PONTIFICIA UNIVERSIDAD JAVERIANA

FACULTAD DE INGENIERÍA

INGENIERÍA DE SISTEMAS

TALLER 21- SISTEMAS DISTRIBUIDOS

PATRÓN PUBLICADOR-SUSCRIPTOR (ZEROMQ)

SEBASTIAN ORJUELA SÁNCHEZ

CAMILO ANDRES RODRIGUEZ RUIZ

CARLOS ANDRES BORJA GONZÁLEZ

SAMIR ALEJANDRO SANCHEZ ROMERO

PROFESOR

OSBERTH DE CASTRO CUEVAS

FECHA ENTREGA: 8/11/2023

Informe de implementación de una aplicación de cálculo distribuido utilizando el patrón publicador-suscriptor con broker, usando el middleware zeromq

Introducción

El presente informe detalla la implementación de la aplicación de cálculo distribuido desarrollada en el taller anterior (T11), utilizando el patrón publicador-suscriptor con un broker mediante el middleware ZeroMQ. La estructura general de la aplicación permanece inalterada, manteniendo los nodos existentes (Cliente, Server, IvaNode, Suma) y añadiendo un nodo adicional denominado "Broker". Esta nueva arquitectura permite una mayor flexibilidad al introducir la capacidad de agregar más servidores de cálculo o servidores de operación según las necesidades del sistema.

Objetivos del proyecto

- Implementar el patrón Publicador-Suscriptor con ZeroMq con una arquitectura ya establecida en el taller 11.
- Modificar el cliente para que funcione como publicador.
- Modificar los demás nodos para que funcionen como un suscriptor en el patrón diseñado.
- Implementar el broker para que gestione la comunicación entre el cliente y los diferentes nodos.
- Implementar fallos y desconexiones para poder encontrar soluciones.
- Implementar la escalabilidad en el código para que se pueda implementar en otros lenguajes o SOPs

Descripción general

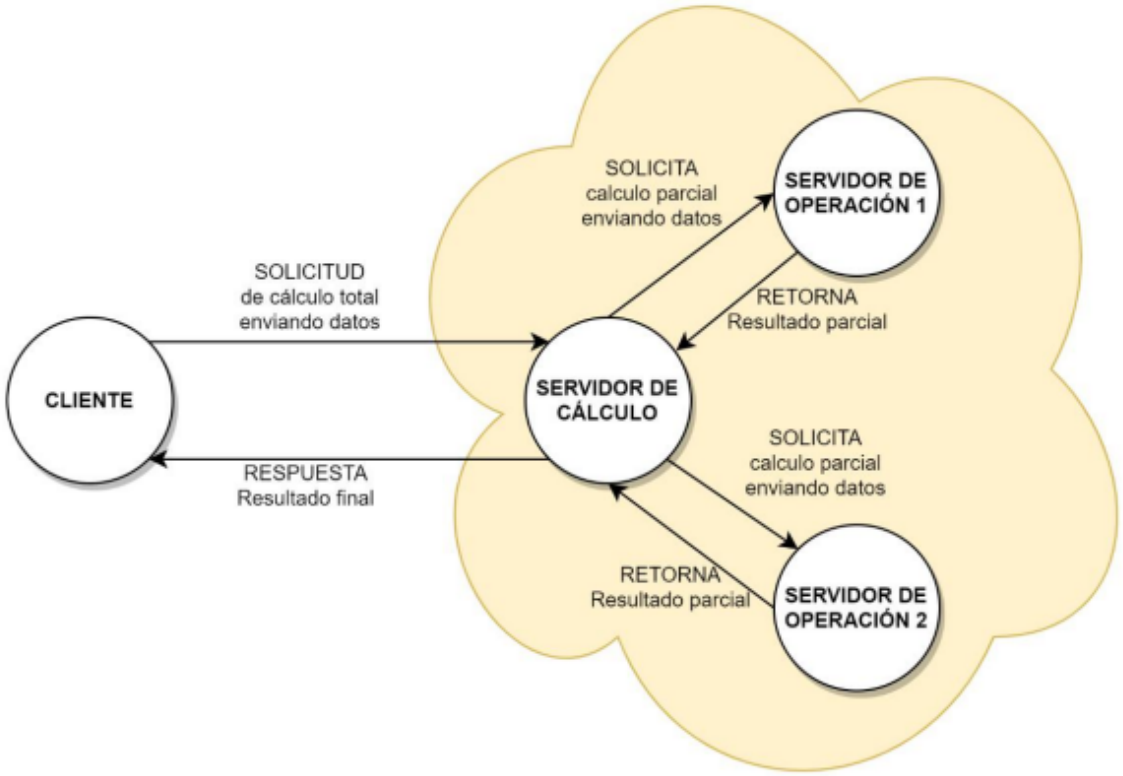

Imagen1: estructura mínima del cálculo distribuido mediante la arquitectura propuesta en el taller anterior T11.

La aplicación de cálculo distribuido consiste en varios nodos interconectados, cada uno desempeñando un papel específico en la ejecución de operaciones distribuidas. A continuación, se presenta una descripción general de los nodos principales y su funcionalidad:

Cliente (Publicador):

- Interactúa con el usuario para recopilar información sobre productos (nombre, categoría y precio).

- Envía la lista de productos al Broker a través de mensajes publicados.

- Puede solicitar promociones, si el cliente lo desea y si el cliente tiene más de 60 años de edad.

Broker

- Actúa como intermediario central en la red, gestionando las conexiones y comunicaciones entre nodos.

- Recibe mensajes de los clientes (Publicadores) y los enruta a los nodos suscriptores adecuados.

- Gestiona dinámicamente la conexión del CopiaServidor con el Cliente en caso de que el servidor principal no esté disponible por diversas razones como lo sería: daño, desconección, actualización, entre otros problemas .

- Crea un nuevo nodo para consultar promociones si el Cliente lo solicita.

Servidor (Suscriptor)

- Recibe la lista de productos del Cliente a través del Broker.

- Propaga la lista al IvaNode y luego recibe la lista modificada de vuelta.

- Envía la lista con iva al nodo Suma y recibe el total a pagar.

- Devuelve el total al Cliente a través del Broker.

IvaNode (Suscriptor)

- Recibe la lista de productos del Servidor a través del Broker.

- Agrega un 19% de IVA a los productos que no pertenecen a la categoría "Canasta".

- Devuelve la lista modificada al Servidor a través del Broker.

Suma (Suscriptor)

- Recibe la lista con IVA incluido del Servidor a través del Broker.

- Realiza la suma total de los precios con IVA.

- Devuelve el total a pagar al Servidor a través del Broker.

CopiaServidor (Suscriptor)

- Conecta al Cliente con el Servidor en caso de que el Servidor principal no esté disponible.

- Recibe y reenvía los mensajes entre el Cliente y el broker.

Descuento(Suscriptor)

- Recibe la confirmación, la lista de productos y la edad por parte del cliente.
- Se encarga de hacerle un 10 porciento de descuento en todos los productos y retorna el total a pagar al cliente con ayuda de la conexión del broker

Esta reimplementación proporciona una mayor flexibilidad y robustez al sistema, permitiendo la expansión y gestión eficiente de los nodos mediante el uso del patrón publicador-suscriptor con un broker basado en ZeroMQ.

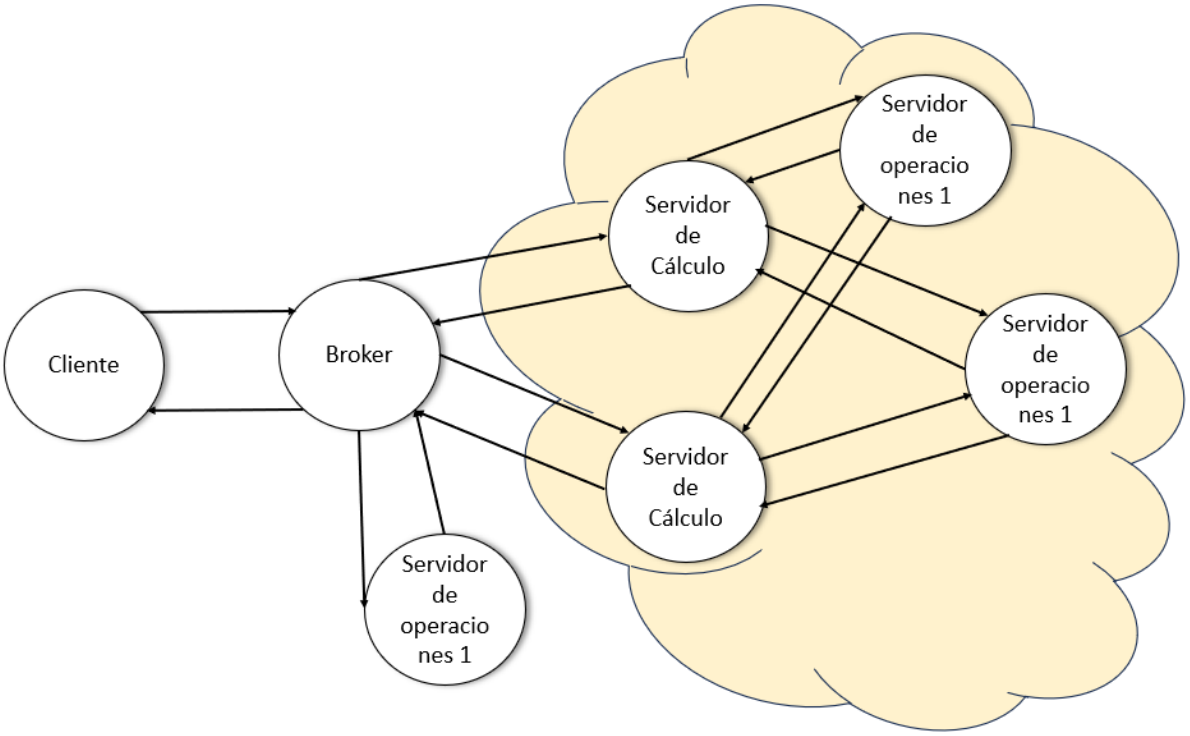

Imagen 2: estructura mínima del cálculo distribuido mediante la arquitectura propuesta por los estudiantes en este taller T21.

Planteamiento del problema

La implementación de sistemas distribuidos presenta desafíos significativos en términos de escalabilidad, flexibilidad y gestión eficiente de recursos. En el contexto de la aplicación de cálculo distribuido desarrollada en el Taller 11, se identifican varias problemáticas que limitan su capacidad de adaptación y expansión. Estas problemáticas sirven como base para la necesidad de la reimplementación utilizando el patrón publicador-suscriptor con un broker mediante el middleware ZeroMQ.
Escalabilidad Limitada: El sistema actual carece de la capacidad para adaptarse dinámicamente a la inclusión de nuevos nodos, ya sean servidores de cálculo o servidores de operación. Esto limita la escalabilidad y dificulta la gestión eficiente de recursos en entornos cambiantes.

Conectividad del Cliente: La conexión directa entre el Cliente y el Servidor principal puede generar inconvenientes cuando el Servidor principal no está disponible. La necesidad de una gestión más robusta de la conexión entre el Cliente y los nodos de operación se convierte en una prioridad.

Gestión de Promociones: El sistema actual no incorpora un mecanismo eficiente para gestionar consultas de promociones por parte del Cliente. La falta de un nodo específico para estas consultas limita la capacidad de adaptación a requisitos específicos, como descuentos para clientes mayores de 60 años.

Dependencia de Middleware Específico: El uso exclusivo de sockets como medio de comunicación en la implementación original limita la flexibilidad y dificulta la adaptación a

cambios en los requisitos del sistema. La introducción de un middleware específico, como ZeroMQ, se presenta como una solución para superar esta limitación y mejorar la eficiencia en la comunicación entre nodos.

Gestión de Errores: La aplicación actual carece de un mecanismo robusto para gestionar situaciones de fallo, como la falta de respuesta de nodos específicos. La necesidad de una gestión adecuada de errores y una mayor robustez en la comunicación entre nodos se vuelve esencial para garantizar la integridad y confiabilidad del sistema distribuido.

La reimplementación de la aplicación utilizando el patrón publicador-suscriptor con un broker mediante ZeroMQ busca abordar estas problemáticas, ofreciendo una arquitectura más modular, flexible y adaptativa para satisfacer las demandas de un entorno distribuido dinámico. Este enfoque se centra en mejorar la escalabilidad, la gestión de conexiones y la capacidad de respuesta a requisitos específicos del sistema, proporcionando así una solución más eficiente y robusta.

Estrategia utilizada

Para abordar las limitaciones identificadas en la implementación original del sistema de cálculo distribuido, se ha optado por una estrategia integral que involucra la reestructuración de la arquitectura mediante la adopción del middleware ZeroMQ y la implementación del patrón publicador-suscriptor. A continuación, se detallan los pasos clave de esta estrategia:

Integración de ZeroMQ
- Se seleccionó ZeroMQ como middleware para mejorar la comunicación entre los nodos de la aplicación distribuida. ZeroMQ ofrece una arquitectura ligera y eficiente que facilita la implementación de patrones de comunicación avanzados, como el publicador-suscriptor.

Adopción del Patrón Publicador-Suscriptor
- Se reconfiguró la arquitectura original para seguir el patrón publicador-suscriptor. Cada nodo actúa como un suscriptor que se suscribe a un tema específico (o canal), permitiendo una comunicación más flexible y modular entre los componentes del sistema.

Incorporación del Nodo Broker
- Se introdujo un nuevo nodo llamado "Broker" que actúa como intermediario central en la red. El Broker recibe mensajes publicados por los clientes y enruta dinámicamente estos mensajes a los nodos suscriptores correspondientes, facilitando así una comunicación eficiente y escalable.

Gestión Dinámica de Conexiones
- El Broker asume la responsabilidad de gestionar las conexiones entre los clientes y los servidores. En caso de que el servidor principal no esté disponible, el Broker conecta dinámicamente al Cliente con un nodo alternativo llamado "CopiaServidor". Esto mejora la robustez del sistema al garantizar la continuidad de las operaciones incluso en ausencia del servidor principal.

Creación Dinámica de Nodos

- El Broker tiene la capacidad de crear dinámicamente un nuevo nodo para gestionar consultas específicas, como las solicitudes de promociones para clientes mayores de 60 años. Esto permite una adaptabilidad y expansión eficiente del sistema según las demandas del usuario.

Manejo de Errores Mejorado
- Se implementaron mecanismos robustos para manejar situaciones de error, como la falta de respuesta de nodos específicos. El Broker gestiona de manera eficiente las situaciones de fallo, garantizando una mayor confiabilidad y recuperación ante posibles interrupciones en la red.

Flexibilidad y Escalabilidad Mejoradas
- La adopción de ZeroMQ y el patrón publicador-suscriptor ofrece una mayor flexibilidad y escalabilidad al sistema. La arquitectura resultante permite la fácil adición de nuevos nodos, ya sean servidores de cálculo o servidores de operación, sin afectar la operación global del sistema.

Esta estrategia de reestructuración se enfoca en superar las limitaciones de la implementación original, mejorando la modularidad, la eficiencia en la comunicación y la capacidad de respuesta a requisitos específicos. La combinación de ZeroMQ y el patrón publicador-suscriptor proporciona una base sólida para un sistema distribuido más robusto y adaptable.

Implementación del planteamiento del problema

```python
import zmq
import time


1 usage  Carlos *
def cliente():
    context = zmq.Context()
    socket = context.socket(zmq.REQ)

    # Intentar conectarse al servidor principal durante 10 segundos
    tiempo_inicio = time.time()
    while time.time() - tiempo_inicio < 10:
        try:
            socket.connect("tcp://localhost:5559")
            break
        except zmq.error.ZMQError:
            print("Intentando conectar al servidor principal...")
            time.sleep(1)
    else:
        # Si no se conecta al servidor principal, intentar con el CopiaServidor
        print("No se pudo conectar al servidor principal. Intentando con CopiaServidor...")
        socket.connect("tcp://localhost:5560")  # Conectar al CopiaServidor

    # Crear lista de productos
    productos = []
    while True:
        nombre = input("Ingrese nombre del producto (o 'fin' para terminar): ")  # La peticion de los datos del prducto
        if nombre.lower() == 'fin':
            break
        categoria = input("Ingrese la categoría del producto: ")
        precio = float(input("Ingrese el precio del producto: "))
```

```python
        productos.append({'nombre': nombre, 'categoria': categoria, 'precio': precio})

    # Enviar lista de productos al servidor
    socket.send_pyobj(productos)

    # Recibir el total a pagar desde el servidor
    total = socket.recv_pyobj()

    # Preguntar la edad al usuario
    edad = int(input("¿Cuántos años tienes? "))

    # Si la edad es mayor a 60, solicitar descuento
    if edad > 60:
        print(f"El total a pagar es: ${total:.2f}")
        descuento = input("¿Desea aplicar el descuento? (si/no): ").lower() == "si"

        if descuento:
            descuento_socket = context.socket(zmq.REQ)
            descuento_socket.connect("tcp://localhost:5690")  # peticion con el nodo Descuento
            # Enviar el total y la edad al nodo de descuento
            descuento_socket.send_pyobj((total, edad))

            # Recibir el total con descuento y la lista de premios
            total_con_descuento, premios = descuento_socket.recv_pyobj()

            print(f"Total con descuento: ${total_con_descuento:.2f}")

            if premios:
                print("Lista de premios:")
                for premio in premios:
                    print(premio)
```

```
            if premios:
                print("Lista de premios:")
                for premio in premios:
                    print(premio)
        else:
            print("No se aplicará el descuento.")
    else:
        print(f"El total a pagar es: ${total:.2f}")


if __name__ == "__main__":
    cliente()
```

Imagen 3: implementación del Nodo cliente(Publicador) según lo que propusimos.

```
import zmq

1 usage  Carlos *
def broker():
    context = zmq.Context()
    frontend = context.socket(zmq.ROUTER)  # Maneja las conexiones de los clientes, actúa como enrutador
    backend = context.socket(zmq.DEALER)  # Socket maneja la conexión al servidor, actúa como distribuidor

    frontend.bind("tcp://*:5559")  # Puerto que recibe el mensaje del cliente.
    backend.bind("tcp://*:5560")  # Puerto que recibe los mensajes del servidor.

    # Creación del socket con el nodo Descuento
    descuento_socket = context.socket(zmq.REQ)
    descuento_socket.connect("tcp://localhost:5690")

    # Ciclo principal del broker
    poller = zmq.Poller()
    poller.register(frontend, zmq.POLLIN)
    poller.register(backend, zmq.POLLIN)

    while True:
        socks = dict(poller.poll(10000))  # Espera 10 segundos por eventos

        if frontend in socks:
            # Enviar mensaje desde el cliente al servidor
            mensaje = frontend.recv_multipart()
            backend.send_multipart(mensaje)

        if backend in socks:
            # Enviar mensaje desde el servidor al cliente
            mensaje = backend.recv_multipart()
```

```
                frontend.send_multipart(mensaje)


if __name__ == "__main__":
    broker()
```

imagen 4: Implementación del nodo broker y gestión de comunicación del cliente con el servidor.

```python
import zmq # libreria para utilizar el middleware ZeroMQ
import time # Libreria para utilizar el tiempo de espera
import logging

# Configuración básica de logging
logging.basicConfig(level=logging.INFO)

1 usage  Carlos *
def server():
    context = zmq.Context()
    socket = context.socket(zmq.REP)
    socket.connect("tcp://localhost:5560")# coneccion al nodo broker a traves del socket REP
    logging.info("Servidor esperando un publicador...")

    # Ciclo principal del servidor
    while True:
        #Se crean sockets REQ (Request) para la comunicación con nodos especializados en IVA y suma.
        iva_socket = context.socket(zmq.REQ)
        suma_socket = context.socket(zmq.REQ)

        try:
            #Se establece la conexión con el nodo de IVA
            iva_socket.connect("tcp://localhost:5570")
            #Se establece la conexión con el nodo Suma
            suma_socket.connect("tcp://localhost:5580")

            # Recibir la lista de productos desde el cliente
            message = socket.recv_pyobj()

            # Mostrar la lista original
            logging.info("Lista de productos recibida:")
```

```
        logging.info("Lista de productos recibida:")
        for producto in message:
            logging.info(producto)

        # Intentar aplicar el IVA con IvaNode
        try:
            #Se intenta conectar con un nodo de IVA (IvaNode) para aplicar el IVA. Si no se puede conectar,
            # se aplica el IVA localmente a aquellos productos que no pertenecen a la categoría 'Canasta'.
            iva_socket.send_pyobj(message)
            iva_response = iva_socket.recv_pyobj()
        except zmq.Again:
            logging.warning("No se pudo conectar con IvaNode. Aplicando IVA localmente...")
            iva_response = message
            for producto in iva_response:
                if producto['categoria'] != 'Canasta':
                    producto['precio'] *= 1.19

        # Mostrar la nueva lista con el IVA
        logging.info("\nNueva lista con IVA aplicado:")
        for producto in iva_response:
            logging.info(producto)

        # Intentar realizar la suma con SumaNode
        try:
            suma_socket.send_pyobj(iva_response)
            total_response = suma_socket.recv_pyobj()
        except zmq.Again:
            logging.warning("No se pudo conectar con SumaNode. Realizando la suma localmente...")
            total_response = sum([producto['precio'] for producto in iva_response])

        # Mostrar el total a pagar
```

```
            # Mostrar el total a pagar
            logging.info(f"\nTotal a pagar (con suma): ${total_response}")

            # Enviar el total al cliente
            socket.send_pyobj(total_response)

            logging.info("Servidor esperando un publicador...")

        except Exception as e:

            logging.error(f"Error en el servidor: {e}")

        finally:
            # Cerrar sockets
            try:
                iva_socket.close()
                suma_socket.close()
            except zmq.error.ZMQError as e:
                logging.error(f"Error al cerrar sockets: {e}")

        # Esperar 10 segundos antes de intentar nuevamente la conexión
        time.sleep(10)

if __name__ == "__main__":
    server()
```

Imagen 5: implementación del Nodo servidor(Suscriptor) según lo que propusimos.

```python
def iva_node():
    #Se crea un contexto de ZeroMQ y se vincula un socket REP al puerto 5570 para recibir mensajes del servidor.
    context = zmq.Context()
    socket = context.socket(zmq.REP)
    socket.bind("tcp://*:5570")

    # Ciclo principal del IvaNode
    while True:
        try:
            # Recibir la lista de productos del servidor
            message = socket.recv_pyobj()

            # Aplicar el IVA a los productos según la categoría
            for producto in message:
                if producto['categoria'] != 'Canasta':
                    producto['precio'] *= 1.19

            # Mostrar la nueva lista con el IVA
            print("\nNueva lista con IVA aplicado en IvaNode:")
            for producto in message:
                print(producto)

            # Enviar la lista con IVA al servidor
            socket.send_pyobj(message)
    #En caso de que el contexto de ZeroMQ se termine, se sale del bucle.
        except zmq.ContextTerminated:
            break

if __name__ == "__main__":
    iva_node()
```

Imagen 6: implementación del Nodo IvaNode(Suscriptor) según lo que propusimos.

```python
import zmq

1 usage  Carlos *
def suma_node():
    #Se crea un contexto de ZeroMQ y se vincula un socket REP al puerto 5580 para recibir mensajes del servidor.
    context = zmq.Context()
    socket = context.socket(zmq.REP)
    socket.bind("tcp://*:5580")

    # Ciclo principal del SumaNode
    while True:
        try:
            # Recibir la lista de productos con IVA del servidor
            message = socket.recv_pyobj()

            # Sumar los precios
            total_response = sum([producto['precio'] for producto in message])

            # Mostrar el total con la suma realizada en SumaNode
            print(f"\nTotal a pagar (con suma en SumaNode): ${total_response}")

            # Enviar el total al servidor
            socket.send_pyobj(total_response)
        except zmq.ContextTerminated:
            #En caso de que el contexto de ZeroMQ se termine, se sale del bucle.
            break

if __name__ == "__main__":
    suma_node()
```

Imagen 7: Implementación del nodo Suma(Suscriptor) según la propuesta que establecimos.

```python
import zmq

1 usage  Carlos *
def descuento():
    #Se crea un contexto de ZeroMQ y se vincula un socket REP al puerto 5690 para recibir mensajes del broker.
    context = zmq.Context()
    socket = context.socket(zmq.REP)
    socket.bind("tcp://*:5690")

    while True:
        # Recibir el total y la edad desde el broker
        total_pagar, edad = socket.recv_pyobj()

        # Aplicar descuento si la edad es mayor a 60 años
        if edad > 60:
            descuento = total_pagar * 0.2
            total_con_descuento = total_pagar - descuento
            premios = ["Premio A", "Premio B", "Premio C"]

            # Enviar el total con descuento y la lista de premios al broker
            socket.send_pyobj((total_con_descuento, premios))
        else:
            # Enviar el total sin descuento al broker
            socket.send_pyobj((total_pagar, None))

if __name__ == "__main__":
    descuento()
```

Imagen 8: Implementación del nodo Descuento(Suscriptor) según la propuesta que establecimos.

```python
import zmq # libreria para utilizar el middleware ZeroMQ
import time # Libreria para utilizar el tiempo de espera
import logging

# Configuración básica de logging
logging.basicConfig(level=logging.INFO)

new *
def CopiaServer():
    context = zmq.Context()
    socket = context.socket(zmq.REP)
    socket.connect("tcp://localhost:5560")# coneccion al nodo broker a traves del socket REP
    logging.info("Servidor esperando un publicador...")

    # Ciclo principal del servidor
    while True:
        #Se crean sockets REQ (Request) para la comunicación con nodos especializados en IVA y suma.
        iva_socket = context.socket(zmq.REQ)
        suma_socket = context.socket(zmq.REQ)

        try:
            #Se establece la conexión con el nodo de IVA
            iva_socket.connect("tcp://localhost:5570")
            #Se establece la conexión con el nodo Suma
            suma_socket.connect("tcp://localhost:5580")

            # Recibir la lista de productos desde el cliente
            message = socket.recv_pyobj()

            # Mostrar la lista original
            logging.info("Lista de productos recibida:")
```

```
            for producto in message:
                logging.info(producto)

            # Intentar aplicar el IVA con IvaNode
            try:
                #Se intenta conectar con un nodo de IVA (IvaNode) para aplicar el IVA. Si no se puede conectar,
                # se aplica el IVA localmente a aquellos productos que no pertenecen a la categoría 'Canasta'.
                iva_socket.send_pyobj(message)
                iva_response = iva_socket.recv_pyobj()
            except zmq.Again:
                logging.warning("No se pudo conectar con IvaNode. Aplicando IVA localmente...")
                iva_response = message
                for producto in iva_response:
                    if producto['categoria'] != 'Canasta':
                        producto['precio'] *= 1.19

            # Mostrar la nueva lista con el IVA
            logging.info("\nNueva lista con IVA aplicado:")
            for producto in iva_response:
                logging.info(producto)

            # Intentar realizar la suma con SumaNode
            try:
                suma_socket.send_pyobj(iva_response)
                total_response = suma_socket.recv_pyobj()
            except zmq.Again:
                logging.warning("No se pudo conectar con SumaNode. Realizando la suma localmente...")
                total_response = sum([producto['precio'] for producto in iva_response])

            # Mostrar el total a pagar
            logging.info(f"\nTotal a pagar (con suma): ${total_response}")
```

```
            # Mostrar el total a pagar
            logging.info(f"\nTotal a pagar (con suma): ${total_response}")

            # Enviar el total al cliente
            socket.send_pyobj(total_response)

            logging.info("Servidor esperando un publicador...")

        except Exception as e:

            logging.error(f"Error en el servidor: {e}")

        finally:
            # Cerrar sockets
            try:
                iva_socket.close()
                suma_socket.close()
            except zmq.error.ZMQError as e:
                logging.error(f"Error al cerrar sockets: {e}")

        # Esperar 10 segundos antes de intentar nuevamente la conexión
        time.sleep(10)

if __name__ == "__main__":
    CopiaServer()
```

Imagen 9: Implementación del nodo CopiaServer(Suscriptor) según la propuesta que establecimos.

Pruebas del programa

Para evaluar el desempeño de nuestra aplicación, realizamos pruebas utilizando diferentes factores en la simulación de ejecución de un sistema distribuido pequeño en la vida real, las pruebas realizadas son las siguientes:

- El mejor de los casos: en la siguiente prueba correremos los nodos con el nodo Cliente realizando una petición a broker y este realizando la coneccion con el servidor principal, y este servidor principal enviando la lista de productos al nodo IvaNode para que implemente el Iva a cada uno de los productos que no tienen la categoría de Canasta después que este retorne al nodo servidor la lista con el iva implementado, ya con esta nueva lista el nodo Servidor le pasa la lista con el iva al nodo Suma y este realiza la suma de todos los productos con su iva y el nodo retorna al servidor el total a praga. ya con el total a pagar en el servidor este se lo retorna al cliente con ayuda de la conexión del Broker para que reciba el total a pagar. Después el cliente se le mostrará por pantalla si desea verificar si tiene descuento o no de su compra y recibirá el total definitivo a pagar. Hay que tener en cuenta que el nodo Copia Servidor también está pendiente a recibir una coneccion por parte del broker si llegado el caso el servidor principal no sirve

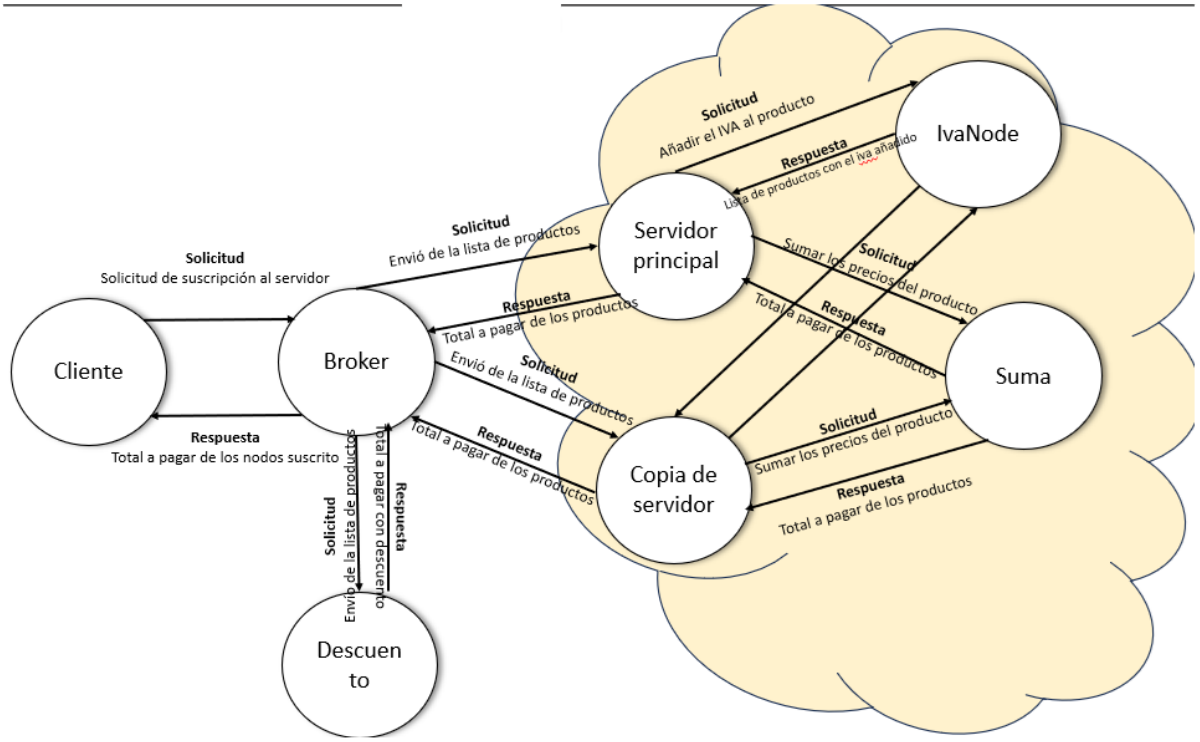

Imagen 10: estructura de la prueba del mejor de los casos

```
Windows PowerShell
Copyright (C) Microsoft Corporation. Todos los derechos reservados.

Instale la versión más reciente de PowerShell para obtener nuevas características y mejoras. https://aka.ms/PSWindows

(venv) PS D:\Carlos\Documentos\Materia de 8vo semestre\Sitemas distribuidos\Taller21> cd .\Taller\
(venv) PS D:\Carlos\Documentos\Materia de 8vo semestre\Sitemas distribuidos\Taller21\Taller> python .\Server.py
INFO:root:Servidor esperando un publicador...
```

Imagen 11: Espera de un publicador por parte del servidor

```
(venv) PS D:\Carlos\Documentos\Materia de 8vo semestre\Sitemas distribuidos\Taller21\Taller> python Cliente.py
Ingrese nombre del producto (o 'fin' para terminar): Manzana
Ingrese la categoría del producto: Canasta
Ingrese el precio del producto: 2000
Ingrese nombre del producto (o 'fin' para terminar): Carro
Ingrese la categoría del producto: Vehiculo
Ingrese el precio del producto: 147852369
```

Imagen 12:Los datos suministrados por el publicador en donde el podrá realizar un montón de productos hasta que escriba fin.

```
(venv) PS D:\Carlos\Documentos\Materia de 8vo semestre\Sitemas distribuidos\Taller21> cd .\Taller\
INFO:root:{'nombre': 'Manzana', 'categoria': 'Canasta ', 'precio': 2000.0}
INFO:root:{'nombre': 'Carro', 'categoria': 'Vehiculo', 'precio': 147852369.0}
INFO:root:
Nueva lista con IVA aplicado:
INFO:root:{'nombre': 'Manzana', 'categoria': 'Canasta ', 'precio': 2380.0}
INFO:root:{'nombre': 'Carro', 'categoria': 'Vehiculo', 'precio': 175944319.10999998}
INFO:root:
Total a pagar (con suma): $175946699.10999998
INFO:root:Servidor esperando un publicador...
```

Imagen 13: El servidor cuando recibe un listado por parte del publicador y lo muestra por consola lo que recibió y sigue prendido esperando otro publicador

```
Nueva lista con IVA aplicado en IvaNode:
{'nombre': 'Manzana', 'categoria': 'Canasta ', 'precio': 2380.0}
{'nombre': 'Carro', 'categoria': 'Vehiculo', 'precio': 175944319.10999998}
```

Imagen 14: El nodo IvaNode cuando recibe un listado por parte del publicador y lo muestra por consola lo que recibió y sigue prendido esperando otra lista del servidor. añade el iva al producto y retorna la lista con el iva agregado.

```
(venv) PS D:\Carlos\Documentos\Materia de 8vo semestre\Sitemas distribuidos\Taller21\Taller> python .\Suma.py

Total a pagar (con suma en SumaNode): $175946699.10999998
```

Imagen 15: El nodo Suma muestra por pantalla el total a pagar y lo retorna a el servidor

```
(venv) PS D:\Carlos\Documentos\Materia de 8vo semestre\Sitemas distribuidos\Taller21\Taller> python Cliente.py
Ingrese nombre del producto (o 'fin' para terminar): Manzana
Ingrese la categoría del producto: Canasta
Ingrese el precio del producto: 2000
Ingrese nombre del producto (o 'fin' para terminar): Carro
Ingrese la categoría del producto: Vehiculo
Ingrese el precio del producto: 147852369
Ingrese nombre del producto (o 'fin' para terminar): fin
¿Cuántos años tienes? 10
El total a pagar es: $175946699.11
```

Imagen 16: Se le pregunta al cliente la edad para hacerle el descuento, como es menor a 60 años no se aplica el descuento.

```
Ingrese nombre del producto (o 'fin' para terminar): fin
¿Cuántos años tienes? 70
El total a pagar es: $175945025.97
¿Desea aplicar el descuento? (si/no): si
```

Imagen 17: Se le pregunta al cliente la edad para hacerle el descuento, como es mayor a 60 años se le pregunta si se quiere hacer el descuento y se realiza el descuento.

- Prueba cuando el nodo 2: nodo servidor de cálculo (principal) se encuentra desconectado o sin servicio:

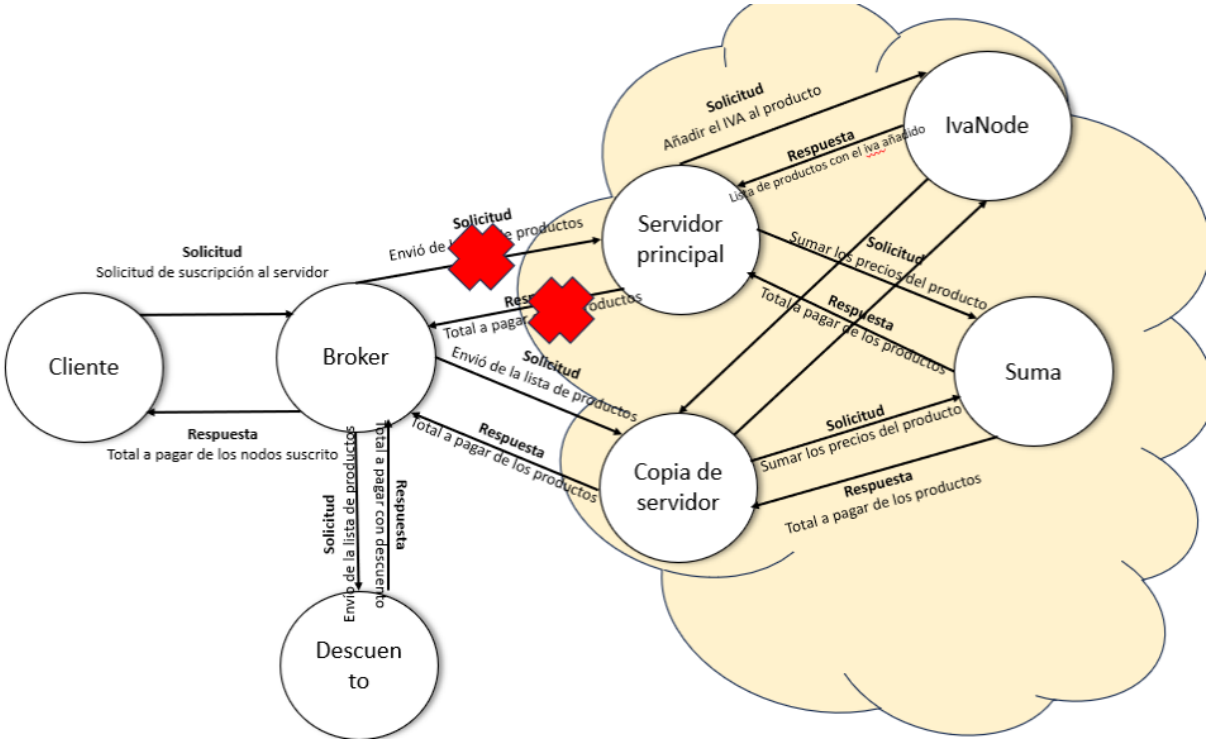

Imagen 18: modelo cuando no funciona la comunicación entre el nodo cliente, broker y el nodo servidor principal

En un modelo de sistemas distribuido siempre existirá un backup del servidor el cual lo reemplazará al momento de que dicho servidor se dañe o no de respuesta al cliente que está solicitando entrar a hacer el papel del servidor principal hasta que arreglen al servidor principal. Para poder identificar si el servidor no responde usamos la estrategia de los threads con un timer o método reloj en donde si no se tiene un respuesta en 1000 milisegundos se conecte a este backup de servidor.

```python
while True:
    socks = dict(poller.poll(10000))  # Espera 10 segundos por eventos

    if frontend in socks:
        # Enviar mensaje desde el cliente al servidor
        mensaje = frontend.recv_multipart()
        backend.send_multipart(mensaje)

    if backend in socks:
        # Enviar mensaje desde el servidor al cliente
        mensaje = backend.recv_multipart()
        frontend.send_multipart(mensaje)
```

Imagen 19: pedazo del código del nodo Broker el cual es la que se encarga de realizar la coneccion entre el servidor y el cliente, espera 10 segundos de coneccion y como no se conectó entonces se conecta a la copia del servidor

Con ayuda de ese pedazo validamos que el cliente siempre tenga una respuesta y no se de cuenta de que se daño o se cayeron los servidores. Cabe aclarar que como es un producto controlado mostramos al cliente el aviso de que se está conectando a un nuevo servidor, por otro lado en la vida real este mensaje no se mostrará al usuario.

```
INFO:root:Servidor esperando un publicador...
INFO:root:Lista de productos recibida:
INFO:root:{'nombre': 'Manzana', 'categoria': 'Canasta', 'precio': 10000.0}
INFO:root:
Nueva lista con IVA aplicado:
INFO:root:{'nombre': 'Manzana', 'categoria': 'Canasta', 'precio': 10000.0}
INFO:root:
Total a pagar (con suma): $10000.0
INFO:root:Servidor esperando un publicador...
```

Imagen 20: comprobación de la conexión del nodo cliente a la copia del servidor principal y el sigue la misma propuesta previamente descrito y retorna el total a pagar

Con lo anterior podemos ver que cumplimos la arquitectura propuesta como solución de la imagen 18, no mostramos los demás servidores de operaciones ya que son los mismo que la imagen 14 y 15. para comprobar que se conecto el pc al servidor copia podemos ver que en su consola se observa la conexión con la ip del pc del cliente.

- Prueba cuando el servidor nodo Iva se encuentra desconectado con el servidor principal:

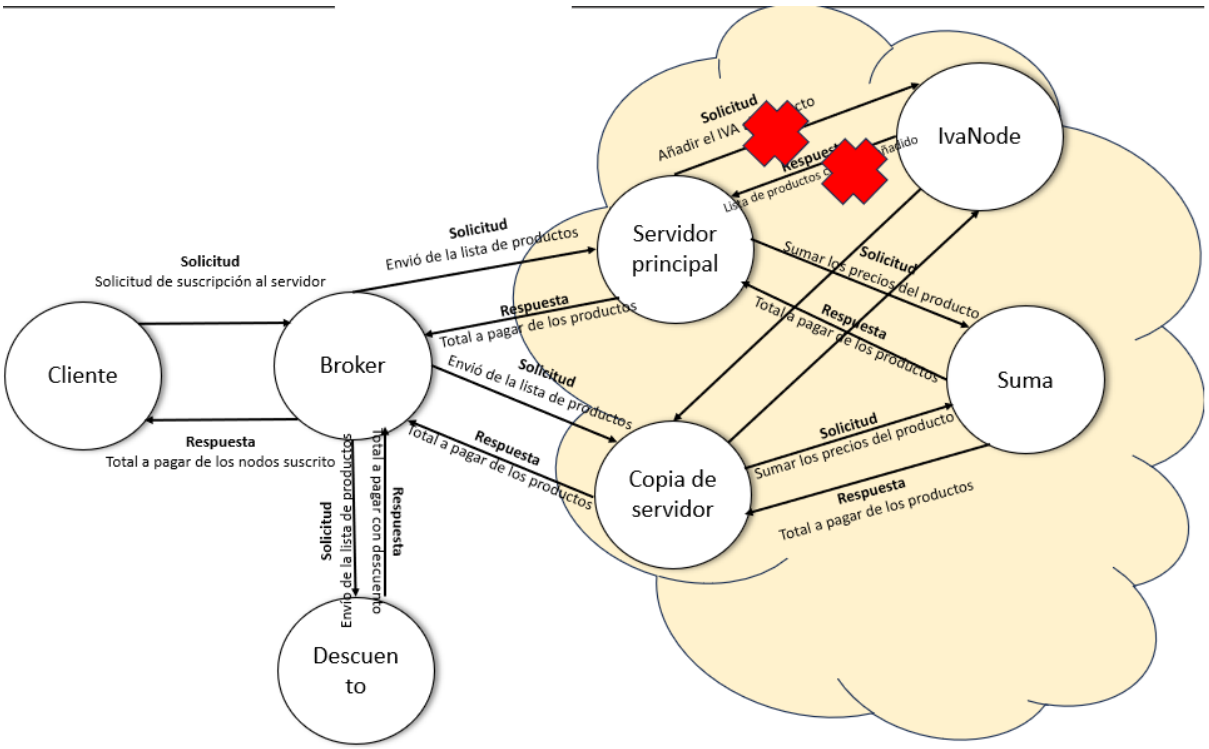

Imagen 21: modelo cuando no funciona la comunicación entre el nodo servidor de cálculo y el nodo servidor Nodo IVA

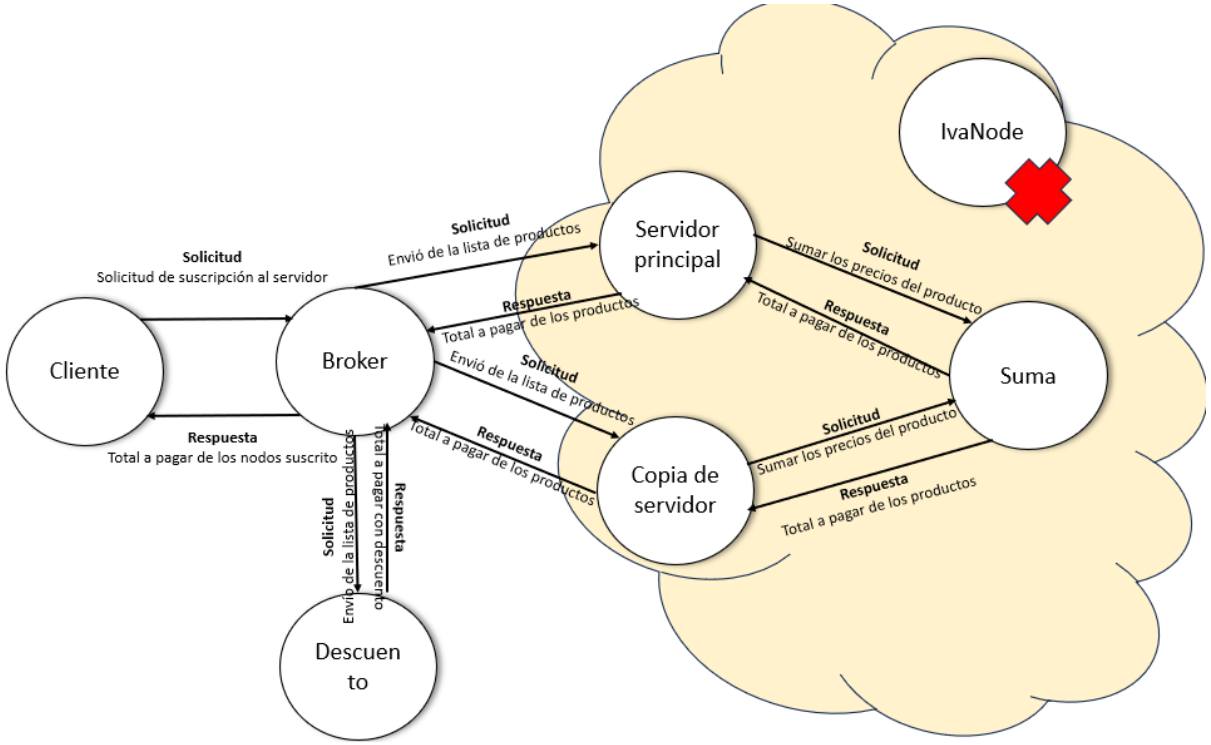

Imagen 22: modelo de la arquitectura con la solución propuesta por los programadores al momento de identificar que el servidor nodo Iva está dañado o desconectado

```
Ingrese la cantidad de productos: 2
Ingrese el nombre del producto 1: Naranja
Ingrese la categoría del producto 1: canasta
Ingrese el precio del producto 1: 4152
Ingrese el nombre del producto 2: Jabon
Ingrese la categoría del producto 2: Belleza
Ingrese el precio del producto 2: 7589
Total a pagar: 13971.79
PS D:\Carlos\Documentos\Materia de 8vo semestre\Sitemas distribuidos\DistribuidosServer\src\main\java>
```

Imagen 23: la compilación en consola del nodo Cliente con nuevos datos

```
        except zmq.Again:
            logging.warning("No se pudo conectar con IvaNode. Aplicando IVA localmente...")
            iva_response = message
            for producto in iva_response:
                if producto['categoria'] != 'Canasta':
                    producto['precio'] *= 1.19

        # Mostrar la nueva lista con el IVA
        logging.info("\nNueva lista con IVA aplicado:")
        for producto in iva_response:
            logging.info(producto)
```

imagen 24:pedazo del código en donde el servidor principal realiza la estrategia de reloj y espera 10000 milisegundos de respuesta y si no el realiza la operación del nodo IVA

- Prueba cuando el servidor IvaNode y el servidor Suma se encuentran desconectados:

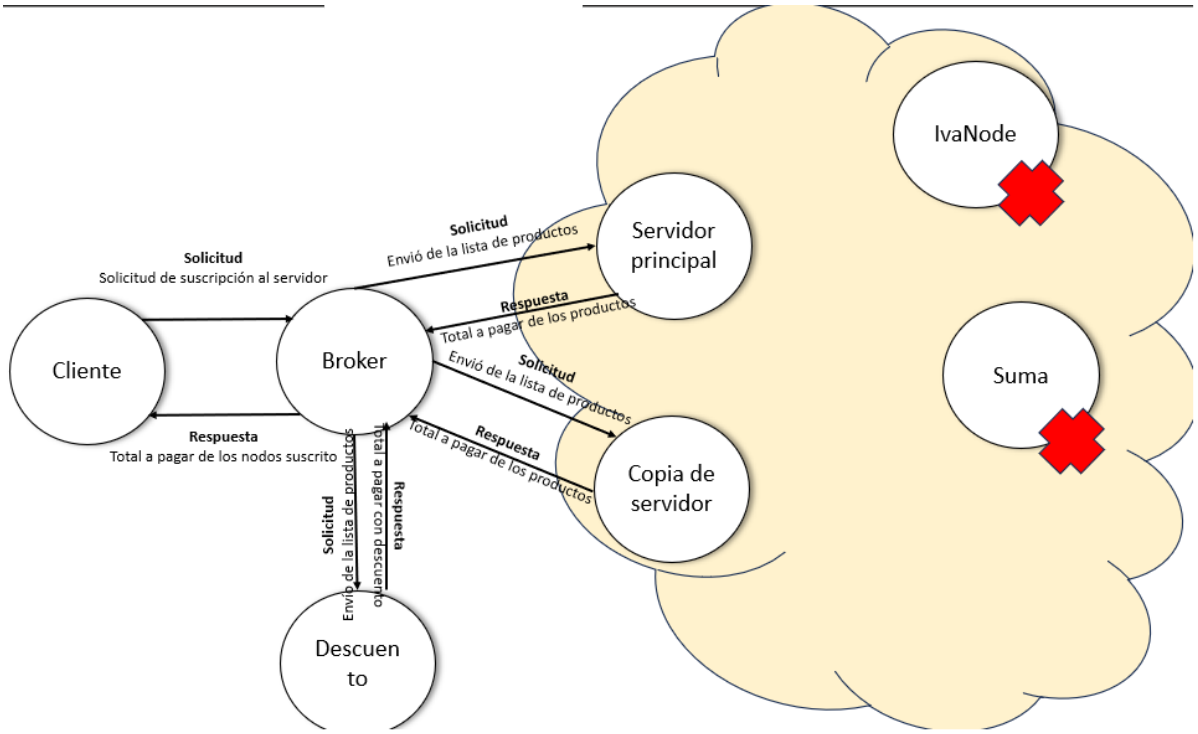

Imagen 25: modelo de fallo cuando la comunicación con el nodo IvaNode y la conexión Nodo Suma no se encuentra habilitada.

```
Ingrese la cantidad de productos: 2
Ingrese el nombre del producto 1: Naranja
Ingrese la categoría del producto 1: canasta
Ingrese el precio del producto 1: 4152
Ingrese el nombre del producto 2: Jabon
Ingrese la categoría del producto 2: Belleza
Ingrese el precio del producto 2: 7589
Total a pagar: 13971.79
PS D:\Carlos\Documentos\Materia de 8vo semestre\Sitemas distribuidos\DistribuidosServer\src\main\java>
```

Imagen 26: la compilación en consola del nodo Cliente con nuevos datos.

```python
# Intentar aplicar el IVA con IvaNode
try:
    #Se intenta conectar con un nodo de IVA (IvaNode) para aplicar el IVA. Si no se puede conectar,
    # se aplica el IVA localmente a aquellos productos que no pertenecen a la categoría 'Canasta'.
    iva_socket.send_pyobj(message)
    iva_response = iva_socket.recv_pyobj()
except zmq.Again:
    logging.warning("No se pudo conectar con IvaNode. Aplicando IVA localmente...")
    iva_response = message
    for producto in iva_response:
        if producto['categoria'] != 'Canasta':
            producto['precio'] *= 1.19

# Mostrar la nueva lista con el IVA
logging.info("\nNueva lista con IVA aplicado:")
for producto in iva_response:
    logging.info(producto)

# Intentar realizar la suma con SumaNode
try:
    suma_socket.send_pyobj(iva_response)
    total_response = suma_socket.recv_pyobj()
except zmq.Again:
    logging.warning("No se pudo conectar con SumaNode. Realizando la suma localmente...")
    total_response = sum([producto['precio'] for producto in iva_response])
```

imagen 22: Pedazo de código donde el servidor realiza lo del iva por su lado

Conclusiones

Reimplementación de la Aplicación de Cálculo Distribuido con ZeroMQ y Patrón Publicador-Suscriptor

1. Mejora en la Flexibilidad y Escalabilidad:

- La adopción del patrón publicador-suscriptor junto con ZeroMQ ha mejorado la flexibilidad y escalabilidad de la aplicación. Ahora, es posible añadir fácilmente más servidores de cálculo o servidores de operación sin afectar la arquitectura general.

2.Introducción del Nodo Broker

- La incorporación del nodo `Broker` como un intermediario central en la comunicación ha simplificado la gestión de conexiones entre el cliente y el servidor. El `Broker` facilita la publicación y suscripción de mensajes, gestionando de manera eficiente las solicitudes de cálculo.

3.Manejo de CopiaServer por el Broker

- El nodo `Broker` ahora es responsable de conectar al cliente con `CopiaServer` en caso de que el servidor principal no esté disponible. Esta mejora asegura una mayor disponibilidad y continuidad de la operación, ya que el `Broker` toma la decisión de conectar al cliente a la copia según la situación.

4. Integración del Nodo Breaker

- La introducción del nodo `Breaker` ha fortalecido la aplicación al manejar la conexión entre el cliente (publicador) y el servidor (suscriptor). Este nodo actúa como un punto de control central, mejorando la gestión y control de la comunicación entre ambas partes.

5. Consulta de Promociones para Personas Mayores

- La capacidad del `Broker` para crear dinámicamente un nodo cuando el cliente quiere consultar una promoción por ser una persona mayor a 60 años agrega un componente adicional de personalización. Esta funcionalidad demuestra la adaptabilidad del sistema para atender requisitos específicos del usuario.

6. Aumento en la Robustez del Sistema

- La reestructuración utilizando ZeroMQ y el patrón publicador-suscriptor ha mejorado la robustez del sistema. El `Broker` gestiona las conexiones y posibles fallos, lo que resulta en un comportamiento más confiable y una mejor capacidad de recuperación frente a situaciones inesperadas.

7. Simplificación en la Lógica del Cliente

- La reimplementación ha simplificado la lógica del cliente, ya que ahora se encarga principalmente de la publicación de mensajes. El cliente no está directamente involucrado en la gestión de conexiones o la toma de decisiones complejas, lo que facilita su comprensión y mantenimiento.

En resumen, la reestructuración de la aplicación de cálculo distribuido ha mejorado significativamente la flexibilidad, escalabilidad y robustez del sistema. La introducción de nodos como `Broker` y `Breaker` ha contribuido a una arquitectura más modular y fácil de mantener, proporcionando una base sólida para futuras expansiones y adaptaciones del sistema.