FACULTAD
DE INGENIERIA
Universidad de Buenos Aires

75.74 - Sistemas Distribuidos I

TP1 - Concurrencia y Comunicaciones

Metrics && Alert Server

1° Cuatrimestre de 2022

Eliana Gamarra - 100016

Introducción

En el siguiente informe se presenta una solución a un sistema distribuido para el monitoreo de métricas y un sistema de alertas.
Los usuarios se podrán comunicar con el servidor para enviar métricas y hacer consultas de agregación.
A su vez el servidor se encargará de escribir en un archivo de logs las alertas para estas métricas en caso de exceder cierto valor límite previamente configurado.

A continuación se presentan una serie de diagramas para facilitar el entendimiento del sistema.

Diagrama de robustez

Diagrama de los distintos procesos involucrados y de qué manera se comunican por medio de queues.
Para implementar los procesos se uso la libreria Process y a la izquierda se tiene el punto de entrada del usuario a la app, luego el request handler va a procesar el pedido que este haga por una comunicación por socket TCP, ya sea para reportar una nueva métrica o hacer una consulta de agregación que se encolan en queue_report y queue_query respectivamente.

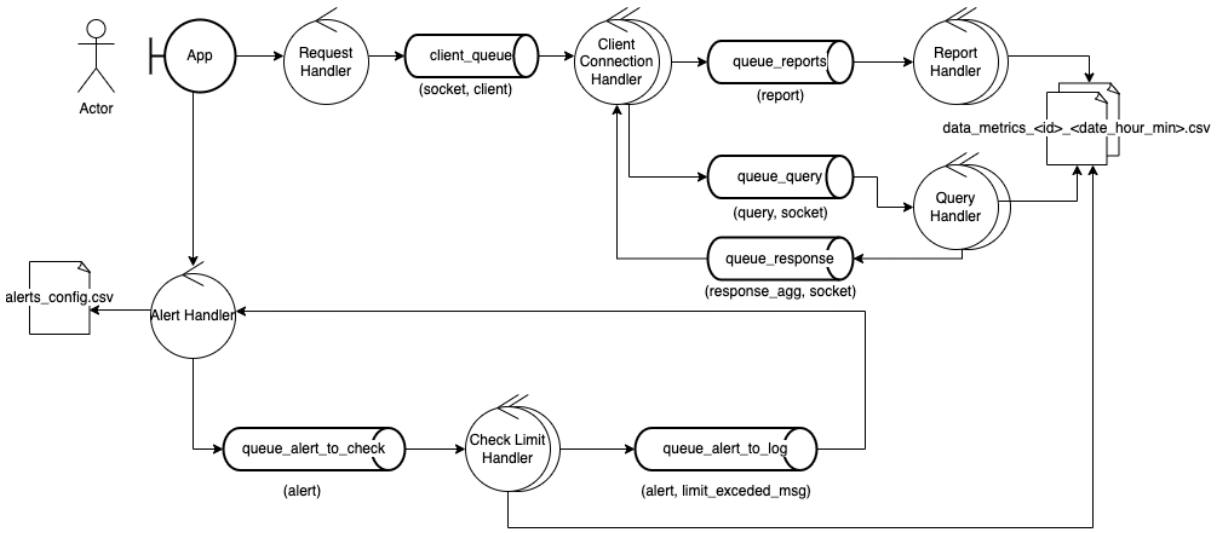
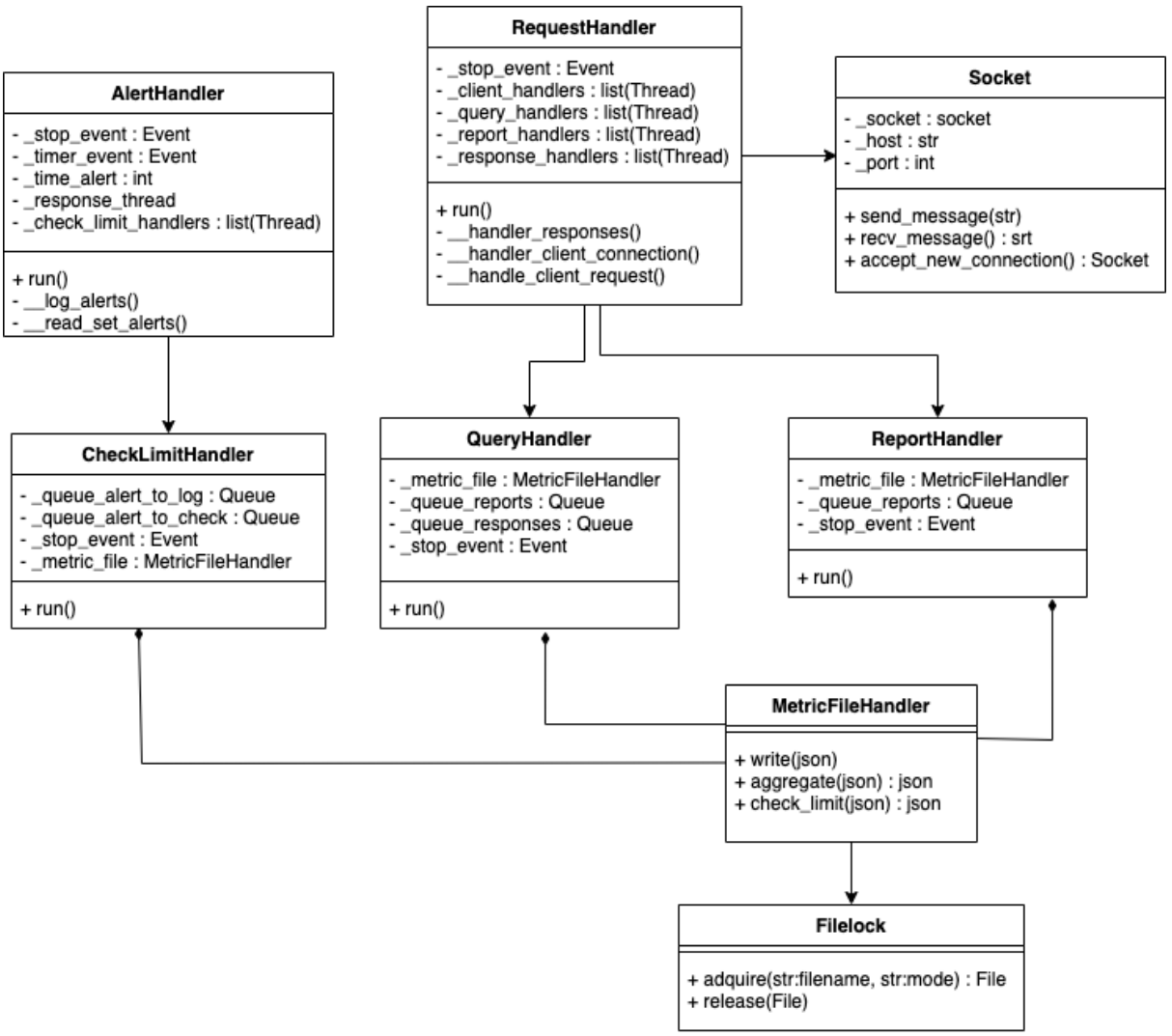


Diagrama de clases



Se implementaron las clases:

- **RequestHandler**
  Se encarga de aceptar las conexiones de los clientes y procesar los mensajes enviados para luego encolarlos en las queue correspondientes que se encargan de realizar las operaciones de escritura o lectura según corresponda a los archivos de métricas para luego enviar al cliente las respuestas a estos mensajes según corresponda.
- **ReportHandler**:
  Se encarga de leer *queue_report* y procesar el reporte que se obtuvo
- **QueryHandler**
  Se encarga de leer *queue_query* y procesar la query que se obtuvo leyendo el archivo de métricas y aplicando las agregaciones pedidas por el cliente. La respuesta luego se encola en queue_responses
- **MetricFileHandler**

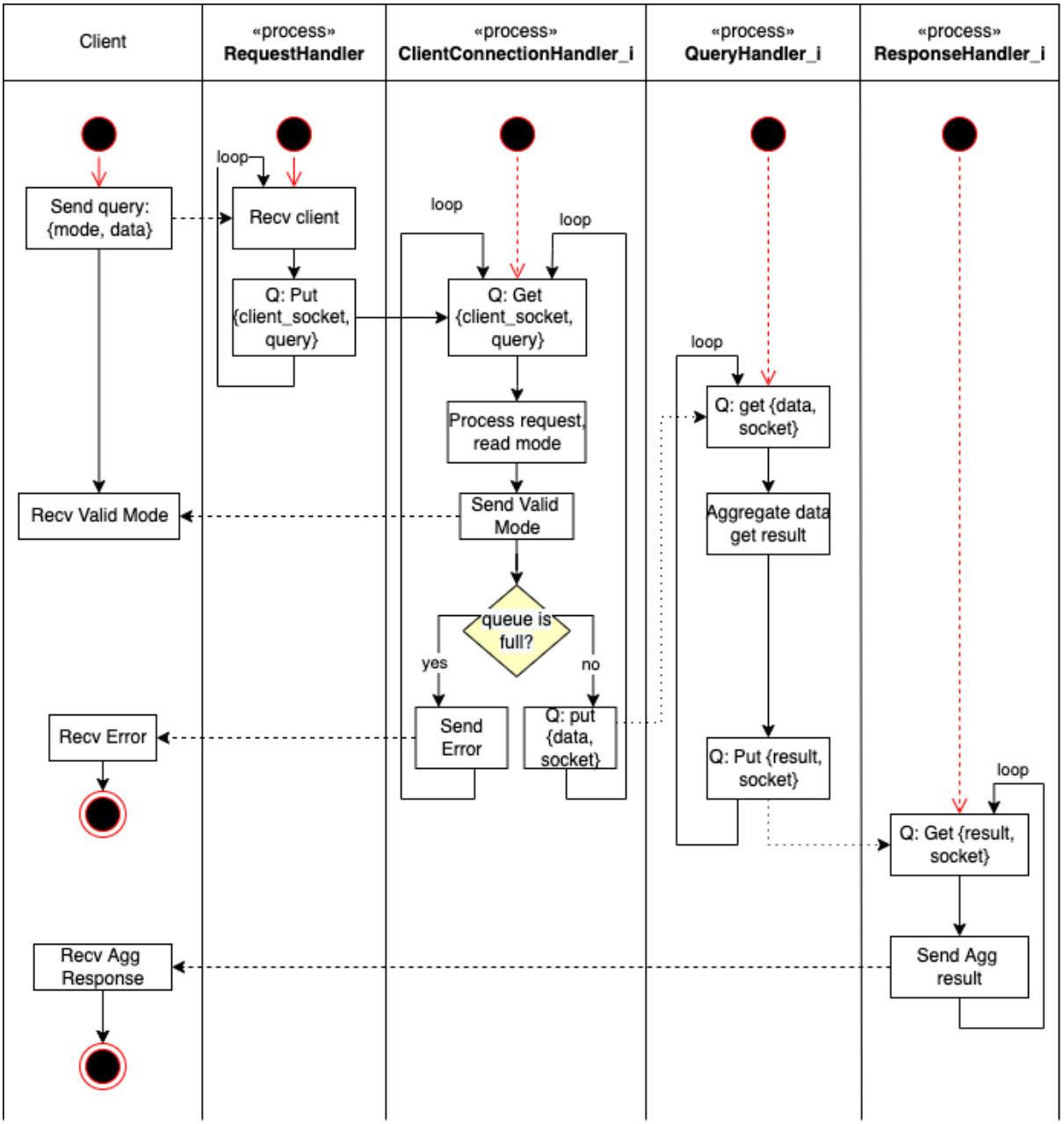
Se encarga de escribir y leer los archivos de métricas para guardar nuevos reportes o leer los datos y realizar las consultas de agregación.
Los archivos son guardados con el formato:
metric_file_<id>_<date>_<hour>-<minutes>

- **AlertHandler**
  Se encarga, cada 1 minuto de leer el archivo de alertas configuradas y escribirlas en la *queue_check_limit* por cada una de ellas. También si alguna métrica superó el valor límite establecido esta es leída de *queue_alert_to_log* y escribe en el archivo de logs una alerta por exceder este rango
- **CheckLimitHandler**
  Se encarga de leer las alertas encoladas en queue_alert_to_check
- **Socket**
  Encapsula el comportamiento del socket, así como los métodos de enviar y recibir mensajes.
  Para esto primero se envía la longitud del mensaje, un tipo int encodeado en 4 bytes y luego se envía el mensaje. Al recibir se obtiene la longitud y luego se recibe el mensaje hasta completar todos los bytes del mensaje anterior.
- **Filelock**
  Crea un lock para los archivos de métricas que impide que estos quieran ser accedidos más de una vez al mismo tiempo por distintos procesos.

Concurrencia

Diagrama de Actividad

Para mostrar la concurrencia se realizó un diagrama de actividades del momento en el que un cliente hace una consulta de agregación
Para eso se representaron los envíos de datos por socket con líneas a rayas y la comunicación entre procesos por medio de queues con líneas punteadas.



Manejo de archivos

Los archivos son guardados con el formato: *metric_file_<id>_<date>_<hour>-<minutes>.csv*

Estos son escritos por la clase **MetricFileHandler** que es instanciada en múltiples procesos del **ReportHandler.**
Para manejar una escritura segura de estos archivos compartidos se usa un FileLock para que dos o más procesos no accedan al mismo tiempo

Se tiene en cuenta la afinidad de un worker para guardar los datos usando una función de hashing. A cada worker se le asigna un id y conoce la cantidad total de workers, cuando lee una métrica de la queue utiliza la función de hash, si hay afinidad guarda la métrica sino vuelve a encolar la métrica en la misma queue.

A su vez son leídas por los procesos de **QueryHandler** y **AlertLimitHandler**.