqlclient-dev libjson-c-dev
```

2.- Clonar el repositorio. Ejecuta el comando:

```
# git clone https://github.com/jmu809/JtR-Cluster.git
```

3. Base de datos MySQL

- Instalar y configurar MySQL:

```
# apt install mysql-server
# mysql -u root
CREATE DATABASE miweb_db CHARACTER SET utf8mb4 COLLATE utf8mb4_unicode_ci;
CREATE USER 'user1'@'localhost' IDENTIFIED BY 'user1';
```

```
GRANT ALL PRIVILEGES ON miweb_db.* TO 'user1'@'localhost';
FLUSH PRIVILEGES;
EXIT;
```

- Importa la estructura y datos:

```
# tar xfz BD.tgz
# mysql -u user1 -p miweb_db < miweb_db.sql
```

4. Servidor Web y PHP

- Instala Apache y PHP:

```
# apt install apache2 php libapache2-mod-php php-mysql
```

- Descomprime la web y establece los permisos:

```
# tar -xvzf Web.tgz -C /var/www/
# mkdir -p /var/www/html/uploads
# chmod 770 /var/www/html/uploads
# chmod +x /var/www/html/bin/*
```

5. John the Ripper y dependencias

- Instalar soporte OpenCL y MPI:

```
# apt install opencl-headers ocl-icd-opencl-dev libopenmpi-dev openmpi-bin
```

- Descargar y compilar John:

```
# git clone https://github.com/openwall/john -b bleeding-jumbo john
# cd john/src
#./configure --enable-opencl
# make -sj$(nproc)
```

6. Scripts y servicios

- Descomprimir y compilar servidor:

```
# tar xfz Server.tgz
# gcc -o server server.c -lmysqlclient -ljson-c -lpthread
```

7.- Ejecutamos el servidor

```
# ./server
```

NODO CLIENTE (PROCESAMIENTO)

1. Sistema Operativo y herramientas

- Ubuntu Server 24.04 o similar, preparado para GPU.
- Instalar utilidades:

```
# apt update
# apt install build-essential libmysqlclient-dev libjson-c-dev git pkg-config opencl-headers ocl-icd-opencl-dev
```

2.- Clonar el repositorio. Ejecuta el comando:

```
# git clone https://github.com/jmu809/JtR-Cluster.git
```

3. Drivers GPU (AMD)

- Para Open Source (Rusticl):

```
# apt install mesa-opencl-icd clinfo
# echo 'export RUSTICL_ENABLE=radeonsi' | sudo tee /etc/profile.d/rusticl.sh
# chmod +x /etc/profile.d/rusticl.sh
# source /etc/profile.d/rusticl.sh
```

- Para drivers propietarios:
  Descargar e instalar desde la web oficial de AMD, https://www.amd.com/en/support/download/linux-drivers.html.

```
# wget https://repo.radeon.com/amdgpu-install/6.4.1/ubuntu/noble/amdgpu-install_6.4.60401-1_all.deb
```

Instala los drivers

```
# apt install ./amdgpu-install_6.4.60401-1_all.deb
# amdgpu-install --usecase=opencl --no-dkms
```

4. Comprobación de GPUs

```
# clinfo | grep -i device
```

5. John the Ripper (GPU)

- Instalar soporte OpenCL y MPI:

```
# apt install opencl-headers ocl-icd-opencl-dev libopenmpi-dev openmpi-bin
```

- Descargar y compilar John:

```
# git clone https://github.com/openwall/john -b bleeding-jumbo john
# cd john/src
#./configure --enable-opencl
# make -sj$(nproc)
```

6. Scripts de procesamiento

- Descomprimir y compilar:

```
# tar -xvzf Cliente.tgz
# gcc -o procesamiento procesamiento.c -ljson-c -lpthread
# gcc -o gpu_report gpu_report.c -ljson-c
# gcc -o cpu_report cpu_report.c -ljson-c
```

7. Ejecutamos el nodo de procesamiento

```
# ./procesamiento
```

4.- Utilización

A continuación, se presenta un ejemplo práctico detallado para ilustrar el funcionamiento de JtR Cluster en un escenario real de criptoanálisis de contraseñas. Este caso de uso demostrará la capacidad del sistema para gestionar y ejecutar ataques de fuerza bruta en un entorno distribuido y heterogéneo. Para hacer las pruebas se ha utilizado el nodo servidor y tres nodos de procesamiento que cuentan cada uno con cinco tarjetas gráficas.

La infraestructura utilizada para esta demostración se compone del siguiente hardware:

- **Servidor:** Intel Xeon E5-2650 v2 @ 2.60GHz – 16 cores / 128 GB RAM / Ubuntu 24.04.

- **3 nodos de procesamiento**: Cada nodo (ubuser1, ubuser2, ubuser3) está configurado con un procesador Intel CPU G4560 3.50GHz / 8GB de RAM / 256 GB SSD, y cuenta con 5 x GPU Radeon RX 580 / 2304 Stream Processors / 8GB / OpenCL.

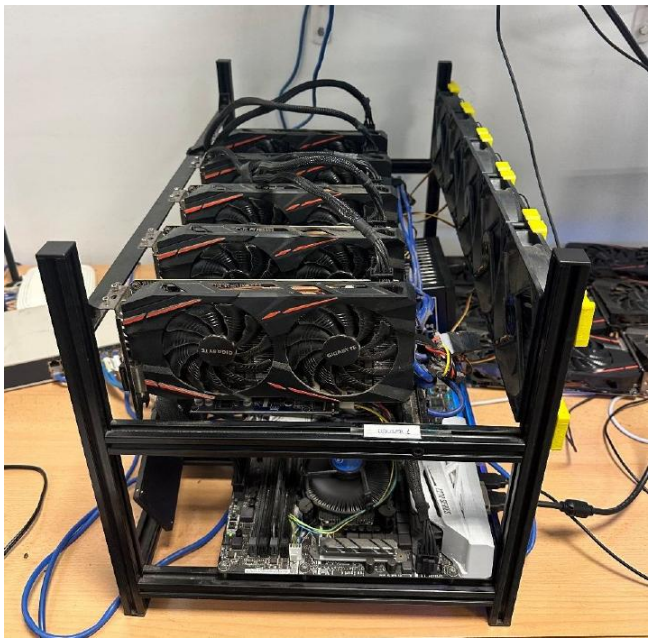

Figura 1.- Nodo de procesamiento con 5 tarjetas gráficas

Inicio del sistema

Una vez realizada la instalación, se ejecuta el *JTR Cluster* en el nodo servidor, actuando como coordinador principal. Permanece a la espera de conexiones de los nodos de procesamiento, escuchando a través del puerto 10000

```
root@ubuser1:~/0_server# ./server
● server.c listening on port 10000...
```

Figura 2.- Servidor escuchando en el puerto 10000

Y en en los diferentes nodos (ubuserX) se ejecuta *./procesamiento*. Al iniciar, cada nodo detecta su hardware (CPU/GPU) y envía esta información al servidor, el cual, una vez recibida esta información. Una vez recibida la informacion, registra el nodo en la base de datos (figura 4) y la información se encuentra disponible en la web del sistema (figura 5).

```
root@ubuser3:~/01_procesamiento# ./procesamiento
🚀 Starting procesamiento.c MULTI-GPU + MULTI-CPU
🔄 Registering GPUs:
🔄 Registering CPUs:
📝 Node registered and 'configuracion' file created.
💪 Detected 3 GPU(s) — starting 3 cracking thread(s)...
[GPU 4] ✊ No pending jobs. Waiting 10 seconds...
[GPU 5] ✊ No pending jobs. Waiting 10 seconds...
[GPU 6] ✊ No pending jobs. Waiting 10 seconds...
```

Figura 3.- Nodo procesamiento envía información del hardware al servidor

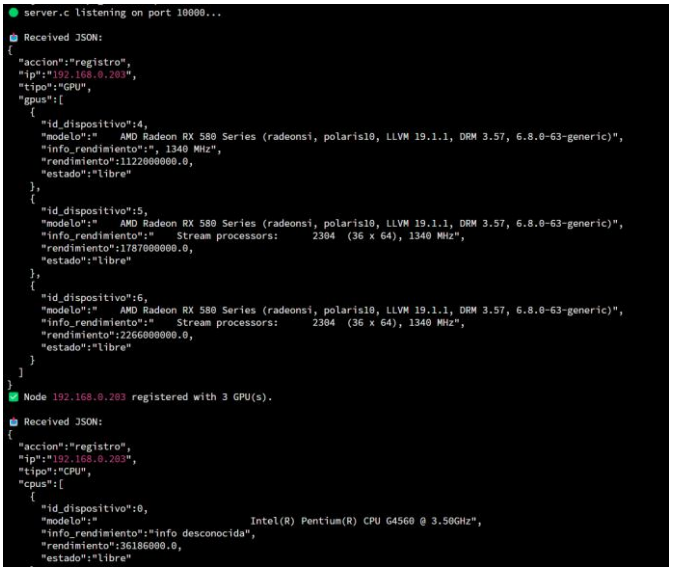

Figura 4.- Servidor recibe información del nodo procesamiento

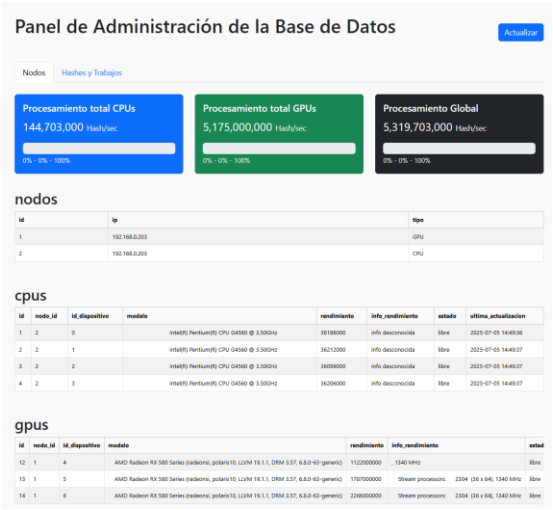

Figura 5.- Carga de la información de los nodos en la base de datos

Envío de hash

A continuación se muestra un ejemplo que detalla el proceso completo de ruptura de un hash introducido manualmente a través de la interfaz web.

1. Introducción del hash en la web. El usuario accede a la interfaz web e introduce manualmente el hash que desea **crackear**, seleccionando el charset (conjunto de caracteres) y la longitud estimada de la contraseña.

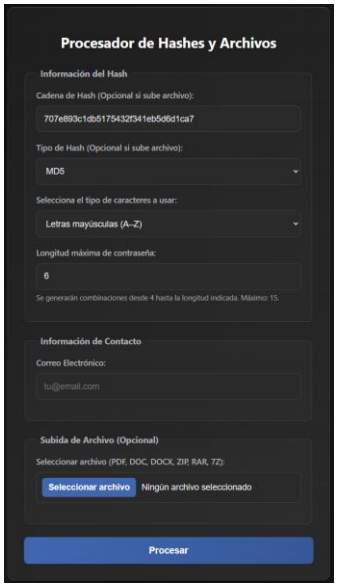

Figura 6.- Pagina web con datos del hash rellenos por le usuario

2. Registro en la base de datos. El servidor recibe la petición y almacena la información en la base de datos, donde se registra el hash y los parámetros asociados. Al registrarse el hash, el sistema crea automáciamente los diferentes “hash jobs” que se irán asignando dinámicamente a los diferentes nodos de procesamiento.

Figura 7.-. Base de datos con el hash cargado

3. Asignación de “hash jobs”. Los nodos de procesamiento solicitan trabajo al servidor. Este, tras consultar la base de datos, asigna a cada nodo un "hash job" (o un conjunto de "hash jobs") específico.

```
[ASSIGNED] Job 1 assigned to node 192.168.0.203 (GPU 5) at 2025-07-05 14:55:23
[ASSIGNED] Job 1 assigned to node 192.168.0.203 (GPU 4) at 2025-07-05 14:55:23
[ASSIGNED] Job 1 assigned to node 192.168.0.203 (GPU 4) at 2025-07-05 14:55:23
```

Figura 8.- Asignación de trabajo a la petición de trabajo del nodo

Los nodos ejecutan el ataque sobre el trabajo asignado utilizando los recursos disponibles (CPU/GPU).

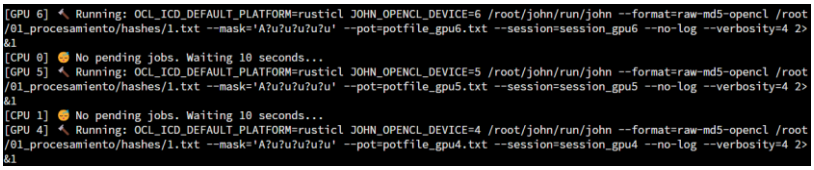

Figura 9.- Ejecución del ataque sobre el trabajo asignado

4.- Monitorizcion del proceso.

Durante el proceso, es posible consultar la interfaz web para ver el porcentaje de procesamiento en uso y el estado de cada trabajo, comprobando la actualización en tiempo real del sistema.

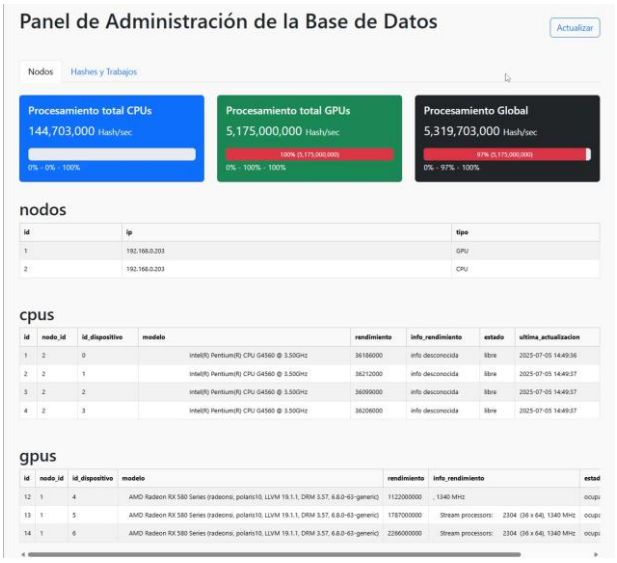

Figura 10.- Monitorización del procesamiento ocupado mientras se ejecutan los trabajos

5. Recepción y procesamiento de resultados.

Cuando un nodo finaliza su trabajo, comunica el resultado al servidor. Este, a su vez, actualiza la base de datos y registra el estado (found/not_found). Si la contraseña es encontrada, el servidor detiene el resto de trabajos asociados a ese hash y marca como "not found" todos los trabajos pendientes relacionados con dicho hash.

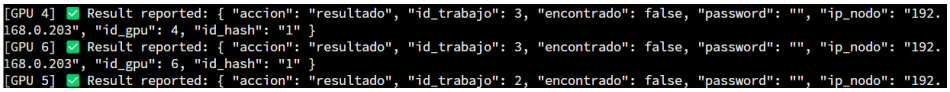

Figura 11.- Reporte de resultado de ejecución del trabajo al nodo servidor

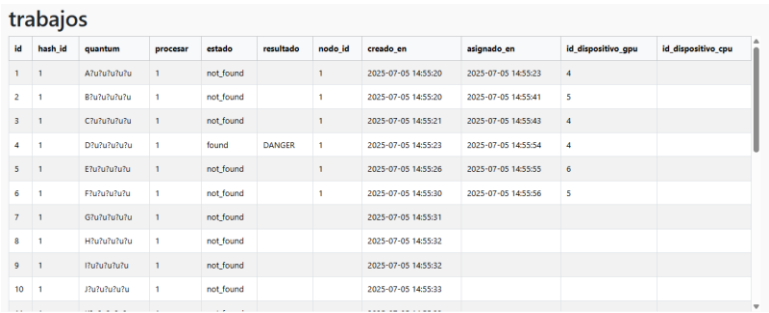

Figura 12.- Actualización de la base de datos después de recibir el resultado

6. Notificación al usuario. Finalmente, el servidor envía automáticamente un correo electrónico al usuario informándole de la contraseña recuperada.

5.- Autoría

Software desarrollado como parte del Trabajo de Fin de Grado en Ingeniería Informática de D. Juan Rafael Madolell TFG y dirigido por Dr. D. Julio Gómez López (Universidad de Almería, 2025).

Autores: D. Juan Rafael Madolell y Dr. D. Julio Gómez López (Universidad de Almería)