UNIVERSIDAD DE BUENOS AIRES

FACULTAD DE INGENIERÍA

75.74 Sistemas Distribuidos I

Trabajo práctico 1

1° cuatrimestre 2022

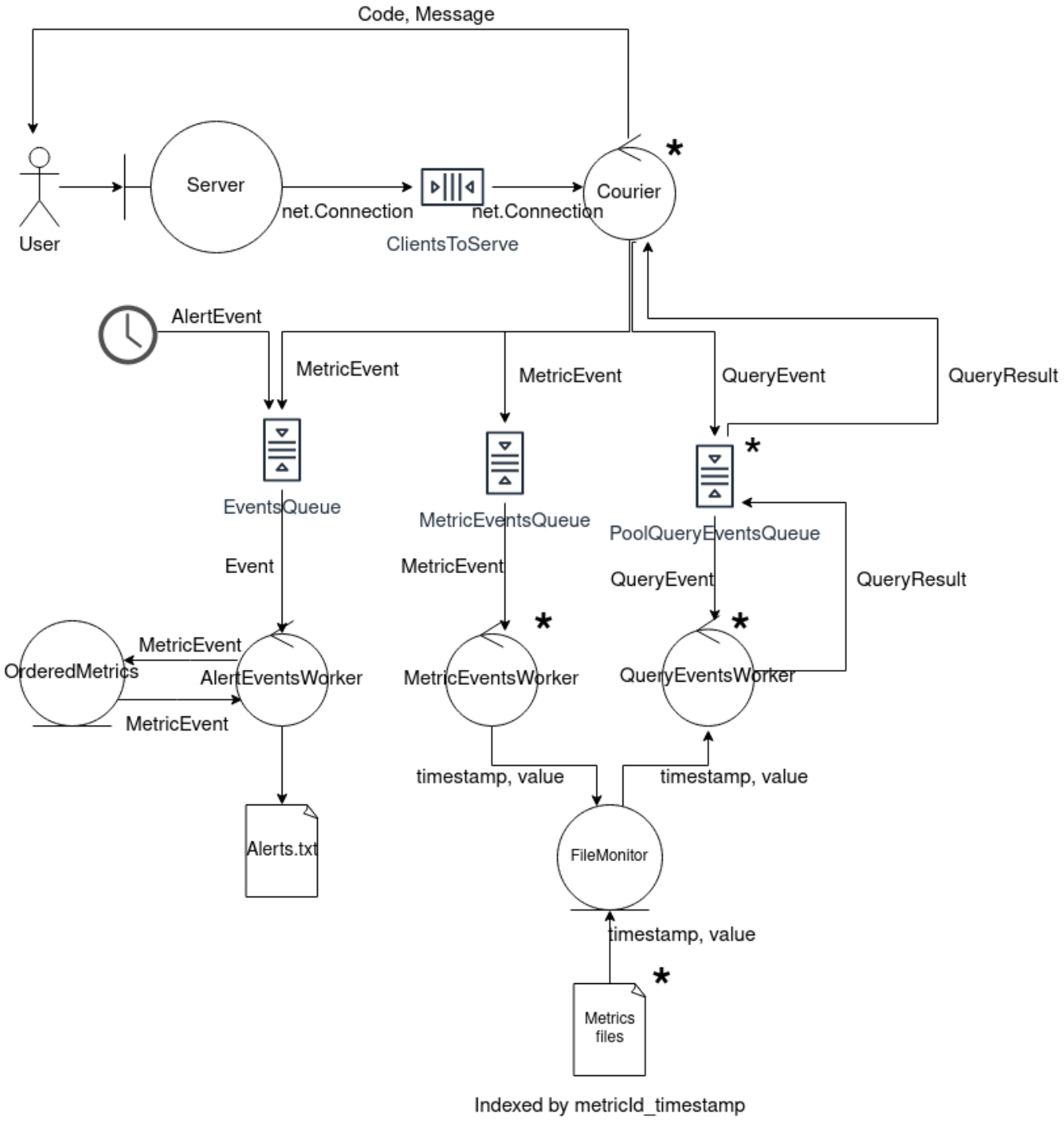
Apellido y Nombre	Padrón	Mail
Del Carril, Manuel	100772	mdelcarril@fi.uba.ar

Introducción

El presente trabajo práctico tiene como objetivo la implementación de un servidor de métricas y alertas aplicando los conocimientos adquiridos en clase sobre protocolos y concurrencia.

Arquitectura

El siguiente diagrama permite visualizar las distintas entidades que conforman el sistema:



Como se puede observar, el punto de entrada del sistema es *Server,* quien se encuentra a la escucha de conexiones. Dichas conexiones serán almacenadas en una cola para que los distintos *Couriers* se encarguen de atender el pedido.

Una vez un *Courier* se encuentra disponible, decodificará el mensaje recibido. Según el tipo, enviará a la cola correspondiente, para que sea delegado en el *Worker* adecuado. Únicamente se queda esperando un mensaje de respuesta por parte del *Worker* en caso de que sea una *Query*.

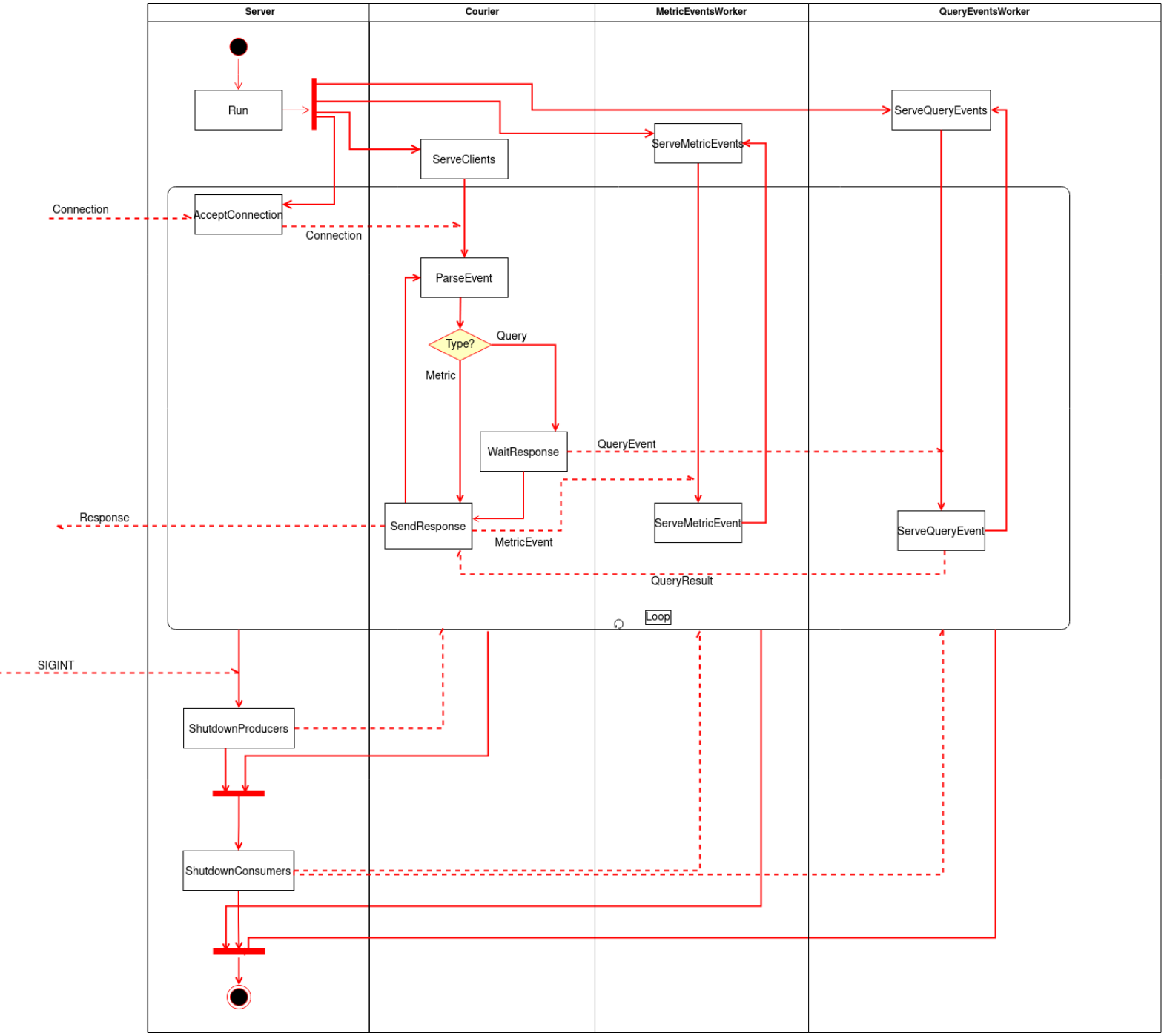
MetricEventsWorker

Es el encargado de procesar las métricas que desean agregarse al sistema. Recibe un MetricEvent conformado por un timestamp (en formato UNIX), un metric Id y un valor.
Para la escritura, se ayuda del *FileMonitor*, quien resguarda el acceso a los archivos. No devuelve ningún valor al *Courier* porque el sistema garantiza consistencia eventual.

QueryEventsWorker

Es el encargado de procesar los *QueryEvent*, (conformados por un metric Id, tipo de agregación, ventana de agregación y un intervalo de tiempo) pasadas por el *Courier*. Las reciben a través de una cola única y responden el resultado a través de ella. Accede a los archivos a través del *FileMonitor*.

Esta dinámica recién descrita se puede ver en el siguiente diagrama de estados:



Se puede ver como el servidor crea estos Workers al inicio y a medida que va recibiendo conexiones nuevas, se va generando este pasaje de manos de la información relatado anteriormente. También podemos observar que sucede cuando llega una señal de terminación al sistema, cambiando de estado al hilo *Server* y generando la emisión de comandos para que cada hilo cuando termine la tarea que se encuentra realizando en el momento, cierre ordenadamente.

FileMonitor

Es quien protege el acceso a los archivos para evitar caer en inconsistencias a causa de la concurrencia en lecturas y escrituras. El sistema de archivos se encuentra particionado por id de métrica y por minuto.

AlertEventsWorker

Es quien se encarga de detectar y disparar las alertas. Para ello, recibe todos los *MetricEvents* que el *Courier* cree para almacenar en forma ordenada. Una vez le llegue la petición de procesar las alertas (enviadas por *ClockWorker*), calculará para cada métrica almacenada un intervalo de tiempo en el cual aplicará la función de agregación pedida y lo comparará con el límite pedido.

Protocolos

Para los protocolos de recepción de eventos se optó por mensajes binarios (ya sea para recibir los pedidos del servidor como para escribir los archivos de métricas) para buscar optimizar el tamaño de los mensajes además de su lectura (a través de reducir la cantidad de veces que se accede al file descriptor en cuestión).

Para ello, siempre se buscó implementar valores de tamaño fijo conocido (como los distintos números que se utilizan) y si era necesario un valor variable (como es el caso de metric id por ejemplo) se colocaba un entero de tamaño conocido indicando el largo de dicho campo.

Por otro lado, para la escritura de las métricas, se escribía en el archivo correspondiente el valor UNIX del timestamp de la métrica y su valor. De esta manera, cada métrica siempre tiene un largo fijo lo cual hace más fácil la lectura y el parseo.

Por último, dado que el archivo de alertas tiene más sentido que sea leído por una persona, se escribió cada alerta como una línea de texto.

Concurrencia

Para la resolución de la concurrencia en la atención de pedidos por parte de los usuarios, se utilizaron hilos para recibir mensajes, decodificarlos y procesarlos en forma paralela.

Cada hilo tiene su función dentro del sistema con responsabilidades claras y cuya tarea es generalmente delegada por un hilo superior. Para evitar la generación de hilos ilimitada ante el crecimiento de pedidos, se buscó definir una cantidad fija de cada tipo de

trabajador, donde si la carga de trabajo excede la capacidad de este grupo, la tarea será rechazada.

Para sincronizar estos distintos hilos, se usaron colas bloqueantes. Esto resultó particularmente útil porque permite separar las responsabilidades en capas donde entre capa y capa tengo 1 o más colas bloqueantes. Estos canales almacenan las tareas pendientes a ser completadas por la capa inferior y además permiten ser utilizados como medio para devolver el resultado de dicha tarea.

El único lugar donde no se implementó este sistema fue en el acceso a archivos, donde se le brinda a cada hilo que lo necesite, una referencia a la estructura que encapsula el acceso a archivos. Es decir, se hizo uso de memoria compartida protegida a través de *Locks* para evitar problemas. El motivo de esta decisión fue pura y exclusivamente por cuestiones de simplicidad. Además, para permitir un fácil acceso y evitar tiempos altos de bloqueo por acceso a archivos, se definió una granularidad de los mismos por métrica y por minuto.

Conclusiones

Se logró la implementación de un sistema capaz de procesar métricas y generar alertas con cierto grado de concurrencia y escalabilidad. Además se aplicaron los conocimientos teóricos aprendidos en clase de la manera en que se consideró conveniente.