Desarrollo de un clúster heterogéneo de bajo consumo para inferencia sobre redes neuronales

Development of a low-power heterogeneous cluster for neural network inference

Trabajo Fin de Grado

Curso 2023-2024

Autor

Adrián Martín Tiscar

Directores

Francisco D. Igual Peña

Sandra Catalán Pallarés

Grado en Ingeniería Informática

Facultad de Informática

Universidad Complutense de Madrid

Desarrollo de un clúster heterogéneo de bajo consumo para inferencia sobre redes neuronales

Development of a low-power heterogeneous cluster for neural network inference

Trabajo de Fin de Grado en Ingeniería Informática

Autor

Adrián Martín Tiscar

Directores

Francisco D. Igual Peña

Sandra Catalán Pallarés

Convocatoria: Junio 2024

Grado en Ingeniería Informática

Facultad de Informática

Universidad Complutense de Madrid

Junio de 2024

Todos los ficheros utilizados en este TFG se pueden encontrar en el siguiente repositorio de GitHub AdrianMartinT/TFG (github.com).

Resumen

Desarrollo de un clúster heterogéneo de bajo consumo para inferencia sobre redes neuronales

Este proyecto plantea el desarrollo de un clúster heterogéneo de bajo consumo energético, para optimizar la ejecución de procesos de inferencia en redes neuronales. El clúster se construye con Raspberry Pi, con un nodo especialmente configurado con un acelerador Google Coral. El objetivo es mejorar la eficiencia de las inferencias neuronales al usar la Unidad de Procesamiento Tensorial (TPU - del inglés Tensor Processing Unit) Google Coral como recurso a disposición del usuario.

La administración del clúster se lleva a cabo a través del sistema de gestión de trabajos SLURM, que distribuye y controla las tareas en los distintos nodos. Los sistemas de colas como SLURM, aunque muy útiles, no dan un soporte para el manejo de TPUs, por ello este trabajo aborda su manejo para ponerlo a disposición del usuario. Esto se logró mediante una implementación que permite a SLURM gestionar las TPU como recursos asignables, consiguiendo usarlas de forma concurrente y transparente, logrando así el objetivo del proyecto.

El resultado es un clúster que aprovecha la diversidad de recursos disponibles, desde las Raspberry Pi estándar hasta el nodo con acelerador y las TPU. Este enfoque permite una ejecución más rápida y eficiente de las inferencias en redes neuronales. La propuesta busca ayudar con soluciones eficientes en términos energéticos y temporales, contribuyendo al avance en investigación de inteligencia artificial y procesamiento neuronal.

Palabras clave

Clúster, Inferencia, SLURM, TPU, Rendimiento

Abstract

Development of a low-power heterogeneous cluster for neural network inference

This project proposes the development of a low-energy heterogeneous cluster to optimize the execution of inference processes in neural networks. The cluster is built with Raspberry Pi, with a specially configured node equipped with a Google Coral accelerator. The goal is to improve the efficiency of neural inferences by using the Google Coral Tensor Processing Unit (TPU) as a resource available to the user.

The cluster management is carried out through the SLURM job management system, which distributes and controls tasks across the various nodes. Queue systems like SLURM, although very useful, do not support handling TPUs. Therefore, this project addresses their management to make them available to the user. This was achieved through an implementation that allows SLURM to manage TPUs as assignable resources, enabling their concurrent and transparent use, thereby achieving the project's objective.

The result is a cluster that leverages the diversity of available resources, from standard Raspberry Pi units to the node with an accelerator and TPUs. This approach allows for faster and more efficient execution of inferences in neural networks. The proposal aims to provide efficient solutions in terms of energy and time, contributing to advancements in artificial intelligence research and neural processing.

Keywords

Cluster, Inference, SLURM, TPU, Performance

ÍNDICE

ÍNDICE DE FIGURAS

Índice de Scripts y Ficheros

Capítulo 1 - Introducción

1.1 Introducción

Un clúster informático [1] es un conjunto de computadoras conectadas entre sí para trabajar conjuntamente, lo que permite aumentar la potencia y gestionar mejor las cargas de trabajo. Son una de las innovaciones más destacadas en el ámbito de la informática de altas prestaciones. Los clústeres han revolucionado la forma en que abordamos problemas computacionales complejos gracias a su mayor potencia de cálculo y el poder paralelizar tareas, entre otros aspectos. Esto ha permitido e impulsado avances significativos en diversas áreas de la ciencia, la industria y la investigación.

Esta capacidad de trabajo a nivel computacional nos ha proporcionado un potencial asombroso. Desde la simulación de fenómenos científicos complejos hasta el análisis de enormes conjuntos de datos en el campo del aprendizaje automático y la inteligencia artificial, los clústeres informáticos han permitido abordar desafíos que antes parecían insuperables. Los avances en áreas como la medicina, la meteorología, la física y la ingeniería se han acelerado gracias a la capacidad de realizar cálculos y experimentos, usando la capacidad proporcionada por los clústeres.

En resumen, los clústeres informáticos representan una herramienta muy útil en nuestra era. Su capacidad para combinar recursos y habilidades de manera colaborativa ha ampliado las posibilidades de la informática y ha ofrecido un nuevo camino para descubrimientos y desarrollos.

Para poder aprovechar toda la capacidad de un clúster es conveniente disponer de un sistema que organice los nodos y se encargue de distribuir los trabajos y programas que el clúster va a ejecutar. En nuestro caso, hemos usado SLURM [2], un programa de gestión de trabajos que distribuye eficientemente los cálculos entre los nodos para aprovechar al máximo la potencia de los recursos disponibles.

SLURM es un sistema de gestión de clústeres y programación de trabajos de código abierto el cual es perfecto para el despliegue de clústeres cuyos nodos ejecutan el sistema operativo Linux. Es altamente escalable y puede manejar fallos de manera eficiente. SLURM no necesita modificaciones significativas en el kernel para funcionar, lo que hace que sea muy práctico y autónomo.

SLURM tiene tres funciones principales que lo hacen destacar. En primer lugar, se encarga de asignar acceso a los recursos, como nodos de cómputo, de manera exclusiva o no exclusiva, según las necesidades de los usuarios durante un período de tiempo. Esto asegura que todos puedan realizar su trabajo sin problemas. En segundo lugar, proporciona un marco realmente útil para iniciar, ejecutar y supervisar trabajos, especialmente aquellos

que son paralelos, en los nodos que se les asignan. Y por último, maneja la asignación de recursos y evita conflictos, todo gracias a cómo gestiona la cola de trabajos pendientes.

Los sistemas de colas como SLURM soportan la definición y programación de recursos genéricos GRES (del inglés - Generic RESources) [20] entre los que se encuentran las GPU o dispositivos CUDA entre otros, pero no permite de forma nativa gestionar las TPU.

En resumen, SLURM es una herramienta muy útil para optimizar la administración de clústeres y la programación de trabajos. Sin duda, es una herramienta de gran utilidad para cualquiera que trabaje con clústeres y necesite una solución confiable y escalable.

La eficiencia en informática siempre ha sido un parámetro de gran importancia, la energía y el tiempo no son ilimitados, y queremos poder resolver problemas en un tiempo razonable y a cambio de un consumo energético contenido y adaptado a las tareas a realizar. Por ello es tremendamente útil reducir gastos en tiempo y energía para cualquier tipo de cómputo, por lo que, cualquier avance o investigación hacia una mayor eficiencia siempre ha sido de gran interés para los investigadores.

1.2 Objetivos

El objetivo de este Trabajo Final de Grado es el de poder tratar las TPU conectadas a los nodos como un recurso más desde SLURM. Esto le dará al sistema gestor de colas la nueva posibilidad de usar aceleradores a su antojo para poder aprovechar la capacidad de las TPU. Esto ofrecerá a los usuarios más control y capacidad de decisión a la hora de trabajar con distintos tipos de tareas en un clúster, sobre todo si se trata de inferencia de redes neuronales.

Este objetivo, podemos tratarlo logrando los siguientes hitos:

1. Estudiar el funcionamiento y configuración de redes, como comunicar entre sí varias máquinas, acceso a internet, ficheros compartidos, etc.
2. Aprender sobre el funcionamiento de las herramientas a usar durante el proyecto, así como de su configuración.
3. Aplicar estos conocimientos a Raspberry Pi para tener un control sobre ellas y tenerlas como nodos de un clúster con el que podremos trabajar.
4. Aplicar una forma de control sobre las TPU dentro del clúster ofreciendo a los usuarios la posibilidad de aprovecharlo.

1.3 Plan de trabajo

El plan a seguir ha sido dividido en cuatro fases:

La primera será simular las máquinas reales (Raspberry Pi) con máquinas virtuales, con el objetivo de entender y gestionar las herramientas y recursos necesarios para la elaboración de este proyecto (dichas herramientas serán enunciadas a lo largo de este documento). Será necesario poder manejar las máquinas e instalar y hacer funcionar las herramientas. Este estudio de conexión con las máquinas virtuales y configuración de los programas necesarios para lograr el objetivo tendrá un tiempo estimado de 120 horas.

La segunda fase consistirá en replicar este proceso con las raspberries con las que se generará el clúster, sabiendo que sobre una de ellas se encuentra el acelerador que da interés a este trabajo. Habrá que instalar y configurar de nuevo los programas y hacer alguna prueba sobre las mismas. Esto tendrá un tiempo estimado de 70 horas.

Y la tercera fase, será en configurar la gestión de la TPU desde SLURM para poder darle al usuario la capacidad de usarla y aprovechar las ventajas que nos da, aparte de comprobar dichas ventajas. Esta fase se basa en búsqueda de información y pruebas, y tendrá una duración estimada de 60 horas.

Y la última fase se basará en documentar y estudiar los resultados así como de hacer comparaciones de tiempos. Esta fase final tendrá una duración estimada de 50 horas.

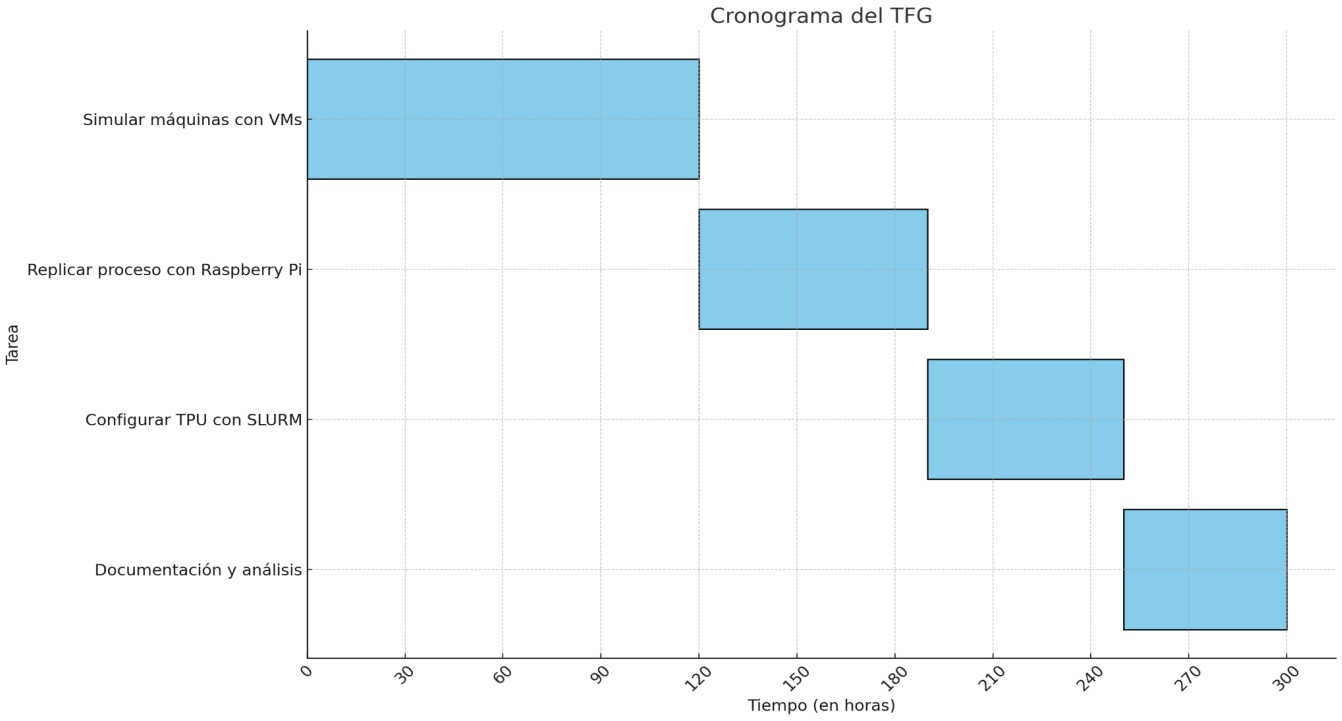

Figura 1.3 - Cronograma del TFG

1.4 Estructura del documento

Este documento se dividirá en varias secciones:

- El segundo capítulo consta de los conceptos clave y necesarios para entender el proyecto. En el veremos que es un sistema de colas, como funciona SLURM y en que nos ayuda. También comprenderemos la utilidad de la TPU y conceptos importantes de IA como las redes neuronales y la inferencia. Por último veremos detalles sobre los entornos donde vamos a llevar a cabo nuestro proyecto, las máquinas virtuales y las Raspberry Pi.
- En el tercer capítulo se mostrará cómo se ha desarrollado el entorno y la configuración de todos los elementos necesarios en las máquinas virtuales. Este capítulo tendrá información vital, ya que este mismo proceso luego se llevará a cabo en las Raspberry.
- En el cuarto capítulo se verá la implementación en las Raspberry y el uso de la TPU de forma transparente desde SLURM. Aquí veremos distintas alternativas de uso y las diferencias que nos ofrece.
- Finalmente en el quinto capítulo veremos las conclusiones a las que hemos llegado tras la realización de este trabajo.

Capítulo 2 - Conceptos generales

En este capítulo hablaremos de los conceptos necesarios para la realización de este trabajo, mencionaremos las herramientas que usaremos y los motivos que dan interés a este proyecto.

2.1 Sistemas de colas

Un sistema de colas es un modelo que se encarga de gestionar a los clientes que llegan demandando un servicio, dirigiendo en que orden y donde son atendidos. Se encarga principalmente de ordenar y optimizar tareas. Proporciona un marco para iniciar, ejecutar y supervisar el trabajo en un conjunto de nodos asignados. Y su característica más interesante es que arbitra la disputa por los recursos mediante la gestión de una cola de trabajos pendientes.

Existen varios sistemas de gestión de colas utilizados en diferentes entornos. Entre los más comunes se encuentran PBS (Portable Batch System), SLURM (Simple Linux Utility for Resource Management) y HTCondor. Estos sistemas permiten a los usuarios enviar trabajos a una cola, gestionar la prioridad y asignación de recursos, y monitorear el estado de los trabajos.

Las características que se esperan de un sistema de gestión de colas son escalabilidad, flexibilidad, eficiencia, fiabilidad y facilidad de uso. Cada sistema ofrece diferentes niveles de estas características, adaptándose a distintas necesidades. Por ejemplo, SLURM es altamente escalable y eficiente, mientras que HTCondor destaca por su flexibilidad en entornos distribuidos.

2.2 SLURM

SLURM (Simple Linux Utility for Resource Management) [2] es un programa de código abierto que se encarga de la administración de un conjunto de cargas de trabajo. Está diseñado principalmente para gestionar y distribuir tareas computacionales en un conjunto de computadoras o nodos en un entorno de cómputo de alto rendimiento o HPC (High-Performance Computing) y es altamente escalable.

SLURM permite a los usuarios enviar tareas (programas, scripts, etc) al sistema, especificando requisitos de recursos, prioridades y otros parámetros. Luego, el planificador de SLURM asigna recursos a los trabajos en función de los recursos disponibles, las prioridades de los trabajos y políticas definidas por el usuario.

2.2.1 Características de SLURM

SLURM es una herramienta muy usada por las diversas características y funcionalidades que ofrece en entornos HPC. Es usado, por ejemplo, por Summit, uno de los supercomputadores más potentes del mundo, y por MareNostrum, supercomputador en Barcelona. Sus características más importantes son:

- Planificación de trabajos: SLURM permite especificar requisitos como CPU, memoria u otros parámetros, y el planificador se encarga de asignar dichos recursos en función de la disponibilidad y las políticas definidas por el usuario.
- Eficiencia y recursos: SLURM comprueba el uso y cantidad de recursos en tiempo real, asignándolos de forma eficiente para optimizar el rendimiento. También gestiona recursos como GPU y FPGA, dando la posibilidad al usuario de especificarlos si así lo requiere en los scripts de lanzamiento.
- Escalabilidad: SLURM es capaz de gestionar clústeres de miles de nodos, es altamente escalable e ideal para grandes instalaciones de HPC.
- Accounting (contabilización): SLURM incluye un sistema para llevar un registro del uso de recursos, categorizando por usuario, trabajo y partición, lo cual es útil para el análisis de costes.
- Tolerancia a fallos: SLURM está diseñado para ser tolerante a fallos, y tiene mecanismos para recuperar trabajos o reanudarlos desde puntos de control.
- Colas y prioridades: SLURM utiliza colas para la gestión de trabajos, y nos permite definir múltiples colas con distintas prioridades. Los trabajos pueden ser enrutados a colas concretas dependiendo de sus requisitos.

2.2.2 Componentes

Los componentes de SLURM interactúan entre sí, comunicándose a través de la red interna (los nodos deben conocerse entre ellos), para garantizar una eficiente administración de recursos y la correcta ejecución de los trabajos en el clúster. Se divide en tres componentes principales:

- Controlador de SLURM (slurmctld): se encarga de gestionar y coordinar todas las operaciones del clúster, planificando y asignando los trabajos a los distintos nodos de cómputo. El nodo centralizado donde se despliega el controlador es imprescindible y de gran importancia para la administración del clúster, y nos referiremos a él a continuación como *frontend*.
- Demonios de gestión (slurmd): son los encargados de procesar los trabajos recibidos por el controlador, por lo que necesitan comunicación con el mismo, para recibir trabajos y enviar el estado de la ejecución. Típicamente, este elemento se encuentra replicado en los nodos de cómputo (normalmente el frontend no se considera como tal), y permanece en constante comunicación con el controlador.

- Base de datos interna de SLURM: es donde se almacena la información sobre la configuración, los nodos y las tareas, usada por el controlador y los demonios para coordinarse.

2.2.3 Funcionalidad y gestión de recursos

SLURM se encarga de la planificación de trabajos, asigna recursos para la realización de los mismos optimizando el uso de estos recursos, como CPU y memoria. Hace un seguimiento del estado de los trabajos y se encarga de su ejecución. También se ocupa de la administración del acceso al clúster verificando las credenciales de los usuarios.

SLURM utiliza colas para gestionar los trabajos y la asignación de los recursos disponibles. Dichas colas se definen en el archivo de configuración de SLURM bajo el nombre 'PartitionName'. Podemos definir las particiones que queramos y asignar los nodos que estarán en ellas. Esta característica nos ayudará a poder definir nodos con ciertas características, y así poder usarlo cuando lo queramos.

2.2.4 Objetivos de un sistema de colas

Los objetivos de este programa son la eficiencia, minimizando de forma equitativa los tiempos de ejecución de las tareas haciendo una asignación de tareas y recursos eficiente. La escalabilidad, para poder hacer crecer el clúster todo lo que sea necesario. La tolerancia a fallos, minimizando el impacto en los datos. Y la facilidad de uso, ayudando al usuario dándole una interfaz consistente para gestionar sus trabajos y simplificando su uso ocultando la complejidad del clúster y los recursos.

2.2.5 Scripts de lanzamiento

Los scripts de lanzamiento de SLURM comienzan con una serie de campos que permiten configurar cómo queremos lanzar el trabajo. Con estos campos podemos especificar la reserva de recursos o como queremos ver la salida entre otras cosas. Dichos campos comienzan con #SBATCH. Estos scripts están escritos en bash o sh. A continuación voy a detallar un script de lanzamiento de ejemplo:

```
#!/bin/bash
#SBATCH   --job-name=test_tfg                          #   Job   name
#SBATCH  --output=output.txt                  #  Standard  output  file
```

```
#SBATCH  --error=error.txt                    #  Standard error file
#SBATCH  --nodes=1                               #  Number of nodes
#SBATCH --ntasks-per-node=1              # Number of tasks per node
#SBATCH --cpus-per-task=1             # Number of CPU cores per task
#SBATCH --time=1:00:00               # Maximum runtime (D-HH:MM:SS)
#SBATCH --gres=gpu:2                              # Reserve 2 GPUs
#SBATCH --cpus-per-task=8                      # 8 CPUs per task
#SBATCH --mem=32G                                 # 32 GB of memory
python mi_programa.py
```

Script 2.2.5 - Script de lanzamiento de SLURM ejemplo

En este script podemos diferenciar varios campos:

- '--job-name': Define el nombre del trabajo.
- '--output' y '--error':Definen archivos para la salida estándar y la salida de error.
- '--nodes': Define el número de nodos a utilizar.
- '--ntask-per-node': Define el número de tareas por nodo.
- '--cpus-per-task': Define el número de núcleos de CPU que se asignan por tarea.
- '--time': Define el tiempo máximo de ejecución para el trabajo.
- '`--gres`': Define el número de GPUs que se reservan para el trabajo.
- '`--cpus-per-task`': Define el número de CPUs que se asignan por tarea.
- '`--mem`': Define la cantidad de memoria.

2.2.6 Comandos de SLURM

SLURM nos proporciona una gran variedad de comandos para la gestión de trabajos, colas, nodos y recursos. Vamos a explicar los más esenciales:

- sbatch y srun: Envía un trabajo a los nodos de trabajo.
- squeue: Muestra la lista de trabajos encolados con su información (tiempo, nodo asignado, etc).
- scancel <id_job>: Cancela el trabajo con el identificador que le demos.
- sinfo: Nos muestra información sobre el clúster, sus nodos y sus recursos.
- scontrol: Con sus distintas opciones podemos administrar el estado del clúster

2.2.7 Munge

Munge [6] es un programa específico para autenticación y cifrado entre los nodos de clústeres. Su funcionamiento principal es el de generar y validar tokens con el propósito de verificar la identidad de los usuarios en los nodos de un clúster. Básicamente es una herramienta que nos permite tener autenticación segura con eficiencia en sistemas como el nuestro. Además Munge está integrado con SLURM, lo cual nos permite mayor facilidad en su uso y configuración, y que funcione sin problemas y con eficiencia.

2.3 Recursos computacionales (CPU, GPU y TPU)

Una CPU (del inglés - Central Processing Unit) [3] es un dispositivo hardware fundamental en cualquier dispositivo programable. Tiene la función de interpretar las instrucciones de los programas informáticos. Los computadores actuales pueden contener más de una CPU.

Una GPU (del inglés - Graphics Processing Unit) [4] es un dispositivo hardware que se encarga del procesamiento gráfico u operaciones en coma flotante, con el objetivo de aligerar la carga de la CPU.

Una TPU (del inglés - Tensor Processing Unit) [5] es un dispositivo hardware creado con el objetivo de acelerar operaciones de cómputo relacionadas con la inteligencia artificial. Como su nombre indica, está especializado en operaciones matriciales y cálculos vectoriales en paralelo, las cuales son una gran parte de operaciones que se realizan en la inteligencia artificial, por ello. Una TPU es mucho más rápida y eficiente que una CPU o una GPU para operaciones relacionadas con la IA, ya que está especializada en ello, aunque la CPU es más versátil y la GPU es mejor para gráficos.

En nuestro caso le dedicaremos especial atención a la TPU, ya que es la unidad a la que vamos a dar uso e intentar aprovechar su potencial para operaciones de inferencia de redes neuronales. Más concretamente, nos vamos a centrar en la Google Coral TPU.

Google Coral es un dispositivo USB especializado para acelerar los cálculos de inferencia en machine learning. Nos proporciona un buen equilibrio entre alto rendimiento y bajo consumo energético. Cuenta con un procesador Google Edge TPU, una ASIC (del inglés - Application Specific Integrated Circuit) diseñada para ejecutar modelos de aprendizaje automático. Es capaz de realizar 4 trillones de operaciones por segundo con solo 2W de potencia. Es compatible con el framework de TensorFlow, lo cual usaremos para nuestras pruebas.

La página de Coral [21] nos ofrece un ejemplo sencillo (que usaremos más adelante) para poner a prueba el acelerador. El primer paso es instalar las dependencias necesarias:

```
echo "deb https://packages.cloud.google.com/apt
coral-edgetpu-stable main" |
sudo tee /etc/apt/sources.list.d/coral-edgetpu.list
```

```
curl https://packages.cloud.google.com/apt/doc/apt-key.gpg |
sudo apt-key add -
sudo apt-get update
sudo apt-get install libedgetpu1-std
sudo apt-get install python3-pycoral
```

Tras esto tenemos que crear un directorio, clonar los elementos necesarios de su repositorio en GitHub y probar a ejecutarlo.

```
mkdir coral && cd coral
git clone https://github.com/google-coral/pycoral.git
cd pycoral
bash examples/install_requirements.sh classify_image.py
python3 examples/classify_image.py \
--model
test_data/mobilenet_v2_1.0_224_inat_bird_quant_edgetpu.tflite \
--labels test_data/inat_bird_labels.txt \
--input test_data/parrot.jpg
```

Comando 2.3 - Lista de comandos para la ejecución del ejemplo de Coral

Una vez ejecutado nos mostrará una salida como esta que nos confirmará que se ha ejecutado correctamente.

```
----INFERENCE TIME----
Note: The first inference on Edge TPU is slow because it includes
loading the model into Edge TPU memory.
11.8ms
3.0ms
2.8ms
2.9ms
2.9ms
-------RESULTS--------
Ara macao (Scarlet Macaw): 0.75781
```

Salida 2.3 - Resultado de la ejecución del ejemplo de Coral

2.4 IA, Redes neuronales e inferencia

La Inteligencia artificial o IA es el término que se le da a las aplicaciones informáticas que intentan simular el comportamiento humano en la realización de tareas, es un campo que combina la computación y conjuntos de datos para la resolución de problemas. Abarca los subcampos del aprendizaje automático y el aprendizaje profundo. Este campo se compone de algoritmos de IA que buscan crear sistemas expertos que hagan predicciones o clasificaciones basadas en datos.

Una red neuronal [7] es un modelo basado en el funcionamiento del cerebro humano, el cual tiene neuronas interconectadas por las cuales pasa la información. Estas neuronas se organizan en capas, su número es variable así como el número de neuronas por capa. La red recibe datos de entrada y va pasando por las distintas capas atravesando sus neuronas a través de ponderaciones, las cuales son dadas en la etapa de entrenamiento. Para que una red neuronal funcione es necesario entrenarla con numerosos datos de entrada y los resultados correspondientes, al ir viendo la entrada y lo que debería devolver va ajustando sus valores para poder evaluar distintas entradas. Tras la etapa de entrenamiento podemos usar la red neuronal, a esto le llamamos inferencia.

La inferencia de redes neuronales consiste en la utilización de una red neuronal entrenada para hacer predicciones o tomar decisiones basadas en nuevos datos de entrada. En una red neuronal lo primero que hay que hacer es entrenarla usando un conjunto de datos de entrenamiento para aprender patrones y relaciones en los datos. Una vez entrenada es cuando usamos el proceso de inferencia para aplicar ese conocimiento a datos nuevos.

Durante la fase de inferencia, la red neuronal toma los datos de entrada y los procesa a través de sus capas (neuronas) y conexiones ponderadas para generar una salida. Esta salida puede ser una clasificación, una predicción numérica, una respuesta a una pregunta, o cualquier otro resultado para el cual la red haya sido diseñada.

Cuando hablamos del coste de la inferencia nos referimos al costo computacional (cuanto va a tardar) y al costo energético. Según donde se ejecute esta tarea podemos tener grandes diferencias respecto al coste, según pasamos de CPU a GPU y a TPU vamos aumentando el rendimiento de estas tareas, reduciendo así su tiempo de ejecución y haciendo que sean más eficientes, pero por otro lado, aumenta también el coste económico.

2.5 Máquinas virtuales

Una máquina virtual (VM - por sus siglas en inglés) [8] en un software que simula una computadora en la cual podemos tener otro sistema operativo que el de la máquina donde la simulamos, otro sistemas de ficheros y aplicaciones independientes. Tienen grandes ventajas como el permitirnos poder simular distintas máquinas en una sola. También se pueden usar para probar aplicaciones, sistemas operativos, y se pueden clonar fácilmente.

Las máquinas virtuales también tienen sus desventajas, principalmente el coste de recursos, a fin de cuentas están reservando parte del procesador y la memoria del sistema principal para ellas, lo que puede afectar en el rendimiento de las tareas que hagamos.

En nuestro caso las usaremos para hacer un entorno simulado en el que probar la configuración de red entre las máquinas y la instalación y el funcionamiento de los distintos programas que necesitaremos. Nos servirá como paso previo a usar las Raspberries, y así poder entender mejor el funcionamiento de los componentes y probarlos.

2.6 Raspberry Pi

La Raspberry Pi [9] es una pequeña computadora, que puede conectarse a un monitor y usarse con teclado y ratón y funciona con un sistema operativo Linux. Este pequeño dispositivo es capaz de llevar a cabo la mayoría de las tareas típicas de una computadora de escritorio, incluyendo navegación en internet, reproducción de videos, manejo de aplicaciones, así como también programación. A diferencia de las computadoras convencionales, tienen una potencia inferior y suelen utilizarse para propósitos específicos en lugar del propósito general de las computadoras.

Tiene la capacidad de interactuar con el entorno exterior (mediante la instalación de sensores) y puede ser utilizada en una amplia gama de proyectos, como reproductores de música, pequeños robots y detectores de movimiento. Su arquitectura está basada en la tecnología ARM (del inglés - Advanced RISC Machine), lo que la hace eficiente con respecto a energía y coste.

La Raspberry Pi tiene varias interfaces de red, lo que la permiten conexiones en redes locales y a internet. Además de sus interfaces también dispone de una variedad de puertos y conectores, como USB y HDMI. Tiene la capacidad de interactuar con el entorno a través de la instalación de sensores.

Capítulo 3 - Entorno de pruebas basado en máquinas virtuales

Antes de realizar el trabajo con máquinas físicas reales (Raspberry Pi), se ha simulado el trabajo con máquinas virtuales con el objetivo de hacer pruebas sobre la conexión de red de los nodos y de los distintos softwares que se usarán. Tras esto se llevó este conocimiento a las Raspberry Pis (véase figura 3.0) ofrecidas por la UCM e interconectadas entre ellas. En una de ellas es donde estaba instalado el acelerador, y por tanto, donde se han ejecutado las pruebas de inferencia. He trabajado con ellas remotamente mediante el comando ssh.

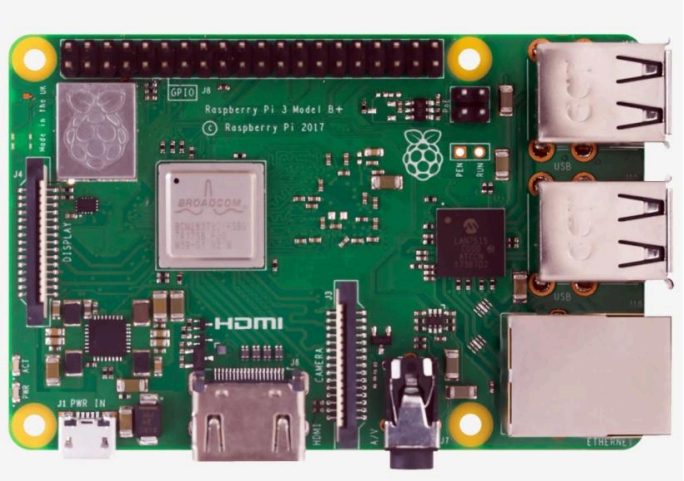

Figura 3.0 Raspberry Pi Fuente: [18]

3.1 Conexión y topología de la red

Primero hemos creado 4 máquinas virtuales con el sistema operativo centOS distribuido por Red Hat [10] de Linux. Ha sido necesario habilitar la conexión a internet para poder descargar los distintos programas necesarios para este trabajo.

Hemos conectado las máquinas con una red local para que puedan comunicarse entre ellas como se puede observar en la figura 3.1. También se han configurado con nombres todas las máquinas para que puedan acceder al resto sin necesidad de utilizar la dirección IP, editando el archivo /etc/hosts

Para que las IP se guardaran de forma persistente y se mantuvieran incluso cuando se apagasen las máquinas, se ha editado el archivo /etc/sysconfig/network-scripts/ifcfg-eth1 (eth1 era por donde había conectado las máquinas) añadiendo la variable IPADDR=<IP>. Se ha comprobado que la configuración persistente funciona y todos los nodos se conectan entre sí.

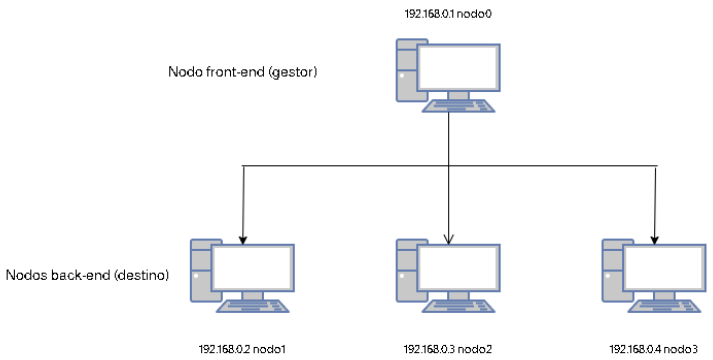

Figura 3.1 Diagrama de la topología de las máquinas virtuales

3.2 Munge

Para hacer la instalación se ha usado la guía que ofrecía el propio Munge en su página de github [11] en la cual se muestran los pasos de instalación y configuración. Primero se descargo el fichero comprimido y al descomprimirlo hay que navegar al carpeta donde esta el make, pero antes de ejecutarlo se preparó la configuración con:

```
./configure  --prefix=/usr --sysconfdir=/etc --localstatedir=/var --runstatedir=/run
```

Esto se hace para definir las rutas donde se almacenarán los distintos recursos de munge, sysconfdir es donde se almacenará la clave, localstatedir es donde se almacenará la semilla y los logs de ejecución, y runstatedir es donde estará el socket para que los clientes se conecten.

En una máquina hubo un problema a la hora de encontrar un archivo de configuración necesario para el funcionamiento correcto de Munge, y para ello se tuvo que editar las variables de entorno:

```
export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:/usr/lib
```

También ha sido necesario instalar un paquete criptográfico como openssl-devel, ya que la criptografía es una parte fundamental de Munge y necesita poder hacer estas operaciones eficazmente. Para que el programa funcione correctamente es necesario instalarlo en todos los nodos y al generar la clave Munge, la cual es un archivo que garantiza la seguridad autenticando los nodos del clúster y sirviendo también para cifrar las

comunicaciones entre ellos. Es necesario copiarla al resto de nodos mediante scp [12] u otro medio seguro.

3.3 SLURM

Para su instalación en todos los nodos se han seguido las instrucciones de la página oficial de SLURM [13]. Primero es necesario descargar el archivo y descomprimirlo, también se creó un directorio en Documents para tener ahí todos los archivos del programa.

Para su correcto funcionamiento es necesario preparar un archivo de configuración que tendrá que estar exactamente igual en todas las máquinas. Para generarlo SLURM ofrece una herramienta web para ello [14]. El documento de configuración de SLURM generado y usado en el clúster está en el anexo de esta memoria.

Hay que tener en cuenta que SLURM tiene dos tipos de procesos: slurmctld para la máquina controladora, que se encarga de hacer de front-end y gestionar las tareas, y slurmd para los nodos destino de las tareas, los cuales recibirán y ejecutarán los trabajos mandados por el nodo gestor. Por ello hay un comando distinto para cada proceso, *systemctl start slurmctld* para el nodo front-end y *systemctl start slurmd* para los nodos de cómputo.

Para generar el documento de configuración se tuvo que tener en cuenta varios aspectos importantes, como donde se almacenaban el estado de los controladores para poder consultarlo y conocerlo, el de Slurmctld se introducía en StateSaveLocation y el de Slurmd en SlurmdSpoolDir. Se cambió la selección de recursos a lineal, ya que según su definición está basada en nodos y no administra la asignación de procesadores individuales. El almacenamiento de contabilidad de trabajos hubo que ponerlo a none para evitar que diera problemas al no poder SLURM hacer las métricas en raspbian, y como no era necesario, no se usó. Se modificó también el parámetro ProctrackType, a LinuxProc para que el SO se encargará de gestionar los identificadores de los procesos de SLURM.

Tras preparar el fichero para su posterior uso se ha usado el comando ./configure que venía en el paquete y los comandos make y make install para instalar slurm en los nodos. Para el configure era necesario usar los parámetros --prefix=/usr --sysconfdir=/etc para que usara esos directorios, el prefix defina la ruta donde estarán los ficheros de SLURM, y el sysconfdir la ruta donde leerá el archivo de configuración. Este proceso se hizo también en el resto de nodos, copiando el fichero de configuración e instalando y configurando SLURM. Había que tener en cuenta que tras cierto tiempo los nodos podían ponerse en estado DOWN, lo que implicaba que no trabajaban, para solucionar esto se usaba el comando scontrol update NodeName=<nodeX> State=RESUME.

3.4 Network File System (NFS)

Para poder mandar los trabajos y que los nodos los puedan reconocer necesitamos una carpeta compartida en red, para ello hemos usado nfs [15].

Hemos creado un directorio compartido para que todas las máquinas puedan leer y escribir archivos ahí. Luego simplemente hay que arrancar el servicio nfs en las máquinas y montarlo para que sea funcional.

Para su instalación ha sido necesario instalarlo, con *sudo apt install nfs-kernel-server* en el front-end y *sudo apt install nfs-common* en el resto. Luego hemos creado el directorio compartido en el nodo front-end, y hemos editado su archivo */etc/exports* con la siguiente linea:

/home/workDirSlurm
*(rw,all_squash,insecure,sync,no_subtree_check,anonuid=1000,anongid=1000)

la cual declara el directorio que queremos compartir. Finalmente habra que montar dicho directorio en el resto de nodos para que puedan acceder a el con el siguiente comando:
sudo mount <dirección_ip_servidor>:/ruta/al/directorio_compartido /ruta/al/directorio_local

3.5 Pruebas del servicio

Tras hacer toda la configuración e instalación (Munge, SLURM, NFS) en las Raspberry Pi, hemos arrancado el servicio de Munge y SLURM en todas las máquinas y probado a lanzar scripts de prueba en la máquina controladora (en el directorio compartido), para poder así lanzarlo con SLURM al resto de máquinas y comprobar que ha sido ejecutado por ellas. Se comprobó que su funcionamiento era el deseado. Un script de prueba sencillo que ejecutamos es el Script 3.5.

```
#!/bin/bash

echo "El trabajo se está ejecutando en el nodo: $HOSTNAME" >> salida.txt
echo "Hola, soy un script de prueba" >> salida.txt
ls -a >> salida.txt
sleep 5
```

Script 3.5 - Script de prueba en Raspberry Pi

Capítulo 4 - Entorno real basado en RPis

4.1 Configuración del entorno

Para las Raspberry Pi La configuración de red entre las máquinas ya venía dada y funcionaba perfectamente. Disponemos de 4 Raspberry, la controladora llamada raspi3-frontend y las que realizarán los trabajos son de la raspi3-1 a la raspi3-4. Comprobamos que las máquinas se conocieran y pudieran interconectarse entre ellas. Tuvimos que instalar aquí también tanto Munge como SLURM, de la misma forma, y configurarlos adecuadamente para su funcionamiento.

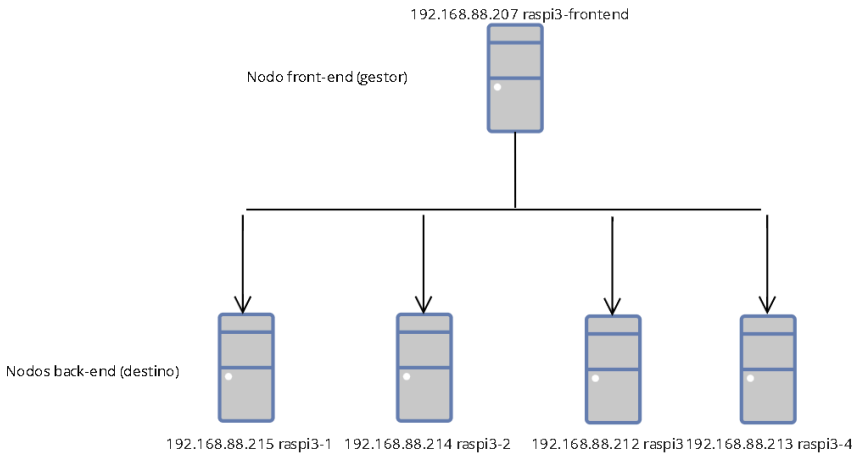

Figura 4.1 - Diagrama de la topología de las Raspberry

En estas máquinas nos encontramos con la característica de que en uno de los nodos tenemos un acelerador de Google Coral conectado a un puerto USB. Esto convierte este nodo en el objetivo perfecto al cual mandar trabajos con operaciones tensoriales las cuales podrá acelerar y con ello lograr un mayor rendimiento de nuestro clúster. Esto no lo teníamos en las máquinas virtuales, por tanto ahora será interesante saber detectarlo y usarlo lanzando algún programa de inferencia.

Para detectar el acelerador desde la propia máquina que lo tiene conectado, se puede utilizar el comando *lsusb* el cual muestra si tiene dispositivos conectados a sus puertos y una descripción de los mismos. En el caso de la Raspberry Pi con la TPU he visto la descripción del acelerador de Google. Conociendo este comando podemos generar scripts (como el Script 4.2) que se encarguen de lanzar los trabajos si encontramos un acelerador en el nodo al que enviamos el trabajo.

Ya teniendo todo preparado solo falta poder tener la capacidad de usar a nivel de usuario la TPU (acelerador de Google Coral, véase figura 4.1) conectada a uno de los nodos, de esta forma se podrá aprovechar y elegir qué tareas deseamos enviar a dicho nodo y conseguir la eficiencia que buscamos.

Figura 4.1 - Acelerador Google Coral Fuente: [19]

4.2 Uso del acelerador con lsusb

Para detectar el acelerador desde la propia máquina que lo tiene conectado, se puede utilizar el comando *lsusb* el cual muestra si tiene dispositivos conectados a sus puertos y una descripción de los mismos. En el caso de la Raspberry Pi con la TPU he visto la descripción del acelerador de Google. Conociendo este comando podemos generar scripts (como el Script 4.2) que se encarguen de lanzar los trabajos si encontramos un acelerador en el nodo al que enviamos el trabajo.

```
#!/bin/bash
if lsusb | grep -q "Google Inc."  then
      export TPU_AVAILABLE=1
else
      export TPU_AVAILABLE=0
fi

if    "$TPU_AVAILABLE" -eq 1 ]; then
      echo "TPU disponible en este nodo. Ejecutando trabajo que utiliza
      TPU..." >> salida.txt
      echo "Hola, soy un script de prueba" >> salida.txt
else
```

```
	echo "No se encontró TPU en este nodo. Ejecutando trabajo que no
	utiliza TPU..." >> salida.txt
fi
```

Script 4.2.1- Script para detectar TPU

El problema de este enfoque es que si el trabajo se envía a un nodo sin acelerador simplemente entrará por el else y no ejecutará el programa, por tanto si queremos que se ejecute sí o sí habría que lanzarlo hasta que sea enviado a uno de los nodos deseados.

La parte buena es que si nos da igual el nodo al que se envíe, podemos lanzar igualmente el programa modificándolo a nuestro gusto según donde se ejecute. Por ejemplo, si queremos que escriba en distintos ficheros para hacer pruebas, o si queremos lanzarlo en segundo plano, o también para lanzar distintos programas sabiendo que en un caso tendremos la potencia del acelerador y en el otro no.

Teniendo la forma de poder detectar la TPU podemos usar un programa de inferencia para enviar a ese nodo y hacer cálculos de tiempos para comprobar como se ha comportado y si ha funcionado todo como debería.

Se descubrió un problema a la hora de ejecutar sobre la coral, y es que el usuario que lanza el trabajo tiene que pertenecer al grupo plugdev, ya que los grupo del usuario se toman desde ahí, por ello hubo que modificar el fichero /etc/groups.

Para ejecutar esto he usado la página de Coral [16], y el ejemplo que ofrecen [17], el cual consiste en una red clasificadora de imágenes con la que probaremos si acierta a saber que en una imagen de un loro hay un loro. Para ejecutarlo simplemente hemos generado el Script 4.2 que ejecute el programa si encuentra el acelerador y escriba la salida en un fichero de texto.

```
#!/bin/bash
if lsusb | grep -q "Google Inc."; then
	export TPU_AVAILABLE=1
else
	export TPU_AVAILABLE=0
fi

if [ "$TPU_AVAILABLE" -eq 1 ]; then
	echo "TPU disponible en este nodo. Ejecutando trabajo que utiliza
TPU..." >> inferencia.txt
	python3 examples/classify_image.py --model
test_data/mobilenet_v2_1.0_224_inat_bird_quant_edgetpu.tflite --labels
test_data/inat_bird_labels.txt --input test_data/parrot.jpg >> inferencia.txt
2>&1
else
	echo "No se encontró TPU en este nodo. Ejecutando trabajo que no
utiliza TPU..." >> inferencia.txt
fi
```

Script 4.2.2 - Script inferencia con TPU

Al enviarlo con sbatch y terminar su ejecución pudimos comprobar el tiempo que tardo con el comando de SLURM sacct, el cual nos muestra en su Salida 4.2 diversa información sobre las tareas, entre la cual está el tiempo que ha tardado en ejecutarse.

```
JobID        JobName    Elapsed
------------ ----------       ----------
62             inferenci+    00:00:05
63             inferenci+    00:00:05
```

Salida 4.2 - Resultado del comando sacct

Esta ejecución hace uso del acelerador Google Coral usando el modelo preparado para ello. Tras ejecutarlo varias veces, lo cual es necesario para cargar el modelo en la TPU, hemos podido ver los resultados en el Fichero 4.2.1.

```
TPU disponible en este nodo. Ejecutando trabajo que utiliza TPU...
----INFERENCE TIME----
Note: The first inference on Edge TPU is slow because it includes
loading the model into Edge TPU memory.
132.6ms
14.7ms
14.7ms
14.6ms
14.5ms
-------RESULTS--------
Ara macao (Scarlet Macaw): 0.75781
```

Fichero 4.2.1 - Salida inferencia con TPU

En contraposición a esto y para mostrar la ventaja en coste de la TPU sobre la CPU, se ha enviado a ejecución también un código que no usaba la TPU con su modelo correspondiente. En el cual se esperaba ver un aumento considerable del tiempo de la inferencia. Tras su ejecución hemos podido confirmar nuestra sospecha en el Fichero 4.2.2.

```
TPU disponible en este nodo. Ejecutando trabajo que utiliza TPU...
----INFERENCE TIME----
Note: The first inference on Edge TPU is slow because it includes
loading the model into Edge TPU memory.
252.2ms
248.8ms
248.9ms
248.9ms
249.0ms
-------RESULTS--------
```

```
Ara macao (Scarlet Macaw): 0.77734
```

Fichero 4.2.2 - Salida inferencia sin TPU

Al final depende de cómo deseemos que procedan nuestros programas y nuestro clúster. Si lo que deseamos es un tratamiento distinto según donde se lance, la forma mostrada anteriormente nos funcionará, sin embargo, si lo que deseamos es que se envíe sí o sí a un nodo con un acelerador tenemos otra posibilidad.

4.3 Uso del acelerador con *constraint*

Vamos a explorar también la otra alternativa, que es más acertada ya que no necesita acceder al nodo para saber si tiene una TPU o no. SLURM nos provee de una variedad de comandos como sinfo para conocer el estado del clúster, o sbatch para lanzar los trabajos. El interés de esto radica en que podemos usar dichos comandos en un script para hacer comprobaciones de la configuración del sistema.

Para entender por qué esto nos ayuda es necesario conocer una característica de SLURM que es la de etiquetar nodos. Es posible dar etiquetas a los nodos según sus características para que SLURM pueda dirigirse a ellos como si fuera un filtro de búsqueda.

Por ejemplo, si tenemos un nodo con una TPU y queremos etiquetarlo, tenemos que editar el archivo de configuración y al final, en la parte de definición de nodos podemos ponerlos por separado, especificando si a alguno queremos añadirle una etiqueta. En nuestro caso lo hemos especificado así:

```
# COMPUTE NODES
NodeName=raspi3-1 Feature=TPU CPUs=1 State=UNKNOWN
NodeName=raspi3-[2-4] CPUs=1 State=UNKNOWN
PartitionName=Acelerador Nodes=ALL Default=YES State=UP
```

En este fragmento de la configuración se definen como se identifican y gestionan los nodos de cómputo de nuestro clúster. En las dos primeras líneas vemos como se definen conjuntos de nodos. El primer conjunto está formado únicamente por el nodo con el acelerador y por ello usamos la propiedad Feature, que nos permitirá solicitar los nodos de este conjunto para las operaciones de inferencia. En la segunda línea se definen el resto de nodos sin la característica antes mencionada. Finalmente en la última línea definimos una única cola (partition) llamada Acelerador, incluimos en ella todos los nodos con Nodes=ALL y ponemos los nodos listo para trabajar con State=UP.

Como usuarios podemos definir los nodos que queremos que ejecute cierto tipo de trabajos, como los de inferencia, etiquetarlos para tal propósito. Por ello se ha modificado el archivo de configuración como se ha mostrado anteriormente, y luego para lanzar los trabajos sobre unos nodos etiquetado basta con añadir el flag --constraint=<nombre_etiqueta> a la hora de ejecutar los trabajos, así:

sbatch --constraint="TPU" inferencia_constraint.sh

Con esto conseguimos que el trabajo vaya exclusivamente a los nodos con esa etiqueta, por tanto siempre tendrá el acelerador disponible. Si encolamos varias veces el mismo trabajo sin esta opción vemos como va a cualquier nodo. Hemos necesitado lanzarlo múltiples veces para ver donde se encolaban los trabajos, para ello hemos lanzado un bucle sencillo:

```
n=5
for ((i = 1; i <= $n; i++)); do
        sbatch inferencia_constraint.sh
done
```

Y este es el resultado del encolamiento en la partición:

```
atiscar@raspi3-frontend:~$ squeue
JOBID  PARTITION  NAME    USER  ST             TIME    NODES  NODELIST(REASON)
110  Acelerado  inferencia_constraint.sh  atiscar  PD  0:00  1  (Resources)
111 Acelerado inferencia_constraint.sh atiscar PD    0:00      1 (Priority)
112 Acelerado inferencia_constraint.sh atiscar PD    0:00      1 (Priority)
113 Acelerado inferencia_constraint.sh atiscar PD    0:00      1 (Priority)
114 Acelerado inferencia_constraint.sh atiscar PD    0:00      1 (Priority)
100 Acelerado inferencia_constraint.sh atiscar  R     0:04      1 raspi3-1
107 Acelerado inferencia_constraint.sh atiscar  R     0:00      1 raspi3-2
108 Acelerado inferencia_constraint.sh atiscar  R     0:00      1 raspi3-3
109 Acelerado  inferencia_constraint.sh atiscar  R 0:00       1  raspi3-4
```

Salida 4.3.1 - Resultado del comando sbatch sin constraint

Como podemos ver, el trabajo se encola en todos nuestros nodos, se reparten las distintas veces que lo hemos lanzado entre todos. Por ello ahora vamos a ver qué pasa si en lugar de usar sbatch inferencia_constraint.sh usamos el flag constraint, quedando de esta forma:

```
n=5
for ((i = 1; i <= $n; i++)); do
        sbatch --constraint="TPU" inferencia_constraint.sh
done
```

Y este es el resultado del encolamiento en la partición:

```
atiscar@raspi3-frontend:~$ squeue
JOBID  PARTITION    NAME    USER  ST            TIME    NODES  NODELIST(REASON)
117 Acelerado    inferencia_constraint.sh  atiscar PD 0:00    1 (Resources)
```

```
118 Acelerado     inferencia_constraint.sh   atiscar PD    0:00    1 (Priority)
119 Acelerado     inferencia_constraint.sh   atiscar PD    0:00    1 (Priority)
120 Acelerado    inferencia_constraint.sh    atiscar PD   0:00  1 (Priority)
116 Acelerado   inferencia_constraint.sh    atiscar  R   0:03   1   raspi3-1
```

Salida 4.3.2 - Resultado del comando sbatch con constraint

Se puede apreciar que ahora los trabajos únicamente se encolan en raspi3-1, que es la que posee la TPU y que tenemos definida en el fichero de configuración de SLURM.

Ahora vamos a comprobar la paralización de tareas, para comprobar que esto es correcto vamos a modificar el script anterior añadiendo la hora al principio y al final de la ejecución y comprobaremos que si lo ejecutamos sobre cualquier nodo se solaparan las horas. El script quedaría así:

```
#!/bin/bash

echo "============================================="
date "+%H:%M:%S"
python3   ~/coral/pycoral/examples/classify_image.py   --model
~/coral/pycoral/test_data/mobilenet_v2_1.0_224_inat_bird_quant_edg
etpu.tflite                                                --labels
~/coral/pycoral/test_data/inat_bird_labels.txt --input ~/coral>
date "+%H:%M:%S"
echo "============================================="
```

Script 4.3 - Script con horas para concurrencia

Ahora probemos a ejecutarlo varias veces con el mismo bucle de antes, y veamos la salida de los distintos trabajos:

```
atiscar@raspi3-frontend:~/coral/pycoral$ cat slurm-166.out
     =============================================
     21:01:49
     ----INFERENCE TIME----
     Note: The first inference on Edge TPU is slow because it
includes loading the model into Edge TPU memory.
     121.0ms
     15.0ms
     14.9ms
     15.1ms
     15.1ms
     -------RESULTS--------
     Ara macao (Scarlet Macaw): 0.75781
     21:01:54
     =============================================
```

```
atiscar@raspi3-frontend:~/coral/pycoral$ cat slurm-167.out
==========================================
21:01:49
Traceback (most recent call last):
                                                                File
"/home/atiscar/coral/pycoral/examples/classify_image.py", line 37,
in <module>
    from pycoral.adapters import classify
ModuleNotFoundError: No module named 'pycoral'
21:01:50
==========================================
atiscar@raspi3-frontend:~/coral/pycoral$ cat slurm-168.out
==========================================
21:01:51
Traceback (most recent call last):
                                                                File
"/home/atiscar/coral/pycoral/examples/classify_image.py", line 37,
in <module>
    from pycoral.adapters import classify
ModuleNotFoundError: No module named 'pycoral'
21:01:52
==========================================
```

Salida 4.3.3 - Resultado del script 4.3 sin constraint

Como podemos apreciar el primer trabajo ha ido desde las `21:01:49` hasta las `21:01:54` y el tercero desde las `21:01:50` hasta las `21:01:52`, lo que demuestra que la ejecución ha sido paralela, ya que mientras se ejecutaba el primero, lo ha hecho también el tercero. Con esto podemos ver que los trabajos que no hacen uso de la TPU, en los que no usamos el flag del constraint, se ejecutan concurrentemente.

Ahora por último vamos a ver que si ejecutamos con el constraint efectivamente se ejecutan secuencialmente.

```
atiscar@raspi3-frontend:~/coral/pycoral$ cat slurm-171.out
==========================================
21:10:13
----INFERENCE TIME----
Note: The first inference on Edge TPU is slow because it includes
loading the model into Edge TPU memory.
123.4ms
15.1ms
14.9ms
14.6ms
14.9ms
-------RESULTS--------
```

```
Ara macao (Scarlet Macaw): 0.75781
21:10:18
============================================
atiscar@raspi3-frontend:~/coral/pycoral$ cat slurm-172.out
============================================
21:10:18
----INFERENCE TIME----
Note: The first inference on Edge TPU is slow because it includes
loading the model into Edge TPU memory.
123.4ms
15.1ms
15.1ms
14.7ms
15.0ms
-------RESULTS--------
Ara macao (Scarlet Macaw): 0.75781
21:10:23
============================================
```

Salida 4.3.4 - Resultado del script 4.3 con constraint

Como podemos apreciar el primer trabajo ha ido desde las `21:10:13` hasta las `21:10:18` y el segundo desde las `21:10:18` hasta las `21:10:23`, lo que demuestra que hasta que no ha terminado el primer trabajo no ha empezado el siguiente, y así con todos, ya que esos trabajos necesitan el uso de la TPU y deben esperar a que esté disponible.

Capítulo 5 - Conclusiones y trabajo futuro

En este trabajo se han estudiado los modos de gestión de trabajos en un clúster compuesto por varios nodos. Se ha hecho con dos tipos distintos de máquinas (virtuales y físicas) y en este caso con distintos sistemas operativos. Esto nos muestra que es posible replicarlo con distintos sistemas operativos y arquitecturas.

Usar SLURM ha sido un acierto ya que es altamente escalable y se ha adaptado perfectamente a las necesidades del proyecto y funcionaría con otras necesidades distintas realizando los ajustes oportunos.

En primer lugar, se estudió el funcionamiento de un clúster y su configuración para saber trabajar con él, para ello ha sido necesario conocer conceptos de redes para la comunicación entre los distintos nodos. Sobre esto fue también necesario conocer el funcionamiento de las herramientas de Munge, SLURM y NFS, con las cuales se ha podido tener un mayor control sobre los nodos y diferenciar el nodo front o nodo gestor, y los nodos donde se hacían las distintas operaciones de cómputo.

A continuación, se replicaron y usaron todos estos conocimientos sobre un clúster formado por Raspberry Pis con la característica de que una de ellas tenía un acelerador Google Coral instalado. Gracias al estudio y simulación previa, este paso fue relativamente sencillo de hacer.

En último lugar se estudió el uso de la TPU a petición del usuario, desde el nodo gestor para poder aprovechar su potencia. Aquí se descubrió que no solo se podía hacer, sino que además, había varias formas de hacerlo. Y esto ofrece una gran ventaja, ya que como se ha comprobado, el uso de la TPU hace mucho más eficiente las tareas relacionadas con la inferencia de redes neuronales.

En conclusión, se puede decir que es posible gestionar una TPU como un recurso más desde SLURM, permitiendo a los usuarios del clúster, poder lanzar tareas de inferencia o relacionadas con cálculos de tensores, aprovechando al máximo la potencia de un acelerador en un nodo o varios del clúster. Y permitiendo adaptar los trabajos o tareas que se envíen al clúster según sus necesidades.

Capítulo 6 - Introduction

6.1 Introduction

A computer cluster [1] is a group of interconnected computers that work together, allowing for increased computing power and better workload management. They are one of the most notable innovations in the field of computing. Clusters have revolutionized the way we approach complex computational problems due to their enhanced computational power and the ability to parallelize tasks, among other things. This has enabled and driven significant advancements in various areas of science, industry, and research.

This computational work capacity has provided us with astonishing potential. From simulating complex scientific phenomena to analyzing vast datasets in the field of machine learning and artificial intelligence, computer clusters have made it possible to tackle challenges that once seemed insurmountable. Progress in fields such as medicine, meteorology, physics, and engineering has been accelerated by the ability to perform calculations and experiments using the capabilities provided by clusters.

In summary, computer clusters represent a highly valuable tool in our era. Their ability to combine resources and skills collaboratively has expanded the possibilities of computing and paved the way for discoveries and developments.

To harness the full capacity of a cluster, it's beneficial to have a system that organizes the nodes and handles the distribution of tasks and programs the cluster will execute. In our case, we have utilized SLURM [2], a job management program responsible for allocating which nodes will perform computations, thus enabling more efficient utilization of these resources.

SLURM is an open-source cluster management and job scheduling system that's particularly well-suited for Linux clusters. It is highly scalable and can efficiently handle failures. SLURM doesn't require complex kernel modifications to function, making it very practical and self-contained.

SLURM excels in three primary functions. Firstly, it manages resource access, such as computing nodes, either exclusively or non-exclusively, based on users' needs over a specified time frame, ensuring smooth job execution for all. Secondly, it provides a valuable framework for initiating, executing, and monitoring jobs, particularly those that are parallel, on the assigned nodes. Lastly, it handles resource allocation and conflict avoidance, all thanks to how it manages the queue of pending jobs.

In summary, SLURM is an incredible tool for optimizing cluster management and job scheduling. Undoubtedly, it is an immensely valuable tool for anyone working with clusters, in need of a reliable and scalable solution.

Efficiency in computing has always been a critical parameter, as both time and energy are not limitless resources, and we aim to solve problems within a reasonable timeframe. Therefore, it is tremendously beneficial to reduce time and energy expenditures for any type of computation, making any advancements or research towards greater efficiency a topic of significant interest for researchers.

6.2 Objectives

The objective of this Final Degree Project is to treat the TPUs connected to the nodes as an additional resource within SLURM. This will empower the job queue management system with the new capability to use accelerators as needed, enhancing user control and decision-making when working on various types of tasks within a cluster, especially those related to neural network inference.

This objective can be achieved by accomplishing the following milestones:

1. Study the operation and configuration of networks, including communication between multiple machines, internet access, file sharing, etc.
2. Learn about the operation of the tools to be used during the project and their configuration.
3. Apply this knowledge to Raspberry Pi to gain control over them and incorporate them as nodes within a cluster for collaborative work.
4. Implement a means of control over the TPUs within the cluster, providing users with the ability to harness their capabilities effectively.

6.3 Work Plan

The plan has been divided into four phases:

The first phase involves simulating real machines (Raspberry Pi) using virtual machines. The goal is to understand and manage the tools and resources required for this project, which will be detailed throughout this document. It will be necessary to handle the virtual machines, install, and make the tools work. This study of connecting to virtual machines and configuring the necessary programs to achieve the goal is estimated to take approximately 120 hours.

The second phase consists of replicating this process with the Raspberry Pis that will create the cluster. It's important to note that one of them contains the accelerator, which is of interest for this work. We'll need to reinstall and configure the programs and conduct some tests on them. This phase is estimated to take around 70 hours.

The third phase will involve configuring TPU management from SLURM to give users the ability to use it and take advantage of its benefits, as well as verifying these advantages. This phase is based on information search and testing and is expected to last approximately 60 hours.

The final phase will revolve around documenting and studying the results, as well as making time comparisons. This concluding phase is estimated to require about 50 hours."

6.4 Structure of the Document

This document will be divided into several sections:

- The second chapter consists of the key concepts necessary to understand the project. In it, we will see what a queue system is, how SLURM works, and how it helps us. We will also understand the utility of the TPU and important AI concepts such as neural networks and inference. Finally, we will see details about the environments where we will carry out our project, the virtual machines, and the Raspberry Pi.
- In the third chapter, the development environment and the configuration of all necessary elements on the virtual machines will be shown. This chapter will contain vital information, as this same process will later be carried out on the Raspberry Pi.
- The fourth chapter will cover the implementation on the Raspberry Pi and the transparent use of the TPU from SLURM. Here we will see different usage alternatives and the differences they offer.
- Finally, in the fifth chapter, we will present the conclusions we have reached after completing this work.

Capítulo 7 - Conclusions and Future Work

In this study, we have explored job management methods within a cluster composed of multiple nodes. This investigation was conducted using two different types of machines (virtual and Raspberry Pi devices) and, consequently, with different operating systems. This demonstrates that it is possible to replicate this approach with various operating systems and architectures.

The decision to use SLURM was a smart one, as it is highly scalable and adapted perfectly to the project's requirements. It can also work seamlessly with other systems with the necessary adjustments.

Firstly, we examined the operation of a cluster and its configuration to understand how to work with it. This required acquiring knowledge of networking concepts for communication between the various nodes. Additionally, we needed to understand the functionality of tools such as Munge, SLURM, and NFS, which allowed us to exert greater control over the nodes, distinguishing between the front node or manager node and the nodes where various computing operations were performed.

Next, we applied and utilized these insights to a cluster composed of Raspberry Pi devices, with the unique feature that one of them had a Google Coral accelerator installed. Thanks to the prior study and simulation, this step was relatively straightforward to execute.

Finally, we examined the use of the TPU at the user's request, from the manager node, to harness its power. This investigation revealed that not only was it possible to do so, but there were also several ways to achieve this.

In conclusion, it is feasible to manage a TPU as just another resource within SLURM. This empowers users of the cluster to launch inference tasks or tasks related to tensor calculations, fully leveraging the potential of an accelerator on one or more nodes of the cluster. It also allows users to adapt jobs or tasks sent through the cluster according to their specific needs.

Apéndice A - Configuración de SLURM

Documento de configuración de SLURM generado y usado en las Raspberry Pi

```
# slurm.conf file generated by configurator.html.
# Put this file on all nodes of your cluster.
# See the slurm.conf man page for more information.
#
ClusterName=virtualCluster
SlurmctldHost=raspi3-frontend
#SlurmctldHost=
#
#DisableRootJobs=NO
#EnforcePartLimits=NO
#Epilog=
#EpilogSlurmctld=
#FirstJobId=1
#MaxJobId=67043328
#GresTypes=
#GroupUpdateForce=0
#GroupUpdateTime=600
#JobFileAppend=0
#JobRequeue=1
#JobSubmitPlugins=lua
#KillOnBadExit=0
#LaunchType=launch/slurm
#Licenses=foo*4,bar
#MailProg=/bin/mail
#MaxJobCount=10000
#MaxStepCount=40000
#MaxTasksPerNode=512
MpiDefault=none
#MpiParams=ports=#-#
#PluginDir=
#PlugStackConfig=
#PrivateData=jobs
ProctrackType=proctrack/linuxproc
#Prolog=
#PrologFlags=
#PrologSlurmctld=
#PropagatePrioProcess=0
#PropagateResourceLimits=
#PropagateResourceLimitsExcept=
```

```
#RebootProgram=
ReturnToService=1
SlurmctldPidFile=/var/run/slurmctld.pid
SlurmctldPort=6817
SlurmdPidFile=/var/run/slurmd.pid
SlurmdPort=6818
SlurmdSpoolDir=/var/spool/slurmd
SlurmUser=root
#SlurmdUser=root
#SrunEpilog=
#SrunProlog=
StateSaveLocation=/var/spool/slurmctld
SwitchType=switch/none
#TaskEpilog=
TaskPlugin=task/affinity
#TaskProlog=
#TopologyPlugin=topology/tree
#TmpFS=/tmp
#TrackWCKey=no
#TreeWidth=
#UnkillableStepProgram=
#UsePAM=0
#
#
# TIMERS
#BatchStartTimeout=10
#CompleteWait=0
#EpilogMsgTime=2000
#GetEnvTimeout=2
#HealthCheckInterval=0
#HealthCheckProgram=
InactiveLimit=300
KillWait=30
#MessageTimeout=10
#ResvOverRun=0
MinJobAge=300
#OverTimeLimit=0
SlurmctldTimeout=120
SlurmdTimeout=300
#UnkillableStepTimeout=60
#VSizeFactor=0
Waittime=0
#
#
# SCHEDULING
```

```
#DefMemPerCPU=0
#MaxMemPerCPU=0
#SchedulerTimeSlice=30
SchedulerType=sched/backfill
SelectType=select/linear
#SelectTypeParameters=
#
#
# JOB PRIORITY
#PriorityFlags=
#PriorityType=priority/basic
#PriorityDecayHalfLife=
#PriorityCalcPeriod=
#PriorityFavorSmall=
#PriorityMaxAge=
#PriorityUsageResetPeriod=
#PriorityWeightAge=
#PriorityWeightFairshare=
#PriorityWeightJobSize=
#PriorityWeightPartition=
#PriorityWeightQOS=
#
#
# LOGGING AND ACCOUNTING
#AccountingStorageEnforce=0
#AccountingStorageHost=
#AccountingStoragePass=
#AccountingStoragePort=
#AccountingStorageType=accounting_storage/slurmdbd
#AccountingStorageType=accounting_storage/filetxt
#AccountingStorageUser=
#AccountingStoreFlags=
#JobCompHost=
#JobCompLoc=Documents/slurm/finalizacion
#JobCompPass=
#JobCompPort=
JobCompType=jobcomp/none
#JobCompUser=
#JobContainerType=job_container/none
JobAcctGatherFrequency=30
JobAcctGatherType=jobacct_gather/none
SlurmctldDebug=info
SlurmctldLogFile=/var/log/slurmctld.log
SlurmdDebug=info
SlurmdLogFile=/var/log/slurmd.log
```

```
#SlurmSchedLogFile=
#SlurmSchedLogLevel=
#DebugFlags=
#
#
# POWER SAVE SUPPORT FOR IDLE NODES (optional)
#SuspendProgram=
#ResumeProgram=
#SuspendTimeout=
#ResumeTimeout=
#ResumeRate=
#SuspendExcNodes=
#SuspendExcParts=
#SuspendRate=
#SuspendTime=
#
#
# COMPUTE NODES
NodeName=raspi3-1 Feature=TPU CPUs=1 State=UNKNOWN
NodeName=raspi3-[2-4] CPUs=1 State=UNKNOWN
PartitionName=Acelerador Nodes=ALL Default=YES State=UP
```

Bibliografía

[1] Clúster informático Introducción: Clústeres - Documentación de IBM
[2] Documentación SLURM Slurm Workload Manager - Documentation (schedmd.com)
[3] CPU https://es.wikipedia.org/wiki/Unidad_central_de_procesamiento
[4] GPU https://es.wikipedia.org/wiki/Unidad_de_procesamiento_gr%C3%A1fico
[5] TPU TPUs de Google: el imparable avance del hardware especializado para ML - Paradigma (paradigmadigital.com)
[6] Munge github https://github.com/dun/munge
[7] Red neuronal El modelo de redes neuronales - Documentación de IBM
[8] Máquina virtual ¿Qué es una máquina virtual? | Glosario de VMware | ES
[9] Raspberry Pi Inicio - Raspberry Pi
[10] Red Hat Red Hat | Líder mundial del código abierto
[11] Munge Guia de instalación https://github.com/dun/munge/wiki/Installation-Guide
[12] scp Copia remota con el comando scp - Gestión de sistemas remotos en Oracle Solaris 11.1
[13] Guía instalación SLURM Slurm Workload Manager - Quick Start Administrator Guide (schedmd.com)
[14] Herramienta configuración SLURM Slurm System Configuration Tool (schedmd.com)
[15] NFS Cómo configurar NFS Mount en Ubuntu 20.04 | DigitalOcean
[16] Acelerador coral USB Accelerator | Coral
[17] Ejemplo inferencia Coral Get started with the USB Accelerator | Coral
[18] Imagen Rapberry Pi Raspberry Pi 3 Model B+ | RASPBERRY PI RPI-B BCM2837B0 64bit 1GB| 292268 (soselectronic.com)
[19] Imagen acelerador Google Coral USB Accelerator | Coral
[20] Slurm Workload Manager - Programación de recursos genéricos (GRES) (schedmd.com)
[21] https://coral.ai/docs/accelerator/get-started/#1-install-the-edge-tpu-runtime