Entrega N°1 Proyecto My-Uber

Facultad de Ingeniería

Pontificia Universidad
JAVERIANA
Colombia

Nicolas Camacho Alarcón

Daniel Felipe León Pérez

María Paula Rodríguez Mancera

Profesora: Rafael Vicente Paez Mendez

Introducción a Sistemas Distribuidos

10 de octubre de 2024

1. Modelos del Sistema

Modelo Arquitectónico

El sistema **My-Uber** sigue una arquitectura cliente-servidor distribuida, diseñada para gestionar de manera eficiente la información de taxis y usuarios dentro de una cuadrícula que representa una ciudad. Los componentes principales son:

1. **Servidor Central**
    - Gestiona información de taxis y usuarios.
    - Asigna taxis a servicios solicitados por los usuarios.
    - Almacena información relevante en tiempo real (posiciones de taxis, solicitudes y servicios completados).
    - Abre puertos de comunicación específicos (e.g., TCP port 5555 para taxis y TCP port 5556 para usuarios) utilizando **ZeroMQ** para manejar diferentes tipos de mensajes y conexiones.
    - **Autenticación de Mensajes**: Valida los mensajes recibidos.
2. **Taxis**
    - Procesos individuales que representan a los taxis en la ciudad.
    - Comunican su posición actual al servidor y reciben asignaciones de servicios.
    - Emplean **sockets ZeroMQ (Publisher)** para enviar su posición periódicamente al servidor (e.g., cada 30 segundos de tiempo real, equivalente a 30 minutos simulados).
    - Cuentan con un módulo de control para moverse en la cuadrícula según su velocidad, actualizando su posición en intervalos regulares.
3. **Usuarios**
    - Representados como **hilos** creados por un generador, simulando solicitudes de taxis en la cuadrícula.
    - Se comunican con el servidor central a través de **ZeroMQ (Request/Reply)** para solicitar un servicio de taxi.
    - Generan solicitudes de servicio basadas en un tiempo `t` y establecen un tiempo de espera (`timeout`) antes de cancelar la solicitud si no hay respuesta del servidor.
4. **Respaldo del Servidor (Replica)**
    - Proceso espejo del servidor central, ejecutado en una máquina diferente (dirección IP diferente) para asegurar tolerancia a fallos.
    - Mantiene una sincronización continua de estado con el servidor principal para garantizar que, en caso de falla, pueda asumir la operación de manera transparente.
5. **Proceso de Health-Check**
    - Monitorea el estado del servidor central enviando **heartbeats cada 5 segundos** usando **ZeroMQ**.
    - Si no recibe respuesta tras 3 intentos consecutivos, activa el servidor de respaldo para asegurar la continuidad del servicio.

Balanceo de Carga (Opcional)

- Si el servidor principal y el de respaldo están activos simultáneamente, se puede implementar un balanceo de carga para distribuir solicitudes de usuarios y la gestión de taxis.

Modelo de Interacción

1. Comunicación entre Taxis y Servidor

- **Registro Inicial**: Los taxis se conectan al servidor central usando **ZeroMQ (Publisher)**, proporcionando su identificador único y posición inicial.
- **Actualización de Posiciones**: Envían su posición al servidor cada intervalo definido. En caso de cuadrículas grandes, el intervalo de envío se ajusta dinámicamente para evitar congestión.
- **Asignaciones de Servicios**: Los taxis disponibles reciben notificaciones del servidor central para atender solicitudes de usuarios, a través de **ZeroMQ (Push-Pull)**.

2. Comunicación entre Usuarios y Servidor

- **Solicitud de Servicio**: Los usuarios se comunican con el servidor usando **ZeroMQ (Request/Reply)** para solicitar taxis, enviando su posición y un identificador único.
- **Timeouts**: Si el servidor no responde dentro de un tiempo límite (e.g., 5 segundos), el usuario marca la solicitud como fallida y registra el evento. Este timeout previene solicitudes sin atender por falta de taxis o congestión.

3. Comunicación entre Servidor Central y Respaldo

- **Sincronización de Estado**: El servidor de respaldo mantiene su estado sincronizado con el servidor central mediante un mecanismo de replicación continua. Esto se logra utilizando un **protocolo de replicación asíncrona**, como **write-ahead logging (WAL)**, que permite que cada cambio en el estado del servidor central (e.g., actualizaciones de posición de taxis, asignaciones de servicios) sea inmediatamente replicado al respaldo. Para asegurar la consistencia de los datos, se implementa un proceso de confirmación de transacciones (acknowledgments), por el cual el respaldo confirma la recepción de cada actualización. Si se produce una pérdida de conexión momentánea, el servidor de respaldo solicitará todos los eventos que no se hayan sincronizado una vez que la comunicación se restablezca.
- **Detección de Fallos y Conmutación por Error**: El proceso de health-check verifica el estado del servidor central enviando **heartbeats** (mensajes de "ping") cada 5 segundos. Si no se recibe una respuesta tras 3 intentos consecutivos (indicando que el servidor principal está inactivo o ha fallado), el servidor de respaldo se activa automáticamente y toma el control del sistema. Esto garantiza que las operaciones continúen sin interrupciones. La conmutación es transparente para los taxis y usuarios, quienes seguirán comunicándose con el servidor ahora respaldado, manteniendo la continuidad del servicio. Una vez que el servidor principal se recupere, se debe implementar un mecanismo para sincronizar cualquier cambio que se haya producido mientras el respaldo estaba activo, antes de devolver el control al servidor principal.

Modelo de Fallos

1. Tolerancia a Fallos del Servidor Central

- **Servidor de Respaldo**: Mantiene sincronización de estado en tiempo real con el servidor principal.

- **Conmutación por Error**: Si el health-check detecta una falla (3 intentos fallidos de respuesta), el respaldo asume el control automáticamente, continuando las operaciones sin interrupciones.

2. Gestión de Fallas de Taxis y Usuarios

- Si un taxi deja de enviar su posición o falla durante un servicio, el servidor central lo marca como inactivo y reasigna el servicio.
- Si un usuario no recibe un taxi dentro del `timeout`, el hilo termina y registra la solicitud como fallida, simulando que el usuario busca otro proveedor.

Modelo de Seguridad

1. Autenticación y Autorización

- **Taxis**: Cada taxi tiene un identificador único, y el servidor valida su autenticidad (e.g., con una llave API para asegurar que solo taxis registrados se comuniquen).
- **Usuarios**: Los hilos de usuarios se identifican con sus credenciales y sus solicitudes son autenticadas por el servidor.
- **Control de Acceso y Sesiones**: Tokens de sesión temporales se implementan para verificar la legitimidad de las solicitudes.

2. Integridad y Confidencialidad de Datos

- **Cifrado de Comunicación**: Todas las comunicaciones entre taxis, usuarios y servidor se cifran con **ZeroMQ (CurveZMQ)** para garantizar la privacidad.
- **Validación de Datos Recibidos**: El servidor valida que las coordenadas de taxis y solicitudes estén dentro de los límites de la cuadrícula y cumplan con el formato correcto.

3. Disponibilidad

- **Tolerancia a Fallos**: Una arquitectura redundante con un servidor de respaldo asegura la disponibilidad del servicio.
- **Timeouts**: Se implementan para liberar recursos y finalizar hilos inactivos si el servidor no responde, garantizando la continuidad de operaciones.

---

2. Diseño del Sistema

Diagrama de Despliegue

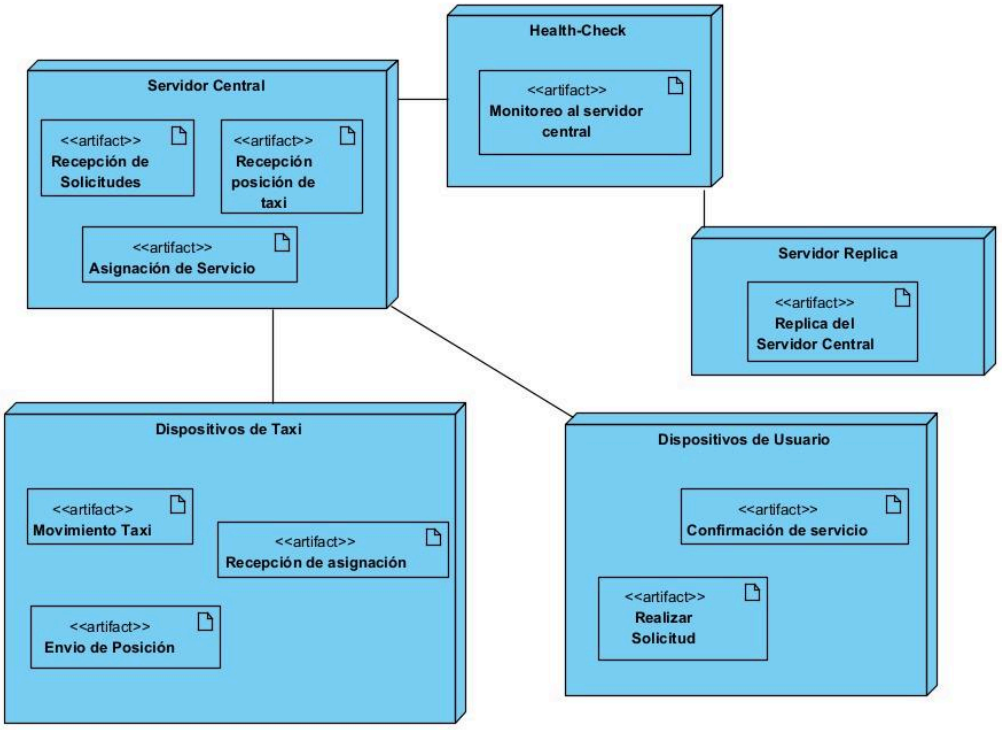

Imagen 1. Diagrama de Despliegue - Autoría Propia.

Descripción:

- **Servidor Central (IP1)**: Gestiona todas las operaciones del sistema, incluyendo la ubicación de taxis, solicitudes de usuarios, y asignación de taxis a servicios. Abre puertos TCP dedicados para taxis (e.g., TCP port 5555) y para usuarios (e.g., TCP port 5556), asegurando una separación clara de los mensajes.
- **Servidor de Respaldo (IP2)**: Actúa como respaldo del servidor central, ejecutando sincronización continua de estado para tolerancia a fallos. Funciona en una máquina separada con una IP diferente.
- **Proceso de Health-Check (IP3)**: Monitorea el servidor central para detectar fallas. Si el servidor central falla, el health-check activa el servidor de respaldo. Puede residir en una tercera máquina para asegurar independencia de operación y evitar un punto único de fallo.
- **Taxis (IP4, IP5)**: Cada taxi es un proceso independiente, desplegado en una o varias máquinas, lo que permite simular múltiples taxis en la cuadrícula de la ciudad. Los taxis se comunican directamente con el servidor central para enviar su posición y recibir asignaciones de servicios.
- **Usuarios (IP6)**: Un proceso generador de usuarios crea hilos que simulan solicitudes de servicios de taxi. Los usuarios envían solicitudes de taxi al servidor central y esperan una respuesta para el servicio. Cada solicitud y su respuesta se manejan a través de ZeroMQ.

Conexiones:

- **Taxis al Servidor Central**: Comunicación a través de ZeroMQ con el patrón Publisher-Subscriber para enviar posiciones y recibir actualizaciones de estado. Adicionalmente, el patrón Push-Pull puede utilizarse para enviar órdenes de asignación.

- **Usuarios al Servidor Central**: Comunicación a través de ZeroMQ con el patrón Request-Reply para enviar solicitudes de taxi y recibir confirmaciones o rechazos.
- **Servidor Central y Respaldo**: Comunicación mediante sockets TCP para sincronizar datos en tiempo real y estados del sistema, asegurando que el servidor de respaldo siempre esté listo para tomar el control.

Diagrama de Componentes

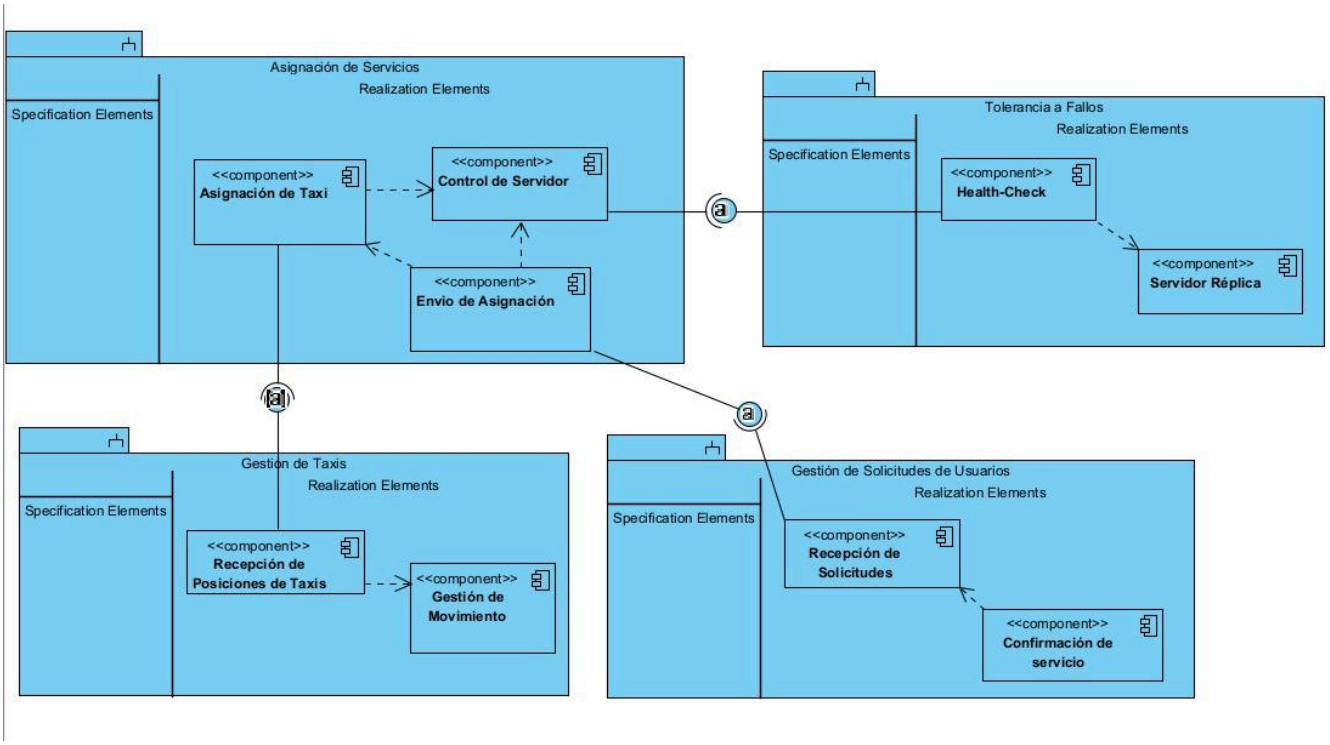

Imagen 2. Diagrama de Componentes - Autoría Propia.

Componentes Principales:

- **Servidor Central**:
  - **Módulo de Registro de Taxis**: Gestiona el registro inicial de taxis y guarda su información relevante (ID, posición inicial, estado).
  - **Módulo de Recepción de Posiciones**: Recibe actualizaciones periódicas de la posición de cada taxi.
  - **Módulo de Gestión de Solicitudes de Usuarios**: Recibe solicitudes de taxis de los usuarios, maneja el tiempo de espera (timeout) y calcula el taxi más cercano.
  - **Módulo de Asignación de Taxis**: Asigna un taxi disponible a un usuario, comunicándose con el taxi seleccionado para notificarle del nuevo servicio.
  - **Módulo de Persistencia de Datos**: Almacena datos históricos y actuales, como posiciones de taxis, servicios realizados, y respuestas a usuarios, en una base de datos o archivo seguro.
- **Servidor de Respaldo**:
  - **Sincronización de Estado**: Replica la misma estructura de módulos que el servidor central y recibe actualizaciones de estado en tiempo real para poder asumir el control si se detecta una falla.
- **Proceso de Health-Check**:

    - **Módulo de Monitoreo**: Verifica la disponibilidad del servidor central mediante heartbeats o pings periódicos.
    - **Módulo de Conmutación por Error**: Activa el servidor de respaldo si detecta una falla en el servidor principal tras tres intentos fallidos de conexión.
- **Taxi**:
    - **Módulo de Comunicación**: Gestiona la conexión con el servidor central para enviar actualizaciones de posición y recibir asignaciones de servicio.
    - **Módulo de Movimiento y Actualización de Posición**: Calcula la nueva posición del taxi en la cuadrícula de la ciudad y actualiza su estado basado en su velocidad y dirección.
    - **Módulo de Recepción de Asignaciones**: Recibe y maneja asignaciones de servicios provenientes del servidor central, cambiando el estado del taxi a "ocupado".
- **Generador de Usuarios**:
    - **Módulo de Creación de Hilos**: Crea hilos para representar a cada usuario en la cuadrícula, generando solicitudes de servicio.
    - **Módulo de Solicitud de Servicio**: Gestiona la comunicación entre los hilos de usuarios y el servidor central para solicitar taxis.

Diagrama de Clases

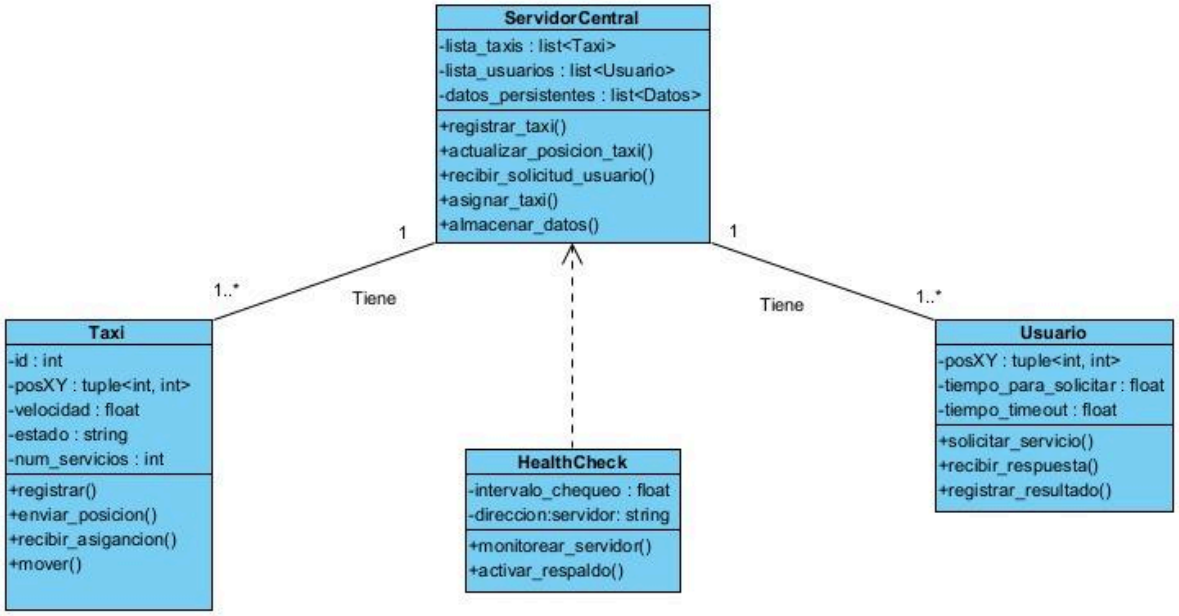

Imagen 3. Diagrama de Clases- Autoría Propia.

Clases Principales:

1. **Taxi**:
    - **Atributos**:
        - `id`: Identificador único del taxi.
        - `posicion (x, y)`: Coordenadas actuales del taxi en la cuadrícula.
        - `velocidad`: Velocidad de movimiento del taxi en km/h.
        - `estado`: Estado actual (disponible/ocupado).
        - `num_servicios`: Número de servicios completados en el día.
    - **Métodos**:

        - `registrar()`: Registra el taxi con el servidor central.
        - `enviar_posicion()`: Envía la posición actual al servidor.
        - `recibir_asignacion()`: Recibe asignaciones de servicio desde el servidor.
        - `mover()`: Actualiza la posición del taxi en función de su velocidad y dirección.
2. **ServidorCentral**:
    - **Atributos**:
        - `lista_taxis`: Lista de taxis registrados y sus datos.
        - `lista_usuarios`: Lista de usuarios esperando servicio.
        - `datos_persistentes`: Datos de posición y servicios almacenados.
    - **Métodos**:
        - `registrar_taxi()`: Registra un nuevo taxi en el sistema.
        - `actualizar_posicion_taxi()`: Actualiza la posición de un taxi dado.
        - `recibir_solicitud_usuario()`: Recibe solicitudes de taxi de los usuarios.
        - `asignar_taxi()`: Calcula y asigna el taxi más cercano a un usuario.
        - `almacenar_datos()`: Guarda la información de servicios y movimientos.
3. **Usuario**:
    - **Atributos**:
        - `posicion (x, y)`: Coordenadas del usuario en la cuadrícula.
        - `tiempo_para_solicitar`: Tiempo que el usuario espera antes de solicitar un taxi.
        - `tiempo_timeout`: Tiempo máximo de espera para obtener una respuesta de servicio.
    - **Métodos**:
        - `solicitar_servicio()`: Envía una solicitud de servicio al servidor central.
        - `recibir_respuesta()`: Recibe la respuesta del servidor (asignación de taxi o rechazo).
        - `registrar_resultado()`: Guarda el resultado de la solicitud (satisfactorio o fallido).
4. **HealthCheck**:
    - **Atributos**:
        - `intintervalo_chequeo`: Intervalo de tiempo entre cada verificación de estado del servidor.
        - `direccion_servidor`: Dirección IP del servidor central a verificar.
    - **Métodos**:
        - `monitorear_servidor()`: Envía heartbeats al servidor central para verificar su estado.
        - `activar_respaldo()`: Activa el servidor de respaldo si se detecta una falla.

Diagrama de Secuencia

1. **Escenario: Taxi envía su posición al Servidor Central**:
    - Taxi envía su posición actual al servidor central cada 30 segundos de tiempo real.
    - Servidor central actualiza la posición del taxi en su base de datos y mantiene el registro actualizado para posibles asignaciones.

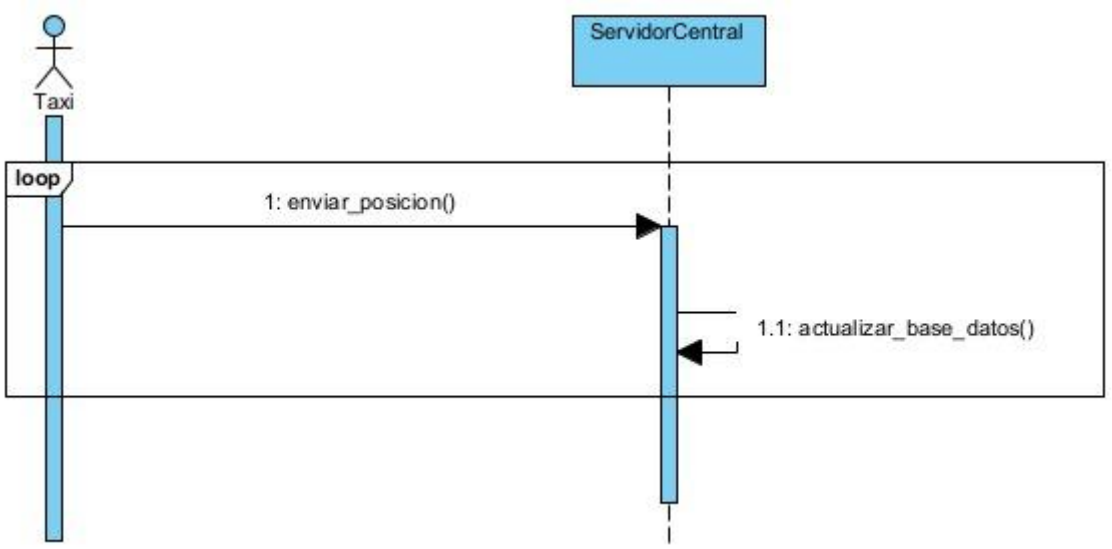

Imagen 4. Diagrama de Secuencia “Taxi envía su posición al Servidor Central” - Autoría Propia.

2. **Escenario: Usuario solicita un taxi**:
    - Usuario envía una solicitud de servicio al servidor central.
    - Servidor verifica la lista de taxis disponibles y calcula cuál está más cercano al usuario.
    - Servidor asigna el taxi más cercano al usuario, enviando una confirmación tanto al taxi como al usuario.
    - Taxi recibe la asignación, actualiza su estado a "ocupado" y se prepara para el servicio.

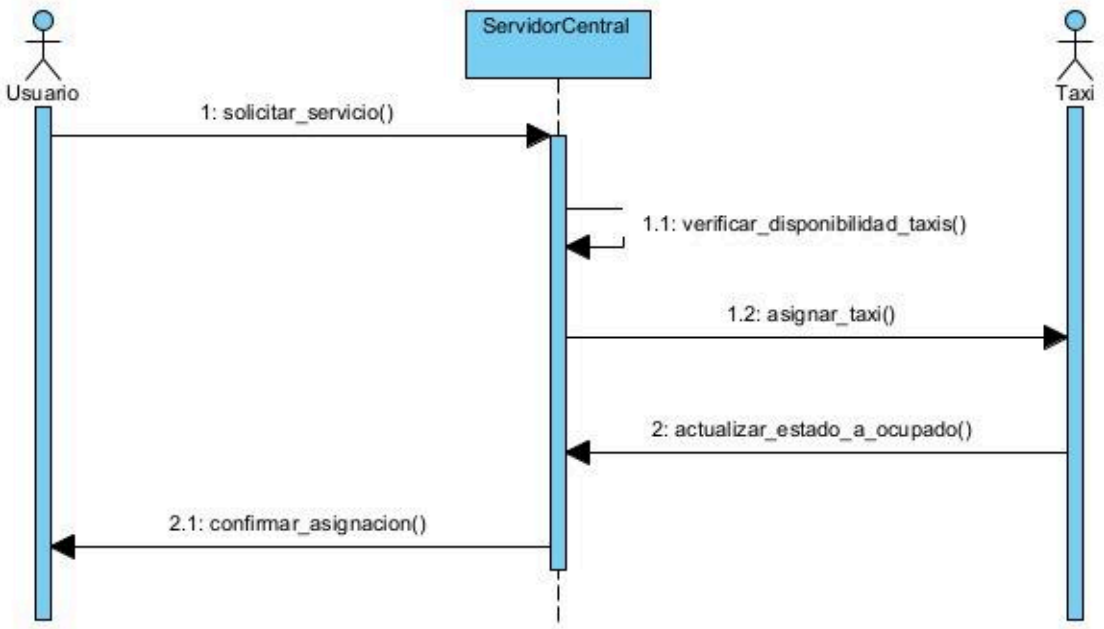

Imagen 5. Diagrama de Secuencia “Usuario solicita un taxi” - Autoría Propia.

3. **Escenario: Fallo del Servidor Central**:
    - Proceso de Health-Check detecta que el servidor central no responde tras 3 intentos consecutivos de comunicación.
    - Health-Check activa el servidor de respaldo, el cual asume todas las funciones del servidor principal.
    - Todos los taxis y usuarios continúan su comunicación con el servidor de respaldo sin interrupción del servicio, garantizando la continuidad operativa.

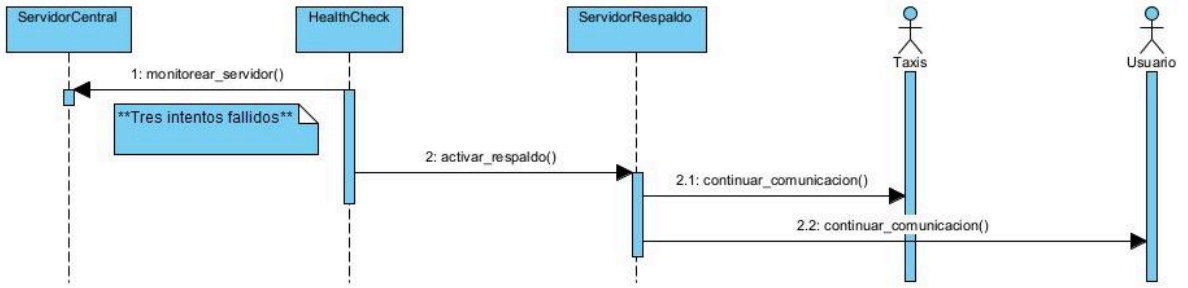

Imagen 6. Diagrama de Secuencia “Fallo del Servidor Central” - Autoría Propia.

---

3. Protocolo de Pruebas

Para garantizar el correcto funcionamiento del sistema My-Uber, se llevarán a cabo una serie de pruebas que abarcan diferentes niveles de la arquitectura del sistema. Se utilizarán tanto pruebas unitarias como de integración, de sistema, de rendimiento, de seguridad y de tolerancia a fallos. A continuación, se detallan los planes para cada tipo de prueba, junto con sus objetivos y enfoques específicos:

Pruebas Unitarias

Estas pruebas se centrarán en verificar la funcionalidad de los componentes individuales del sistema.

1. **Taxi**:
    - **Registro en el Servidor**: Verificar que cada taxi se registre correctamente en el servidor central al inicio del proceso.
    - **Envío de Posiciones**: Asegurar que los taxis envíen sus posiciones periódicamente al servidor central según el intervalo definido.
    - **Recepción de Asignaciones**: Confirmar que los taxis reciben las asignaciones de servicios correctamente y cambian su estado a "ocupado" de manera apropiada.
    - **Simulación de Movimiento**: Comprobar que la función `mover()` actualiza la posición del taxi de acuerdo con su velocidad y dirección.
2. **Servidor Central**:
    - **Registro de Taxis**: Validar que el servidor central recibe y almacena correctamente los registros de taxis.
    - **Actualización de Posiciones**: Comprobar que el servidor actualiza correctamente las posiciones de los taxis conforme a la información recibida.
    - **Asignación de Taxis**: Verificar que el servidor selecciona el taxi más cercano a la ubicación del usuario y asigna correctamente el servicio.
    - **Persistencia de Datos**: Confirmar que el servidor almacena de forma correcta y segura todos los eventos relevantes (posiciones de taxis, servicios asignados).
3. **Usuarios**:
    - **Solicitud de Servicio**: Verificar que cada usuario envía una solicitud de taxi correctamente.
    - **Recepción de Respuestas**: Comprobar que los usuarios reciben la respuesta del servidor (asignación de taxi o rechazo) y que manejan adecuadamente los casos de éxito y de timeout.
    - **Registro de Resultados**: Confirmar que cada usuario guarda correctamente el resultado de su solicitud (satisfactorio o fallido).
4. **Proceso de Health-Check**:
    - **Detección de Fallas**: Asegurar que el health-check detecta la falta de respuesta del servidor principal después de 3 intentos fallidos de comunicación.
    - **Conmutación por Error**: Verificar que, en caso de falla, el health-check activa el servidor de respaldo y este asume las funciones sin pérdida de información.

Pruebas de Integración

Estas pruebas verifican la interacción entre los diferentes componentes y el flujo completo del sistema.

1. **Comunicación entre Taxis y Servidor Central**:
    - **Flujo Completo**: Probar el proceso completo desde el registro inicial del taxi en el servidor hasta la recepción de asignaciones de servicio, incluyendo la actualización periódica de posiciones.
    - **Sincronización de Estado**: Validar que el servidor de respaldo recibe correctamente las actualizaciones del servidor central.
2. **Comunicación entre Usuarios y Servidor Central**:

    - **Solicitud y Asignación de Servicios**: Probar el flujo completo desde que un usuario solicita un servicio, el servidor central asigna un taxi, y el usuario recibe la confirmación.
3. **Conmutación por Error (Failover)**:
    - **Falla del Servidor Central**: Simular una falla en el servidor central y verificar que el respaldo toma el control sin interrupciones, asegurando que todas las solicitudes y actualizaciones continúan procesándose correctamente.
    - **Verificación de Sincronización**: Confirmar que después de la conmutación, el servidor de respaldo tiene el estado actualizado y consistente.

Pruebas de Sistema

Estas pruebas aseguran que el sistema completo funcione correctamente en un entorno distribuido real.

1. **Despliegue en Múltiples Máquinas**:
    - **Distribución de Componentes**: Ejecutar taxis en al menos dos máquinas diferentes, el servidor central en otra, y los usuarios en una cuarta máquina. Verificar la comunicación efectiva y la coordinación entre los componentes distribuidos.
    - **Rendimiento de Red**: Asegurar que la comunicación entre los diferentes componentes es estable y eficiente, con tiempos de latencia dentro de los límites aceptables.

Pruebas de Rendimiento

Estas pruebas medirán la capacidad de respuesta y escalabilidad del sistema bajo diferentes condiciones de carga.

1. **Mediciones de Tiempo de Respuesta**:
    - **Tiempos de Asignación**: Registrar el tiempo desde que un usuario envía una solicitud hasta que recibe la respuesta del servidor con la asignación de un taxi.
    - **Análisis de Tiempos Mínimos, Máximos y Promedios**: Realizar un análisis detallado de los tiempos de respuesta para diferentes escenarios (bajo carga normal, alta carga).
2. **Pruebas de Carga y Escalabilidad**:
    - **Incremento de Taxis y Usuarios**: Aumentar progresivamente el número de taxis y usuarios en la cuadrícula, observando cómo afecta al tiempo de respuesta y al rendimiento general del sistema.
    - **Medición de Uso de Recursos**: Monitorear el uso de CPU, memoria y red en el servidor central, taxis y usuarios para garantizar que el sistema pueda manejar el incremento de carga.

Pruebas de Seguridad

Estas pruebas validarán la capacidad del sistema para manejar datos de manera segura y evitar ataques maliciosos.

1. **Validación de Datos**:

    - **Datos Inválidos**: Enviar datos mal formados o inválidos (e.g., coordenadas fuera de la cuadrícula) y verificar que el sistema los rechaza correctamente.
    - **Control de Acceso**: Intentar registrar un taxi o usuario con identificadores duplicados o no autorizados y comprobar que el sistema no permite el acceso.
2. **Pruebas de Cifrado y Comunicación Segura**:
    - **Cifrado de Mensajes**: Confirmar que todos los mensajes enviados entre taxis, usuarios y servidores están cifrados utilizando `CurveZMQ`.
    - **Prevención de Ataques (opcional)**: Simular intentos de intercepción de mensajes o ataques de "man-in-the-middle" para asegurar que la comunicación permanece segura.

Pruebas de Tolerancia a Fallos

Estas pruebas verifican que el sistema puede recuperarse de fallas y continuar operando sin pérdida de datos ni interrupciones.

1. **Falla del Servidor Central**:
    - **Simulación de Desconexión**: Apagar el servidor principal abruptamente y verificar que el servidor de respaldo se activa y continúa todas las operaciones desde el punto donde se dejó.
    - **Verificación de Integridad de Datos**: Asegurar que todos los datos de posición de taxis, servicios asignados y registros se mantienen intactos durante la conmutación.
2. **Falla de un Taxi**:
    - **Simulación de Desconexión de Taxi**: Desconectar un taxi de manera abrupta y verificar que el servidor lo marca como inactivo, re-ubicando la solicitud de servicio a otro taxi disponible.
    - **Reconexión de Taxi**: Simular la reconexión de un taxi previamente desconectado y verificar que se reincorpora correctamente al sistema.
3. **Falla del Health-Check (opcional)**:
    - **Prueba de Redundancia del Health-Check**: Simular una falla en el proceso de health-check y verificar que hay un proceso de respaldo (si se implementa) que continúa monitoreando el servidor central.
    -

---

4. Obtención de Métricas de Desempeño

La recopilación y análisis de métricas de desempeño se enfocarán en medir la eficiencia y la efectividad del sistema My-Uber. A continuación, se detallan los métodos y herramientas que se utilizarán para registrar, analizar y presentar las métricas relevantes.

Logs y Registros

1. **Registro de Eventos Críticos**:
    - **Formato y Contenido**: Cada evento importante (registro de taxi, solicitud de usuario, asignación de servicio, actualización de posición) se registrará en archivos de log, usando formato JSON o CSV para asegurar uniformidad y facilidad de análisis.

- **Uso de Timestamps**: Para cada evento, se utilizará `time.time()` de Python para marcar el inicio y fin de la operación, permitiendo medir los tiempos de respuesta y otros indicadores en milisegundos.
- **Nivel de Detalle**: Los logs incluirán detalles como:
  - ID del taxi o usuario.
  - Coordenadas de posición (para taxis y usuarios).
  - Estado de la solicitud (en proceso, completada, fallida).
  - Tiempos de inicio y fin de cada operación.

2. **Estructura de Almacenamiento de Logs**:
   - Los logs se generarán en archivos separados por tipo de evento (e.g., `taxis_logs.json`, `usuarios_logs.json`, `servidor_logs.json`) para facilitar su análisis posterior.
   - Se podría optar por almacenar estos logs en una base de datos ligera (e.g., SQLite) para búsquedas y filtrado eficientes, especialmente cuando el número de eventos es muy alto.

Medición de Tiempo de Respuesta

1. **Cálculo de Tiempos de Respuesta para Solicitudes de Usuario**:
   - **Inicio de Medición**: Cuando un usuario envía una solicitud de taxi al servidor central, se marca el tiempo de inicio de la operación.
   - **Fin de Medición**: El tiempo de fin se marca cuando el usuario recibe una respuesta (asignación de taxi o rechazo). La diferencia entre ambos tiempos representa el tiempo de respuesta del servidor.
   - **Registro de Resultados**: Cada tiempo de respuesta se guarda en el log del usuario, con información detallada sobre la solicitud, el tiempo de espera y el estado final (satisfactorio o fallido).
2. **Medición de Tiempos para Asignación de Servicios**:
   - El servidor central registrará el tiempo desde que recibe una solicitud de servicio hasta que asigna un taxi.
   - Si hay demoras significativas en la asignación (por ejemplo, debido a un alto número de solicitudes simultáneas), se registrará esta información para análisis de desempeño y posibles cuellos de botella.

Cantidad de Respuestas Exitosas y No Exitosas

1. **Respuestas Exitosas (Asignación de Taxi)**:
   - Cada vez que un taxi es asignado a un usuario de manera satisfactoria, se registra esta operación tanto en el servidor central como en el log del usuario.
   - Los datos incluyen la identificación del taxi asignado, la posición inicial del usuario, y el tiempo total de respuesta.
2. **Respuestas No Exitosas (Rechazos y Timeouts)**:
   - Se registran todas las solicitudes que no pudieron ser atendidas, ya sea porque no había taxis disponibles o porque se alcanzó el tiempo límite (timeout) antes de recibir una respuesta.
   - Estos eventos se registrarán en logs separados para análisis detallado de los motivos de fallas.
3. **Almacenamiento de Datos**:

- Tanto las respuestas exitosas como las no exitosas se guardarán en archivos de log detallados o en una base de datos, con una estructura que permita identificar tendencias y analizar la causa raíz de las fallas.

Herramientas de Monitoreo y Análisis

1. **Logs Detallados y Dashboard (Opcional)**:
   - Se utilizarán herramientas de logging como `logging` en Python para crear registros detallados en el servidor, taxis y usuarios.
   - **Herramientas de Monitoreo en Tiempo Real**: Si se desea obtener una visión más dinámica del rendimiento, se puede integrar una herramienta de monitoreo como **Prometheus** para recolectar métricas de desempeño y generar dashboards con visualizaciones en tiempo real de los tiempos de respuesta y estadísticas de solicitudes.
2. **Análisis Automatizado de Logs**:
   - Un proceso automatizado se ejecutará para analizar los logs y calcular métricas clave como tiempos promedio, mínimos y máximos de respuesta, así como tasas de éxito y fallas.
   - Se pueden crear scripts en Python para leer los archivos de log y producir reportes detallados con información gráfica (por ejemplo, usando `matplotlib` o `pandas`).

Análisis Posterior de Desempeño

1. **Cálculo de Métricas de Tiempos de Respuesta**:
   - **Tiempo Promedio**: Se calculará el tiempo promedio de respuesta para las solicitudes de servicio exitosas, asegurando que esté dentro de los límites aceptables de desempeño.
   - **Tiempos Mínimos y Máximos**: Se analizarán los tiempos mínimos y máximos de respuesta para identificar posibles anomalías o variaciones inusuales en el rendimiento.
   - **Distribución de Tiempos de Respuesta**: Se creará un histograma de los tiempos de respuesta para entender la distribución y detectar patrones.
2. **Reporte de Estadísticas de Solicitudes**:
   - **Número de Servicios Exitosos**: Contabilizar la cantidad de servicios completados con éxito y determinar si hay correlación con el número de taxis y la carga del sistema.
   - **Número de Rechazos y Timeouts**: Analizar los eventos fallidos y determinar las causas (por ejemplo, falta de taxis disponibles o fallas de comunicación).
3. **Generación de Reportes Gráficos y Dashboards**:
   - Crear reportes gráficos que presenten claramente los resultados de las métricas de desempeño (e.g., tiempos de respuesta, tasa de éxito/falla).
   - Si se utiliza una herramienta como **Prometheus**, se pueden generar dashboards con visualizaciones en tiempo real y estadísticas históricas para facilitar la interpretación de los datos.
4. **Evaluación de Uso de Recursos (Opcional)**:
   - Monitorear el uso de CPU, memoria y tráfico de red en los diferentes componentes (servidor, taxis, usuarios) durante las pruebas de carga para detectar posibles cuellos de botella.

- Presentar un análisis sobre la capacidad de escalabilidad del sistema basado en el uso de recursos y la respuesta bajo diferentes cargas de trabajo.