Programación Científica y HPCI

Máster Universitario en Ingeniería Matemática y Computación

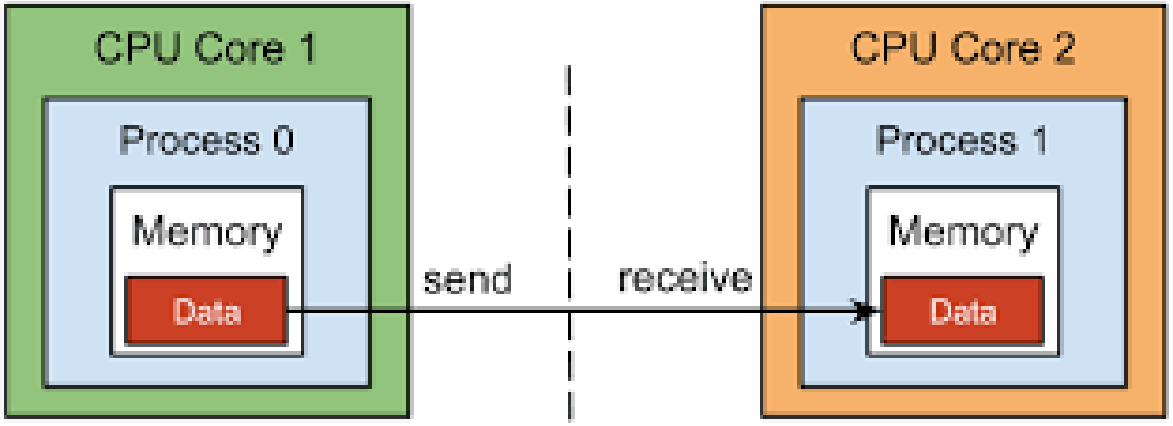
Tema 8 / Programación Paralela - HPC

MPI (Message Passing Interface)

- MPI como Interfaz de Paso de Mensajes estándar.
- MPI como una especificación de sintaxis y semántica de funciones para la comunicación de datos entre procesos en sistemas distribuidos.
- Aprobación y publicación de la primera versión de MPI en 1994.
- Permite la comunicación entre procesos en múltiples procesadores.

Características de MPI

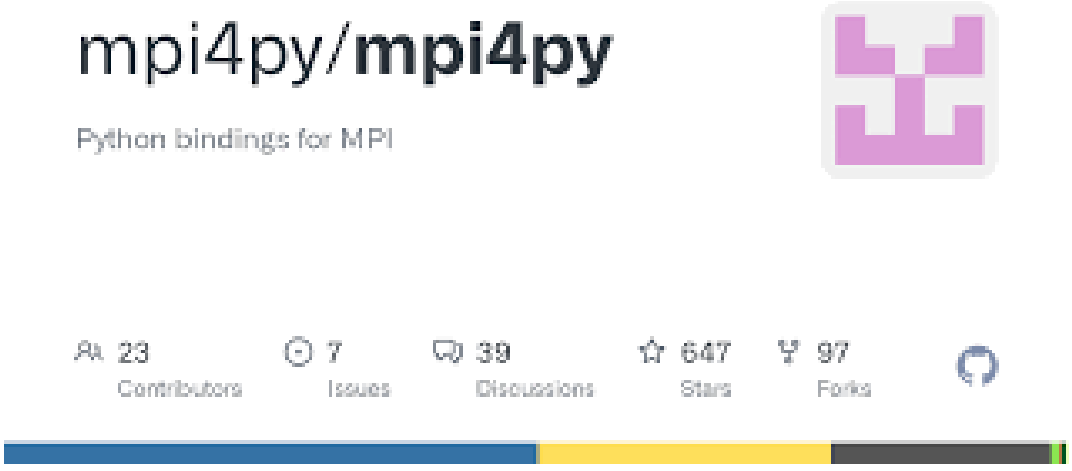
- Amplia funcionalidad y portabilidad
- Uso en entornos de multiprogramación, programación distribuida y entornos heterogéneos



Implementación de MPI en Python

- Módulo mpi4py
- Importante: instalación de módulos y características adicionales

Logo MPI

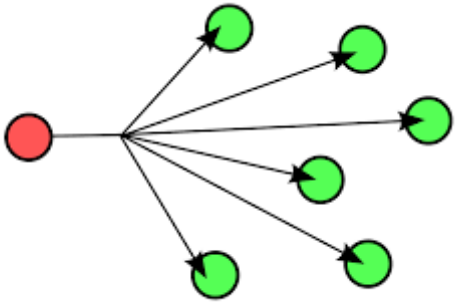


Comunicación punto a punto en MPI

- Comunicación punto a punto:
  - Uso de los métodos send y recv para pasar datos entre procesos.
  - El método send envía datos desde un proceso a otro indicando el destinatario (rango del proceso receptor).
  - El método recv recibe datos en un proceso indicando la fuente (rango del proceso emisor).
- Difusión de datos:
  - Comunicación en grupo enviando una copia exacta de la información a todos los procesos de un comunicador.
  - Uso del método bcast para realizar la difusión, especificando el dato a difundir y la raíz del comunicador.

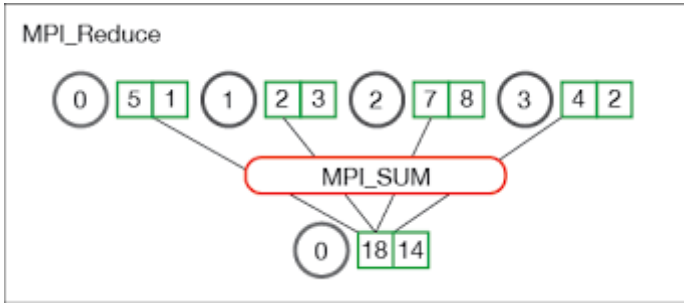
Difusión en MPI

- Comunicación en grupo mediante el envío de una copia exacta de la información a todos los procesos de un comunicador.
- Utilización del método bcast en mpi4py para realizar la difusión.
  - El método bcast requiere especificar el dato a difundir y la raíz del comunicador.
- Importante difundir aplicaciones distribuidas para compartir información de manera eficiente.



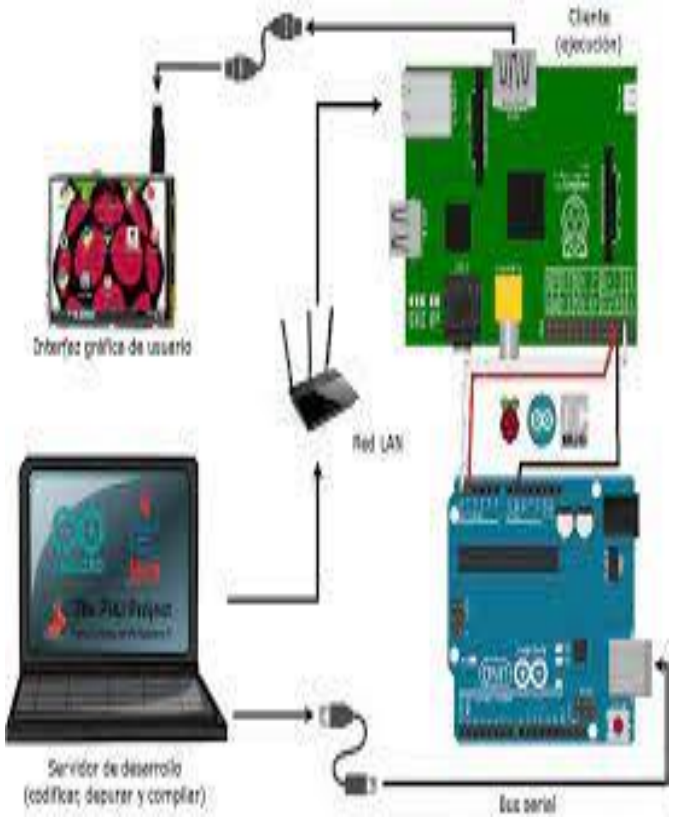
Operaciones de dispersión de datos, recolección y reducción con arrays

- **Dispersión:** Envío de porciones de un array a distintos procesos.
- **Recolección:** Recopilación de datos de diferentes procesos en un solo proceso.
- **Reducción:** Combina los datos de diferentes procesos en uno solo.
- **Funciones clave:** Scatter, Gather y Reduce en mpi4py.
- **Parámetros:** Especificar los datos a dispersar/recolectar, el tamaño, y los procesos involucrados.



Introducción a la programación distribuida

1. **Programación distribuida:** Diseño e implementación de sistemas cooperativos formados por múltiples ordenadores que se comunican entre sí.
2. **Importancia de la programación distribuida:** Permite la cooperación entre los ordenadores para realizar tareas de forma conjunta y eficiente.
3. **Beneficios:** Mejora del rendimiento y capacidad de procesamiento al distribuir las tareas en diferentes máquinas.
4. **Ejemplos de sistemas cooperativos:** Aplicaciones web, sistemas de procesamiento de datos a gran escala, redes de sensores, entre otros.



Características de los sistemas distribuidos

1. **Comunicación mediante mensajes:** Los sistemas distribuidos se comunican a través de mensajes enviados entre los diferentes componentes de la red.
2. **Red de comunicación:** Utilización de una red para transmitir los mensajes entre los nodos del sistema distribuido.
3. **No hay memoria física compartida:** A diferencia de los sistemas centralizados, en los sistemas distribuidos no existe una memoria compartida entre los nodos. Cada nodo tiene su propia memoria local.

Ventajas de la programación distribuida

- **Mejora del rendimiento:** La programación distribuida permite distribuir las tareas entre diferentes máquinas, lo que mejora el rendimiento en programas con alta demanda de computación.
- **Uso de recursos:** La programación distribuida aprovecha los recursos de las distintas máquinas para realizar las tareas de forma más eficiente.
- **Características de conectividad:** Los sistemas distribuidos pueden utilizar características de conectividad para integrar ejecuciones y almacenamiento de datos de forma transparente.

Consideraciones en la programación distribuida

- Al programar en un entorno distribuido, es importante considerar el número de "brokers" o colas de mensajes que utilizaremos.
- Los "brokers" actúan como intermediarios entre los emisores y receptores de los mensajes, y pueden ayudar a gestionar la carga y la escalabilidad del sistema.
- Es crucial determinar correctamente los destinatarios de los mensajes para asegurar que las tareas se envíen a los trabajadores o procesadores adecuados.
- Los protocolos de envío de mensajes y enrutamiento también deben ser tenidos en cuenta para garantizar una comunicación eficiente y segura entre los componentes distribuidos.
- La elección de los protocolos adecuados dependerá de los requisitos y características específicas del sistema distribuido.

Más beneficios de la programación distribuida

- **Rendimiento:** Mayor velocidad y capacidad de procesamiento.
- **Eficiencia:** Aprovechamiento óptimo de recursos disponibles.
- **Utilización efectiva:** Distribución equilibrada de tareas y carga.

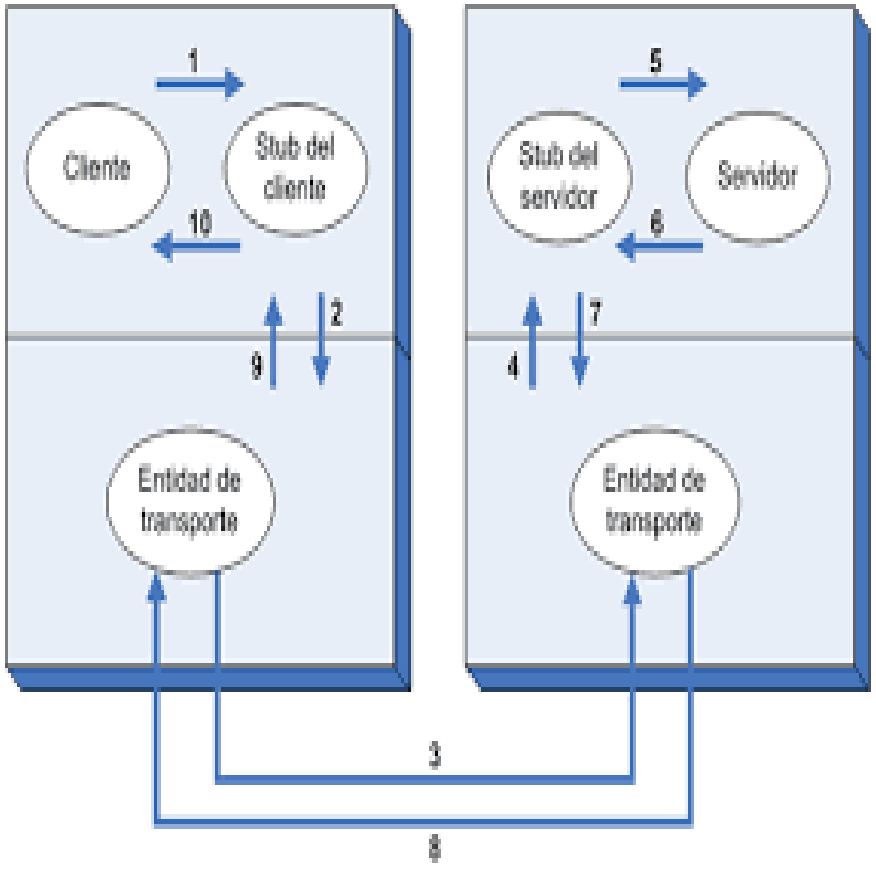
Desafíos de la programación distribuida

- **Complejidad del diseño e implementación:** Los sistemas distribuidos implican abordar desafíos adicionales en términos de la arquitectura, el diseño de la comunicación y la gestión de recursos.
- **Coordinación y sincronización:** La coordinación entre los distintos nodos y la sincronización de las tareas distribuidas pueden ser desafiantes debido a la naturaleza distribuida del sistema.
- **Mantenimiento y escalabilidad:** A medida que los sistemas distribuidos crecen, es necesario mantener y escalar la infraestructura para asegurar un funcionamiento óptimo.

Casos de uso de la programación distribuida

- **Aplicaciones con altos requerimientos de procesamiento y computación:** La programación distribuida se utiliza en aplicaciones que requieren un gran poder de procesamiento, como análisis de datos, simulaciones complejas y renderizado de gráficos en tiempo real.
- **Escalabilidad y paralelización en el procesamiento de grandes volúmenes de datos:** La programación distribuida permite dividir el procesamiento de grandes conjuntos de datos en tareas más pequeñas y distribuir esas tareas en múltiples nodos, lo que acelera el procesamiento y mejora la eficiencia.

Llamada a procedimientos remotos (RPC)



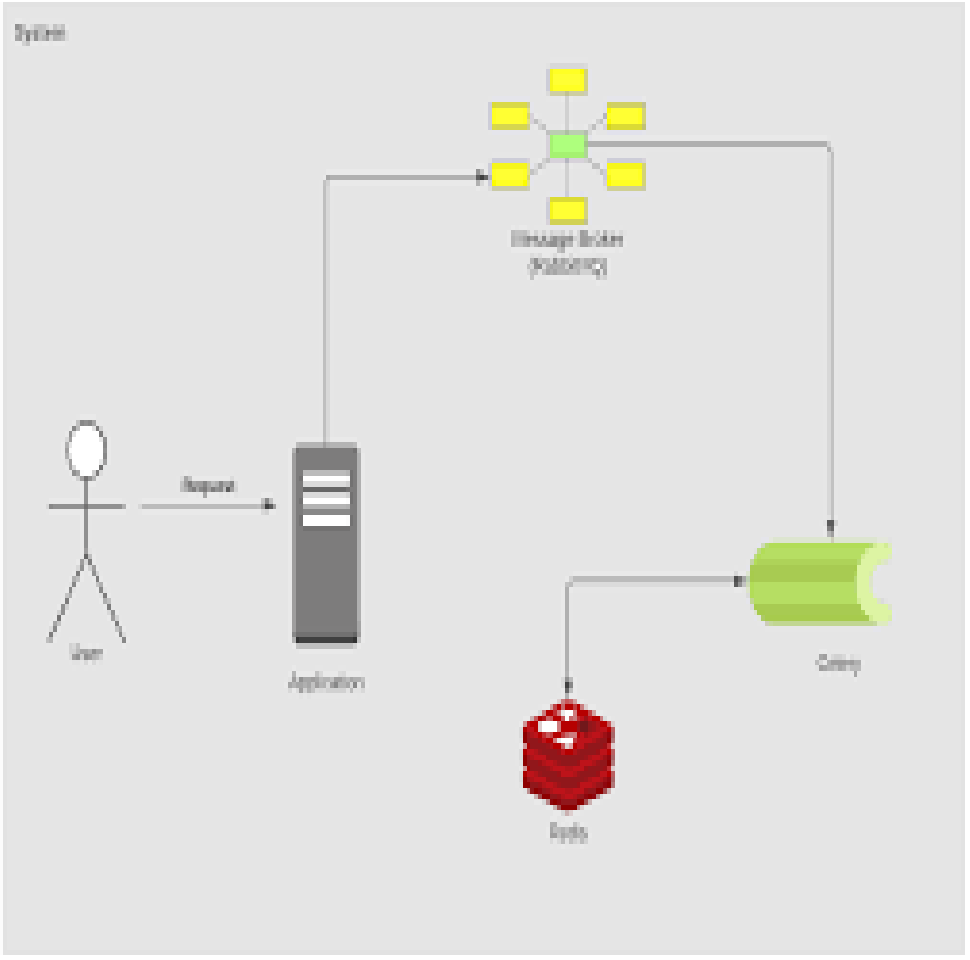
- **Mecanismo de comunicación:** La llamada a procedimientos remotos (RPC) es un mecanismo de comunicación síncrona utilizado en sistemas distribuidos.
- **Comunicación asimétrica:** La comunicación en RPC es asimétrica, donde se indica el receptor de la llamada y la información se envía en un solo sentido.
- **Uso de interfaz:** RPC se implementa mediante una interfaz que proporciona funciones para establecer la comunicación de manera transparente entre los componentes del sistema distribuido.

RPyC (Remote Python Call)

1. **Módulo transparente:** RPyC es un módulo de Python que facilita las llamadas a procedimientos remotos.
2. **Características principales:** RPyC ofrece transparencia, funciona en diferentes plataformas, presenta simetría en las comunicaciones, tiene baja sobrecarga, y se enfoca en la seguridad.

Gestión con Celery

- **Biblioteca de Python:** Celery es una biblioteca de Python diseñada para la gestión de colas distribuidas.
- **Enrutamiento de tareas asíncronas:** Celery permite el enrutamiento y gestión de tareas asíncronas, lo que facilita la ejecución distribuida.
- **Paso de mensajes distribuidos:** Celery se basa en el paso de mensajes distribuidos para la comunicación entre los componentes del sistema.



Configuración básica de Celery

Pasos iniciales: Para utilizar Celery, se deben seguir algunos pasos iniciales.
Creación de un servicio: Se crea un servicio utilizando Celery, el cual se configura con un "broker".
"Broker": El "broker" es un componente esencial en Celery y se utiliza para transportar los mensajes entre los distintos procesos y nodos del sistema distribuido.

Definición de tareas en Celery

- **Uso del decorador:** En Celery, se utiliza un decorador para identificar una tarea. Este decorador se aplica a una función que representa la tarea que se desea ejecutar de forma distribuida.
- **Identificación de la tarea:** Al aplicar el decorador a una función, se indica que dicha función es una tarea que puede ser ejecutada por Celery.

```python
from celery import Celery

app = Celery('myapp', broker='pyamqp://guest@localhost//')

@app.task
def my_task(arg1, arg2):
    # Código de la tarea
    return result
```

Lanzamiento de tareas en Celery

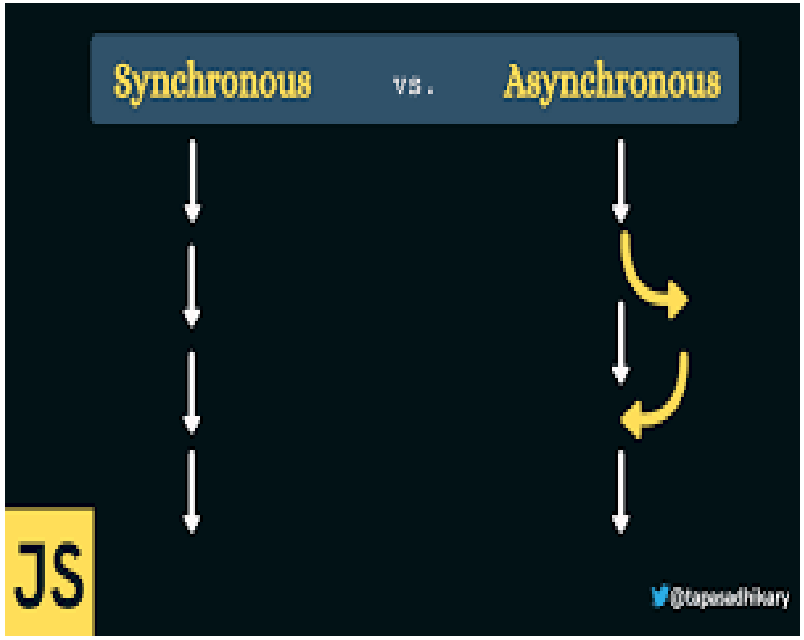
- Para lanzar tareas en Celery, utilizamos el método send_task().
- El método send_task() nos permite enviar una tarea específica para su ejecución.
- Podemos especificar el nombre de la tarea y proporcionar los argumentos necesarios.
- Celery se encargará de encolar y distribuir la tarea a los trabajadores disponibles.
- La tarea se ejecutará de forma asíncrona, lo que nos permite continuar con otras operaciones sin esperar su finalización.
- Esto proporciona una forma eficiente de gestionar y ejecutar tareas en entornos distribuidos.

```
from celery import Celery
app = Celery('myapp', broker='pyamqp://guest@localhost//')

result = app.send_task('my_task', args=[arg1, arg2])
```

Ventajas de Celery en la gestión distribuida

- **Flexibilidad en la ejecución de tareas asíncronas:** Celery permite ejecutar tareas de manera asíncrona, lo que proporciona mayor flexibilidad en la gestión de procesos y tiempos de ejecución.
- **Capacidad para gestionar servicios de manera eficiente:** Celery facilita la gestión de servicios distribuidos al proporcionar herramientas para encolar y distribuir tareas, optimizando el uso de recursos y garantizando un rendimiento eficiente.



Uso de Celery en aplicaciones reales

- **Procesamiento de colas de tareas en tiempo real:**
  - Gestión eficiente de flujos de trabajo complejos.
  - Distribución de tareas en múltiples nodos para un procesamiento paralelo.

- **Aplicaciones web escalables:**
  - Procesamiento asíncrono de solicitudes HTTP.
  - Distribución de tareas en varios servidores para una mayor capacidad de respuesta.

- **Procesamiento y análisis de grandes volúmenes de datos:**
  - División de tareas en tareas más pequeñas para un procesamiento paralelo.
  - Utilización de recursos de manera eficiente para un análisis de datos rápido.

- **Programación de tareas programadas y procesamiento en segundo plano:**
  - Ejecución de tareas programadas en intervalos específicos.
  - Procesamiento en segundo plano para mejorar la capacidad de respuesta de la aplicación.

Casos de éxito con Celery

- **Spotify:** Implementación de Celery en su sistema de gestión de tareas distribuidas. Resultados: mejora del rendimiento en un 50% y capacidad para manejar tareas concurrentes a gran escala.
- **Instagram:** Utilización de Celery en el procesamiento de imágenes y el manejo de colas de mensajes. Resultados: escalabilidad mejorada y reducción de la latencia en la entrega de imágenes.
- **Airbnb:** Integración de Celery en su plataforma de reserva y gestión de alojamientos. Resultados: mejora en la eficiencia del procesamiento de solicitudes y capacidad para gestionar un alto volumen de reservas.

Ejemplo en Python

In [ ]:
```
# Instalación:
# Asegúrate de tener instalado Celery en tu entorno Python. Puedes instalarlo usando pip:
```

In [ ]:
```
pip install celery
```

In [ ]:
```
# Configuración:
# Crea un archivo de configuración celeryconfig.py para definir la configuración de Celery. Por ejemplo:
```

In [ ]:
```
# celeryconfig.py

broker_url = 'amqp://guest:guest@localhost:5672//'  # URL del broker (RabbitMQ)
result_backend = 'redis://localhost:6379/0'  # URL del backend de resultados (Redis)
```

In [ ]:
```
# Creación de tareas:
# Define las tareas que deseas ejecutar de forma distribuida en un archivo tasks.py.
# Por ejemplo, vamos a crear una tarea que calcula la suma de dos números:
```

In [ ]:
```
# tasks.py
from celery import Celery

app = Celery('myapp')
app.config_from_object('celeryconfig')

@app.task
def suma(a, b):
    return a + b
```

In [ ]:
```
# Lanzamiento de tareas:
# En otro archivo o en la consola interactiva de Python, puedes lanzar las tareas utilizando Celery. Por ejemplo:
```

In [ ]:
```
from tasks import suma

# Lanza la tarea de suma de forma distribuida
result = suma.delay(4, 6)

# Espera a que la tarea se complete y obtén el resultado
print(result.get())
```

www.unir.net